AMPlayer

Introduction and Overview

AMPlayer is a module for playing Audio MPEG data through the computers sound output, or to a streaming destination. Where sound is to be output, it will select the 8 or 16 bit output depending on the normal configuration (and availability).

The module has been designed so it is easy to make other frontends, or add support for the module to existing player-frontends.

The AMPlayer module supports MPEG version 1, 2 and 2.5, for layers I, II and III. Mixed data may be safely used with the module. The AMPlayer is resilient when faced with corrupt data, and will skip unknown data in a safe manner. Streams of data may contain ID3v2 tags, and may be terminated by ID3v1.1 tags. ID3v2 footers are skipped and not parsed.

Terminology

MPEG audio data is used as a generic term, covering MPEG 1/2/2.5 audio data, using layers I, II, or III.

The term VBR is used to mean 'Variable Bit-Rate'. VBR data is data which contains non-constant bitrate through the track. It is still assumed that the MPEG version and layer remain constant.

Technical Details

In order to play MPEG audio data, AMPlayer processes data in the background using callbacks. This allows the module to continue in, and out of the desktop with no supervision from any other component.

MPEG data input methods

MPEG audio data can come from either a file, or from a data stream. When operating in both of these modes, the functionality of AMPlayer is similar.

File playback

During file playback, the AMPlayer module continually takes data from the file as it needs it. File data is buffered by the module.

File playback will initially read the ID3v1.1 tags from the file if present, and store these for later retrieval by front ends. During playback, ID3v2 tags will be processed (if enabled) and services issued announcing their arrival.

Streaming playback

When streaming, an application must feed data to AMPlayer in a timely fashion. Data is supplied to a ring of buffers which are drained individually by the AMPlayer module. As each buffer is emptied it is the job of the streaming application to provide further data or to close the stream.

The player will take data from buffers supplied by the streaming application in the order given. When the end of a buffer is reached, the player will continue seamlessly to the next buffer. There is no necessity to provide frame-aligned data to the buffers. Where frames straddle the buffer end, it may be necessary for the player to retain the buffer until the frame can be processed. Because of this streaming application must supply at least two buffers and at least 2000 bytes in total for streaming playback. When a buffer is no longer required, a service will be issued to inform the streaming application.

Metadata can be supplied to the AMPlayer module during streaming. This data will be inserted in-line with the stream and made available at the point at which it is played. This data is incompatible with ID3v1.1 data, and with ID3v2 data. During playback, ID3v1.1 and ID3v2 tags are skipped. ID3v2 tags larger than the total size of the ring of buffers will cause the player to

stall.

File output methods

AMPlayer is able to output to a number of different destinations. Where the destination is the physical sound system (8bit, 16bit or SharedSound), the interface to the module is unchanged.

8 bit sound output

All sound output is generated as 16bit-stereo data. When only the 8bit sound system is available, a lookup table will be used by the output code to generate the correct logarithmic output. This will result in a slight degradation in quality, but only to the level of the system accuracy. The overall frequency of the sound system will track that of the sound being generated to the limits of the sound system itself. This may result in further inaccuracies.

Because AMPlayer takes over the entire sound system to handle its output, no other sounds (for example, the system beep) will be heard in this mode.

16 bit sound output

Where 16 bit sound is available, but SharedSound support is not, the standard 16 bit sound drivers will be used. The overall frequency of the sound system will track the sound being output to the limits that it is capable.

In this mode, AMPlayer uses the entire 16 bit sound system. As a consequence, no other sound will be generated whilst AMPlayer is playing.

SharedSound output

Where the SharedSound module is available, it will be used by the AMPlayer module. When playing through the SharedSound module, the frequency of the sound system is unaffected by playback. In this mode, the sound most closely matches that which would be generated within the limits of the sound system and its configuration. Importantly, if the overall sound system frequency is configured lower than any of the clients of SharedSound, the output quality will suffer.

Sound from other sources is unaffected by AMPlayer playback, allowing other clients to share the sound system.

Streaming output

AMPlayer can be used to stream generated sound data to any other destination through the use of the streaming output interface. In this mode, the sound system is totally unaffected by processing performed by the module. The streaming interface can obtain the frequency at which the data being read should be played through this interface.

Multiple instantiation

AMPlayer is able to function as a decoder for multiple clients. It achieves this by multiple instantiation of the AMPlayer module. The SWI *SWI AMPlayer_Instance (on page 73)* manages the instances for clients such that it is not necessary for every client to duplicate the same code.

In addition to this, all AMPlayer SWIs may have bit 31 of their flags set to indicate that the operation should be directed to the instance of the AMPlayer module held in R8. This allows background processes to communicate with just the instance to which they are interested. When services are issued to notify clients of events from the decoder, R8 will be set to the instance handle that generated the service.

If a SWI is not directed at a particular instance, then the currently preferred instance will deal with the request.

User front ends

Front end applications which are controlled by the user, queue tracks, or just monitor the state of the player should only communicate with the base instance of AMPlayer. This allows them to function with any number of other concurrent utilities.

Under most circumstances it is advisable that front ends not worry about the existance of multiple instances and merely communicate with the currently preferred instance. This allows for the greatest flexibility with clients selecting alternative instances as the private for control if necessary

Plugins

Plugins should be aware of the existance of plugins, and only register themselves with the base instance unless explicitly requested otherwise. Because of their nature, plugins are only really suitable for the base instance, or a secondary instance which is being mixed with the base.

* Commands

*Commands will always be issued to the currently 'preferred' instance. In general this will be the base, but under specialised circumstances another instance may be preferred. The AMPlayer module itself will retain the current preferred instance through all operations, and therefore the only mechanism by which another AMPlayer instance may be the preferred is by explicitly issuing the relevant OS_Module, or by issuing a * Command directly to an instance.

Data structures

In order to communicate with AMPlayer, a number of data structures are required. These provide information about the streams being processed.

File Information Block

The File Information Block provides information about the file currently being played.

Offset	Required flags	Contents
+0	none	flags : **Bit(s) Meaning** 0 Total time valid 1 Elapsed time valid 2 ID3 tag info pointers valid 3 VU values valid 4 Error message pointer valid 5 Next filename pointer valid 6 File uses variable bit rate, high and low rates valid 7 ID3v1.1 track valid 8 Data comes from stream 9-31 Reserved, must be zero
+4	none	buffer usage ratio in % (*)
+8	bit 0	projected total time in cs
+12	bit 1	time elapsed in cs
+16	bit 2	pointer to ID3 song title
+20	bit 2	pointer to ID3 artist
+24	bit 2	pointer to ID3 album name
+28	bit 2	pointer to ID3 year string
+32	bit 2	pointer to ID3 comment
+36	bit 3	left channel VU
+40	bit 3	right channel VU
+44	none	main volume (0..127) (*)
+48	bit 4	pointer to most recent error/warning message, or 0 if no message is pending
+52	bit 5	pointer to filename of the "next" file, or 0 if no file currently queued
+56	bit 2	ID3v1 genre (a number)
+60	bit 2+7	ID3v1.1 track (a number), or 0 if not specified
+64	bit 6	lowest bitrate used
+68	bit 6	highest bitrate used

Fields marked with (*) are not valid when returned from an AMPlayer_FileInfo call.

Projected total time

The projected total time for the track is based on the bitrate used by the file so far (unless supplied in another manner). It is assumed that the bitrate remains constant for this calculation. The total time given will be wrong if :

- the file size is unknown, e.g. if playing a stream
- the frame type will later change in a way that alters the number of bytes per frame (eg change in bit rate)
- the data is partially corrupt or contains skippable data (ID3v2 tags, rogue unsynchronised data, etc)

None of the above exceptions are true in the vast majority of MPEG files. The first case is determined by the module, and bit 0 of the flags will be clear. The second case cannot be known in advance, and it will also affect the elapsed time. No matter what happens, the time will always move forward, it just might not be counting centiseconds in these cases.

Within VBR files generated by the Xing encoder (or applying a Xing compatible header) the total time will be calculated from the header. If the file has been truncated, this time will be estimated based on the information in the header.

ID3v1 genre values

ID3v1 genre values are defined elsewhere. See http://www.id3.org for more details.

ID3v1.1 track numbers

ID3v1.1 is an extension to ID3 which, if present, declares the track number within an album.

VU-var values

When the VU level is available, it is a number between 0 and 255. The value is from -42 to 0 dB, in 1/6th dB steps. The level is the peak of the average level since last calling this SWI.

Bitrate values

Where VBR Audio MPEG data is being processed, the high and low bitrate values are used to indicate the current known limits of the data.

Frame Information Block

The Frame Information Block provides information about the most recent frame processed.

Offset	Contents
+0	MPEG version as 3 ASCII chars and a 0 terminator, e.g. "2.0"
+4	layer type (1..3). 0 is unknown layer
+8	sampling frequency in Hz
+12	bitrate in kbit/sec
+16	mode :

Mode	Meaning
0	Stereo
1	Joint-stereo
2	Dual channel
3	Single channel

+20	number of channels
+24	frame flags :

Bit(s)	Meaning
0	Copyright
1	Original
2	CRC

+28	pointer to left channel DCT array (*)
+32	pointer to right channel DCT array (*)

Fields marked with (*) are not valid when returned from an AMPlayer_FileInfo call.

Plugin Information Block

The Plugin Information Block is used when registering and enumerating plugins present.

Offset	Contents
+0	Filter name, padded with 0's (16 chars)
+16	Filter author, padded with 0's (32 chars)
+48	Filter version, padded with 0's (8 chars)

IDTag Information block

The IDTag Information Block is passed to the service handlers for ID3v2 processing. It provides information on the overall structure of the ID3v2 tag.

Offset	Contents
+0	Version of original tag data (major * 256 + minor)
+4	Header flags :

Bit(s)	Name	Meaning
7	HEADERF_UNSYNCHRONISED	Data was unsynchronised
6	HEADERF_EXTENDEDHEADER	Extended header was present (ignored at present)
5	HEADERF_EXPERIMENTAL	Tag is experimental (should never be seen by decoder)
4	HEADERF_FOOTER	Footer was included (has been ignored)
other		reserved, must be 0

IDFrame Information block

The IDFrame Information Block is passed to the service handlers for ID3v2 processing. It provides information on the specific ID3v2 frame being processed.

Offset	Contents
+0	frame number (within this tag)
+4	frame name (0 terminated)
+12	flags for this frame :

Bit(s)	Name	Meaning
0	FRAMEF_HASLENGTH	Length field was given
1	FRAMEF_UNSYNCHRONISED	Frame was unsynchronised
2	FRAMEF_ENCRYPTED	Frame was encrypted (N/I)
3	FRAMEF_COMPRESSED	Frame was compressed
6	FRAMEF_GROUP	Frame is one of a group
12	FRAMEF_READONLY	Frame should not be edited
13	FRAMEF_FILEDISCARD	Discard frame if file changed
14	FRAMEF_TAGDISCARD	Discard frame if tag changed
other		reserved, must be 0

+16	Pointer to frame data (decompressed, de-unsynchronised)
+20	Frame data length
+24	Encryption type, or -1 if not given
+28	Compressed length, or -1 if not compressed
+32	Frame group, or -1 if not given

Flags will be promoted to those used by ID3v2.4, if they are of a lower version than that.

Frame data will be terminated by a 0 (not included in the length) for ease of decoding text fields.

System variables

AMPlayer$Buffer$*

Default AMPlayer output buffer sizes

Use

When starting playback, AMPlayer checks the file being played against the variable `AMPlayer$Buffer$<part path>`, where *<part path>* is the longest component of the path name which is set as a system variable. For example, if you were to play `ADFS::Music.$.Lennon.Imagine` and had the system variables `AMPlayer$Buffer$` and `AMPlayer$Buffer$ADFS` set, the latter would be used in preference to the former.

The value given is used to determine the number of 'blocks' of output data that will be buffered in the 'Antishock buffer'. The larger this buffer is, the longer the system can be busy before playback ceases. Larger buffers have a greater initial load on the machine as more data is decoded to fill the buffer when the first file is played.

A 'block' is an arbitrary size, currently around 4.5K. The reason for supplying a buffering value in blocks is to provide a more robust means of storing the buffering size. If in future the block size changes, the amount of time that that buffer corresponds to will remain constant (for a given frequency of data).

In the case of streams, the variable `AMPlayer$Buffer$Stream` will be used to determine the initial output buffer size.

Related SWIs

SWI AMPlayer_Control 1 (on page 39)

AMPlayer$FileBuffer

Input file buffer size

Use

When starting playback, AMPlayer allocates a buffer for data from the file. Whereas `AMPlayer$Buffer$*` determines the output buffer size, `AMPlayer$FileBuffer` determines the input buffer size. If you are accessing files on a filing system which has a slow start up time, (for example networks or CDs) you may wish to set this higher value than the default.

The value this variable is set to is in Kilobytes. Unlike the output buffer, the input cannot be measured in blocks because that would require knowing in advance the data contained in the blocks.

For Streams `AMPlayer$FileBuffer` has no meaning and is not used.

AMPlayer$Volume

Volume level to use when AMPlayer starts

Use

When AMPlayer initialises, it reads `AMPlayer$Volume` to determine the initial volume. This is a linear volume level, with a maximum at 127, a minimum at 0, and a default level of 112.

Whenever the volume level is changed in the base instance, this system variable is updated to reflect this. When an instance starts (in the same manner as the base AMPlayer instantiation intialising), it reads the state of `AMPlayer$Volume` and sets its volume to that specified. The result of this is that at any time that an instance is created it starts with the same volume level as the base instance. Should an instance wish to control the volume level of its instance, it should do so with care and pay attention to the initial volume level where appropriate. For example, if the user is playing their base instance at a volume level of 12, they will not wish to have a new instance playing at 112 unless they specifically requested it.

Related SWIs

SWI AMPlayer_Control 1 (on page 39)

AMPlayer$DecimationThreshold

Select decimation threshold to use when AMPlayer starts

Use

When AMPlayer initialises, it reads `AMPlayer$DecimationThreshold` to determine the initial decimation threshold. This threshold is used to determine during playback whether decimation of input data is used to provide output data.

Whenever the decimation threshold is changed in the base instance, this system variable is updated to reflect this. When an instance starts (in the same manner as the base AMPlayer instantiation intialising), it reads the state of `AMPlayer$DecimationThreshold` and sets its threshold to that specified. The result of this is that at any time that an instance is created it starts with the same threshold as the base instance.

When decimating input data, the upper half of the frequency data is discarded, resulting in a frequency of half that normally required for the input data. This reduces the processing required by the data, and therefore reduces the load that AMPlayer places on the system. This speed increase is to the detriment of the quality of the output data.

Related SWIs

SWI AMPlayer_Control 5 (on page 51)

Service calls

Service_AMPlayer (Service Call &52E00)

Events issued by AMPlayer

On entry

R0 = reason code :

Reason	Meaning
0	*AMPlayer module is initialising (on page 17)*
0	*AMPlayer module is initialising (on page 17)*
1	*AMPlayer module is dying (on page 18)*
2	*Playback is about to start (on page 19)*
3	*Playback has stopped (on page 20)*
4	*Playback has moved on to another track (on page 21)*
5	*ID3v2 tag has been found (on page 22)*
6	*One or more buffers previously passed to an AMPlayer have been marked as free (on page 23)*

R1 = service call number

R2 - RR7 = dependant on reason code

R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer is issued by the AMPlayer module to inform clients of a change in state, or other information about playback. Consult the individual reason codes for more details.

Related APIs

None

Service_AMPlayer 0 Initialising (Service Call &52E00)

AMPlayer module is initialising

On entry

R0 = reason code
R1 = service call number
R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer 0 is issued by the AMPlayer module when it initialises. Clients wishing to add plugins to the output of the module should register themselves.

Related SWIs

SWI AMPlayer_Plugin (on page 53)

Service_AMPlayer 1 Dying (Service Call &52E00)

AMPlayer module is dying

On entry

R0 = reason code
R1 = service call number
R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer 1 is issued by the AMPlayer module (or an instance of AMPlayer) when it is killed. Clients wishing to only run during the lifetime of AMPlayer should either become dormant or terminate.

Related SWIs

SWI AMPlayer_Instance 2 (on page 79)

Service_AMPlayer 2 Start (Service Call &52E00)

Playback is about to start

On entry

R0 = reason code
R1 = service call number
R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer 2 is issued by the AMPlayer module when it is about to stat playing a track. Clients wishing to monitor the progress of the track in the background, or to schedule new tracks may wish to watch for this service.

Related SWIs

SWI AMPlayer_Play (on page 24)

Service_AMPlayer 3 Stop (Service Call &52E00)

Playback has stopped

On entry

R0 = reason code
R1 = service call number
R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer 3 is issued by the AMPlayer module when it has stopped playing and moved to state 'Dormant'.

Related SWIs

SWI AMPlayer_Stop (on page 26)

Service_AMPlayer 4 Change (Service Call &52E00)

Playback has moved on to another track

On entry

R0 = reason code
R1 = service call number
R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer 4 is issued by the AMPlayer module when it has changed to playing the next track queued. Clients wishing to track the file being played should watch for this service.

Related SWIs

SWI AMPlayer_Play (on page 24)

Service_AMPlayer 5
ID3v2
(Service Call &52E00)

ID3v2 tag has been found

On entry

R0 = reason code
R1 = service call number
R2 = pointer to IDTag Information Block
R3 = pointer to IDFrame Information Block
R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer 5 is issued by the AMPlayer module during playback, when an ID3v2 tag has been encountered. Clients wishing to process ID3v2 as they arrive should watch for this service.

Related SWIs

SWI AMPlayer_Control 3 (on page 43)

Service_AMPlayer 6 StreamBuffersAvailable (Service Call &52E00)

One or more buffers previously passed to an AMPlayer have been marked as free

On entry

R0 = reason code
R1 = service call number
R2 = stream handle
R3 = flag word (currently 0)
R8 = instance handle of issuing instance, or 0 for the base

On exit

R1 preserved

Use

Service_AMPlayer 6 is issued by the AMPlayer module during playback from an AMPlayer stream for which service call reporting was requested at creation time when one or more blocks previously passed to that stream have been marked as being freed. Clients wishing to be informed when their blocks are no longer being held by the module should watch for this service.

Related SWIs

SWI AMPlayer_StreamOpen (on page 57)

SWI calls

AMPlayer_Play (SWI &52E00)

Plays or queues a file

On entry

R0 = flags :

Bit(s)	Name	Meaning
0	Queue	Places the named file in the queue of tracks to play. If there is no file currently playing, the behaviour is exactly as if bit 0 were clear.
1	Cue	Starts the named file immediately, but paused at the first frame. Use *SWI AMPlayer_Pause (on page 28)* to start playback.
2	Transient	Creates a new instance, sets the volume level, marks the instance as transient, and starts the named file within that instance.
3-30		Reserved, must be 0.
31		R8 contains the instance handle to which this call should be directed.

R1 = pointer to filename
R2 = if bit 2 of R0 set:
volume level to set, or -1 for default
R3 = if bit 2 of R0 set:
pointer to instance name, or 0 to name automatically
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 = if bit 2 of R0 set:
handle of created instance

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to play or queue a file for playback. When starting play transiently, the current play is unchaged and only the new instance is affected.

Related * commands

*AMPlay (on page 89)

Related SWIs

SWI AMPlayer_Stop (on page 26)
SWI AMPlayer_Pause (on page 28)
SWI AMPlayer_Instance (on page 73)
SWI AMPlayer_Control 4 (on page 49)

AMPlayer_Stop (SWI &52E01)

Stops playback

On entry

R0 = flags :

Bit(s)	Name	Meaning
0	Cut	Playback continues with the queued file if any. If no file is queued, playback will stop.
1-30		Reserved, must be 0.
31		R8 contains the instance handle to which this call should be directed.

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to stop playback, or start the skip to the queued file.

Related * commands

*AMStop (on page 90)

Related SWIs

SWI AMPlayer_Play (on page 24)
SWI AMPlayer_Pause (on page 28)

AMPlayer_Pause (SWI &52E02)

Pauses playback

On entry

R0 = flags :

Bit(s)	Name	Meaning
0	Resume	Resumes playback.
1-30		Reserved, must be 0.
31		R8 contains the instance handle to which this call should be directed.

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to pause or resume playback. When paused, decoding to the output buffer continues, but at a much reduced rate. There is no sound output.

Pause mode may also be cancelled by stopping. If *SWI AMPlayer_Stop (on page 26)* is used to cut to the next file, or if a different file is started, pause mode will continue to be in effect, freezing the new file at the start of the

file. This can be used to ensure that playback starts at the instant of calling AMPlayer_Pause (as opposed to calling *SWI AMPlayer_Play (on page 24)*, which can have a delay while opening the file etc).

Related * commands

*AMPause (on page 88)

Related SWIs

SWI AMPlayer_Play (on page 24)
SWI AMPlayer_Stop (on page 26)

AMPlayer_Locate (SWI &52E03)

Locates a position in the playback

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = target time in centi-seconds
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI locates the position of the target time, and continues playback (or pausing) from there. This has no effect unless the status is either Playing, Locating or Paused. This may take some time, and the playback buffer may empty (which will mute the sound).

The time given here corresponds to the elapsed time returned from *SWI AMPlayer_Info (on page 32)*. This is true even when the elapsed time is

wrong. So when, at time X, the Info call returns the wrong time Y, giving time Y to this call will still start playing at the right time X.

Playback can only start on a frame boundary, so the resolution of the start point is around 2 cs (for 128kbit/sec, 44.1kHz frames).

Related * commands

*AMLocate (on page 87)

AMPlayer_Info (SWI &52E04)

Return information on the state of playback

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 = Player status :

State	Meaning and registers
0	Dormant R1 - R4 preserved
1	Starting R1 = pointer to filename R2 - R4 preserved
2	Locating R1 = pointer to filename R2 = pointer to *File Information Block (on page 6)*, or 0 if not set up yet R3 = pointer to *Frame Information Block (on page 9)*, or 0 if not set up yet R4 = target time
3	Playing R1 = pointer to filename R2 = pointer to *File Information Block (on page 6)*, or 0 if not set up yet R3 = pointer to *Frame Information Block (on page 9)*, or 0 if not set up yet R4 preserved
4	Pausing R1 = pointer to filename R2 = pointer to *File Information Block (on page 6)*, or 0 if not set up yet R3 = pointer to *Frame Information Block (on page 9)*, or 0 if not set up yet R4 preserved
5	Stopping R1 = pointer to filename R2 - R4 preserved
6	Changing R1 = pointer to filename R2 - R4 preserved
7	Cueing R1 = pointer to filename R2 - R4 preserved

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This call will return information about the current state of the player.

When locating, the current time can be read from the file info block, as it moves toward the target time returned in R4.

This call might be made from BASIC with :

```
SYS "AMPlayer_Info",,"" TO ,Filename$,FIB%
```

This will set Filename$ to either "" or the filename. Similarily, FIB% will be 0 if there is no info at this stage, or a pointer to it if there is.

There is a brief period when the status might be returned as Locating (2) or Playing (3), but where there is no valid FIB or FRIB, because the first frame has yet to be read.

Related * commands

*AMInfo (on page 86)

Related SWIs

SWI AMPlayer_FileInfo (on page 55)

AMPlayer_Control (SWI &52E05)

Configure the operation of the player

On entry

R0 = flags :

Bit(s)	Meaning
0-7	reason code :

Reason	Configuration
0	*Read or write the volume level (on page 37)*
1	*Read or write the output buffer size (on page 39)*
2	*Set SVC stack check level (on page 41)*
3	*Control the ID3v2 tag processing facilities (on page 43)*
4	*Read or write the 'transience' flag (on page 49)*
5	*Read or write the decimation threshold (on page 51)*

8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to configure various aspects of the AMPlayer module's operation.

Related APIs

None

AMPlayer_Control 0 Volume (SWI &52E05)

Read or write the volume level

On entry

R0 = flags :

Bit(s)	Meaning
0-7	0 (reason code)
8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = new volume level (0-127), or -1 to read current level
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R1 = old volume level

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read or write the main volume level of the player.

Related * commands

*AMVolume (on page 91)

Related SWIs

SWI AMPlayer_Control (on page 35)

Related system variables

AMPlayer$Volume (on page 14)

AMPlayer_Control 1
BufferSize
(SWI &52E05)

Read or write the output buffer size

On entry

R0 = flags :

Bit(s)	Meaning
0-7	1 (reason code)
8	Use blocks, rather than bytes
9-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = new buffer size in bytes or blocks, or -1 to read current size
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R1 = old size in bytes or blocks

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This control call is used to read or write the size of the audio buffer used by AMPlayer to store decoded data. The size can be specified in bytes or in blocks. It is recommended that you use the blocks size to be compatible with the system variable usage, and to ensure that similar amounts of data are buffered in future.

If the buffer isn't currently created, this controls how large it will be when it eventually is. If it exists, OS_ChangeDynamicArea is used to change the size. This may fail with an error, even if some of the job was done (this can happen when reducing the size, as the amount that can be released depends on what is currently being played). This effect is greatly reduced in AMPlayer 1.29 and later.

If it succeeds, the sound may be broken up slightly. On versions of AMPlayer prior to 1.29, this will always cause at least one 'jump' in the playback.

Related SWIs

SWI AMPlayer_Control (on page 35)

Related system variables

AMPlayer$Buffer$* (on page 12)

AMPlayer_Control 2 StackSize (SWI &52E05)

Set SVC stack check level

On entry

R0 = flags :

Bit(s)	Meaning
0-7	2 (reason code)
8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = new level (in words), or 0 for default
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R1 = old level

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI call is used to set the stack check level used when a callback occurs

When receiving a callback, the SVC stack depth is checked to see if the

kernel is reasonably unthreaded. By using this call, you can control what is considered "reasonable". The default value is currently 64, i.e. if there are more than 64 words on the stack by the time of the callback, a new callback will be registered later instead. Setting this too low will cause the player to stall, and you can only stop it by killing the module (or putting the level back up).

It is not expected that users of the AMPlayer API will need to use this SWI.

Related SWIs

SWI AMPlayer_Control (on page 35)

AMPlayer_Control 3
ID3v2Control
(SWI &52E05)

Control the ID3v2 tag processing facilities

On entry

R0 = flags :

Bit(s)	Meaning
0-7	3 (reason code)
8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = sub-reason code :

Sub-reason	Meaning
0	*Enable or disable ID3v2 processing (on page 45)*
1	*Select ID3v2 frame filtering (on page 47)*

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R1 = old level

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This control call is used to control ID3v2 processing within the player.

ID3v2 processing is quite intensive and can have a performance hit, especially when compressed frames are used.

Related SWIs

SWI AMPlayer_Control 3 (on page 43)

AMPlayer_Control 3, 0
ID3v2State
(SWI &52E05)

Enable or disable ID3v2 processing

On entry

R0 = flags :

Bit(s)	Meaning
0-7	3 (reason code)
8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = 0 (sub-reason code)
R2 = type of change :

Type	Meaning
0	disable
1	enable
-1	read current state

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R2 = enable count, or 0 if disabled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to enable or disable the processing of ID3v2 tags by the AMPlayer module.

When disabled, no ID3v2 processing at all is performed - the tags are merely skipped. This will improve performance when such tags are encountered.

Related SWIs

SWI AMPlayer_Control 3 (on page 43)

AMPlayer_Control 3, 1
ID3v2Filtering
(SWI &52E05)

Select ID3v2 frame filtering

On entry

R0 = flags :

Bit(s)	Meaning
0-7	3 (reason code)
8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = 1 (sub-reason code)
R2 = pointer to frame name, or 0 for 'all frames'
R3 = filtering operation :

Type	Meaning
0	disable frame processing
1	enable frame processing
-1	read current state of processing
-2	read frame state as matched by processing engine

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R2 = enable count, or 0 if disabled

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to enable or disable specific frames or groups of frames for processing through the service interface.

A single character frame name will select all frames starting with that character. Three character frame names select the ID3v2.2 and earlier frames that match. Four character frame names select the ID3v2.3 and ID3v2.4 frame explicitly.

Claimants of the ID3v2 service should enable the frames they wish to see, and disable them when they no longer require them.

Related SWIs

SWI AMPlayer_Control 3 (on page 43)

AMPlayer_Control 4 Transience (SWI &52E05)

Read or write the 'transience' flag

On entry

R0 = flags :

Bit(s)	Meaning
0-7	4 (reason code)
8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = Operation to apply:

Value	Meaning
0	Mark as intransient
1	Mark as transient
-1	Read current flag

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R1 = old transience flag

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read or write the transiency flag for an instance. Transiency has no meaning for the base instance, but for created instances, it means that when playback completes, the instance will be destroyed automatically.

Related SWIs

SWI AMPlayer_Control 3 (on page 43)

AMPlayer_Control 5 DecimationControl (SWI &52E05)

Read or write the decimation threshold

On entry

R0 = flags :

Bit(s)	Meaning
0-7	5 (reason code)
8-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = Operation to apply:

Value	Meaning
0	Read decimation threshold
1	Set decimation threshold

R2 = new decimation threshold
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R1 = current decimation threshold

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read or write the 'decimation threshold'. AMPlayer will normally decode data at full accuracy. If the 'decimation threshold' frequency is met or exceeded, the module will apply automatic decimation to the output.

The initial decimation value is based on the version of the module in use (for AMPlayer, this is 1000000, for AMPlayerFP, it is 22050, and for AMPlayer6 it is 0). This can be overridden by setting the `AMPlayer$DecimationThreshold` variable to a number before starting the module. Setting the decimation threshold of the base instance will set the variable to the same value.

On builds that do not support decimation, the calls are ignored, and reading the decimation always returns 1000000.

Related SWIs

SWI AMPlayer_Control (on page 35)

Related system variables

AMPlayer$DecimationThreshold (on page 15)

AMPlayer_Plugin (SWI &52E06)

Plugin operations

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = reason code :

Value	Meaning
0	Register plugin
1	Deregister plugin

R2 = private word to pass in R0
R3 = pointer to *pre-processor (pre-DCT) (on page 84)*, or 0 for none
R4 = pointer to *post-processor (post-DCT) (on page 85)*, or 0 for none
R5 = pointer to static *Plugin Information Block (on page 9)*
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to register or deregister plugins.

Related APIs

None

AMPlayer_FileInfo (SWI &52E07)

Return information on the a file

On entry

R0 = flags for requested information, or 0 to return size of buffers :

Bit(s)	Meaning
0	Return total time, and Frame Information Block for the first frame
1	Reserved, must be 0
2	Return ID3 tag information
3-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = pointer to filename
R2 = pointer to buffer for *File Information Block (on page 6)*
R3 = pointer to buffer for *Frame Information Block (on page 9)*
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R2 = required size of the File Information Buffer, if reading size of buffer
R3 = required size of the Frame Information Buffer, if reading size of buffer

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI attempts to return useful information about the file given. The more bits you set in R0, the slower it gets, as it requires more of the file to be read. In particular, requesting the ID3 tag information means seeking to the end of the file, which may be very slow on some filing systems (like DATFS).

Therefore, it sometimes makes sense to make 3 calls:

1. With R0=&000, to get the size of the buffers.
2. With R0=&001, to read the quick things and determine whether the file is interesting at all.
3. With R0=&004, to read only the ID3 tag fields.

Remember to read the buffer sizes first. These information blocks will no doubt be extended, and if you assume the old size, your program will stop working when a new AMPlayer module comes out. Just read the size, and your program will continue to work for eternity. You don't need to supply a frame info pointer in R3 unless bit 0 is set.

Related * commands

*AMInfo (on page 86)

Related SWIs

SWI AMPlayer_Info (on page 32)

AMPlayer_StreamOpen (SWI &52E08)

Starts a stream playing

On entry

R0 = flags :

Bit(s)	Name	Meaning
0	Queue	Places the named file in the queue of tracks to play. If there is no file currently playing, the behaviour is exactly as if bit 0 were clear.
1	Cue	Starts the named file immediately, but paused at the first frame. Use *SWI AMPlayer_Pause (on page 28)* to start playback.
2	GenerateService	Indicates that *Service_AMPlayer 6 (on page 23)* should be generated every time one or more blocks passed into the stream become free.
3-30		Reserved, must be 0.
31		R8 contains the instance handle to which this call should be directed.

R1 = pointer to stream name (for information)
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 = stream handle, or 0 if failed to start

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to start streamed input, ready for playback.

Called by a streaming application to get a Stream Handle. Most streamers will start the stream paused and feed data in until the buffer full flag (see *SWI AMPlayer_StreamInfo (on page 63)*) is set.

Note: The combination of Queue and Start Paused may not work.

Related SWIs

SWI AMPlayer_Play (on page 24)
SWI AMPlayer_StreamClose (on page 59)

AMPlayer_StreamClose (SWI &52E09)

Informs AMPlayer that a stream has ended

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = stream handle
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to inform the AMPlayer module that no more data will be supplied to the stream. This does NOT release outstanding buffers that have been passed to AMPlayer. If you need to get these back, call AMPlayer_StreamClose, then *SWI AMPlayer_Stop (on page 26)*.

Related SWIs

SWI AMPlayer_StreamOpen (on page 57)

AMPlayer_StreamGiveData (SWI &52E0A)

Inform AMPlayer of the location of input data

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = stream handle

R2 = pointer to streaming data block :

Offset	Contents
+0	Usage word
+4	Meta data list associated with this buffer, or 0 for no data
+8	Length of data following

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

Called by a streamer application to pass a block of data to AMPlayer. All but the first word may be considered read-only and must be in interrupt sharable space (i.e. in a dynamic area, module area, sprite pool etc, NOT in the application). By calling this SWI, the application must guarantee that it will keep the buffer around (and unchanged) at least until AMPlayer sets the usage word to zero.

The streamer application should monitor the first word of the buffers it has previously passed in to see when it gets set to 0 to allow reuse. Buffers will be released strictly in the same order they were passed in.

The metadata list blocks must be kept intact for the same length of time as the data blocks, and can be considered 'released' by AMPlayer when their corresponding data blocks are.

Metadata blocks are in the following format:

Offset	Contents
+0	Pointer to next metadata block, or 0 for the last entry
+4	Pointer to a 0-terminated 'Key name' field
+8	Pointer to a value field (often a 0-terminated string, but may be binary data)
+12	Length of value buffer in bytes

It is envisaged that there will be a SWI added later to allow metadata to be supplied for non-streaming sources.

Related SWIs

SWI AMPlayer_StreamOpen (on page 57)
SWI AMPlayer_StreamClose (on page 59)

AMPlayer_StreamInfo (SWI &52E0B)

Read information about the streaming data

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = stream handle
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 = flags word :

Bit(s)	Meaning
0	Stream is active (actively being processed)
1	Output buffer has been full
2	Stream is paused

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is called by a streaming application to monitor the state of the decoder. Typically a streamer will start a stream up paused and feed it data. The streamer application will then wait until the stream becomes active, and until either the output buffer becomes full, or until it runs out of buffer space itself. Then it can unpause the stream knowing that the maximum amount of buffering is in use.

Related APIs

None

AMPlayer_MetaDataPollChange (SWI &52E0C)

Read meta-data state value

On entry

None

On exit

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = opaque unique value, guaranteed to change to a new unique value when any metadata items change.
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is called by streaming applications to discover when meta-data provided by the application has been 'passed' by the decoder and is now active.

Related SWIs

SWI AMPlayer_MetaDataLookup (on page 67)

AMPlayer_MetaDataLookup (SWI &52E0C)

Read meta-data token value

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = pointer to key field to match
R2 = pointer to buffer for result (or NULL to read length)
R3 = length of buffer, or 0 to read required length
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 preserved
R1 = buffer, filled with data to length given
R2 = length of buffer required, including terminator

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is called by applications to read meta-data passed to the AMPlayer module. Every element of meta-data is tagged with a 'key' which is used to return its value.

Related SWIs

SWI AMPlayer_MetaDataPollChange (on page 65)

AMPlayer_SoundSystem (SWI &52E0E)

Read sound systems available, or set output sound system

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = Operation type :

Type	Meaning
-1	Read available sound systems
0	Select any available system (best possible)
1	Select 8 bit sound system
2	Select 16 bit sound system
3	Select SharedSound
4	Select 'User' sound system (data read via *SWI AMPlayer_StreamReadData (on page 71)*)

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 = Bit mask of avialable sound systems :

Bit(s)	Meaning
0	8 bit output available
1	16 bit output available
2	SharedSound output available
3	User output available

R1 = Last sound system in use (as on input)

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to determine what sound systems are available on the current machine, and to select a different sound system. Its most common use is to select the User sound system for streaming output data.

Related SWIs

SWI AMPlayer_StreamReadData (on page 71)

AMPlayer_StreamReadData (SWI &52E0F)

Read data from streaming sound system

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = Unused
R2 = pointer to next byte of data to read, or 0 for first call
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R2 = pointer to first byte of data to read, or 0 if none available
R3 = number of bytes of data available
R4 = frequency of data

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to take data from the 'user' sound system. Initially, users would call with R2 = 0, to determine where to read data from. They then take use from this address up to the limit supplied. Subsequently, the call this SWI again to inform AMPlayer how much data has been read. AMPlayer will only remove data after it has been informed that it has been read.

Decoding will continue in the background. It may be necessary to delay processing if no data is ready. Data can only be generated on callbacks. It may, therefore, be necessary to wait for a moment or so for the decode to function.

VU-bars are inactive when use user sound state is in use.

Callers should note that only complete frames fed in will produce any output - not all the input stream may be used.

Related SWIs

SWI AMPlayer_SoundSystem (on page 69)

AMPlayer_Instance (SWI &52E10)

Manipulate AMPlayer instances

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = Reason code :

Reason	Meaning
0	*Read current instance handle (on page 75)*
1	*Create a new instance of the module (on page 77)*
2	*Destroy an instance of the module (on page 79)*
3	*Read handle of base instance (on page 81)*
4	*Enumerate handles of known instances (on page 82)*

R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 - R7 = dependant on reason code

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used control instances of AMPlayer. You should use it in preference to directly creating instances of the AMPlayer module yourself. Consult the reason code descriptions for more details.

Related APIs

None

AMPlayer_Instance 0 Current (SWI &52E10)

Read current instance handle

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = 0 (reason code)
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 = instance handle

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used return the instance handle of the current instance of AMPlayer. This may be used to discover the handle of the preferred instance such that the preferred instance may be changed to allow fading from one track to another. It is not expected that this call be used often.

Related APIs

None

AMPlayer_Instance 1 Create (SWI &52E10)

Create a new instance of the module

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = 1 (reason code)
R2 = pointer to zero terminated suffix for the instance name. Ideally you should keep this to < 16 characters.
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R0 = instance handle of new instance

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used create a new instance of the AMPlayer module. This instance may be used in exactly the same manner as any other, if SWIs are directed at it. Set bit 31 of any SWIs flags to direct the call to the instance whose handle is in R8.

Related APIs

None

AMPlayer_Instance 2 Destroy (SWI &52E10)

Destroy an instance of the module

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = 2 (reason code)
R2 = instance handle to destroy
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

None

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used destroy an instance of the AMPlayer module. You should destroy instances when you have no further use for them, to free up resources and processing time.

Related APIs

None

AMPlayer_Instance 3 ReadBase (SWI &52E10)

This SWI call is for internal use only. You must not use it in your own code.

AMPlayer_Instance 4 Enumerate (SWI &52E10)

Enumerate handles of known instances

On entry

R0 = flags :

Bit(s)	Meaning
0-30	Reserved, must be 0.
31	R8 contains the instance handle to which this call should be directed.

R1 = 4 (reason code)
R2 = last instance handle, or -1 to start enumerating
R3 = pointer to buffer to write name into
R4 = length of buffer
R8 = if bit 31 of R0 set:
instance handle to direct at, or 0 for the base

On exit

R2 = instance handle, or -1 if no more
R4 = length written to buffer, or -ve length if failed to write

Interrupts

Interrupts are undefined
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to enumerate the instances currently in use by AMPlayer. A 'visualisation' application might use this call to select what source to provide a representation of.

Related APIs

None

Entry points

Plugin_PreProcess

Pre-process DCT blocks

On entry

R0 = private word
R1 = pointer to 'out 1' buffer
R2 = pointer to 'out 2' buffer
R3 = pointer to 32 frequencies (16.16 format)

On exit

R0 = 0 if frequencies processed
1 if DCTs done (not recommended)

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is not re-entrant

Use

This entry is called prior to 'dct64' which decodes the frequencies into the buffer. Note that there is no way in which to identify whether the buffer is for the left or right channel.

Related APIs

None

Plugin_PostProcess

Post-process DCT output buffers

On entry

R0 = private word
R1 = pointer to 'out 1' buffer
R2 = pointer to 'out 2' buffer
R3 = pointer to samples buffer

On exit

R0 = 0 if buffer processed

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

Entry point is not re-entrant

Use

This entry is called after the 'dct64' which decodes the frequencies into the buffer. Note that there is no way in which to identify whether the buffer is for the left or right channel.

Related APIs

None

*Commands

*AMInfo

Read information on playback, plugins or file

Syntax

```
*AMInfo [-plugins] [-file <filename>]
```

Parameters

`-plugins`	- Display information about the plugins currently active.
`-file <filename>`	- Display information about a file, rather than about the current playback state.

Use

This command is used to display information about the current playback state, or the state of the plugins.

Examples

```
*AMInfo -file
Corrs,The.01ForgivenNotForgotten.01ErinShore(TraditionalIntro)
```

Related SWIs

SWI AMPlayer_Info (on page 32)
SWI AMPlayer_FileInfo (on page 55)
SWI AMPlayer_Plugin (on page 53)

*AMLocate

Skip to a position in the current file

Syntax

```
*AMLocate [+ | -] <hours>:<minutes>:<seconds>
*AMLocate <minutes>:<seconds>
```

Parameters

`<hours>` - Number of hours to search for

`<minutes>` - Number of minutes to search for

`<seconds>` - Number of seconds to search for

Use

This command is used to jump to a point in the playback of the file. Use + and - to indicate a location relative to the current playback position.

Examples

```
*AMLocate 7:23
```

Related SWIs

SWI AMPlayer_Locate (on page 30)

*AMPause

Pause, or resume, playback

Syntax

```
*AMPause [-off]
```

Parameters

`-off` - resume playback, instead of pausing

Use

This command is used to pause, or resume playback.

Examples

```
*AMPause
```

Related SWIs

SWI AMPlayer_Pause (on page 28)

*AMPlay

Play a new file

Syntax

```
*AMPlay [-next] [-queue] [-cue] [-transient] <filename>
```

Parameters

`-next`	- Start playback of queued file immediately
`-queue`	- Queue this file, rather than playing it immediately
`-cue`	- Start decoding this track, but leave the player paused
`-transient`	- Start decoding this track in a new instance which will terminate when playback completes.

Use

This command is used to play a new file.

Examples

```
*AMPlay -queue
ADFS::4.$.Music.Artists.Dido.NoAngel.06Thankyou
```

Related SWIs

SWI AMPlayer_Play (on page 24)

*AMStop

Stops playback

Syntax

```
*AMStop
```

Parameters

None

Use

This command is used to stop playback.

Examples

```
*AMStop
```

Related SWIs

SWI AMPlayer_Stop (on page 26)

*AMVolume

AMVolume

Syntax

```
*AMVolume [+ | -] <volume>
```

Parameters

<*volume*> - volume level (0-127)

Use

This command is set the playback volume. Initially, 113 is selected.

Examples

```
*AMVolume 113
```

Related SWIs

SWI AMPlayer_Control 0 (on page 37)

Related system variables

AMPlayer$Volume (on page 14)

Document information

Maintainer(s): AMPlayer developers <amplayer@amplayer.org>

History:

Revision	Date	Author	Changes
1		JRF	First XML monitored version ● First version of the document which includes revision information.

Related: None

Disclaimer: This document is, to the best of our knowledge, a correct representation of the API of AMPlayer.