KRAWALL GBA
Developer's Handbook

Sebastian Kienzl

November 14, 2004

Contents

Document changes

14th November,2004:

- Slight update concerning initialization

7th July, 2004:

- Updated the converter-section, sample-limit of 512 is gone
- Updated the mutex-solution in Appendix A, it wasn't correct
- Updated paragraph about ramping

23rd March, 2003:

- Updated sections on 3.5 callbacks, 2.1.3 library-names, 2.4 interrupts, 3.6 quality and 3.4 volumes
- Added Appendix B

13rd January, 2003:

- Updated S3M/XM-effect description for channel-volume

16th November, 2002:

- Updated multiple-interrupt explanation in section 2.4
- Fixed some typos

30th October, 2002:

- Added example on how to avoid problems when calling `kramWorker()` in an interrupt
- Elaboration on different versions of the library
- Added description how channel-management is done

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 sound effects 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 examples that come with Krawall illustrate compiling and linking projects with Krawall very well, also startup-code and a linkscript is included.

2.1.1 Header files

There are various header files you might want to include. `krawall.h` defines all Krawall-related calls. The conversion-tool also creates the files `modules.h` which defines all modules that have been converted and `samples.h` / `instruments.h` which `#define`'s some samples/instruments by name to 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 symbols you need to take care of is `samples` and `instruments`. They are defined in a compiled file generated by the conversion tool — see 4.1 for details.

2.1.3 Available libraries

A various of builds are included. There are builds with different IWRAM-usage, different sample-frequencies and for different retrace-rates.

The default library is the one with medium IWRAM-Usage with 16kHz sampling-frequency at 60Hz retrace-rate and all the figures within this document (for speed and memory-usage) are valid for this version.

IWRAM-usage

The speed-differences between the IWRAM-usage-versions of the library are very small, you might gain (large version) or loose (small version) maybe one or two percent of CPU-time. So it's actually just to fit your needs: If you were very low on IWRAM then you'd use the small version – If you had lots of IWRAM it'd be a pity not to use it; in that case you'd take the large version.

Sample-frequency

There's a 32kHz sampling-frequency-library available, as twice as much samples must be mixed so Krawall will take twice as much time to process music.

Normally it's not necessary to have music output at 32kHz, 16kHz does already give you pretty good quality as you can hear from the demo. But if someone really has a lot of CPU-time to spare, why not go for that version? You must also know that 32kHz requires a bigger buffer to mix to, so IWRAM-usage will be more, too.

Retrace-rate

At last there are two different version for 30Hz and 60Hz, the latter takes up more IWRAM because it needs to mix more samples at once. The reason you might want to use the 30Hz-library is elaborated in 2.5.

Which is which?

The libraries names are composed like this:

```
krawall-SF-RT-IW-[sf].lib
```

Where `SF` is the sample-frequency (16kHz/32kHz), `RT` is the retrace rate (60Hz/30Hz) and `IW` is the IWRAM-usage (medium/small/large). If there's a `-sf`-suffix then this library is compiled with `-msoft-float` – you might need this if your program is compiled with the same flag.
The default-library is:

```
krawall-16k-60-medium.lib
```

Now it's not immediately clear how much IWRAM a library actually takes. Hence, there's an `info.txt` in the `lib`-directory which contains stats about each version. The column `Size` in the `.iwram`-row displays the size of each library in hex.

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. There are three different libraries available, named small, medium and large. This is the medium (and default) usage:

ROM	32kb + module/sample-data
EWRAM	8.6kb
IWRAM	6.6kb (16kHz)

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()` 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, or you will get audible clicks in the first few seconds of playback.

If you want to switch from Stereo-output to Mono-output or vice versa you should call `kragReset()` before you call `kragInit()` again – if there is sound output in that moment an audible hickup will occur.

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 be aware that solely enabling multiple interrupts doesn't really solve the problem of `kradInterrupt()` not getting called fast enough because `kradInterrupt()` itself might get interrupted by another interrupt, although `kradInterrupt()` is a very short routine (less than 30 cycles). The only thing that might happen in that case is that an audible click occurs. So to ensure `kradInterrupt()` never ever gets interrupted the Crt0 must be altered to take care of this. The Crt0 in the examples-directory does that already, see the comment at the beginning of the file for more information.

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

Another thing to be aware of is DMA-transfers. DMA-transfers completely lock the CPU from the bus hence when a DMA-transfer is going on `kradInterrupt()` will not get called so keep your transfers short! If you have lots of memory to copy you should use the BIOS-functions `CpuSet` or `CpuFastSet`. `kradInterrupt()` might tolerate a delay up to 60usec - that's about 10000 cycles.

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 once 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. See Appendix A for a solution.

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 (`kramWorker()` returns zero then).

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 is provided for that purpose, too – see 2.1.3.

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 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 XM/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 Active voices

In the current implementation Krawall can mix 32 channels at once. For both music and sfx-playback channels are allocated dynamically and free'd when they are no longer used. So if you have for example a 28-channel tune playing and want to use 8 sfx's at once you might run into problems. In that case calls to `kramPlay()` etc. will return zero, because they couldn't allocate a channel. So be sure not to exceed 32 channels, that shouldn't be much of a problem, tho.

3.3 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 was a one-shot sample and has already ended.

There's also the possibility to use XM-instruments as sound effects, the functions `krapInstPlay()`, `krapInstRelease()` and `krapInstStop()` can be used for that. If you use these functions you must call the function `krapInstProcess()` periodically, it is responsible for updating the instrument's envelopes. It's a good idea to call it right before `kramWorker()`.

For details on these functions see the API-documentation.

3.4 Volumes

Since the GBA's DAC has only 8-bit resolution it is important to have music and SFX as loud as possible all the time because playing 8-bit samples (that already have quite of a quantization-error) at a low amplitude will further decrease the quality.

In order to have maximum possible output volume follow these steps:

- Determine how many channels are going to be used (max 8, 16 or 32) and call `kramSetMasterVol()` with the default-volume of 128 and OR the according `KRAM_MV_CHANNELSxx`-parameter
- If needed lower the volume of 128 passed to `kramSetMasterVol()` (not lower than 16) until you have distortion-free (caused by clipping) playback
- In order to adjust volume-differences between different tunes use the global-volume-option when converting (see 4.1)

Notes:

- You must not use more channels than set up with `kramSetMasterVol()`. Doing otherwise might overflow Krawall's internal mixing-buffers which introduces non-clipable distortion!
- Specifying volumes bigger than 128 when converting might do the same, so adjust differences between modules by using smaller numbers!
- Make sure the tracked music is as loud as possible – See also document *Musician's Guide*.

Krawall also 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.

You might also have the problem that SFX are very hard to hear if both music and SFX are at full volume (128). A workaround would be to lower the music-volume which you wouldn't want to do for reason mentioned before. See Appendix B for tips.

3.5 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 3.4).

KRAP_CB_DONE Once a non-looping module has reached it's end this will occur.

KRAP_CB_LOOP When a looping song restarts this will get triggered.

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 prototype:

```
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.

3.6 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* in S3M's and the sample name in XM's with a ~ in the tracker.

Krawall also supports ramping: When a sample is forced to stop there might be a click due to the sudden change in amplitude – it heavily depends on the tune and samples that are used whether a click occurs. Until version 20040707 ramping had to be enabled explicitly, now it's enabled by default.

If you need that little CPU that is taken up by ramping, you can disable ramping by calling `kramQualityMode()` with the parameter `KRAM_QM_RAMP_OFF`.

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/XM-files to compilable data. Just pass it a list of S3M/XM'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.

instruments.s Contains all instruments. Exports the name `instruments`.

instruments.h As `samples.h`, just for the instruments.

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

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

The .s must be compiled and linked against your project. The examples 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/instruments will be added anyway, so you can put all your sound-effects in an other than that empty mod-file.

Until version 20040707, Krawall had a limit of 512 samples that could be used in pattern-data. Now the limit is 64K, which isn't an actual limit any more.

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 mod-file itself.

As mentioned in 3.4 you can specify a per-module global volume. Just postfix the filename with `:vol` on the commandline where `vol` ranges from 0 to 255 (128 is the default).

Chapter 5

Effects

This chapter is about the S3M/XM-effects Krawall implements.

5.1 S3M

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:

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, implemented

Wxx Global Volume Slide: MPT Extension, 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`.

5.2 XM

Krawall implements all XM-effects as defined by the original Fast Tracker 2 and some extensions done by MPT.

Although not implemented in FT2 and Modplug Tracker (I don't know about other trackers) *Set Channel Volume* is `Ixx` and *Channel Volume Slide* is `Jxx` (`Mxx` and `Nxx` for S3M) in Krawall so if you're able to enter these effects in your tracker you might want to use them.

Appendix A

kramWorker() in interrupt

As mentioned before, calling `kramWorker()` in an interrupt (VCount-Match for instance) will run you into synchronization-problems if a call to for example `kramPlay()` will get interrupted by just that interrupt. In the worst case, this will lead to a crash. However there's a simple workaround to avoid this problem:

```
Declarations:

    volatile bool gamelogicRunning;
    volatile bool kramWorkerCallNeeded;

"Main"-program:

    gamelogicRunning = true;
    // don't worry about calls to kramPlay() etc...
    gamelogic();

    if( kramWorkerCallNeeded ) {
        kramWorker();
    }

    gamelogicRunning = false;

The VCountMatch-interrupt could look like this:

    void VCountMatchInterrupt()
    {
        if( !gamelogicRunning ) {
            kramWorkerCallNeeded = false;
            kramWorker();
        }
        else {
            kramWorkerCallNeeded = true;
        }
    }
```

This might look a little obscure at first, but if you think about it you'll find out it's a simple and correct solution. See example `interrupt` for an implementation.

Appendix B

SFX isn’t loud enough!

As said before you might have the problem that SFX can hardly be heard when both music and SFX are playing at full volume (128). Reducing the music’s volume is not a good idea because it will introduce noise and the music might be too silent on he GBA’s internal speaker.

Here’s a posting on that topic from the Krawall-mailinglist by Neil Voss from Alinear (thanks!):

> There are a number of methods of sound design for games that would help here, besides lowering the GBA music volume, which you don’t really want to do...
>
> 1) Design the music and sounds so their spectrums do not ’compete’... In other words... if the music has a lot of very full mid range, it may bury your mid range sounds. I usually do this by leaving BGM music a little ’empty’ in certain regards – in electronic music I omit cymbals as much as I can, keep some mid-high range in the spectrum somewhat free of sounds, etc. If for example I need bassy sound effects, I usually make the bass of these some 20-30 Hz away from (either lower or higher than) the bass register being used in the music.
>
> 2) Make sure everything is processed with good compression. Not filesize compression, but dynamics compression. I always pre-process the samples in a mix through a compressor as if they were being mixed into a studio track... even though its not perfect (the nature of the sound compression changes as the sound changes playback frequency) it works pretty well to help the clarity of things. Because the GBA is so limited, I’ve done

a lot of very heavy, tight compression on any sounds which are rhythmic, or have a hard attack on them (drums, etc.).

For GBA sounds, I usually run everything with a hard attack through a compressor using a threshold of -12db, ratio between 1:4 and 1:8, attack time of about 5ms, release of about 100ms, with no pre or post gain, then normalize the sounds... I use the BlueLine compressor (DX+VST) a lot, because it has a nice somewhat analog sound to it... but just about any – even free ones that come with sound forge/cooledit/etc. should do the trick.

3) To keep the SFX within the same dynamic range (volume range) but make them 'louder' you can use any number of loudness maximization plugins or techniques. One method would be to EQ the sounds, removing all the registers which are not very present in the sounds, and gaining the registers the sound actually resides in. The best way to do this is if you can do it by ear and hear what registers you want to shape, but you can also look at a basic spectrum VU meter to see where the sound 'sits'.

Another method it to use a compressor/limiter plugin to pump up the sounds a bit. I'd seriously recommed the Direct-X plugins from 'Waves'... I use the Waves 'L2' a lot for this purpose. Normalization only typically normalizes the peaks of the sound, whereas a limiter can actually set a max volume level for everything and 'pump up' (or subdue) the amplitudes across the sound sample (over time) to the desired level... For example if you have a 'pad' or 'strings' sound which has a bit of a 'pop' at the beginning, and then subdues after the pop, you can use a limiter to pump up the body of the sound which comes after the attack... etc.

4) Use panning/stereo field to set where sounds are. If you have enough CPU/channels it helps a lot just to give everything its own 'space' in a mix. For example, a voice-over will be a LOT clearer if it is not directly in the mono center while music also sits there – you can either pan it off to one side a little bit or, ideally give it a slight stereo effect but putting it both in L/R channels and offsetting them slightly (either in time or frequency, or both). This of course doesn't do a lot for the GBA internal speaker...

Really 'clarity' in sound/music design is a lot more than simply having things louder/softer than each other... you can accomplish a fair bit in mixing, but most of it will really come from working the sounds themselves so everything sits together nicely in terms of spectrum and dynamics. Again with GBA you want to play things as loud as possible in terms of volume level... it doesn't have very good low volume sound reproduction.

Hope this helps...