KRAWALL
Developer's Handbook

Sebastian Kienzl

June 10, 2002

Contents

Chapter 1

Introduction

1.1 Preamble

This documentation is definitely not as detailed as it could be. However controlling Krawall is very simple and most API-functions are self-explanatory. So these are only some annotations on things that aren't immediately clear by looking at the API only. If however any questions arise don't hesitate to contact me, I will either compile a FAQ or add it to the documentation. There are also plenty of grammatical errors — I hope you, dear reader, will bear with it. If you stumble across a really serious mistake I'd be happy to correct it.

1.2 What is Krawall?

KRAWALL is a complete sound solution for the Gameboy Advance by Nintendo. It is is geared towards game-development and therefore supports playback of music and low-latency sound effects both based on 8-bit samples.

Speed is, besides quality, the most important viewpoint and we have designed and written Krawall with that clearly in mind. An averagely loaded eight-channel song with four soundeffects replayed at 16kHz/stereo usually doesn't peak at more than 20% cpu-time, the average usage is considerably lower.

The sound quality at the same sample-frequency is quite stunning in our opinion: Krawall doesn't click or make other noises. If you have a little more cpu-time to spare Krawall provides a high-quality resampling-mode which is even smoother.

Chapter 2

Implementing Krawall

This chapter discusses how to use Krawall in your project.

We have tried to keep Krawall's surface as simple as we could and we're sure you will have your GBA making noise in no time.

2.1 Compiling and linking

Krawall is written in C and ASM but can of course also used in C++-projects. The library `krawall.lib` is an incrementally linked object file in ELF-format and must be linked with GCC's `-mthumb-interwork` switch against your project.

The example that comes with Krawall illustrates compiling and linking projects with Krawall very well, it also includes startup-code and a linkscript.

There's also a version for use with the ARM SDT-compiler, it's available for commercial use only, tho.

2.1.1 Header files

There are various header files you might want to include. `krawall.h` defines all Krawall-related calls, `mtypes.h` defines the data-structures you will need. The conversion-tools also creates the files `modules.h` which defines all modules that have been converted and `samples.h` which `#define`'s some samples by name to use make them available for easy use as SFX. See 4.1 for details.

2.1.2 Imported symbols

Krawall needs some standard libc-calls like `bzero()` or `memcpy()`. Besides that symbols like `__divsi3`, `_call_via_r2` and so on are imported, but these are symbols GCC takes care of while linking.

The only symbol you need to take care of is `samples`. It is defined in `<mtypes.h>` and represents, as the name says, all samples. It's defined in a compiled file generated by the conversion tool — see 4.1 for details.

2.2 Resources

You need to be aware of what hardware is used by Krawall so your program doesn't interfere with it: Timers 0/1, DMA-Channels 1/2 and of course the Directsound-Hardware.

Do not fiddle with any of these or any sound-registers while Krawall is active. Krawall does not utilize the synthesis-chip so it could be used. If you do be especially careful with the register `SGCNT0` and `SGCNT1`: do not change any bits that are associated with the Directsound-hardware.

2.2.1 Memory Usage

Krawall uses very little memory. The patterndata in the ROM is compressed.

ROM	17kb + module/sample-data
EWRAM	4.4kb
IWRAM	6.6kb (8kHz), 8.7kb (16kHz), 13kb (32kHz)

2.2.2 CPU Usage

This is one of Krawall's strength: it's speed. Playback of an ordinary eight-channel module at 16kHz can be done with a peak-usage of less than 18% of a frame, the average usage is usually much less. These measures were taken at 16kHz.

Constant Usage	2.6%	
	Usage per voice	
Panning position	Normal	High Quality
Far left/right	1.13%	2.55%
Centered	1.25%	2.67%
Stereo (arbitrary panning)	1.49%	2.85%

So playing eight channels panned to either far left or right would take 2.6 + 8 * 1.13 = 11.64%. The *High Quality* mode is documented later.

The player-logic will peak at max. 6% in heavy-load situations (measured for a 14-channel tune).

2.3 Initialization

All you need to do is call `kragInit()` *once* at start-up with the appropriate parameters. After that Krawall is ready to go. You must start mixing (see 2.5 Mixing) after `kragInit()` as soon as possible!

2.4 Interrupts

Around every fourth or eighth frame (depending on replaying-frequency) the Timer1-IRQ is triggered. You must tie the function `kradInterrupt()` to that interrupt or Krawall will not work properly. This function's purpose is to reset the DMA (which is unfortunately needed on the GBA) and should get executed with ideally no delay.

If you have interrupt-service routines that take time (more than a few rasterlines) you should consider allowing multiple interrupts to happen to ensure that the Timer1-IRQ gets serviced as soon as possible. If there are audible clicks in the replay then `kradInterrupt()` probably doesn't get called fast enough.

You must enable interrupts in the "Interrupt Master Enable Register" as well; `kragInit()` doesn't do that.

2.5 Mixing

So far we have only been talking about initialization! Now let's see how the actual work is done.

Most games are frame-based. This means that every frame the same stuff is done: Check user input, do game logic, mix sound and do the gfx once the 'beam' is off-screen and so on. Hence the function `kramWorker()` needs to get called one per frame. It will mix as much samples as it takes to provide playback without skips.

If you plan to call `kramWorker()` in an interrupt (i.e. VCount Match) make sure you don't call any Krawall-functions which might get interrupted by just that interrupt or you will very likely run into typical problems that occur in such multi-process situations.

In the case that `kramWorker()` doesn't get called for one or two frames (i.e. in heavy-load situations) Krawall might cope with it but you really should avoid that happening.

A small detail: `kramWorker()` mixes a few samples more than are actually needed and thus will sometimes (around every 400 frames) mix nothing at all and return immediately.

The GBA's screen operates at approximately 60Hz, so you might decide to "do everything" at 30Hz, because your game is too CPU-intense to do 60 fps. An extra-version of the library named `krawall30.lib` is provided for that purpose, too.

Chapter 3

Using Krawall

This chapter focusses on how to control Krawall: play songs, sound effects, pause, fade, use the callback and so on. As said before, if you don't understand some of this (unlikely tho) have a look at the example.

3.1 Controlling music-playback

You'll probably want to play the modules you have converted, the conversion-tool creates a file called `modules.h` that defines all available modules.

Let's explain the distinction between *modules* and *songs* first. A module is the whole S3M-file and is played via `krapPlay()`. If the module is played in song-mode then each partition of the order-list is considered a song. A partition is made by inserting the special marker `+++` into the order-list. If additionally in loop-mode, only the current song gets looped, not the whole module. When in normal mode, those markers are ignored.

You can always directly jump to a partition by specifying a song number other than 0 when calling `krapPlay()`.

The pause mode also needs some annotation: `krapPause()` pauses music-play back but you may also pause all active sfx-channels. Note that you can still play sound effects once Krawall is in pause-mode! `krapUnpause()` will release all paused channels and continue playback.

Another nifty feature is playing jingles. You can play a song/module as a jingle if you give the mode `KRAP_MODE_JINGLE` when calling `krapPlay()`. This will interrupt the currently playing tune and immediately start playing this module. Once the jingle is over, the previously interrupted tune will resume playback.

See API-documentation for detailed information on all other calls available.

3.2 Controlling sound effects

Playing and controlling sound effects is quite simple. `kramPlay()` and `kramPlayExt()` start a sample and return a channel-handle (`chandle`) to identify the channel for subsequent calls. These functions will return zero if no channel could be allocated. The parameter `sfx` specifies whether the sample to be played should be treated as an sfx or a music-sample. This is necessary because music-channels and sfx-channels are in different volume-groups.

Functions like `kramSetFreq()`, `kramSetVol()` and `kramSetPan()` can be used to control a playing channel and return a numeric value indicating whether the call has been successful or not. It will fail if an attempt to control a voice which has been allocated to a different channel is made. This will for example happen if the sample that has been played is a one-shot sample and has already ended.

For details on these functions see the API-documentation.

3.3 Volumes

Krawall distinguishes between the volume of music and volume of sound effects.

`krapSetMusicVol( uint vol, int fade )` where `vol` goes from 0 to 128 will directly scale the global volume of the currently playing module. If `fade` is true then the volume will be faded towards the target-volume. The fading-speed depends on the current tempo of the song. If the player is paused or stopped `fade` is ignored.

The equivalent for sfx is `kramSetSFXVol( uint vol )`, where `vol` ranges from 0 to 128 as well. This will immediately change the volumes of all currently playing sfx's.

3.4 Callback

There's the possibility to install a Callback-function in Krawall with the function `krapCallback()`. It can be used to get notification of certain events. These events are:

KRAP_CP_FADE This will occur when the target-volume has been reached when fading music-volume (see 3.3).

KRAP_CB_DONE Once a module has reached it's end this will occur, no matter if in loop mode or not.

KRAP_CB_MARK This can be handy if you want to synchronize the music with something. When the effect `Zxx` occurs in a pattern this will occur, the parameter `xx` will be passed to the callback-function.

KRAP_CB_SONG This will occur when a `+++`-marker appears in the order-list.

KRAP_CB_JDONE Will occur when the jingle is done.

The callback-function must have the following form:

```
void callBack( int event, int param );
```

`event` specifies the events explained above and `param` is a numeric parameter for that event. Right now it's only meaningful for `KRAP_CB_MARK`, it will be zero for other events.

3.5 Quality modes

Krawall has various quality-modes that can be selected with `kramQualityMode()`. The default is `KRAM_QM_NORMAL`. `KRAM_QM_HQ` will play all music and sfx-channels in high quality and `KRAM_QM_MARKED` will play only those samples in high quality were Krawall is explicitly told to.

You can mark a sample for high quality by prefixing the sample filename (NOT the sample name) with a `~` in the tracker.

Chapter 4

Getting music into Krawall

4.1 The conversion tool

The program with the quite original name 'convert' is used to convert your S3M-files to compilable data. Just pass it a list of S3M's on the command line and it will create the following files:

samples.s Contains all samples and must be compiled with GAS. It exports the name `samples`.

samples.h Contains preprocessor-defines with sample-names. If the name of a sample doesn't contain any other characters than numbers, letters and '_' an entry for it will be created. The sample-names will get uppercased and prefixed with `SAMPLE_`. You will probably want to include this file and use these names for finding sound effects.

modules.h Contains the definition for all converted modules. You might wanna include this, too.

***.c** For each (non-empty) module a .c-file is generated which contains the compressed pattern-data and order-list.

The .s and .c-files must be compiled and linked against your project. The example shows how this can be done in a convenient way. You should know that the name of a module (as defined in `modules.h`) is derived from the module's filename, so the filename must be a valid C-identifier or the compiler will give you errors.

If a module doesn't contain any patterns it's samples will be added anyway, so you can put all your sound-effects in an other than that empty S3M-file.

You should put your sfx-files at the very end of the module list because the pattern-data for songs can only reference the first 512 samples. This is due to the way compression is done: the conversion-tool will print out a warning if this case occurs. If you need to use more than 512 samples in your songs the compression-scheme can be changed to cope with that case, it will become less efficient tho.

To make it clear: the overall amount of samples is NOT limited to any number (well, it is actually limited by the space available on ROM), only the samples referenced in the songs are.

The conversion-tool will also remove redundant samples and will do some optimizations on the S3M-file itself.

Chapter 5

Effects

This chapter is about the S3M-effects Krawall implements. Krawall's implementation is very accurate and implements all S3M-effects as defined by the original Scream Tracker (ST3) and also some extensions done by Modplug Tracker (MPT).

Some effects need annotation, tho:

Mxy Set Channel Volume: MPT Extension, not implemented

Nxy Channel Volume Slide: MPT Extension, not implemented

Pxy Panning Slide: MPT Extension, implemented

Rxy Tremolo: MPT does tremolo on every tick while ST3 only does it inbetween lines, implemented ST3-behavior

S2x Set finetune: not implemented because finetune-values are stored in ROM

S5x Set panbrello waveform: Although an MPT Extension, MPT obviously doesn't reset the panbrello position when a new note is played, Krawall does. To get MPT's behavior you should use `S54-S57`

S6x Pattern delay for x frames: MPT Extension, not implemented

S7x NNA Control: MPT Extension for IT-compatibility, not implemented

S8x Old Panning: It is implemented but you shouldn't use this - use `Xxx` instead

S9x Extended Channel Effects: MPT Extension, not implemented

SAx Set High Offset: MPT Extension, implemented

SFx Select Active Macro: MPT Extension, not implemented

Vxx Set Global Volume: MPT Extension, not implemented

Wxx Global Volume Slide: MPT Extension, not implemented

Xxx Set Panning: MPT Extension, implemented

Yxy Panbrello: MPT Extension, implemented, see S5x

Zxx Used by Krawall to send events via the callback. The Event is `KRAP_CB_MARK` and the parameter is `xx`.