

The Dreamcast Audio 64 API

Table of Contents

1. The AICA Control Layer API.....	AUD-1
acSetMultiPlayMask.....Sets the bit masks for acDigiMultiPlay0	AUD-1
acDspLoadPrograms	Registers a dsp program bank with the driver.AUD-2
acDspLoadOutputMixer.....Registers an output mixer patch with the driver.	AUD-3
acClosePort.....Closes port previously opened.	AUD-4
acDigiSetCurrentPitch.....Change the playback rate (pitch) of audio stream.	AUD-5
acDigiSetVolume.....Adjusts direct output volume of DA stream.	AUD-6
acDigiSetPan.....Adjusts pan placement of a port.	AUD-7
acDigiSetMixer	Set port's audio signal to DSP mixer channel.AUD-8
acDigiSetQSoundAngle.....Sets Q-Sound position.	AUD-9
acDigiRequestEvent.....Used to generate an interrupt when a certain buffer position is reached.	AUD-10
acDigiPlay	Starts a buffer playing.AUD-11
acDigiStop.....Stops a port from playing.	AUD-12
acMidiOpen.....Open a MIDI Port buffer for SMF format 0 playback.	AUD-13
acMidiSetTonebank	Assign a MIDI Program Bank (tonebank) to an active bank.AUD-14
acMidiClose.....Release active MIDI port.	AUD-15
acMidiPlay.....Begins playback on opened MIDI port.	AUD-16
acMidiStop	Stops standard MIDI file playback on port.AUD-17
acMidiRequestEvent.....Generates interrupt to host upon MIDI port reaching specified address.	AUD-18
acMidiPause.....Pause active MIDI port.	AUD-19
acMidiResume.....Resumes playback on active MIDI port.	AUD-20
acMidiSetVolume.....Sets scaled volume setting for MIDI port.	AUD-21
acMidiReset.....Resets MIDI controllers on port to default values.	AUD-22
acMidiSendMessage	Sends raw MIDI messages to ports.AUD-23
acMidiSetTempo.....Set playback tempo of MIDI port.	AUD-24
acDigiOpen	Open a DA Streaming Port for playback.AUD-25
acFileLoad	Mid-level file load.AUD-26
acFileRewind.....Seeks to start of file.	AUD-27
acFileRead	Reads from a file using the GD file system.AUD-28
acFileOpen.....Opens a file using the GD file system.	AUD-29
acFileClose.....Closes a file using the GD file system.	AUD-30
acFileGetSize.....Gets a files size using the GD file system.	AUD-31

acFileReadFromTo.....	Reads a part of a file using the GD file system.	AUD-32
acPrintf	printf's to the Codescape log window.	AUD-33
acSystemRead.....	Use to read values in critical sections.	AUD-34
acSystemDelay	Use to delay for short periods of time.	AUD-35
acSystemEnableArmInterrupts.....	Use to enable the ARM interrupt.	AUD-36
acSystemDisableArmInterrupts	Use to disable the ARM interrupt.	AUD-37
acSystemInit.....	Makes the ac system ready to use.	AUD-38
acGetSystemFlag.....	True if the system has been initialized.	AUD-39
acSystemGetFirstFreeSoundMemory.....	Gets the address of first free sound memory.	AUD-40
acSystemGetCommandFlag.....	Gets address of driver command flag register.	AUD-41
acSystemResetArmInterrupt.....	Resets the ARM interrupt flag.	AUD-42
acSystemInstallDriver.....	Installs the sound driver.	AUD-43

2. The AICA Manager API AUD-45

amBankFetchMidiUspqn.....	Fetches uspn from a MIDI type asset.	AUD-45
amBankFetchMidiLoop	Fetches the loop flag from a MIDI type asset.	AUD-46
amBankFetchMidiPpqn.....	Fetches ppqn from a MIDI type katbank asset.	AUD-47
amBankFetchMidiVolume.....	Fetches master volume from a MIDI type katbank asset.	AUD-48
amBankFetchMidiGmModeFlag.....	Fetches GM mode flag from a MIDI type katbank asset.	AUD-49
amBankLoad.....	The filename and path of the bank to load and area to load to. ...	AUD-50
amBankFetchAssetParameters.....	Fetches parameters from any katbank asset.	AUD-51
amBankFetchWaveLoopFlag.....	Fetches the loop flag from a katbank asset.	AUD-52
amBankFetchWaveRandomPitch.....	Fetches random pitch amount from a katbank asset.	AUD-53
amBankFetchWaveSampleRate.....	Fetches the sample rate from a katbank WAVE asset.	AUD-54
amBankFetchWaveBitDepth.....	Fetches the bit depth of the sample in the bank.	AUD-55
amBankFetchUnknownParameters	Fetches one of the 7 user parameters from a katbank asset.	AUD-56
amBankFetchAsset	Fetches an asset from a katbank.	AUD-57
amBankGetAssetSize.....	Gets the size of an asset from a katbank.	AUD-58
amBankGetNumberOfAssets.....	Gets the number of assets in a katbank.	AUD-59
amBankGetHeaderSize.....	Gets the size of the header portion of a katbank.	AUD-60
amDmaMemCpy	Performs DMA copys to sound memory.	AUD-61
amHeapGetInfo.....	Gets info necessary to start an audio heap.	AUD-62
amHeapGetFree.....	Gets the amount of free memory.	AUD-63
amHeapAlloc.....	Allocates aligned memory from the audio heap.	AUD-64
amHeapGetMaxPurgable.....	Gets amount of memory available from a full purge.	AUD-65
amHeapPurge.....	Purges memory marked as purgable.	AUD-66
amHeapShow.....	Displays the heap stats using acPrintf()	AUD-67
amHeapFree	Frees purgable memory allocated using amHeapAlloc()	AUD-68
amHeapInit.....	Initializes the audio heap.	AUD-69
amHeapCheck.....	Checks the MCB fingerprints for overwrites.	AUD-70
amInit.....	Starts up the AM audio subsystem.	AUD-71
amIntShutdown	Shuts down the am interrupt system.	AUD-72
amIntInit.....	Initializes the am interrupt system.	AUD-73
amFileRewind	Seeks to the start of a file.	AUD-74
amFileLoad.....	Loads specified file into the buffer.	AUD-75
amFileRead.....	Reads from a file that is already open.	AUD-76
amFileOpen	Opens a file for reading.	AUD-77
amFileClose	Closes a file.	AUD-78
amFileGetSize.....	Gets the size of a file.	AUD-79
amFileIoSystemInit.....	Initializes the IO shell.	AUD-80
amFileInstallAlternateIoManager.....	Installs a custom Io proc.	AUD-81
amMemSh4Alloc.....	Sh4 memory allocation shell.	AUD-82

amMemSh4Free	Sh4 memory free shell.	AUD-83
amMemInit.....	Initializes the Sh4 memory shell system.	AUD-84
amMemInstallAlternateMemoryManager.....	Allows redirection of sh4 memory requests.	AUD-85
amMidiSetTempo.....	Sets the tempo of a MIDI sequence.	AUD-86
amMidiSetLoopFlag.....	Sets the loop flag on a MIDI sequence.	AUD-87
amMidiFetchToneBank.....	Installs an MTB asset from a bank file aggregate.	AUD-88
amMidiLoadToneBank.....	Loads a Sega tone bank asset.	AUD-89
amMidiInstallCallback.....	Sets the callback proc for a sequence.	AUD-90
amMidiAllocateSequencePort	Allocates a MIDI port for the sequence.	AUD-91
amMidiFetchSequence.....	Fetches a sequence asset from a katBank.	AUD-92
amMidiPlay.....	Plays a MIDI sequence.	AUD-93
amMidiPlayRaw.....	Plays a MIDI sequence given the basic parameters.	AUD-94
amMidiStop.....	Stops a currently playing MIDI sequence.	AUD-95
amMidiSetVolume.....	Sets the master volume of a MIDI sequence.	AUD-96
amMidiPause.....	Pauses a currently playing MIDI sequence.	AUD-97
amMidiResume.....	Resumes playback of a paused MIDI sequence.	AUD-98
amMidiTransferToneBank.....	Transfers a Sega tone bank to sound memory and sets it as the current bank.	AUD-99
amMidiSetChannelProgram	Sets the current bank slot.	AUD-100
amMidiNoteOn.....	Plays a MIDI triggered sound effect.	AUD-101
amMidiNoteOff.....	Stops a MIDI triggered sound effect.	AUD-102
amMidiSetChannelVolume.....	Sets volume of a midi sound.	AUD-103
amMidiSetChannelPan.....	Sets the pan of a MIDI sound.	AUD-104
amShutdown.....	Shuts down the AM audio subsystem.	AUD-105
amSoundShow.....	Shows the contents of a sound object.	AUD-106
amSoundFetchSample	Fetches a sound and its parameters from a Katapi format bank.	AUD-107
amSoundIsLooping.....	Tells if the given sound has a loop.	AUD-108
amSoundAllocateVoiceChannel.....	Allocates a hardware voice channel.	AUD-109
amSoundGetSampleRate.....	Gets the real world sample rate.	AUD-110
amSoundGetVolume.....	Gets the current volume setting.	AUD-111
amSoundGetPan.....	Gets the current pan position.	AUD-112
amSoundGetVoiceChannel.....	Gets the current voice channel assignment.	AUD-113
amSoundGetCallback	Gets the address of the user callback.	AUD-114
amSoundSetCurrentPlaybackRate.....	Sets the playback rate.	AUD-115
amSoundSetVolume.....	Sets a sounds volume.	AUD-116
amSoundSetPan.....	Sets a sounds pan.	AUD-117
amSoundSetCallback	Sets the user callback.	AUD-118
amSoundIsPlaying.....	Tells if a sound is currently playing.	AUD-119
amSoundStop.....	Stops a currently playing sound.	AUD-120
amSoundPlay	Plays a sound.	AUD-121
amSoundPlayRaw	Plays a sound given all of the required parameters.	AUD-122
amStreamInstallUserCallback	Installs a user callback for a stream.	AUD-123
amStreamRewind	Rewinds an open stream to its start.	AUD-124
amStreamGetMemoryRequirement.....	Gets memory sizes necessary to play the stream.	AUD-125
amStreamSetBufferSizes.....	Sets the sizes for the play and transfer buffers.	AUD-126
amStreamSetBuffers.....	Sets buffer memory pointers in a stream.	AUD-127
amStreamSetIsr.....	Sets the streams data transfer ISR.	AUD-128
amStreamAllocateVoiceChannels.....	Allocates voice channels.	AUD-129
amStreamPrimeBuffers.....	Primes the play buffer.	AUD-130
amStreamGetTrackLengthInFrames.....	Gets the length of a stream in frames.	AUD-131
amStreamGetNibblesPerFrame.....	Gets the number of nibbles in a frame.	AUD-132
amStreamGetSampleRate.....	Gets the real world sample rate of a stream.	AUD-133
amStreamGetTrackLengthInMs.....	Gets the length of a stream in milliseconds.	AUD-134
amStreamGetMsPerIrq	Gets the number of milliseconds per callback.	AUD-135

amStreamSetVolume.....	Sets the volume on a stream.	AUD-136
amStreamSetPan.....	Sets the pan on a mono stream.	AUD-137
amStreamStop	Stops a currently playing stream.	AUD-138
amStreamInit.....	Initializes a stream object.	AUD-139
amStreamPlaying.....	Monitors if a stream is currently playing.	AUD-140
amStreamGetVolume.....	Gets the streams current volume.	AUD-141
amStreamGetPan.....	Gets the streams current pan.	AUD-142
amStreamGetIsrCount	Gets the ISR count.	AUD-143
amStreamClose.....	Closes a stream object.	AUD-144
amStreamStart.....	Starts a stream object playing.	AUD-145
amStreamIsStereo.....	Tells if a stream is stereo.	AUD-146
amStreamIsMono.....	Tells if a stream is mono.	AUD-147
amStreamServer.....	Serves data to a currently playing stream.	AUD-148
amStreamOpen.....	Opens a stream object.	AUD-149
amStreamSetTransferMethod.....	Set amStream transfer method to DMA or non-DMA mode.	AUD-150
amStreamIoSystemInit.....	Initializes the IO shell.	AUD-151
amStreamIoInstallAlternateIoManager.....	Installs a custom io proc.	AUD-152
amUtilGetAicaVolume.....	Converts midi volume units to AICA units	AUD-153
amUtilAlignNumber.....	Performs numerical boundry alignment.	AUD-154
amUtilGetLengthInFrames.....	Gets the length of a stream in frames.	AUD-155
amUtilGetNibblesPerFrame.....	Gets the number of nibbles in a frame.	AUD-156
amUtilGetSampleRate.....	Gets the real world sample rate of a stream.	AUD-157
amUtilGetLengthInMs.....	Gets the length of a stream in milliseconds.	AUD-158
amUtilGetMsPerIrq.....	Gets the number of milliseconds per callback.	AUD-159
amUtilGetAicaSampleType.....	Extrapolates sample bit depth to AICA sample type.	AUD-160
amUtilGetAicaSampleRate.....	Makes a real world sample rate into an AICA sample rate.	AUD-161
amUtilGetMiddleOfBufferInFrames.....	Calculates the middle of the buffer in frames.	AUD-162
amUtilGetEndOfBufferInFrames	Calculates the end of the buffer in frames.	AUD-163
amVoiceInit.....	Initializes the voice pool.	AUD-164
amVoiceAllocate.....	Allocates a voice channel.	AUD-165
amVoiceCheck.....	For debugging the voice pool.	AUD-166
amStreamInstallUserCallback.....	Installs a user callback for a stream.	AUD-167
amStreamRewind	Rewinds an open stream to its start.	AUD-168
amStreamGetMemoryRequirement.....	Gets memory sizes necessary to play the stream.	AUD-169
amStreamSetBufferSizes.....	Sets the sizes for the play and transfer buffers.	AUD-170
amStreamSetBuffers.....	Sets buffer memory pointers in a stream.	AUD-171
amStreamSetIsr	Sets the streams data transfer ISR.	AUD-172
amStreamAllocateVoiceChannels.....	Allocates voice channels.	AUD-173
amStreamPrimeBuffers.....	Primes the play buffer.	AUD-174
amStreamGetTrackLengthInFrames.....	Gets the length of a stream in frames.	AUD-175
amStreamGetNibblesPerFrame.....	Gets the number of nibbles in a frame.	AUD-176
amStreamGetSampleRate.....	Gets the real world sample rate of a stream.	AUD-177
amStreamGetTrackLengthInMs	Gets the length of a stream in milliseconds.	AUD-178
amStreamGetMsPerIrq	Gets the number of milliseconds per callback.	AUD-179
amStreamSetVolume.....	Sets the volume on a stream.	AUD-180
amStreamSetPan.....	Sets the pan on a mono stream.	AUD-181
amStreamStop	Stops a currently playing stream.	AUD-182
amStreamInit.....	Initializes a stream object.	AUD-183
amStreamPlaying.....	Monitors if a stream is currently playing.	AUD-184
amStreamGetVolume.....	Gets the streams current volume.	AUD-185
amStreamGetIsrCount	Gets the ISR count.	AUD-186
amStreamClose.....	Closes a stream object.	AUD-187
amStreamStart.....	Starts a stream object playing.	AUD-188
amStreamIsStereo.....	Tells if a stream is stereo.	AUD-189

amStreamIsMono.....	Tells if a stream is mono.	AUD-190
amStreamServer.....	Serves data to a currently playing stream.	AUD-191
amStreamOpen	Opens a stream object.	AUD-192
amStreamSetTransferMethod.....	Sets the transfer method to DMA or non-DMA.	AUD-193
amMidiResume.....	Resumes playback of a paused MIDI sequence.	AUD-194
amMidiTransferToneBank.....	Transfers a Sega tone bank to sound memory and sets it as the current bank.	AUD-195
amMidiSetChannelProgram	Sets the current bank slot.	AUD-196
amMidiNoteOn.....	Plays a MIDI triggered sound effect.	AUD-197
amMidiNoteOff.....	Stops a MIDI triggered sound effect.	AUD-198
amMidiSetChannelVolume.....	Sets volume of a midi sound.	AUD-199
amMidiSetChannelPan.....	Sets the pan of a MIDI sound.	AUD-200
amSoundShow.....	Shows the contents of a sound object.	AUD-201
amSoundFetchSample	Fetches a sound and its parameters from a Katapi format bank.	AUD-202
amSoundIsLooping.....	Tells if the given sound has a loop.	AUD-203
amSoundAllocateVoiceChannel.....	Allocates a hardware voice channel.	AUD-204
amSoundGetSampleRate.....	Gets the real world sample rate.	AUD-205
amSoundGetVolume.....	Gets the current volume setting.	AUD-206
amSoundGetPan.....	Gets the current pan position.	AUD-207
amSoundGetVoiceChannel.....	Gets the current voice channel assignment.	AUD-208
amSoundGetCallback	Gets the address of the user callback.	AUD-209
amSoundSetPlaybackRate.....	Sets the playback rate.	AUD-210
amSoundSetVolume.....	Sets a sounds volume.	AUD-211
amSoundSetPan.....	Sets a sounds pan.	AUD-212
amSoundSetCallback	Sets the user callback.	AUD-213
amSoundIsPlaying.....	Tells if a sound is currently playing.	AUD-214
amSoundStop.....	Stops a currently playing sound.	AUD-215
amSoundPlay	Plays a sound.	AUD-216
amSoundPlayRaw	Plays a sound given all of the required parameters.	AUD-217
amStreamInstallUserCallback	Installs a user callback for a stream.	AUD-218
amStreamRewind	Rewinds an open stream to its start.	AUD-219
amStreamGetMemoryRequirement.....	Gets memory sizes necessary to play the stream.	AUD-220
amStreamSetBufferSizes.....	Sets the sizes for the play and transfer buffers.	AUD-221
amStreamSetBuffers.....	Sets buffer memory pointers in a stream.	AUD-222
amStreamSetIsr.....	Sets the streams data transfer ISR.	AUD-223
amStreamAllocateVoiceChannels.....	Allocates voice channels.	AUD-224
amStreamPrimeBuffers.....	Primes the play buffer.	AUD-225
amStreamGetTrackLengthInFrames.....	Gets the length of a stream in frames.	AUD-226
amStreamGetNibblesPerFrame.....	Gets the number of nibbles in a frame.	AUD-227
amStreamGetSampleRate.....	Gets the real world sample rate of a stream.	AUD-228
amStreamGetTrackLengthInMs.....	Gets the length of a stream in milliseconds.	AUD-229
amStreamGetMsPerIrq	Gets the number of milliseconds per callback.	AUD-230
amStreamSetVolume.....	Sets the volume on a stream.	AUD-231
amStreamSetPan.....	Sets the pan on a mono stream.	AUD-232
amStreamStop.....	Stops a currently playing stream.	AUD-233
amStreamInit.....	Initializes a stream object.	AUD-234
amStreamPlaying.....	Monitors if a stream is currently playing.	AUD-235
amStreamGetVolume.....	Gets the streams current volume.	AUD-236
amStreamGetPan.....	Gets the streams current pan	AUD-237
amStreamGetIsrCount.....	Gets the ISR count.	AUD-238
amStreamClose	Closes a stream object.	AUD-239
amStreamStart.....	Starts a stream object playing.	AUD-240
amStreamIsStereo.....	Tells if a stream is stereo.	AUD-241

amStreamIsMono.....	Tells if a stream is mono.	AUD-242
amStreamServer.....	Serves data to a currently playing stream.	AUD-243
amStreamOpen.....	Opens a stream object.	AUD-244
amStreamSetTransferMethod.....	Sets whether transfer occurs via DMA or non-DMA	AUD-245
amStreamIoSystemInit.....	Initializes the IO shell.	AUD-246
amStreamIoInstallAlternateIoManager.....	Installs a custom io proc.	AUD-247
amUtilGetAicaVolume.....	Converts midi volume units to AICA units	AUD-248
amUtilAlignNumber.....	Performs numerical boundry alignment.	AUD-249
amUtilGetLengthInFrames.....	Gets the length of a stream in frames.	AUD-250
amUtilGetNibblesPerFrame.....	Gets the number of nibbles in a frame.	AUD-251
amUtilGetSampleRate.....	Gets the real world sample rate of a stream.	AUD-252
amUtilGetLengthInMs.....	Gets the length of a stream in milliseconds.	AUD-253
amUtilGetMsPerIrq.....	Gets the number of milliseconds per callback.	AUD-254
amUtilGetAicaSampleType.....	Extrapolates sample bit depth to AICA sample type.	AUD-255
amUtilGetAicaSampleRate.....	Makes a real world sample rate into an AICA sample rate.	AUD-256
amUtilGetMiddleOfBufferInFrames.....	Calculates the middle of the buffer in frames.	AUD-257
amUtilGetEndOfBufferInFrames	Calculates the end of the buffer in frames.	AUD-258
amVoiceInit.....	Initializes the voice pool.	AUD-259
amVoiceAllocate.....	Allocates a voice channel.	AUD-260
amVoiceCheck.....	For debugging the voice pool	AUD-261
amStreamInstallUserCallback.....	Installs a user callback for a stream.	AUD-262
amStreamRewind	Rewinds an open stream to its start.	AUD-263
amStreamGetMemoryRequirement.....	Gets memory sizes necessary to play the stream.	AUD-264
amStreamSetBufferSizes.....	Sets the sizes for the play and transfer buffers.	AUD-265
amStreamSetBuffers.....	Sets buffer memory pointers in a stream.	AUD-266
amStreamSetIsr	Sets the streams data transfer ISR.	AUD-267
amStreamAllocateVoiceChannels.....	Allocates voice channels.	AUD-268
amStreamPrimeBuffers.....	Primes the play buffer.	AUD-269
amStreamGetTrackLengthInFrames.....	Gets the length of a stream in frames.	AUD-270
amStreamGetNibblesPerFrame.....	Gets the number of nibbles in a frame.	AUD-271
amStreamGetSampleRate.....	Gets the real world sample rate of a stream.	AUD-272
amStreamGetTrackLengthInMs	Gets the length of a stream in milliseconds.	AUD-273
amStreamGetMsPerIrq	Gets the number of milliseconds per callback.	AUD-274
amStreamSetVolume.....	Sets the volume on a stream.	AUD-275
amStreamSetPan.....	Sets the pan on a mono stream.	AUD-276
amStreamStop	Stops a currently playing stream.	AUD-277
amStreamInit.....	Initializes a stream object.	AUD-278
amStreamPlaying.....	Monitors if a stream is currently playing.	AUD-279
amStreamGetVolume.....	Gets the streams current volume	AUD-280
amStreamGetIsrCount	Gets the ISR count.	AUD-281
amStreamClose.....	Closes a stream object.	AUD-282
amStreamStart.....	Starts a stream object playing.	AUD-283
amStreamIsStereo.....	Tells if a stream is stereo.	AUD-284
amStreamIsMono.....	Tells if a stream is mono.	AUD-285
amStreamServer.....	Serves data to a currently playing stream.	AUD-286
amStreamOpen.....	Opens a stream object.	AUD-287

3. The Redbook Playback API.....	AUD-289
acCddaSetVolume.....	Sets Left & Right Channels for Redbook Volume Control (dependent on channel pan). AUD-289
acCddaSetPan.....	Sets Left & Right Channel pan position. AUD-290
acCddaResetChannels.....	Resets CDDA channels to hard pan positions and maximum volume. AUD-291

1. The AICA Control Layer API

acSetMultiPlayMask

Sets the bit masks for acDigiMultiPlay()

FORMAT

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

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

PARAMETERS

KTU32 port,	The port to be started/stopped
KTU32 *uppermask,	A pointer to the upper 32 channel mask
KTU32 *lowermask,	A pointer to the lower 32 channel mask

RETURN VALUE

KTTRUE if command write 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.

acDspLoadPrograms

Registers a dsp program bank with the driver.

FORMAT

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

```
KTBOOL acDspLoadPrograms(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 command write 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.

acDspLoadOutputMixer

Registers an output mixer patch with the driver.

FORMAT

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

```
KTBOOL acDspLoadOutputMixer(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 command write 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.

acClosePort

Closes port previously opened.

FORMAT

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

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

PARAMETERS

port, port number, 0-15.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Close port command. Will allow buffer reassignment upon open.

acDigiSetCurrentPitch

Change the playback rate (pitch) of audio stream.

FORMAT

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

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

PARAMETERS

Port port number, 0-15.

pitchOffsetInCents pitch in cents (1200/octave). Range variable, see remarks below.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Changes pitch (playback rate) of DA stream. This is variable based on original playback rate, but acceptable values are +/-7 octaves, so pitch value can be 1200 x +/-7.

Making this call will not reset the default of the port so calling this with an arg of 1200 will make the sound play up one octave, a further call to this with an arg of 0 will make the // sound play at its starting pitch.

acDigiSetVolume

Adjusts direct output volume of DA stream.

FORMAT

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

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

PARAMETERS

Port	DA stream port number, 0-15.
Volume	0-15, gets louder.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Adjusts direct output aicaVolume of DA stream.

acDigiSetPan

Adjusts pan placement of a port.

FORMAT

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

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

PARAMETERS

port	DA stream port number, 0-15.
aicaPan	0-31, left to right.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Adjusts direct output pan position of DA stream.

acDigiSetMixer

Set port's audio signal to DSP mixer channel.

FORMAT

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

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

PARAMETERS

port	DA stream prot number, 0-15.
mixer	DSP mixer channel number, 0-15.
level	Audio signal level, 0-15.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

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

acDigiSetQSoundAngle

Sets Q-Sound position.

FORMAT

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

```
KTBOOL acDigiSetQSoundAngle(KTU32 port,KTU32 angle)
```

PARAMETERS

mixer	DSP mixer channel number, 0-15. Requires a Qsound DSP program to have been previously loaded and mapped to mixer channel.
angle	0-127, left to right.

RETURN VALUE

KTTRUE	if command write successful; KTFALSE otherwise.
--------	---

FUNCTION

If stream is set for mixer channel, will set the apparent position of sound.
The parameter angle is not error trapped

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

FORMAT

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

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

PARAMETERS

port	DA stream port number, 0-15.
offsetFromBeginningInFrames	Relative offset from start of buffer in samples (frames) offset value must be less than 64k

RETURN VALUE

KTFALSE (ignore)

FUNCTION

When channel playback \geq to position, will generate interrupt. The port number (byte length) is stored in a circular buffer, and the head pointer into this buffer is incremented by the Arm7 driver.

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

acDigiPlay

Starts a buffer playing.

FORMAT

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

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

PARAMETERS

port	DA port number, 0-15.
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).

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Plays buffer assigned by amOpenPort. Total length of buffer must be < 64k samples (frames).

Note: The parameter startOffset is not error trapped.

acDigiStop

Stops a port from playing.

FORMAT

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

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

PARAMETERS

port	DA port number, 0-15.
------	-----------------------

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Stops playback of a previously started port.

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	0 or 0xff, General MIDI format file, allows Bank 0 General MIDI Instrument and Drumset.
address	offset into sound ram of the start of buffer.
MidiBufferSize	buffer length in bytes.
TicksPerQNote	time base in microseconds per quarternote.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Open MIDI port command. This command assigns a memory area which will contain a MIDI Format 0, header-less (tag-less) midi file.

acMidiSetTonebank

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

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	drumset to use, 0-2 for alternate GM Drumset
address	offset into sound memory of start of tone bank
sizeInBytes	size of tone bank in bytes
mttPtr	MIDI translate table pointer (not yet implemented)

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Sets a tonebank for active playback. Assigns a bank number to a tonebank to be used and accessed via a MIDI Bank Select message.

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

acMidiClose

Release active MIDI port.

FORMAT

```
#include <ac.h>

KTBOOL acMidiClose(KTU32 port)
```

PARAMETERS

port MIDI port number, 0-15.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Releases MIDI port, sends All Notes Off message to driver parser.

acMidiPlay

Begins playback on opened MIDI port.

FORMAT

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

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

PARAMETERS

port	MIDI port umber, 0-15.
startOffset	Playback from buffer start position + offset.
loopFlag	Loop MIDI playback buffer, 0 or 0xff.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Starts standard MIDI file playback on port, from location of opened port buffer plus offset.
The default tempo is 120 BPM, until MIDI tempo message is parsed.

acMidiStop

Stops standard MIDI file playback on port.

FORMAT

```
#include <ac.h>

KTBOOL acMidiStop(KTU32 port)
```

PARAMETERS

`port` MIDI port number, 0-15.

RETURN VALUE

`KTTRUE` if command write successful; `KTFALSE` otherwise.

FUNCTION

Stops standard MIDI file playback on port, sends All Notes Off message to 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 command write successful; KTFALSE otherwise.

FUNCTION

Generates interrupt to host upon MIDI port playback reaching the sound ram address specified in PortEventAddress. Will write BYTE indicator of port (port number + 16) into the event interrupt status queue, increment the event interrupt status queue head pointer to the next available position in the queue, and generate and interrupt to host.

Note: The parameter offsetFromBeginningInBytes is not error trapped.

acMidiPause

Pause active MIDI port.

FORMAT

```
#include <ac.h>

KTBOOL acMidiPause(KTU32 port)
```

PARAMETERS

port MIDI port number, 0-15.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Pauses playback on MIDI port. Stops all currently sounding notes on port.

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 command write successful; KTFALSE otherwise.

FUNCTION

Resumes playback on MIDI port. Maintains running status mode.

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 command write successful; KTFALSE otherwise.

FUNCTION

Sets scaled (global) volume setting for MIDI port. Scales all MIDI controller 7 messages for port.

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 command write successful; KTFALSE otherwise.

FUNCTION

Resets controller values to known position, and Bank select per channel is set to 0. The MIDI continuous controller settings affected are as follows:

CC numbers 0x07 = 0x64; 0x0b = 0x7f; 0x0a, 0x47, 0x4a = 0x40; 0x14-0x1c, 0x58 = 0x20; 0x00, 0x34-0x38, 0x46 = 0x00; pitch bend = 0 (center).

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 command write successful; KTFALSE otherwise.

FUNCTION

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

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

acMidiSetTempo

Set playback tempo of MIDI port.

FORMAT

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

PARAMETERS

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

RETURN VALUE

`KTTRUE` if command write successful; `KTFALSE` otherwise.

FUNCTION

Allows real-time control of tempo for port.

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

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	port number, 0-15
address	offset into sound ram of the start of buffer.
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 command write successful; KTFALSE otherwise.

FUNCTION

Open port command. This command assigns a memory area to a voice for playback.

acFileLoad

Mid-level file load.

FORMAT

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

```
KTBOOL acFileLoad(KTSTRING fileName,volatile KTU32 *buffer)
```

PARAMETERS

KTSTRING fileName,	The name of the file to be loaded
volatile KTU32 *buffer,	A pointer to a buffer big enough to hold the file.

RETURN VALUE

KTBOOL, KTTRUE if the load operation was successful

FUNCTION

Uses the low level acFile calls to load a file, i.e. acFileGetSize()->acFileOpen()->acFileRead()->acFileClose()

If any of these calls fail it will send that calls error number to the error system.

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

acFileRewind

Seeks to start of file.

FORMAT

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

```
KTBOOL acFileRewind(ACFILE fd)
```

PARAMETERS

ACFILE fd, the file descriptor obtained by the call to acFileOpen()

RETURN VALUE

KTBOOL, KTTRUE if the seek operation was successful

FUNCTION

Note: All GD file system calls currently require that the buffer pointer passed in be aligned on a 32 byte boundary.

acFileRead

Reads from a file using the GD file system.

FORMAT

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

```
KTBOOL acFileRead(ACFILE fd,KTU8 * buffer,KTU32 size)
```

PARAMETERS

ACFILE fd,	the file descriptor obtained by the call to <code>acFileOpen()</code>
KTU8 * buffer,	a pointer to a buffer of adequate size to accomodate the data
KTU32 size,	the size of the data to be read

RETURN VALUE

KTBOOL, KTTRUE if the read operation was successful

FUNCTION

Reads from a file using the GD file system

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

acFileOpen

Opens a file using the GD file system.

FORMAT

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

```
KTBOOL acFileOpen(KTSTRING fileName,ACFILE *fd)
```

PARAMETERS

KTSTRING fileName,	the name of the file to be opened
ACFILE *fd,	the file descriptor for the file is returned in this value

RETURN VALUE

KTBOOL, KTTRUE if the file was successfully opened

FUNCTION

Uses the GD file system to open a file for reading.

acFileClose

Closes a file using the GD file system.

FORMAT

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

```
KTBOOL acFileClose(ACFILE fd)
```

PARAMETERS

ACFILE fd, the file descriptor of the file to be closed

RETURN VALUE

KTBOOL, KTTRUE if the file was successfully closed

FUNCTION

Closes a file using the GD file system

acFileGetSize

Gets a files size using the GD file system.

FORMAT

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

```
KTBOOL acFileGetSize(KTSTRING fileName, KTU32 * size)
```

PARAMETERS

KTSTRING fileName,	the name of the file for which the size is to be obtained
KTU32 * size,	the size of the named file is returned here if KTTRUE

RETURN VALUE

KTBOOL, KTTRUE if the size was successfully gotten, size is 0 on fail.

FUNCTION

Gets the size in bytes of the named file

acFileReadFromTo

Reads a part of a file using the GD file system.

FORMAT

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

```
KTBOOL acFileReadFromTo(KTSTRING fileName,KTU8 * buffer,KTU32 fromByte,KTU32 toByte)
```

PARAMETERS

KTSTRING fileName,	the name of the file to be opened and read
KTU8 * buffer,	a pointer to a buffer of adequate size to accomodate the data
KTU32 fromByte, zero based,	the byte at which the read will start
KTU32 toByte zero based,	the ending byte of the read

RETURN VALUE

KTBOOL, KTTRUE if the read was successful

FUNCTION

Opens, reads and closes the given file reading into the buffer only those bytes specified in the arguments.

Note: All GD file system calls currently require that the buffer pointer passed in be aligned on a 32 byte boundary.

acPrintf

printf's to the Codescape log window.

FORMAT

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

```
void acPrintf(const char *format,...)
```

PARAMETERS

Standard printf args.

RETURN VALUE

void

FUNCTION

Formatted printing to the debugger log window

acSystemRead

Use to read values in critical sections.

FORMAT

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

```
KTU32 * acSystemRead(volatile KTU32 *address,volatile KTU32 *result)
```

PARAMETERS

<code>volatile KTU32 *address,</code>	the address to read from
<code>volatile KTU32 *result,</code>	the value read from that address

RETURN VALUE

KTU32 *, the result

FUNCTION

Reads data from sound memory in a positive fashion, this is used to avoid problems with the interruptability of the SH4's reads from sound memory. If the ARM is trying to write to an address in sound memory while the SH4 is trying to read from the address it is possible that the read will be interrupted after a word is read with a dword write, this will result in the retrieval of a malformed value.

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>
```

```
void acSystemInit(void)
```

PARAMETERS

void

RETURN VALUE

void

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.

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. This will be changed to read the driver directly into sound memory in the near future, currently it mallocs and frees SH4 memory via the memory shell.

acDigiSetSampleRate

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

FORMAT

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

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

PARAMETERS

Port	port number, 0-63
sampleRate	Set the hardware sample playback rate. Range variable, see remarks below.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Changes sample rate (playback rate) of DA sound. This variable is the desired sample rate, the setting will be the closest approximation the hardware is capable of reproducing.

2. *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 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

KTRUE,	on success
KFALSE,	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 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

The filename and path of the bank to load and area to load to.

FORMAT

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

```
KTBOOL amBankLoad(KTSTRING fileName,AM_BANK_PTR buffer)
```

PARAMETERS

KTSTRING fileName, **The filename and path of the bank to load.**

AM_BANK_PTR buffer, **A 32 byte aligned buffer in sound memory big enough to hold the asset.**

RETURN VALUE

KTTRUE, **on success**

KTFALSE, **fileName is NULL, 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 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

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

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, parameters is NULL, assetNumber 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.

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 number that is set in the katbank build script using the “SampleRate” tag.

amBankFetchWaveBitDepth

Fetches the bit depth of the sample in the bank.

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 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 assetFUNCTION 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_PTR theBank,
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

KTTRUE,	on success
KTFALSE,	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

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 than cycle steal mode as timeliness is important to streaming audio processes.

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

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

RETURN VALUE

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

FUNCTION

Allocates aligned memory from the audio heap zone. The memory can be allocated in alignments of either 4 or 32 bytes. Odd byte address writes to the audio memory area are illegal.

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 boundary. 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.

amHeapShow

Displays the heap stats using `acPrintf()`

FORMAT

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

```
void amHeapShow(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Displays the heap stats in the debugger output window using `acPrintf()`

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,	On success
KTFALSE,	If buffer is NULL, buffer does not point to an allocated block

FUNCTION

This still needs some work.

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

KTTRUE **If the operation was successful**

KTFALSE, **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.`

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 Unable to allocate a temporary buffer in SH4 memory

FUNCTION

Starts up the AM audio subsystem. Allocates a temp buffer in SH4 memory using the `amMem` memory shell. This shell can be taken over with your `mallocfree-newdelete` memory system.

amIntShutdown

Shuts down the am interrupt system.

FORMAT

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

```
void amIntShutdown(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Shuts down the am interrupt system by removing the ARM interrupt callback.

amIntInit

Initializes the am interrupt system.

FORMAT

```
#include <am.h>

void amIntInit(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Initializes the am interrupt system by installing the ARM interrupt callback.

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 Also

```
amFileInstallAlternateIoManager( )
```

An example of this redirection is available in `ammain.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 Also

```
amFileInstallAlternateIoManager()
```

An example of this redirection is available in `ammain.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.

SeeAlso

```
amFileInstallAlternateIoManager()
```

An example of this redirection is available in `ammain.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 Also

```
amFileInstallAlternateIoManager()
```

An example of this redirection is available in `ammain.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.

SeeAlso

amFileInstallAlternateIoManager()

An example of this redirection is available in `ammain.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.

SeeAlso

```
amFileInstallAlternateIoManager()
```

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

amFileIoSystemInit

Initializes the IO shell.

FORMAT

```
#include <am.h>

void amFileIoSystemInit(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Initializes the IO shell.

SeeAlso

`amFileInstallAlternateIoManager()`

An example of this redirection is available in `ammain.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 ammain.c

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 ammain.c as well as a boilerplate copy of the IO proc for modification.

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,	a pointer to the first aligned address after the base address
KTU32 size,KTU32 alignment,	the alignment desired

RETURN VALUE

KTBOOL, KTTRUE if a block was successfully allocated

FUNCTION

This is a shell that verifies a proc pointer then calls it to invoke whatever malloc proc is currently installed. If the developer wishes to control memory allocations made by the audio engine they should see the malloc and free shells in the bottom of amMain.c, fill them in with their malloc and free calls and use the amMemInstallAlternateMemoryManager() call to redirect all memory allocation to their system.

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 verifys a proc pointer then calls it to invoke whatever free proc is currently installed.

If the developer wishes to control memory allocations made by the audio engine they should see the malloc and free shells in the bottom of `amMain.c`, fill them in with their malloc and free calls and use the `amMemInstallAlternateMemoryManager()` call to redirect all memory allocation to their system.

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 `smInit()`

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 `smInit()`

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, the Sequence 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.

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.

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.

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.

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

amMidiInstallCallback

Sets the callback proc for a sequence.

FORMAT

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

```
KTBOOL amMidiInstallCallback(AM_SEQUENCE_PTR theSequence, AM_MIDI_CALLBACK theCallback)
```

PARAMETERS

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

AM_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.

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.

amMidiFetchSequence

Fetches a sequence asset from a katBank.

FORMAT

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

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

PARAMETERS

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

RETURN VALUE

KTTRUE, on success

Or...

KTFALSE 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.

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.

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,AM_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)
AM_MIDI_CALLBACK callback,	The address of a callback proc or KTNUL for no callback

RETURN VALUE

KTTRUE, on success

Or...

KTFALSE 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)
```


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 aquire a new midi port for playback. The callback, if one has been set using `amMidiInstallCallback()`, is still in place.

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

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.

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.

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

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.

amMidiNoteOn

Plays a MIDI triggered sound effect.

FORMAT

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

```
KTBOOL amMidiNoteOn(KTU32 midiPort,KTU32 midiChannel,KTU8 midiNoteNumber,KTU32  
midiNoteOnVelocity)
```

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).

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.

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)

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)

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.

amSoundShow

Shows the contents of a sound object.

FORMAT

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

```
KTBOOL amSoundShow(AM_SOUND_PTR theSound)
```

PARAMETERS

AM_SOUND_PTR sound, **A pointer to an AM_SOUND structure**

RETURN VALUE

KTTRUE **on success**
KTFALSE **if theSound is NULL**

FUNCTION

Shows the data contained in an AM_SOUND structure.

amSoundFetchSample

Fetches a sound and its parameters from a Katapi 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 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,	KTTRUEon 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,	KTTRUEon 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,	KTTRUEon 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>
```

```
KTBOOL 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

KTTRUE	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.

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

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 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>
```

```
KTBOOL 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

<code>AM_SOUND_PTR theSound,</code>	A pointer to a properly initialized sound object
-------------------------------------	--

RETURN VALUE

<code>KTTRUE</code>	if the sound was played
<code>KTFALSE</code>	can't send a command to the driver <code>theSound</code> is <code>NULL</code> 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.

amSoundPlayRaw

Plays a sound given all of the required parameters.

FORMAT

```
#include <am.h>

KTBOOL amSoundPlayRaw(    KTS32 voiceChannel,
                           KTU32 sizeInBytes,
                           KTU32 address,
                           KTS32 sampleRate,
                           AC_AUDIO_TYPE aicaAudioType,
                           KTU32 pitchOffsetInCents,
                           KTU32 aicaLoopFlag,
                           KTU32 volume,
                           KTU32 pan,
                           AM_USER_CALLBACK userCallbackProc
```

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
KTU32 volume,	The normal volume (0-127)
KTU32 pan,	The normal pan (0-127)
AM_USER_CALLBACK userCallbackProc,	A pointer to a user callback proc or KTNUL

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.

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 `am.h`**

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>

KTBOOL amStreamRewind(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, The stream object to rewind.

RETURN VALUE

KTBOOL, 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 without 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: `transferBufferSize=(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, KTTRUE,	If the buffer pointers were set.
KTFALSE,	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	_amStreamMonoIsr0	(unsigned int streamPtr);
void	_amStreamMonoIsr1	(unsigned int streamPtr);
void	_amStreamMonoIsr2	(unsigned int streamPtr);
void	_amStreamMonoIsr3	(unsigned int streamPtr);
void	_amStreamStereoIsr0	(unsigned int streamPtr);
void	_amStreamStereoIsr1	(unsigned int streamPtr);

RETURN VALUE

KTBOOL, KTRUE,	If the ISR was installed successfully.
KFALSE,	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 after the 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 after 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 after the call to amStreamOpen() or it will return 0

CALLS

```
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 after 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 after the call to `amStreamOpen()` or it will return 0

amStreamGetTrackLengthInMs

Gets the length of a stream in milliseconds.

FORMAT

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

```
KTBOOL  amStreamGetTrackLengthInMs(AM_STREAM_PTR theStream,KTU32 trackNumber,KTU32
*lengthInMs)
```

PARAMETERS

AM_STREAM_PTR theStream,

The stream object

KTU32 *lengthInMs,

The length of a stream in milliseconds is returned via this pointer.

RETURN VALUE

KTBOOL

FUNCTION

Gets the length of a stream in milliseconds.

Note: This must be called after 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

KTBOOL

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 after 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.

amStreamInit

Initializes a stream object.

FORMAT

```
#include <am.h>

KTBOOL amStreamInit(      AM_STREAM_PTR theStream,
KTSTRING fileName,
volatile KTU32 volume,
volatile KTU32 pan)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object to be initialized
KTSTRING fileName,	the file name of the .str file to stream
volatile KTU32 volume,	the volume at which to play the stream
volatile KTU32 pan,	the pan of a mono stream, this is disregarded for stereo streams

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.

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,	The stream object to get the volume from
KTU32 *volume,	The volume is returned via this pointer.

RETURN VALUE

KTBOOL,	KTTRUE on success
	KTFALSE 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,	The stream object to get the volume from
KTU32 *pan,	The pan is returned via this pointer.

RETURN VALUE

KTBOOL,	KTTRUE on success
	KTFALSE on NULL parameter or track number out of range.

FUNCTION

Gets the current pan of a stream.

amStreamGetIsrCount

Gets the ISR 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>
```

```
KTBOOL amStreamIsStereo(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be queried.

RETURN VALUE

KTBOOL, 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>
```

```
KTBOOL amStreamServer(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be served

RETURN VALUE

KTBOOL, KTRUE if the stream is currently playing, KFALSE 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 Set amStream transfer method to DMA or non-DMA mode.

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()

amStreamIoSystemInit

Initializes the IO shell.

FORMAT

```
#include <am.h>

void amStreamIoSystemInit(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Initializes the IO shell.

See Also

`amStreamIoInstallAlternateIoManager()`

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

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 ammain.c

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 ammain.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 <code>ac.h</code>
KTU32 aicaSampleRate,	the AICA sample rate of the sound, see <code>ac.h</code>
KTU32 playBufferSizeInBytes,	the size of the playback buffer in bytes
KTU32 *msPerIrq,	the number of milliseconds per callback.

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, KTRUE if the calculation was successful, KFALSE 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 array.

amVoiceCheck

For debugging the voice pool.

FORMAT

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

```
void amVoiceCheck(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Prints a message for any channel whose status flag is set to `KTTRUE`.

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 am.h

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 after the call to `amStreamAllocateVoiceChannels()` and the call to `amStreamOpen()`.

amStreamRewind

Rewinds an open stream to its start.

FORMAT

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

```
KTBOOL amStreamRewind(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, The stream object to rewind.

RETURN VALUE

KTBOOL, KTRUE, 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 without 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: `transferBufferSize= (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, KTTRUE,	If the buffer pointers were set.
KTFALSE,	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      _amStreamMonoIsr0    (unsigned int streamPtr);
void      _amStreamMonoIsr1    (unsigned int streamPtr);
void      _amStreamMonoIsr2    (unsigned int streamPtr);
void      _amStreamMonoIsr3    (unsigned int streamPtr);
void      _amStreamStereoIsr0   (unsigned int streamPtr);
void      _amStreamStereoIsr1   (unsigned int 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()`.

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, KTRUE,	If the voice(s) were successfully allocated.
KFALSE,	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 after the 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 after the call to `amStreamOpen()` or it will return 0

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 after the call to amStreamOpen()

CALLS

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 after 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 after the call to `amStreamOpen()` or it will return 0

amStreamGetTrackLengthInMs

Gets the length of a stream in milliseconds.

FORMAT

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

```
KTBOOL  amStreamGetTrackLengthInMs(AM_STREAM_PTR theStream,KTU32 trackNumber,KTU32
*lengthInMs)
```

PARAMETERS

AM_STREAM_PTR theStream,

The stream object

KTU32 *lengthInMs,

The length of a stream in milliseconds is returned via this pointer.

RETURN VALUE

KTBOOL

FUNCTION

Gets the length of a stream in milliseconds.

Note: This must be called after 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

```
KTBOOL
```

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 after 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 (0-15)

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 (0-31)

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.

amStreamInit

Initializes a stream object.

FORMAT

```
#include <am.h>

KTBOOL amStreamInit(    AM_STREAM_PTR theStream,
KTSTRING fileName,
volatile KTU32 volume,
volatile KTU32 pan)
```

PARAMETERS

AM_STREAM_PTR	theStream, the stream object to be initialized
KTSTRING fileName,	the file name of the .str file to stream
volatile KTU32 volume,	the volume at which to play the stream
volatile KTU32 pan,	the pan of a mono stream, this is disregarded for stereo streams

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.

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, The stream object to get the volume from

RETURN VALUE

KTU32, The current volume of the stream.

FUNCTION

Gets the current volume of a stream.

amStreamGetIsrCount

Gets the ISR 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 aquires 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>
```

```
KTBOOL amStreamIsStereo(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be queried.

RETURN VALUE

KTBOOL, 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>
```

```
KTBOOL amStreamServer(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be served

RETURN VALUE

KTBOOL, 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

Sets the transfer method to DMA or non-DMA.

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 than 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()

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.

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

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.

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).

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.

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)

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)

amSoundShow

Shows the contents of a sound object.

FORMAT

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

```
KTBOOL amSoundShow(AM_SOUND_PTR theSound)
```

PARAMETERS

AM_SOUND_PTR **sound**, A pointer to an AM_SOUND structure

RETURN VALUE

KTTRUE on success

KTFALSE if theSound is NULL

FUNCTION

Shows the data contained in an AM_SOUND structure.

amSoundFetchSample

Fetches a sound and its parameters from a Katapi 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 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>
```

```
KTBOOL 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

KTTRUE 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.

amSoundSetPlaybackRate

Sets the playback rate.

FORMAT

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

```
KTBOOL amSoundSetPlaybackRate(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 < 5012
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

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 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>
```

```
KTBOOL 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 prototyped 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.

amSoundPlayRaw

Plays a sound given all of the required parameters.

FORMAT

```
#include <am.h>

KTBOOL amSoundPlayRaw(KTS32 voiceChannel,
KTU32 sizeInBytes,
KTU32 address,
KTS32 aicaSampleRate,
AC_AUDIO_TYPE aicaAudioType,
KTU32 pitchOffsetInCents,
KTU32 aicaLoopFlag,
KTU32 volume,
KTU32 pan,
AM_USER_CALLBACK userCallbackProc
)
```

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 aicaSampleRate,	The AICA sample rate, i.e. AC_44K_SAMPLE_RATE, see ac.h
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
KTU32 volume,	The normal volume (0-127)
KTU32 pan,	The normal pan (0-127)
AM_USER_CALLBACK userCallbackProc,	A pointer to a user callback proc or KTNUL

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.

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 am.h

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 after the call to `amStreamAllocateVoiceChannels()` and the call to `amStreamOpen()`.

amStreamRewind

Rewinds an open stream to its start.

FORMAT

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

```
KTBOOL amStreamRewind(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, The stream object to rewind.

RETURN VALUE

KTBOOL, 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 rround 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 after 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, KTTRUE,	If the buffer pointers were set.
KTFALSE,	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 after 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 _amStreamMonoIsr0 (unsigned int streamPtr);

void _amStreamMonoIsr1 (unsigned int streamPtr);

void _amStreamMonoIsr2 (unsigned int streamPtr);

void _amStreamMonoIsr3 (unsigned int streamPtr);

void _amStreamStereoIsr0 (unsigned int streamPtr);

void _amStreamStereoIsr1 (unsigned int 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 after 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 after the 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 after 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 after the call to amStreamOpen() or it will return 0

CALLS

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 after 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 after the call to `amStreamOpen()` or it will return 0

amStreamGetTrackLengthInMs

Gets the length of a stream in milliseconds.

FORMAT

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

```
KTBOOL amStreamGetTrackLengthInMs(AM_STREAM_PTR theStream,KTU32 trackNumber,KTU32  
*lengthInMs)
```

PARAMETERS

AM_STREAM_PTR theStream, The stream object

KTU32 *lengthInMs, The length of a stream in milliseconds is returned via this pointer.

RETURN VALUE

KTBOOL

FUNCTION

Gets the length of a stream in milliseconds.

Note: This must be called after 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

KTBOOL

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 after 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, KTRUE 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.

amStreamInit

Initializes a stream object.

FORMAT

```
#include <am.h>

KTBOOL amStreamInit(AM_STREAM_PTR theStream,
KTSTRING fileName,
volatile KTU32 volume,
volatile KTU32 pan)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be initialized
KTSTRING fileName, the file name of the .str file to stream
volatile KTU32 volume, the volume at which to play the stream
volatile KTU32 pan, the pan of a mono stream, this is disregarded for stereo streams

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.

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, The stream object to get the volume from
KTU32 *volume, The volume is returned via this pointer.

RETURN VALUE

KTBOOL, **KTTRUE** on success
KTFALSE 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, The stream object to get the volume from

KTU32 *pan, The pan is returned via this pointer.

RETURN VALUE

KTBOOL, KTTRUE on success

KTFALSE on NULL parameter or track number out of range.

FUNCTION

Gets the current pan of a stream.

amStreamGetIsrCount

Gets the ISR 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, KTRUE 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>
```

```
KTBOOL amStreamIsStereo(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR **theStream**, the stream object to be queried.

RETURN VALUE

KTBOOL, 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>
```

```
KTBOOL amStreamServer(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be served

RETURN VALUE

KTBOOL, 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 Sets whether transfer occurs via DMA or non-DMA

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()
(Call not yet implemented)

amStreamIoSystemInit

Initializes the IO shell.

FORMAT

```
#include <am.h>

void amStreamIoSystemInit(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Initializes the IO shell.

See: `amStreamIoInstallAlternateIoManager()`

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

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 ammain.c

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 ammain.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 after 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 ac.h

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 after 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 after 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 after 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, KTRUE if the calculation was successful, KFALSE 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 array.

amVoiceCheck

For debugging the voice pool

FORMAT

```
#include <am.h>

void amVoiceCheck(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Prints a message for any channel whose status flag is set to `KTTRUE`.

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 am.h

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 after the call to `amStreamAllocateVoiceChannels()` and the call to `amStreamOpen()`.

amStreamRewind

Rewinds an open stream to its start.

FORMAT

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

```
KTBOOL amStreamRewind(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, The stream object to rewind.

RETURN VALUE

KTBOOL, 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 without 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 after 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 amStreamSetBuffer(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, KTTRUE,	If the buffer pointers were set.
KTFALSE,	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 after 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 _amStreamMonoIsr0 (unsigned int streamPtr);

void _amStreamMonoIsr1 (unsigned int streamPtr);

void _amStreamMonoIsr2 (unsigned int streamPtr);

void _amStreamMonoIsr3 (unsigned int streamPtr);

void _amStreamStereoIsr0 (unsigned int streamPtr);

void _amStreamStereoIsr1 (unsigned int 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 after 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 after the 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 after 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 after the call to `amStreamOpen()` or it will return 0

CALLS

`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 after 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 after the call to `amStreamOpen()` or it will return 0

amStreamGetTrackLengthInMs

Gets the length of a stream in milliseconds.

FORMAT

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

```
KTBOOL  amStreamGetTrackLengthInMs(AM_STREAM_PTR theStream,KTU32 trackNumber,KTU32
*lengthInMs)
```

PARAMETERS

AM_STREAM_PTR theStream, The stream object

KTU32 *lengthInMs, The length of a stream in milliseconds is returned via this pointer.

RETURN VALUE

KTBOOL

FUNCTION

Gets the length of a stream in milliseconds.

Note: This must be called after 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

KTBOOL

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 after 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 (0-15)

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 (0-31)

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, KTRUE 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.

amStreamInit

Initializes a stream object.

FORMAT

```
#include <am.h>

KTBOOL amStreamInit(AM_STREAM_PTR theStream,
KTSTRING fileName,
volatile KTU32 volume,
volatile KTU32 pan)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be initialized
KTSTRING fileName, the file name of the .str file to stream
volatile KTU32 volume, the volume at which to play the stream
volatile KTU32 pan, the pan of a mono stream, this is disregarded for stereo streams

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.

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`, The stream object to get the volume from

RETURN VALUE

KTU32, The current volume of the stream.

FUNCTION

Gets the current volume of a stream.

amStreamGetIsrCount

Gets the ISR 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>
```

```
KTBOOL amStreamIsStereo(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be queried.

RETURN VALUE

KTBOOL, 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>
```

```
KTBOOL amStreamServer(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be served

RETURN VALUE

KTBOOL, KTRUE if the stream is currently playing, KFALSE 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.

3. *The Redbook Playback API*

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

FORMAT

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

```
KTBOOL acCddaSetVolume (      KTU32 left_channel_volume, KTU32 right_channel_volume)
```

PARAMETERS

`left_channel_volume` Volume level for left audio channel, 0-127.

`right_channel_volume` Volume level for right audio channel, 0-127.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Sets volume level for CDDA playback channels.

Note: Left and right depend on CDDA Pan position.

acCddaSetPan

Sets Left & Right Channel pan position.

FORMAT

```
#include <ac.h>

KTBOOL acCddaSetPan      (KTU32 left_channel_pan,
                          KTU32 right_channel_pan)
```

PARAMETERS

left_channel_pan	Pan Position for left audio channel, 0-127, left to right.
right_channel_pan	Volume level for right audio channel, 0-127, left to right.

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Sets pan position for CDDA playback channels.

Note: Effects the behavior of volume command, since “left” channel can be set to the right pan position.

acCddaResetChannels Resets CDDA channels to hard pan positions and maximum volume.

FORMAT

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

```
KTBOOL acCddaResetChannels( void )
```

PARAMETERS

none

RETURN VALUE

KTTRUE if command write successful; KTFALSE otherwise.

FUNCTION

Sets pan position and volume for CDDA playback channels.

Note: Immediately sets redbook channels to default pan and maximum volume
