Advanced Gtk+ Sequencer

Developer's Book

Joël Krähemann

Advanced Gtk+ Sequencer: Developer's Book

Joël Krähemann

Copyright (C) Joël Krähemann.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Dedication

This book is dedicated to my friend.

Table of Contents

Foreword vi
1. The application context 1
    Implementations and their interfaces 1
    The main loop property 4
    The config property 4
    The file property 4
    The application mutex 5
    Program start and termination 5
    Abstract data connection 5
    Common interfaces 5
2. XML Input/Output 6
    Writing XML files 6
    Reading XML files 6
3. Multi-/Super-threaded tree 8
    The main loop interface 8
    Threads in general 8
    Pulling threads of thread pool 10
    Worker-threads to do tic-less parallelism 12
        Asynchronously destroy objects 12
    Launching tasks 12
    Async message delivery 12
4. The soundcard and sequencer interface 13
    Gathering PCM information of soundcard 13
    Obtain audio buffer 14
    Read from MIDI device 15
5. AgsAudio a container of AgsChannel 18
    AgsNotation and AgsNote 19
    AgsAutomation and AgsAcceleration 21
    AgsWave and AgsBuffer 21
    AgsRecallID and AgsRecyclingContext 24
    Dealing with recalls 25
        Get port of recall 25
    Open audio files 27
        Audio container 27
        Audio file 27
6. Your tree linked with AgsChannel 28
    The pattern 28
    Linking overview 29
    Limitations 30
    Hands-On 30
7. The recycling tree 33
    Add and remove audio signal 33
8. Your audio data in AgsAudioSignal 35
9. Effects 36
    Play/recall context 36
    Hands-On instantiating an effect 36
        AgsRecallContainer 36
        AgsRecallAudio context 37
        AgsRecallChannel context 37
        AgsRecallAudioRun context 38
        AgsRecallChannelRun context 38
    The basic lifecycle of an effect 39
    A closer look at effects 40
10. Advanced Gtk+ Sequencer's fx engine 41
11. Thread-safe audio ports 44

Get and set values 44
12. Putting all together 45
A. GNU Free Documentation License 55
B. Related projects 61

List of Tables

6.1. AGS network layer table 30

List of Examples

1.1. Thread application context 1
1.2. Audio application context 2
1.3. Get config value 4
1.4. The application context :file property 4
2.1. Writing XML 6
2.2. Reading XML 7
3.1. Calculating tic delay 9
3.2. Starting threads 9
3.3. Pulling threads of thread-pool 10
4.1. PCM information from AgsSoundcard 13
4.2. Get AgsSoundcard buffer 14
4.3. Get AgsSequencer buffer 16
5.1. Using AgsAudio 18
5.2. Using AgsNotation Clipboard 20
5.3. Concat AgsWave 22
5.4. Modify recall port 25
6.1. Adding AgsPattern 28
6.2. Prerequisites 30
6.3. Thread-Unsafe way 31
6.4. Multithread-Safe way 31
7.1. AgsRecycling and AgsAudioSignal 33
9.1. Creating AgsRecallContainer 36
9.2. Creating AgsEchoAudio 37
9.3. Creating AgsEchoChannel 37
9.4. Creating AgsEchoAudioRun 38
9.5. Creating AgsEchoChannelRun 38
10.1. Using ags_fx_factory_create() 41
12.1. Simple pattern sequencer with master playback 45

Foreword

I began to code with C in spring 2002 and hadn't much programming skills, yet. You may ask me why the C programming language? Well, my friend who was already a convient free software user and hacker recomended me it. He told me that C is a standard on Unix like operating systems so it would be a good choice.

After started with language basics and several discussions with my friend about pointers he advised me of Gtk +. While I was doing my first steps in GUI programming with C, I was sure to extensively use it and became a persuaded free software user and programmer.

A year later I really understood the object orientated matter of GObject and how to write objects and widgets myself. C wasn't like Java where you just couldn't implement no classes just everything was a class or at least a method.

First output with AGS happend via Open Sound System device drivers but the entire application lacked of a thread safe concept. But for now you may write tasks.

Chapter 1. The application context

Making Advanced Gtk+ Sequencer objects reachable from different contices was mandatory as introducing AgsApplicationContext. Imagine you are within a GUI callback and want to lookup a soundcard or sequencer the application context shall provide this functionality and provide access to its objects through a well defined interface. As doing it with interfaces you are not limited to one specific implementation rather having the option to choose the appropriate one implementing the interfaces.

- AgsConcurrencyProvider
- AgsServiceProvider
- AgsSoundProvider

There are different contices available e.g. AgsThreadApplicationContext providing its functionality by AgsConcurrencyProvider, AgsAudioApplicationContext giving you the wished objects by implementing AgsConcurrencyProvider and AgsSoundProvider. For example the code below should each giving you the same meaning object but using different contices.

Since AgsApplicationContext is a singleton you create only 1 instance of your desired implementation. The application context is usually obtained by calling `AgsApplicationContext* ags_application_context_get_instance()`. Make sure to instantiate an application context before using this function.

Implementations and their interfaces

The most basic application context implementing AgsConcurrencyProvider is AgsThreadApplicationContext. If you use your very own application context implementation make sure to set main loop, because AgsThread makes use of the appropriate getter `AgsThread* ags_concurrency_provider_get_main_loop(AgsConcurrencyProvider*)`. Further for your very own application context you should set the AgsTaskLauncher in order to take advantage of launching AgsTask implementations. The interface allows you to set a thread pool and some worker threads if needed.

This example instantiates AgsThreadApplicationContext. By instantiating the application context the global variable ags_application_context is initially set. Later you can obtain your application context by calling `AgsApplicationContext* ags_application_context_get_instance()`. Later the code connects to the AgsApplicationContext::setup() signal. The `void setup_callback(AgsApplicationContext*, gpointer)` function sets the global variable start_loader to TRUE. This causes the prior added timeout to do the actual work with AgsTaskLauncher. The task launcher is obtained by calling `AgsThread* ags_concurrency_provider_get_task_launcher(AgsConcurrencyProvider*)`.

Example 1.1. Thread application context

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

void setup_callback(AgsApplicationContext *application_context, gpointer user_data);
gboolean loader_timeout(AgsApplicationContext *application_context);

#define DEFAULT_LOADER_INTERVAL (1000 / 25)

AgsApplicationContext *application_context;
gboolean start_loader;
```

```
application_context = (AgsApplicationContext *) ags_thread_application_context_new();
g_object_ref(application_context);

g_signal_connect_after(application_context, "setup",
         G_CALLBACK(setup_callback), NULL);

start_loader = FALSE;

g_timeout_add(DEFAULT_LOADER_INTERVAL,
       loader_timeout,
       application_context);

ags_application_context_prepare(application_context);
ags_application_context_setup(application_context);

/* main loop run */
g_main_loop_run(g_main_loop_new(g_main_context_default(),
    TRUE));

void
setup_callback(AgsApplicationContext *application_context, gpointer user_data)
{
  start_loader = TRUE;
}

gboolean
loader_timeout(AgsApplicationContext *application_context)
{
  AgsTaskLauncher *task_launcher;

  if(!start_loader){
    return(TRUE);
  }

  task_launcher = ags_concurrency_provider_get_task_launcher(AGS_CONCURRENCY_PROVIDER(ap

  //TODO: add some tasks to task_launcher

  return(FALSE);
}
```

The AgsAudioApplicationContext inherites from AgsApplicationContext and implements the AgsConcurrencyProvider interface, too. So you can retrieve the task launcher the same way. But the context implements one more, the AgsSoundProvider interface. Giving you objects related to threading and audio processing.

This example does the same as the prior, but this time instantiates the AgsAudioApplicationContext. The `gboolean loader_timeout(AgsApplicationContext *application_context)`. does this time add the AgsStartSoundcard task to the task launcher.

Example 1.2. Audio application context

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
```

```
#include <ags/libags-audio.h>

void setup_callback(AgsApplicationContext *application_context, gpointer user_data);
gboolean loader_timeout(AgsApplicationContext *application_context);

#define DEFAULT_LOADER_INTERVAL (1000 / 25)

AgsApplicationContext *application_context;
gboolean start_loader;

application_context = (AgsApplicationContext *) ags_audio_application_context_new();
g_signal_connect_after(application_context, "setup",
         G_CALLBACK(setup_callback), NULL);

start_loader = FALSE;

g_timeout_add(DEFAULT_LOADER_INTERVAL,
       loader_timeout,
       application_context);

ags_application_context_prepare(application_context);
ags_application_context_setup(application_context);

/* main loop run */
g_main_loop_run(g_main_loop_new(g_main_context_default(),
    TRUE));


void
setup_callback(AgsApplicationContext *application_context, gpointer user_data)
{
  start_loader = TRUE;
}

gboolean
loader_timeout(AgsApplicationContext *application_context)
{
  AgsTaskLauncher *task_launcher;
  AgsStartSoundcard *start_soundcard;

  if(!start_loader){
    return(TRUE);
  }

  task_launcher = ags_concurrency_provider_get_task_launcher(AGS_CONCURRENCY_PROVIDER(ap

  start_soundcard = ags_start_soundcard_new();

  ags_task_launcher_add_task(task_launcher,
        start_soundcard);

  return(FALSE);
}
```

The main loop property

AgsApplicationContext:main-loop does usually point to an AgsThread implementing AgsMainLoop interface. libags_thread.so provides you the AgsGenericMainLoop object or if you intend to use libags_audio.so, this property shall point to AgsAudioLoop.

However you should rather use `AgsThread* ags_concurrency_provider_get_main_loop(AgsConcurrencyProvider*)` to obtain the main loop instead.

The config property

The AgsApplicationContext base class provides you an AgsConfig instance. It might load a default configuration or from current users home directory.

Since AgsConfig is a singleton you should obtain it by calling `AgsConfig* ags_config_get_instance()`.

AgsConfig stores its properties as key value pairs within appropriate group. In order to get the config instance, load default configuration and get the threading model do the following.

Example 1.3. Get config value

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

AgsConfig *config;
gchar *str;

config = ags_config_get_instance();
ags_config_load_defaults(config);

str = ags_config_get_value(config,
                           "thread",
                           "model");
```

The file property

You might want to set an AgsFile or AgsSimpleFile instance within your application context. This in view of having your application persisted.

Example 1.4. The application context :file property

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

AgsApplicationContext *application_context;
AgsFile *file;
```

```
application_context = ags_application_context_get_instance();

file = ags_file_new();
g_object_set(application_context,
             "file", file,
             NULL);
```

The application mutex

As version 2.0.x the application mutex was superseeded by the class mutices and a common field :obj-mutex used by various types. The AgsMutexManager is still around but with less importance.

Program start and termination

The application context provides signals to make your application ready to run. You basically implement AgsApplicationContext::prepare, AgsApplicationContext::setup and AgsApplicationContext::register-types. It is upto you how the application shall behave.

Note since version 3.13.0 you call only AgsApplicationContext::prepare, which calls AgsApplicationContext::setup and then enters GLib's main loop. So the prepare signal won't return unless you terminate your application.

AgsApplicationContext::quit signal terminates your application. Feel free to provide your own implementation.

Abstract data connection

AgsDataConnectionManager and AgsConnection are removed in 2.0.x. The object was somehow overhelming because you can have properties.

The AgsConnectable interface provides 2 new functions: `void ags_connectable_connect_connection(AgsConnectable*, GObject*)` and `void ags_connectable_disconnect_connection(AgsConnectable*, GObject*)`.

Dependencies not know an instantiation time can be later connected.

Common interfaces

Use AgsConnectable if you intend to listen to a particular event. If you want to connect an event of an object known during instantiation time use ::connect and ::disconnect. Assumed the object needs to be resolved, you can ::connect-connection ::disconnect-connection, later.

AgsPlugin interface provides persistence to a well known abstract base type. Since it has various implementations, this interface provides `void ags_plugin_read(AgsFile*, xmlNode*, AgsPlugin*)` and `xmlNode* ags_plugin_write(AgsFile*, xmlNode*, AgsPlugin*)`

Likewise there are the interfaces intended to use with sound related objects AgsSoundcard, AgsSequencer, AgsMutable and AgsSeekable.

Chapter 2. XML Input/Output

Saving and restoring your files is done by using XML supporting XPath. The complete persistence layer is described by ags_file.dtd installed on your system. There various classes involved by doing XML IO. It does it in stages as following for reading:

i. Parsing the XML tree and map nodes and objects.

ii. Resolving XPath expressions retrieve objects by their nodes.

iii. Do as needed callbacks of AgsFileLaunch to setup up the application.

Writing files does ommit the last step. The current AgsConfig is going to be embedded in your file. So you can have per project configuration. Certain objects implement AgsPlugin interface to do an abstraction of reading and writing xmlNode.

Writing XML files

Writing files is pretty easy. You just have to instantiate AgsFile, set the application context, open it in read-write mode, call ags_file_write() and finally ags_file_close().

Example 2.1. Writing XML

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

AgsApplicationContext *application_context;
AgsFile *file;

GError *error;

static const gchar *filename = "my_file.xml";

application_context = ags_application_context_get_instance();

file = (AgsFile *) g_object_new(AGS_TYPE_FILE,
                                "application-context", application_context,
                                "filename", filename,
                                NULL);

error = NULL;
ags_file_rw_open(file,
                 TRUE,
                 &error);
ags_file_write(file);
ags_file_close(file);
```

Reading XML files

Normally you instantiate a new application context to be used to load objects into. Create a file object by passing the application context and filename. Then open it and read the content. At the end you close the file descriptor. To use your application start the main loop.

Example 2.2. Reading XML

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

AgsApplicationContext *application_context;
AgsFile *file;

GError *error;

static const gchar *filename = "my_file.xml";

application_context = ags_audio_application_context_new();

file = g_object_new(AGS_TYPE_FILE,
                    "application-context", application_context,
                    "filename", filename,
                    NULL);
error = NULL;
ags_file_open(file,
              &error);

ags_file_read(file);
ags_file_close(file);

ags_thread_start(application_context->main_loop);
```

Chapter 3. Multi-/Super-threaded tree

Advanced Gtk+ Sequencer comes with an AgsThread object. It is organized as a tree structure. The API provides many functions to work with it. These threads do the ::clock event where all threads synchronize.

The AgsTaskLauncher runs synchronized as well but is going to be waited after syncing to run all tasks. The AgsTask signal ::launch runs asynchronous exclusively. Every thread tree shall have at toplevel a thread implementing AgsMainLoop interface.

There is an object call AgsThreadPool serving prelaunched threads. It returns on pull AgsReturnableThread instances. They can be used with a callback ::safe-run.

There is a interface to implement by your application context. Thus the AgsConcurrencyProvider interface is used. It has some common get/set functions to do basic multi-threaded work by well defined objects.

The main loop interface

AgsMainLoop should be implemented by toplevel threads. Within a thread tree this is the topmost element. It has various get and set methods you would expect.

To control the AgsThread::clock signal AgsMainLoop's methods are going to be invoked. The involved functions are:

As it shall be implemented by AGS_TYPE_THREAD subtypes, this parent object provides a mutex to properly lock the object. You should obtain the GRecMutex pointer by accessing its field:

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

AgsThread *thread;

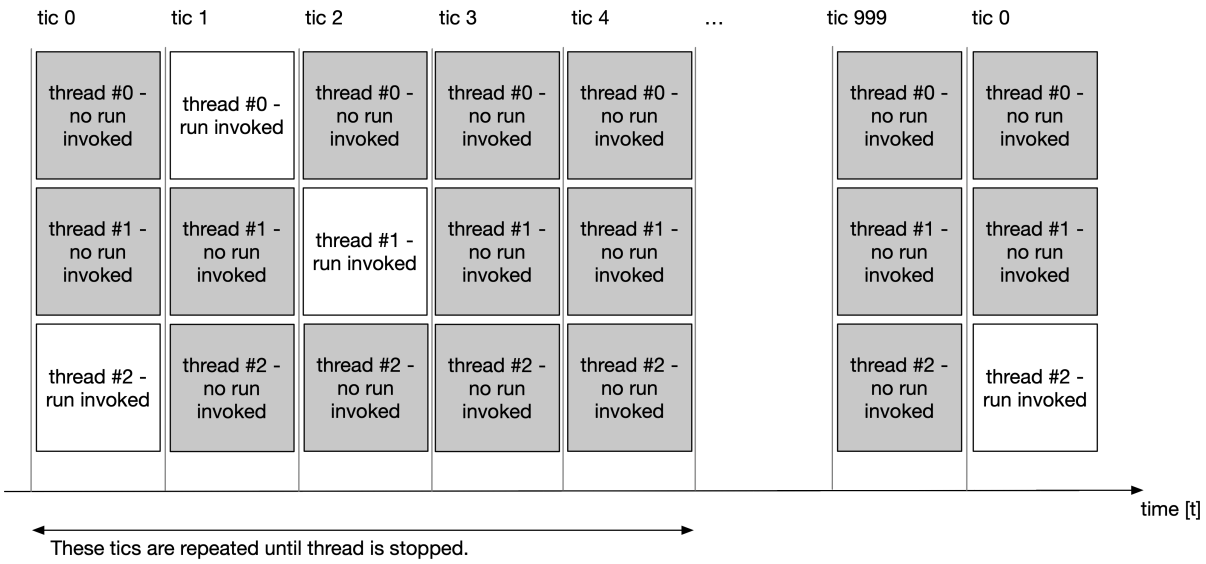
GRecMutex *thread_mutex;

thread = ags_thread_new(NULL);

/* get object mutex */
thread_mutex = AGS_THREAD_GET_OBJ_MUTEX(thread);
```

Threads in general

Libags provides a thread wrapper built on top of GLib's threading API. The AgsThread object synchronizes the thread tree by AgsThread::clock() event. It is somekind of parallelism trap.



All threads within tree synchronize to AgsThread:max-precision per second, because all threads shall have the very same time running in parallel. I talk of tic-based parallelism, with a max-precision of 1000 Hz, each thread synchronizes 1000 times within tree. Giving you strong semantics to compute a deterministic result in a multi-threaded fashion.

Since we want to run tasks exclusively without any interference from competing threads. There is a mutex lock involved just after synchronization and then invokes ags_task_launcher_sync_run(). Be aware the conditional lock can be evaluate to true for many threads.

After how many tics the flow is repeated depends on samplerate and buffer size. If you have an AgsThread with max-precision 1000, samplerate of 44100 common for audio CDs and a buffer size of 512 frames, then the delay until its repeated calculates as following:

Example 3.1. Calculating tic delay

```
tic_delay = 1000.0 / 44100.0 * 512.0; // 11.609977324263039
```

As you might have pre-/post-synchronization needing 3 tics to do its work you get 8 unused tics.

Pre-synchronization is used for reading from soundcard or MIDI device. The intermediate tic does the actual audio processing. Post-synchronization is used by outputing to soundcard or exporting to audio file.

Within thread tree context you have to take care not to hang it up with a dead-lock. Usually you have to use the :start_queue to start threads. Alternatively you may want to use `void ags_thread_start(AgsThread*)`. Use :start_cond, which is protect it with :start_mutex, to notify about running thread.

The following example creates a thread and does add an other thread to :start_queue. This causes it to be started as well. Note you want to access :start_queue using :start_mutex to avoid data races. But there is a convience function which does it for you.

Example 3.2. Starting threads

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

AgsThread *main_loop;
```

```
AgsThread *thread;

AgsApplicationContext *application_context;

application_context = ags_application_context_get_instance();

main_loop = ags_generic_main_loop_new();
ags_concurrency_provider_set_main_loop(AGS_CONCURRENCY_PROVIDER(application_context),
           main_loop);

ags_thread_start(main_loop);

thread = ags_thread_new();
ags_thread_add_child_extended(main_loop,
                              thread,
                              TRUE, TRUE);
ags_thread_add_start_queue(main_loop,
      thread);
```

There many other functions not covered like mutex wrappers ags_thread_lock() and ags_thread_unlock(). As doing a closer look to the API there are functions to lock different parts of the tree. But all these functions should be carefully used, since you might run into a dead-lock.

To find a specific thread type use ags_thread_find(). You can use ags_thread_self() to retrieve your own running thread in case your using Advanced Gtk+ Sequencer thread wrapper.

Pulling threads of thread pool

AgsThreadPool serves you instantiated and running threads. To pull an AgsReturnableThread issue ags_thread_pool_pull(). The following example does instantiate a thread pool and starts it. After, it pulls two threads and the callbacks are invoked.

Example 3.3. Pulling threads of thread-pool

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>

void setup_callback(AgsApplicationContext *application_context, gpointer data);
void thread_run_callback(AgsThread *thread, gpointer data);

gchar *thread_0_str = "thread 0";
gchar *thread_1_str = "thread 1";

void
setup_callback(AgsApplicationContext *application_context, gpointer data)
{
  AgsThread *main_loop;
  AgsThread *thread_0, *thread_1;
  AgsThreadPool *thread_pool;

  main_loop = ags_concurrency_provider_get_main_loop(AGS_CONCURRENCY_PROVIDER(applicati

  thread_pool = ags_thread_pool_new(main_loop);
```

```
  ags_concurrency_provider_set_thread_pool(AGS_CONCURRENCY_PROVIDER(application_context
        thread_pool);

  ags_thread_pool_start(thread_pool);

  /* pull thread 0 */
  thread_0 = ags_thread_pool_pull(thread_pool);

  g_rec_mutex_lock(AGS_RETURNABLE_THREAD_GET_RESET_MUTEX(thread_0));

  g_atomic_pointer_set(&(AGS_RETURNABLE_THREAD(thread_0)->safe_data),
                       thread_0_str);

  ags_returnable_thread_connect_safe_run(AGS_RETURNABLE_THREAD(thread_0),
                                         thread_run_callback);

  ags_returnable_thread_set_flags(thread_0,
      AGS_RETURNABLE_THREAD_IN_USE);

  g_rec_mutex_unlock(AGS_RETURNABLE_THREAD_GET_RESET_MUTEX(thread_0));

  /* pull thread 1 */
  thread_1 = ags_thread_pool_pull(thread_pool);

  g_rec_mutex_lock(AGS_RETURNABLE_THREAD_GET_RESET_MUTEX(thread_1));

  g_atomic_pointer_set(&(AGS_RETURNABLE_THREAD(thread_1)->safe_data),
                       thread_1_str);

  ags_returnable_thread_connect_safe_run(AGS_RETURNABLE_THREAD(thread_1),
                                         thread_run_callback);

  ags_returnable_thread_set_flags(thread_1,
      AGS_RETURNABLE_THREAD_IN_USE);

  g_rec_mutex_unlock(AGS_RETURNABLE_THREAD_GET_RESET_MUTEX(thread_1));
}

void
thread_run_callback(AgsThread *thread, gpointer data)
{
  g_message("%s", (gchar *) data);
}

int
main(int argc, char **argv)
{
  AgsApplicationContext *application_context;

  application_context = ags_thread_application_context_new();
  g_object_ref(application_context);

  g_signal_connect_after(application_context, "setup",
    G_CALLBACK(setup_callback), NULL);

  ags_application_context_prepare(application_context);
  ags_application_context_setup(application_context);
```

```
  /* main loop run */
  g_main_loop_run(g_main_loop_new(g_main_context_default(),
      TRUE));

  return(0);
}
```

Worker-threads to do tic-less parallelism

Worker threads are used to perform heavy load tasks that run completely asynchronous. This means they don't do any sync with the tree. You start worker threads like any other thread by calling `void ags_thread_start(AgsThread*)` or `void ags_thread_stop(AgsThread*)` to stop it.

The AgsWorkerThread overrides ::start of AgsThread class and won't do any synchronization. The worker implementation is responsible to delay computation by calling usleep() or nanosleep().

You can either connect to the ::do-poll signal or inherit of the AgsWorkerThread object. This requires to override ::do-poll.

Asynchronously destroy objects

AgsDestroyWorker is intended to unref or free objects asynchrously. Note the use of this worker for one certain instance, requires it to do it throughout with the worker for all unref calls. Else you would probably end in a data-race ending in accessing a freed instance. This can especially happen as using g_object_run_dispose().

The destroy function takes exactly one parameter like g_free() or g_object_unref(). To add an entry call ags_destroy_worker_add(). The first parameter is the worker, second the pointer to free/unref and third the destroy function.

Launching tasks

It's for thread-safety for sure to run tasks asynchronously exclusive. This means what ever you do it's safe exceptional in view of third-party libraries that might have their own threads. To do your own task you should inherit AgsTask base object and implement ::launch. This signal is invoked after syncing the thread tree.

You can use either ags_task_launcher_add_task() or ags_task_launcher_add_task_all() to add one respectively a GList of tasks. The task shall report failures by calling ::failure signal.

Async message delivery

AgsMessageDelivery is a singleton. In order to get the instance of it call `AgsMessageDelivery* ags_message_delivery_get_instance()`. The library routines only provide messages until you have added an AgsMessageQueue with the appropriate namespace.

- libags - namespace used by libags.so.3, libags_thread.so.3 and libags_server.so.3
- libags-audio - namespace used by libags_audio.so.3

As you usually have one object or widget mapped to a specific object, you can poll the queue by `guint g_timeout_add(guint, GSourceFunc, gpointer)`. Then forward the event as you like. GSequencer does look for matching messages by sender using following `GList* ags_message_queue_find_sender(AgsMessageQueue*, GObject*)`. This not at least because the recipient is most of the time not defined.

Chapter 4. The soundcard and sequencer interface

With AgsSoundcard and AgsSequencer interface you can obtain information about output or input devices. Getting the next buffer for playback something can be achieved, too. As well reading MIDI data from current buffer is supported. Note these operations are performed all delayed in order to avoid concurrent memory access.

Latency is at most one buffer time. Operations on buffers might be performed non-blocking so the thread returns earlier than expected. This has the advantage of controlling timings and let the thread continue to do more synchronization runs. Real-time behaviour is indicated as all pending sync operations were fulfilled as the next buffer is needed.

The Advanced Gtk+ Sequencer framework implements following soundcard objects. Note to register soundcards by a sound server make use of AgsSoundServer interface. This applies to JACK, Pulseaudio and CoreAudio backend.

- AgsDevout ALSA and OSSv4 soundcard output.
- AgsDevin ALSA and OSSv4 soundcard input.
- AgsPulseDevout Pulseaudio output.
- AgsJackDevout JACK Audio Connection Kit output.
- AgsJackDevin JACK Audio Connection Kit input.
- AgsWasapiDevout Windows soundcard output.
- AgsWasapiDevin Windows soundcard input.
- AgsCoreAudioDevout macos soundcard output.
- AgsCoreAudioDevin macos soundcard input.

The Advanced Gtk+ Sequencer framework implements following sequencer objects.

- AgsMidiin ALSA and OSSv4 MIDI input.
- AgsJackMidiin JACK Audio Connection Kit MIDI input.
- AgsCoreAudioMidiin macos MIDI input.

Gathering PCM information of soundcard

In this short example we just get some information out of AgsSoundcard by using `void ags_soundcard_pcm_info(AgsSoundcard*, gchar*, guint*, guint*, guint*, guint*, guint*, guint*, GError*)`. It tells us the card identifier, minimum and maximum supported audio channels, samplerate and buffer size.

Example 4.1. PCM information from AgsSoundcard

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsApplicationContext *application_context;
```

```
GObject *soundcard;

GList *start_list;

guint channels_min, channels_max;
guint rate_min, rate_max;
guint buffer_size_min, buffer_size_max;

GError *error;

application_context = ags_application_context_get_instance();

start_list = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_context));

if(start_list != NULL){
  soundcard = G_OBJECT(start_list->data);

  error = NULL;
  ags_soundcard_pcm_info(AGS_SOUNDCARD(soundcard),
                         &channels_min, &channels_max,
                         &rate_min, &rate_max,
                         &buffer_size_min, &buffer_size_max,
                         &error);

  if(error != NULL){
    g_warning("%s", error->msg);

    g_error_free(error);
  }
}

g_list_free_full(start_list,
   (GDestroyNotify) g_object_unref);
```

Obtain audio buffer

Here we get audio buffer from AgsSoundcard and write some sine synth tone at 440 Hz. First we get presets from soundcard, then we fill the AgsSynthUtil struct and finally compute the sine sound using utility function.

Example 4.2. Get AgsSoundcard buffer

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsApplicationContext *application_context;

GObject *soundcard;

AgsSynthUtil synth_util;

GList *start_list;
```

```
void *buffer;

guint audio_channels;
guint samplerate;
guint buffer_length;
guint format;

application_context = ags_application_context_get_instance();

start_list = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_context));

if(start_list != NULL){
  soundcard = G_OBJECT(start_list->data);

  if(ags_soundcard_is_playing(AGS_SOUNDCARD(soundcard))){
    buffer = ags_soundcard_get_buffer(AGS_SOUNDCARD(soundcard));

    ags_soundcard_get_presets(AGS_SOUNDCARD(soundcard),
         &audio_channels,
         &samplerate,
         &buffer_length,
         &format);

    synth_util.source = buffer;
    synth_util.source_stride = audio_channels;

    synth_util.buffer_length = buffer_length;
    synth_util.audio_buffer_util_format = ags_audio_buffer_util_format_from_soundcard(f
    synth_util.samplerate = samplerate;

    synth_util.synth_oscillator_mode = AGS_SYNTH_OSCILLATOR_SIN;

    synth_util.frequency = 440.0;
    synth_util.phase = 0.0;
    synth_util.volume = 1.0;

    synth_util.frame_count = buffer_length;
    synth_util.offset = 0;

    ags_soundcard_lock_buffer(AGS_SOUNDCARD(soundcard),
         buffer);

    ags_synth_util_compute_sin(&synth_util);

    ags_soundcard_unlock_buffer(AGS_SOUNDCARD(soundcard),
    buffer);
  }
}

g_list_free_full(start_list,
   (GDestroyNotify) g_object_unref);
```

Read from MIDI device

Example 4.3. Get AgsSequencer buffer

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsApplicationContext *application_context;

GObject *sequencer;

GList *start_list;

void *midi_buffer;

guint buffer_length;

application_context = ags_application_context_get_instance();

start_list = ags_sound_provider_get_sequencer(AGS_SOUND_PROVIDER(application_context));

if(start_list != NULL){
  sequencer = G_OBJECT(start_list->data);

  if(ags_sequencer_is_recording(AGS_SEQUENCER(sequencer))){
    buffer_length = 0;
    midi_buffer = ags_sequencer_get_buffer(AGS_SEQUENCER(sequencer),
        &buffer_length);

    if(midi_buffer != NULL &&
       buffer_length > 0){
      guchar *midi_iter;

      /* parse bytes */
      midi_iter = midi_buffer;

      while(midi_iter < midi_buffer + buffer_length){
 ags_sequencer_lock_buffer(AGS_SEQUENCER(sequencer),
      midi_buffer);

 if(ags_midi_util_is_key_on(midi_iter)){
   g_message("key %d on with velocity %d", (0x7f & (midi_buffer[1])), (0x7f & (midi_buf

   midi_iter += 3;
 }else if(ags_midi_util_is_key_off(midi_iter)){
   midi_iter += 3;
 }else if(ags_midi_util_is_key_pressure(midi_iter)){
   midi_iter += 3;
 }else if(ags_midi_util_is_change_parameter(midi_iter)){
   midi_iter += 3;
 }else if(ags_midi_util_is_pitch_bend(midi_iter)){
   midi_iter += 3;
 }else if(ags_midi_util_is_change_program(midi_iter)){
   midi_iter += 2;
 }else if(ags_midi_util_is_change_pressure(midi_iter)){
   midi_iter += 2;
```

```
 }else if(ags_midi_util_is_sysex(midi_iter)){
   guint n;

   /* sysex */
   n = 0;

   while(midi_iter[n] != 0xf7){
     n++;
   }

   midi_iter += (n + 1);
 }else if(ags_midi_util_is_song_position(midi_iter)){
   midi_iter += 3;
 }else if(ags_midi_util_is_song_select(midi_iter)){
   midi_iter += 2;
 }else if(ags_midi_util_is_tune_request(midi_iter)){
   midi_iter += 1;
 }else if(ags_midi_util_is_meta_event(midi_iter)){
   midi_iter += (3 + midi_iter[2]);
 }else{
   g_warning("unexpected byte %x", midi_iter[0]);

   midi_iter++;
 }

 ags_sequencer_unlock_buffer(AGS_SEQUENCER(sequencer),
        midi_buffer);
      }
    }
  }
}

g_list_free_full(start_list,
   (GDestroyNotify) g_object_unref);
```

Chapter 5. AgsAudio a container of AgsChannel

AgsAudio contains a pointer to your notation and automation data. It has its own recall context, AgsRecallAudio. It organizes your recycling contices and thus having an associated AgsRecallID for running contices. Further AgsAudio is your topmost nesting level of AgsAudioSignal. You might traverse the layers in following order:

i. AgsAudio

ii. AgsChannel

iii. AgsRecycling

iv. AgsAudioSignal

In order the audio processing threads are capable to iterate the audio tree, you need to set either (AGS_AUDIO_SYNC) or (AGS_AUDIO_SYNC | AGS_AUDIO_ASYNC) flags. Further if your AgsAudio is a source of AgsAudioSignal you need to set both flags (AGS_AUDIO_OUTPUT_HAS_RECYCLING | AGS_AUDIO_INPUT_HAS_RECYCLING).

If you set AGS_AUDIO_SYNC flag, this causes the output and input channels to be aligned straight. Eg. input line 0 goes to output line 0, input line 1 goes to output line 1 ...

If you set both flags AGS_AUDIO_SYNC and AGS_AUDIO_ASYNC, output and input is not aligned straight. Eg. you have 2 audio channels, 1 output pad and 8 input pads, then input line 0 goes to output line 0, input line 1 goes to output line 1, input line 3 goes to output line 0 ...

It is only possible to have mulitple output pads if you have AgsRecycling assigned to AgsOutput of AgsAudio. This is usually done by sources like instruments.

AgsAudioSignal keeps your audio data as a GList of buffers. AgsRecycling is your nested tree to AgsChannel, giving you the opportunity to emit ::add_audio_signal or ::remove_audio_signal by producer and to have many consumers. AgsChannel is your opposite to an audio channel representing a single line. AgsAudio keeps track of all of them. You might want to add your audio object to an AgsSoundcard.

You may resize the count of pads or audio channels with `void ags_audio_set_pads(AgsAudio*, GType, guint, guint)` and `void ags_audio_set_audio_channels(AgsAudio*, guint, guint)`. Like in the following example the channels are adjusted and notation is added.

Example 5.1. Using AgsAudio

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsAudio *audio;
AgsNotation *notation;

AgsApplicationContext *application_context;

GObject *current_soundcard;

GList *start_soundcard;

guint audio_channels;
guint output_pads, input_pads;
```

```
guint i;

/* get application context and soundcard */
application_context = ags_application_context_get_instance();

start_soundcard = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_contex

current_soundcard = start_soundcard->data;

/* creat audio and resize channels */
audio_channels = 2;

output_pads = 1;
input_pads = 88;

audio = ags_audio_new(current_soundcard);
ags_audio_set_flags(audio,
      (AGS_AUDIO_SYNC |
       AGS_AUDIO_ASYNC |
       AGS_AUDIO_OUTPUT_HAS_RECYCLING |
       AGS_AUDIO_INPUT_HAS_RECYCLING));

ags_audio_set_audio_channels(audio,
                             audio_channels);

ags_audio_set_pads(audio,
                   AGS_TYPE_OUTPUT,
                   output_pads);
ags_audio_set_pads(audio,
                   AGS_TYPE_INPUT,
                   input_pads);

/* add notation */
for(i = 0; i < audio_channels; i++){
  notation = ags_notation_new(audio,
                              i);
  ags_audio_add_notation(audio,
                         notation);
}

g_list_free_full(start_soundcard,
   (GDestroyNotify) g_object_unref);
```

AgsNotation and AgsNote

AgsAudio provides many AgsNotation objects for one single audio channel. They all have a different :timestamp property. Usually a new AgsNotation object is introduced as AGS_NOTATION_DEFAULT_OFFSET is exceeded. So AgsNotation can hold at most 1024 x-positions of AgsNote.

You might want to query a GList of AgsNotation by the matching AgsTimestamp using AGS_TIMESTAMP_OFFSET.

- `void ags_notation_find_near_timestamp(GList*, guint, AgsTimestamp*)`

The notation object stores your notes as a GList. You can add or remove a note by calling appropriate function:

- `void ags_notation_add_note(AgsNotation*, AgsNote*, gboolean)`

- `gboolean ags_notation_remove_note_at_position(AgsNotation, guint, guint)`

The notation object supports selection of notes. There are functions available to select a single point or a region of the notation. You may find specific notes by calling:

- `AgsNote* ags_notation_find_point(AgsNotation*, guint, guint, gboolean)`
- `GList* ags_notation_find_region(AgsNotation*, guint, guint, guint, guint, gboolean)`

To copy & paste notes you might want to select a region first. Then copy the selection and insert it using new x_offset later.

Example 5.2. Using AgsNotation Clipboard

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsAudio *audio;
AgsNotation *notation;
AgsNote *note;

AgsApplicationContext *application_context;

GObject *current_soundcard;

xmlNode *clipboard;

GList *start_soundcard;

guint audio_channels;
guint output_pads, input_pads;
guint i;

/* get application context and soundcard */
application_context = ags_application_context_get_instance();

start_soundcard = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_conte

current_soundcard = start_soundcard->data;

audio_channels = 1;

output_pads = 2;
input_pads = 88;

audio = ags_audio_new(current_soundcard);
ags_audio_set_flags(audio,
      (AGS_AUDIO_SYNC |
       AGS_AUDIO_ASYNC |
       AGS_AUDIO_OUTPUT_HAS_RECYCLING |
       AGS_AUDIO_INPUT_HAS_RECYCLING));

ags_audio_set_audio_channels(audio,
                             audio_channels);
```

```
ags_audio_set_pads(audio,
                   AGS_TYPE_OUTPUT,
                   output_pads);
ags_audio_set_pads(audio,
                   AGS_TYPE_INPUT,
                   input_pads);

notation = ags_notation_new(audio,
       0);
ags_audio_add_notation(audio,
         notation);

for(i = 0; i < 16; i++){
  note = ags_note_new_with_offset(i * 4, (i * 4) + 1,
      0);
  ags_notation_add_note(notation,
   note,
   FALSE);
}

/* select, copy & paste */
ags_notation_add_region_to_selection(notation,
         0, 0,
         64, 1,
         TRUE);

clipboard = ags_notation_copy_selection(notation);
ags_notation_insert_from_clipboard(notation,
       clipboard,
       TRUE, 64,
       FALSE, 0);
```

AgsAutomation and AgsAcceleration

The automation objects stores your accelerations as a GList. There are analogous to notation functions to add or remove accelerations.

- `void ags_automation_add_acceleration(AgsAutomation*, AgsAcceleration*, gboolean)`
- `gboolean ags_automation_remove_acceleration_at_position(AgsAutomation*, guint, gdouble)`

The automation object provides functions to lookup a specific point or region, too.

- `AgsAcceleration* ags_automation_find_point(AgsAutomation*, guint, gdouble, gboolean)`
- `GList* ags_automation_find_region(AgsAutomation*, guint, gdouble, guint, gdouble, gboolean)`

AgsWave and AgsBuffer

The wave objects stores your buffers as a GList. There are analogous to notation functions to add or remove buffers.

- `void ags_wave_add_buffer(AgsWave*, AgsBuffer*, gboolean)`

- `gboolean ags_wave_remove_buffer(AgsWave*, AgsBuffer*, gboolean)`

AgsAudio holds a sorted list of AgsWave objects, `gint ags_wave_sort_func(gconstpointer, gconstpointer)` does the actual sorting. You can use it with `GList* g_list_insert_sorted(GList*, gpointer, GCompareFunc)`.

AgsWave holds a sorted list of AgsBuffer objects, `gint ags_buffer_sort_func (gconstpointer, gconstpointer)` does the actual sorting. You can use it with `GList* g_list_insert_sorted(GList*, gpointer, GCompareFunc)`. AgsWave:timestamp uses sample position with matching samplerate. As using `void ags_timestamp_set_ags_offset (AgsTimestamp*, guint64)` ags_offset equals 0 is your very first sample. You have to introduce after `AGS_WAVE_DEFAULT_BUFFER_LENGTH * samplerate` samples a new AgsWave object. The actual playback recall does bisect AgsWave and AgsBuffer in order to get current playing audio data.

AgsBuffer:data contains your actual audio data of AgsBuffer:format type. AgsBuffer:x is the actual sample position with matching samplerate.

Note audio effects are not applied to AgsWave but to AgsAudioSignal. The program flow is as following:

1. ags-fx-playback does feed AgsWave to AgsAudioSignal of AgsInput.
2. ags-fx-buffer does buffer AgsAudioSignal from AgsInput to AgsOutput.
3. Another AgsAudio containing ags-fx-playback, then it plays it on your soundcard. Assumed you prior linked the the audio tree.

In this example, we first read audio data from 2 different files and concat the returned AgsWave objects. Note if you want to read multi-channel data, you have to modify the example with a for loop or such, to copy overlapping AgsBuffer. AgsBuffer:x shall be unique for specific audio channel.

Example 5.3. Concat AgsWave

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

#define FILENAME_A "test_000.wav"
#define FILENAME_B "test_001.wav"

AgsAudio *audio;
AgsAudioFile *audio_file;

AgsTimestamp *timestamp_a, *timestamp_b;

AgsApplicationContext *application_context;

GObject *current_soundcard;

xmlNode *clipboard;

GList *start_soundcard;
GList *start_wave_a, *end_wave_a;
GList *start_wave_b;

guint64 file_a_frame_count;
guint audio_channels;
guint output_pads, input_pads;
guint i;
```

```
/* get application context and soundcard */
application_context = ags_application_context_get_instance();

start_soundcard = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_contex

current_soundcard = start_soundcard->data;

audio_channels = 1;

output_pads = 1;
input_pads = 1;

audio = ags_audio_new(current_soundcard);
ags_audio_set_flags(audio,
      (AGS_AUDIO_SYNC |
       AGS_AUDIO_OUTPUT_HAS_RECYCLING |
       AGS_AUDIO_INPUT_HAS_RECYCLING));

ags_audio_set_audio_channels(audio,
                             audio_channels);

ags_audio_set_pads(audio,
                   AGS_TYPE_OUTPUT,
                   output_pads);
ags_audio_set_pads(audio,
                   AGS_TYPE_INPUT,
                   input_pads);

/* open first audio file */
audio_file = ags_audio_file_new(FILENAME_A,
    current_soundcard,
    -1);
ags_audio_file_open(audio_file);

ags_sound_resource_info(AGS_SOUND_RESOURCE(audio_file->sound_resource),
   &file_a_frame_count,
   NULL, NULL);

start_wave_a = ags_sound_resource_read_wave(AGS_SOUND_RESOURCE(audio_file->sound_resourc
         current_soundcard,
         0, // change to -1 for all audio channels
         0,
         0.0, 0);

/* open second audio file */
audio_file = ags_audio_file_new(FILENAME_B,
    current_soundcard,
    -1);
ags_audio_file_open(audio_file);

start_wave_b = ags_sound_resource_read_wave(AGS_SOUND_RESOURCE(audio_file->sound_resourc
         current_soundcard,
         0, // change to -1 for all audio channels
         file_a_frame_count,
         0.0, 0);

/* concat AgsWave */
```

```
audio->wave = start_wave_a;

end_wave_a = g_list_last(start_wave_a);

timestamp_a = ags_wave_get_timestamp(end_wave_a->data);

timestamp_b = ags_wave_get_timestamp(start_wave_b->data);

if(ags_timestamp_get_ags_offset(timestamp_a) == ags_timestamp_get_ags_offset(timestamp_
  GList *start_buffer_a, *end_buffer_a;
  GList *start_buffer_b, *buffer_b;

  start_buffer_a = ags_wave_get_buffer(start_wave_a->data);

  end_buffer_a = g_list_last(start_buffer_a->data);

  buffer_b =
    start_buffer_b = ags_wave_get_buffer(start_wave_b->data);

  if(ags_buffer_get_x(buffer_b->data) == ags_buffer_get_x(end_buffer_a->data)){
    AgsBuffer *current_mix_buffer_b;

    current_mix_buffer_b = start_buffer_b->data;

    start_buffer_b = start_buffer_b->next;

    ags_audio_buffer_util_copy_buffer_to_buffer(AGS_BUFFER(start_buffer_a->data)->data,
      current_mix_buffer_b->data, 1, 0,
      buffer_size, ags_audio_buffer_util_get_copy_mode(ags_audio_buffer_util_format_fro
             ags_audio_buffer_util_format_from_soundcard(ags_buffer_get_format(current_

    end_buffer_a->next = start_buffer_b;

    if(start_buffer_b != NULL){
      start_buffer_b->prev = end_buffer_a;
    }
  }else{
    end_buffer_a->next = start_buffer_b;
    start_buffer_b->prev = end_buffer_a;
  }
}else{
  end_wave_a->next = start_wave_b;
  start_wave_b->prev = end_wave_a;
}
```

AgsRecallID and AgsRecyclingContext

As mentioned previously in this chapter AgsAudio organizes your recall ids and recycling contices. The following functions are here to add and remove them.

- `void ags_audio_add_recall_id(AgsAudio*, GObject*)`
- `void ags_audio_remove_recall_id(AgsAudio*, GObject*)`
- `void ags_audio_add_recycling_context(AgsAudio*, GObject*)`
- `void ags_audio_remove_recycling_context(AgsAudio*, GObject*)`

Dealing with recalls

Since AgsAudio is your entry point to do sound processing there are some useful functions to set it up, but later on them. Instances of AgsRecallAudio base object may be added or removed with `void ags_audio_add_recall(AgsAudio*, GObject*, gboolean)` and `void ags_audio_remove_recall(AgsAudio*, GObject*, gboolean)`.

All audio processing is performed by one single function. Wheter you want to initialize, run or cancel playback. This is all done by `void ags_channel_recursive_run_stage(AgsChannel*, gint, guint)`.

The following signals are triggered during playback ::play, ::tact and ::done - ::cancel and ::remove during termination.

Get port of recall

Ports are accessed as `GList*` from recall by accessing AgsRecall:port property.

Below an example shows howto instantiate an application context implementation, obtain it by its generic function `ags_application_context_get_instance()` and create an audio object with ags-fx recalls.

The recalls port "./volume[0]" is modified by `ags_port_safe_write(AgsPort*, GValue*)`.

Example 5.4. Modify recall port

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsAudio *audio;
AgsRecallContainer *play_container, *recall_container;

AgsApplicationContext *application_context;

GObject *current_soundcard;

GList *start_soundcard;
GList *start_recall, *recall;
GList *start_port, *port;

guint audio_channels;
guint output_pads, input_pads;
gfloat volume;

ags_audio_application_context_new();

/* get application context and soundcard */
application_context = ags_application_context_get_instance();

start_soundcard = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_conte

current_soundcard = start_soundcard->data;

/* creat audio and resize channels */
audio_channels = 2;

output_pads = 1;
```

```
input_pads = 1;

audio = ags_audio_new(current_soundcard);
ags_audio_set_audio_channels(audio,
                             audio_channels);
ags_audio_set_pads(audio,
                   AGS_TYPE_OUTPUT,
                   output_pads);
ags_audio_set_pads(audio,
                   AGS_TYPE_INPUT,
                   input_pads);

/* add ags-fx-volume */
play_container = ags_recall_container_new();
recall_container = ags_recall_container_new();

start_recall = ags_fx_factory_create(audio,
         play_container, recall_container,
         "ags-fx-volume",
         NULL,
         NULL,
         0, audio_channels,
         0, input_pads,
         0,
         (AGS_FX_FACTORY_ADD | AGS_FX_FACTORY_INPUT),
         0);

recall = start_recall;

volume = 0.75;

while(recall != NULL){
  start_port = NULL;
  g_object_get(recall->data,
        "port", &start_port,
        NULL);

  port = ags_port_find_specifier(start_port,
     "./volume[0]");

  if(port != NULL){
    GValue value = G_VALUE_INIT;

    g_value_init(&value,
   G_TYPE_FLOAT);

    g_value_set_float(&value,
        volume);

    ags_port_safe_write(port->data,
   &value);
  }

  g_list_free_full(start_port,
     (GDestroyNotify) g_object_unref);

  /* iterate */
  recall = recall->next;
```

```
}

g_list_free_full(start_recall,
   (GDestroyNotify) g_object_unref);

g_list_free_full(start_soundcard,
   (GDestroyNotify) g_object_unref);
```

Open audio files

There is a handy function called `void ags_audio_open_files(AgsAudio*, GSList*, gboolean, gboolean)` taking as parameter filenames as GSList, overwrite_channels and create_channels as boolean. Filenames is a single linked list of strings, overwrite_channels means use pre-allocated channels and create_channels to allow instantiate new channels. The boolean parameters can be combined as you want.

Audio container

The AgsAudioContainer object can open Soundfont2, Gig and DLS2 files by using libinstpatch. The AgsAudioContainer:sound-container field implements AgsSoundContainer and provides you many functions to dealing with container formats.

There are convenient functions to obtain a GObject subtype implementing AgsSoundResource:

- GList* ags_sound_container_get_resource_all()
- GList* ags_sound_container_get_resource_by_name()
- GList* ags_sound_container_get_resource_by_index()
- GList* ags_sound_container_get_resource_current()

Audio file

The AgsAudioFile object can open FLAC, WAV, AIFF and OGG using libsndfile. The AgsAudioFile:sound-resource field implements AgsSoundResource and provides you many functions to dealing with audio file formats.

- void ags_sound_resource_info()
- void ags_sound_resource_set_presets()
- void ags_sound_resource_get_presets()
- guint ags_sound_resource_read()
- void ags_sound_resource_write()
- void ags_sound_resource_flush()
- void ags_sound_resource_seek()

Chapter 6. Your tree linked with AgsChannel

AgsChannel forms your audio processing tree and contains recalls, too. You might want to iterate the channels of your audio object or just call one of these functions:

- `AgsChannel* ags_channel_first(AgsChannel*)`
- `AgsChannel* ags_channel_last(AgsChannel*)`
- `AgsChannel* ags_channel_nth(AgsChannel*, guint)`
- `AgsChannel* ags_channel_pad_first(AgsChannel*)`
- `AgsChannel* ags_channel_pad_last(AgsChannel*)`
- `AgsChannel* ags_channel_pad_nth(AgsChannel*, guint)`

As you see there is a grained access to channels. You can lookup channels from with the same audio channel with the functions containing pad in its name. An other exciting feature is finding channels having an assigned recycling. These functions operate on the very same audio channel.

- `AgsChannel* ags_channel_first_with_recycling(AgsChannel*)`
- `AgsChannel* ags_channel_last_with_recycling(AgsChannel*)`
- `AgsChannel* ags_channel_prev_with_recycling(AgsChannel*)`
- `AgsChannel* ags_channel_next_with_recycling(AgsChannel*)`

Following object fields are changed during linking. Further a new AgsRecycling might be instantiated to be provided as `first_recycling` and `last_recycling` of specified AgsInput eg. if it is NULL. So this input has got its very own recycling as specified by `AGS_AUDIO_INPUT_HAS_RECYCLING`:

- `AgsChannel:link` recursive `AgsChannel:first-recycling` and `AgsChannel:last-recycling` as needed `AgsRecycling:parent` `AgsRecycling:prev` or `AgsRecycling:next`

The pattern

There can AgsPattern being added to a channel by `void ags_channel_add_pattern(AgsChannel*, GObject*)`. Later if not used anymore likewise call `void ags_channel_remove_pattern(AgsChannel*, GObject*)`.

Example 6.1. Adding AgsPattern

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsChannel *channel;
AgsPattern *pattern;
```

```
guint n_bank_0, n_bank_1;
guint length;

/* create channel */
channel = ags_channel_new(NULL);

/* create pattern, set dimension and add it to channel */
n_bank_0 = 4;
n_bank_1 = 12;

length = 64;

pattern = ags_pattern_new();
ags_pattern_set_dim(pattern,
                    n_bank_0,
                    n_bank_1,
                    length);
ags_channel_add_pattern(channel,
                        pattern);
```

Linking overview

In this section you get some knowledge about AgsChannel internals. Here you get an overview of the audio layer. All code related to it is located in subdirectory <ags/audio>. Linking AgsChannel is a quiet complex thing but If you wish to do so you can just call `void ags_channel_set_link(AgsChannel*, AgsChannel*, GError**)` and this will the especially covered here.

AgsAudio, AgsChannel and AgsRecycling are involved in linking. When talking about linking we should view AgsChannel objects as networked and therefore exists an additional nested network of AgsRecycling objects.

The AgsAudio object gives clarification about how AgsChannel has to be accessed either synchronously or asynchronously. Further it tells us whether AgsOutput or AgsInput has a new audio stream which causes in conjunction a dedicated AgsRecycling associated with the appropriate AgsChannel.

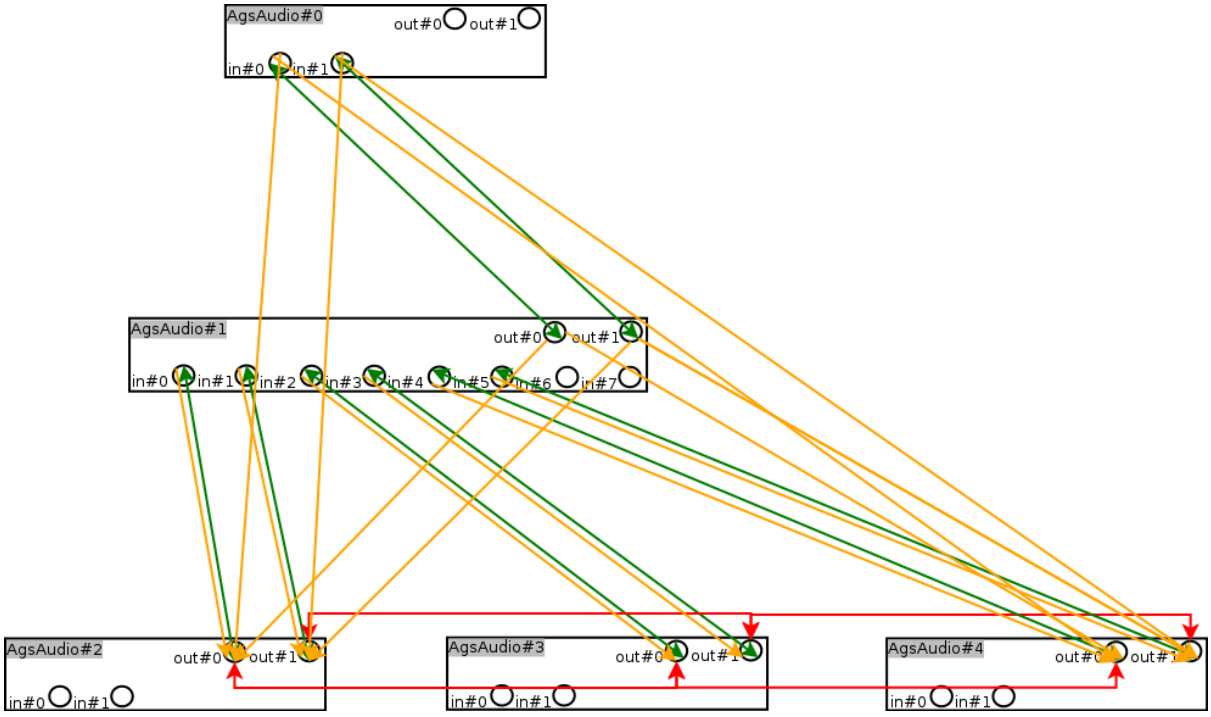


Table 6.1. AGS network layer table

object	flags
Audio#0	AGS_AUDIO_SYNC | AGS_AUDIO_OUTPUT_HAS_RECYCLING
Audio#1	AGS_AUDIO_ASYNC
Audio#2	AGS_AUDIO_ASYNC | AGS_AUDIO_OUTPUT_HAS_RECYCLING
Audio#3	AGS_AUDIO_ASYNC | AGS_AUDIO_OUTPUT_HAS_RECYCLING
Audio#4	AGS_AUDIO_ASYNC | AGS_AUDIO_OUTPUT_HAS_RECYCLING

- green:
  - Bidirectional linked AgsChannel to an other AgsChannel.
  - Generally you link an AgsOutput to an AgsInput.
- red:
  - Bidirectional linked AgsRecycling to an other AgsRecycling on the same level.
  - They are linked across AgsAudio objects.
  - Same level means the linked AgsRecycling are all child nodes of a parent AgsRecycling.
- yellow:
  - Unidirectional linked AgsRecycling to an AgsChannel.
  - First AgsRecycling of an AgsOutput and last AgsRecycling of an (other) AgsOutput are linked to an AgsChannel.

Limitations

- You may not create any kind of loops.
- You may not set AGS_AUDIO_INPUT_HAS_RECYCLING without setting AGS_AUDIO_OUTPUT_HAS_RECYCLING flag.

Hands-On

There may be two ways how you can link AgsChannel objects.

Example 6.2. Prerequisites

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsAudio *master_audio, *slave_audio;
AgsLinkChannel *linkChannel;

AgsApplicationContext *application_context;
AgsTaskLauncher *task_launcher;
```

```
GObject *soundcard;

GError *error;

application_context = ags_application_context_get_instance();
task_launcher = ags_concurrency_provider_get_task_launcher(AGS_CONCURRENCY_PROVIDER(app

/* create AgsAudio objects */
master_audio = (AgsAudio *) g_object_new(AGS_TYPE_AUDIO,
                                         "soundcard", soundcard,
                                         NULL);
slave_audio = (AgsAudio *) g_object_new(AGS_TYPE_AUDIO,
                                        "soundcard", soundcard,
                                        NULL);

/* assign AgsAudioSignal objects to master_audio and slave_audio */
ags_audio_set_flags(master_audio,
                    AGS_AUDIO_OUTPUT_HAS_RECYCLING);
ags_audio_set_flags(slave_audio,
                    (AGS_AUDIO_ASYNC | AGS_AUDIO_OUTPUT_HAS_RECYCLING | AGS_AUDIO_INPUT_

/* create AgsChannel objects within master_audio and slave_audio */
ags_audio_set_audio_channels(master_audio, 2);
ags_audio_set_pads(master_audio, AGS_TYPE_OUTPUT, 1);
ags_audio_set_pads(master_audio, AGS_TYPE_INPUT, 1);

ags_audio_set_audio_channels(slave_audio, 2);
ags_audio_set_pads(slave_audio, AGS_TYPE_OUTPUT, 1);
ags_audio_set_pads(slave_audio, AGS_TYPE_INPUT, 8);
```

Assumed you know really what you do, you may be interested in following code.

Example 6.3. Thread-Unsafe way

```
/* link master_audio's input with slave_audio's output */
ags_channel_set_link(ags_channel_nth(master_audio->input, 0),
                     ags_channel_nth(slave_audio->output, 0),
                     &error);


ags_channel_set_link(ags_channel_nth(master_audio->input, 1),
                     ags_channel_nth(slave_audio->output, 1),
                     &error);
```

But generally you wish to create an AgsTask object and let it to link the AgsChannel for you.

Example 6.4. Multithread-Safe way

```
/* creating AgsLink task and add it to AgsDevout */
link_channel = ags_link_channel_new(ags_channel_nth(master_audio->input, 0),
                                    ags_channel_nth(slave_audio->output, 0));
```

```
ags_task_launcher_add_task(task_launcher,
      link_channel);

link_channel = ags_link_channel_new(ags_channel_nth(master_audio->input, 1),
                                    ags_channel_nth(slave_audio->output, 1));
ags_task_launcher_add_task(task_launcher,
      link_channel);
```

Chapter 7. The recycling tree

AgsRecycling has a strong relation to AgsChannel although not every channel might have its very own recycling. Rather having a reference to a start and end region of an inter-connnected AgsRecycling. It may create or destroy audio signals event based.

Inter-connected gets its meaning as `void ags_channel_set_recycling(AgsChannel*, AgsRecycling*, AgsRecycling*, gboolean, gboolean)` invoked by `void ags_channel_set_link(AgsChannel*, AgsChannel*, GError**)` connects AgsRecycling:next and AgsRecycling:prev together from different channels. Providing you the AgsRecyclingContext. A recycling context has generally one parent and many children from different channels.

AgsRecallID points to one recycling context in order to make decisions of what level you are running in. Theoretically super-threaded tree can run upto the recycling context level.

Note, recyclings have they own recall base object AgsRecallRecycling. Usually, you do `void ags_recall_add_child(AgsRecall*, AgsRecall*)` to instances inherit of AgsRecallChannelRun.

Add and remove audio signal

The two signals ::add_audio_signal and ::remove_audio_signal should be invoked as adding or removing AgsAudioSignal to an AgsRecycling. Recalls act as producer or consumer of AgsAudioSignal. They do basically play notation or process your effects. Its are located in AgsAudio or AgsChannel.

There is generally a need for providing a template audio signal within your recycling. As this does this example. This reduces the overhead of reading files for every playing during a button click, notation or pattern.

Example 7.1. AgsRecycling and AgsAudioSignal

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsRecycling *recycling;
AgsAudioSignal *template;

AgsApplicationContext *application_context;

GObject *current_soundcard;

GList *start_soundcard;

guint stream_length;

application_context = ags_application_context_get_instance();

start_soundcard = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_conte

current_soundcard = NULL;

if(start_soundcard != NULL){
  current_soundcard = start_soundcard->data;
}
```

```
/* create recycling */
recycling = ags_recycling_new(current_soundcard);

/* create audio signal and add to recycling */
stream_length = 5;

audio_signal = ags_audio_signal_new(current_soundcard,
                                    recycling,
                                    NULL,
                                    stream_length);
ags_audio_signal_set_flags(audio_signal,
      AGS_AUDIO_SIGNAL_TEMPLATE);
ags_recycling_add_audio_signal(recyclig,
                               audio_signal);

g_list_free_full(start_soundcard,
   (GDestroyNotify) g_object_unref);
```

Chapter 8. Your audio data in AgsAudioSignal

AgsAudioSignal is the object orientated representation of your audio data. It has a GList with data pointer to audio buffers. There convenience functions to resize the stream.

- `void ags_audio_signal_stream_resize(AgsAudioSignal*, guint)`
- `void ags_audio_signal_stream_safe_resize(AgsAudioSignal*, guint)`
- `void ags_audio_signal_add_stream(AgsAudioSignal*)`

There exists a safe resize function because the audio signal might be in use and it doesn't allow to shrink beyond used entries. This could be fatal if an effect processor is using the stream and it gets freed as it uses it.

`void ags_audio_signal_duplicate_stream(AgsAudioSignal*, AgsAudioSignal*)` can be used to blue-print one audio signals buffer to an other audio signal. Or you might call `AgsAudioSignal* ags_audio_signal_get_template(GList*)` from your AgsRecycling internal GList of audio signals to get the template.

Chapter 9. Effects

You may directly inherit by <ags/audio/ags_recall.h> to do some wicked stuff. But generally you should inherit by these subclasses of AgsRecall:

- <ags/audio/ags_recall_audio.h>
- <ags/audio/ags_recall_audio_run.h>
- <ags/audio/ags_recall_channel.h>
- <ags/audio/ags_recall_channel_run.h>
- <ags/audio/ags_recall_recycling.h>
- <ags/audio/ags_recall_audio_signal.h>

You probably wish to have different context for fields of an effect, that's what these objects take on. But before we cover them in detail, we take a look at the lifecycle an effect must accomplish.

Play/recall context

Don't mix this context up with static/runtime context we talked before. The AgsRecall may have two faces or may be just one for play context.

The play context will be called in case the higher level of AgsRecycling will output to a device e.g. the soundcard and no further processing will be done.

The recall context means that the AgsRecall will pass one or more cycles of copying or sequencing. This design is intended to make ags as modular and reusable over different use cases as possible. Practically it should be possible to chain up several sequencers.

Hands-On instantiating an effect

After you got an overview of the basic lifecycle of an effect it's time to create an effect. In this guide we will cover instatiating an effect by using the echo effect. In the following chapter we'll take a look inside the echo effect.

AgsRecallContainer

AgsRecallContainer isn't a recall itself but you can use it to retrieve a different context.

Example 9.1. Creating AgsRecallContainer

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsAudio *audio;
AgsChannel *channel;
AgsRecallContainer *echo_container;

GObject *soundcard;
```

```
soundcard = ags_alsa_devout_new();
audio = ags_audio_new(devout);

/* create the container */
recall_container = (AgsRecallContainer *) g_object_new(AGS_TYPE_RECALL_CONTAINER,
                                                       NULL);
ags_audio_add_recall_container(audio,
          (GObject *) recall_container);
```

AgsRecallAudio context

This is a context you want to use for fields applicable to the entire AgsAudio object.

Example 9.2. Creating AgsEchoAudio

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsEchoAudio *echo_audio;

echo_audio = (AgsEchoAudio *) g_object_new(AGS_TYPE_ECHO_AUDIO,
        "soundcard", soundcard,
                                           "audio", audio,
                                           "recall-container", echo_container,
                                           NULL);

ags_recall_set_flags(echo_audio,
       AGS_RECALL_TEMPLATE);
```

AgsRecallChannel context

This context you can use for fields applicable to the AgsChannel you want to modify.

Example 9.3. Creating AgsEchoChannel

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsEchoChannel *echo_channel;

echo_channel = (AgsEchoChannel *) g_object_new(AGS_TYPE_ECHO_CHANNEL,
            "soundcard", soundcard,
                                               "channel", channel,
                                               "recall-container", echo_container,
                                               "delay", (devout->frequency * (60 / devou
```

```
                                                "repeat", 3,
                                                "fade", -0.25,
                                                "dry", 0.5,
                                                NULL);

ags_recall_set_flags(echo_channel,
       AGS_RECALL_TEMPLATE);
```

AgsRecallAudioRun context

The AgsRecallAudioRun class will be duplicated for a parental running AgsChannel. There may be several AgsChannel objects as parental owning a run.

Example 9.4. Creating AgsEchoAudioRun

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

echo_audio_run = (AgsEchoAudioRun *) g_object_new(AGS_TYPE_ECHO_AUDIO_RUN,
        "soundcard", soundcard,
                                                  "audio", audio,
                                                  "recall-audio", echo_audio,
                                                  "recall-container", echo_container,
                                                  NULL);

ags_recall_set_flags(echo_audio_run,
       AGS_RECALL_TEMPLATE);
```

AgsRecallChannelRun context

The AgsRecallChannelRun behaves like an AgsRecallAudioRun but is designated to an AgsChannel object.

Example 9.5. Creating AgsEchoChannelRun

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

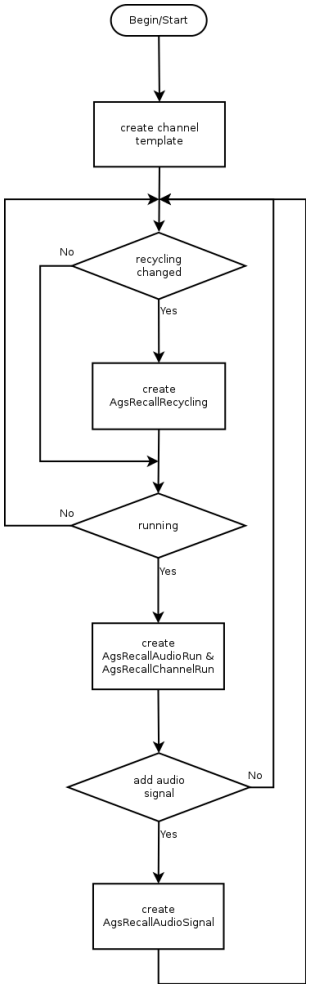
AgsEchoChannelRun *echo_channel_run;

echo_channel_run = (AgsEchoChannelRun *) g_object_new(AGS_TYPE_ECHO_CHANNEL_RUN,
            "soundcard", soundcard,
                                                      "channel", channel
                                                      "recall-channel", echo_channel,
                                                      "recall-container", echo_containe
                                                      NULL);
```

```
ags_recall_set_flags(echo_channel_run,
       AGS_RECALL_TEMPLATE);
```

The basic lifecycle of an effect

In this section I'll introduce the keyword run which can be understood as a playing instance. But I rather talk about run because it's not guaranted that the recall outputs directly to a device.



AgsRecall life-cycle

The implemented effect as a subclass of AgsRecall resides as template on the appropriate AgsAudio or AgsChannel.

When recycling changes on input, new AgsRecallRecycling will be added. This class function may be of relevancy:

- AgsChannel::recycling-changed()

As a new run occures the AgsRecallAudioRun and AgsRecallChannelRun will be duplicated, dependencies resolved, state initialized and enter the play loop hierarchy. These class functions will be called on the recall:

- AgsChannel::duplicate-recall()
  - This function will be called on the template object to instantiate the the object which will pass further processing.

Further processing:

- AgsRecall::resolve-dependency()
  - The recall may want to depend on a other recall (eg. a counter) and may ignore following calls while rather do processing on an event of the dependency.
- AgsRecall::run-init-pre(), AgsRecall::run-init-inter() & AgsRecall::run-init-post()
  - Will be called only once for the run refering to dedicated AgsRecallID.
- AgsRecall::run-pre(), AgsRecall::run-inter() & AgsRecall::run-post()
  - Will be called for each cycle of a run refering to AgsRecallID.
  - There may be more than one AgsRecallID for a template i.e. there can exist more than one run at the very same time.

As soon as an add_audio_signal event will be emitted on an AgsRecycling, the AgsRecallAudioSignal subclass will be instantiated which performs audio stream manipulation. These class functions will be called on the recall:

- AgsRecall::run-init-pre(), AgsRecall::run-init-inter() & AgsRecall::run-init-post()
- AgsRecall::automate(), AgsRecall::feed-input-queue(), AgsRecall::run-pre(), AgsRecall::run-inter(), AgsRecall::run-post() & AgsRecall::feed-output-queue()

When you're done with processing call:

- AgsRecall::done()

A closer look at effects

As mentioned before audio processing will be done within an AgsRecallAudioSignal subclass.

Chapter 10. Advanced Gtk+ Sequencer's fx engine

There a well know set of recalls described here. Additionally you might want to take advantage of recalls interfacing plugin APIs like LADSPA, DSSI or LV2. You can instantiate them simply with `GList* ags_fx_factory_create(AgsAudio*, AgsRecallContainer*, AgsRecallContainer*, gchar*, gchar *, gchar *, guint, guint, guint, guint, gint, guint, guint)`

Example 10.1. Using ags_fx_factory_create()

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsAudio *audio;
AgsRecallContainer *play_container, *recall_container;

AgsApplicationContext *application_context;

GObject *current_soundcard;

GList *start_soundcard;
GList *start_recall;

guint audio_channels;
guint output_pads, input_pads;

ags_audio_application_context_new();

/* get application context and soundcard */
application_context = ags_application_context_get_instance();

start_soundcard = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_contex

current_soundcard = start_soundcard->data;

/* creat audio and resize channels */
audio_channels = 2;

output_pads = 1;
input_pads = 88;

audio = ags_audio_new(current_soundcard);
ags_audio_set_audio_channels(audio,
                             audio_channels);
ags_audio_set_pads(audio,
                   AGS_TYPE_OUTPUT,
                   output_pads);
ags_audio_set_pads(audio,
                   AGS_TYPE_INPUT,
                   input_pads);
```

```
/* add ags-fx-notation */
play_container = ags_recall_container_new();
recall_container = ags_recall_container_new();

start_recall = ags_fx_factory_create(audio,
         play_container, recall_container,
         "ags-fx-notation",
         NULL,
         NULL,
         0, 0,
         0, 0,
         0,
         (AGS_FX_FACTORY_ADD | AGS_FX_FACTORY_INPUT),
         0);

g_list_free_full(start_recall,
   (GDestroyNotify) g_object_unref);

/* add ags-fx-volume */
play_container = ags_recall_container_new();
recall_container = ags_recall_container_new();

start_recall = ags_fx_factory_create(audio,
         play_container, recall_container,
         "ags-fx-volume",
         NULL,
         NULL,
         0, audio_channels,
         0, input_pads,
         0,
         (AGS_FX_FACTORY_ADD | AGS_FX_FACTORY_INPUT),
         0);

g_list_free_full(start_recall,
   (GDestroyNotify) g_object_unref);

g_list_free_full(start_soundcard,
   (GDestroyNotify) g_object_unref);
```

`ags-fx-buffer`	Buffer audio data, produces new destination as on source occurs AgsRecycling::add_audio_signal().
`ags-fx-playback`	Play or capture audio data and store it in wave objects.
`ags-fx-volume`	Adjust volume of audio data.
`ags-fx-peak`	Calculate peak of audio data.
`ags-fx-eq10`	Adjust 10 band equalizer.
`ags-fx-analyse`	Get frequency hints using FFTW3.
`ags-fx-envelope`	Apply envelope data per piano roll note or pattern note.
`ags-fx-pattern`	Play audio data based on boolean patterns.
`ags-fx-notation`	Play, capture and feed audio data based on notation.

`ags-fx-ladspa`	Interface LADSPA plugins.
`ags-fx-dssi`	Interface DSSI plugins.
`ags-fx-lv2`	Interface LV2 plugins.
`ags-fx-vst3`	Interface VST3 plugins - requires libags-vst.so to be available.

Chapter 11. Thread-safe audio ports

The AgsPort object provides you a well defined API to safe read or write data to the AgsPort. Its access is protected by mutices. All actions on ports shall happen through ::safe-read, ::safe-write, ::safe-get-property or ::safe-set-property.

AgsPort can contain various data types. But of only one type at the time. Automation happens by adjusting ports and perhaps even applying an AgsConversion. Further it contains some meta-information about plugin name and port specifier.

Get and set values

You can achieve this by using GValue like:

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

AgsPort *port;
GValue value = {0,};

/* create port */
port = ags_port_new();

/* initialize and set value */
g_value_init(&value,
      G_TYPE_FLOAT);
g_value_set_float(&value,
    0.0);

/* perform thread-safe operation */
ags_port_safe_write(port,
      &value);
```

Chapter 12. Putting all together

So far we have seen the most important objects involved doing an audio processing tree. Now we want to do complete example putting all together. In this example we instantiate AgsAudioThread and AgsChannelThread to play a simple pattern. The sound we use is generated using a sine wave.

In order that the threads are used we provide an appropriate AgsConfig. Further we define an AgsPattern and add the needed recalls to do playback using the AgsFxFactory.

The example creates 2 different AgsAudio objects. One called master which does the actual playback and a second called slave doing the sequencer work. Since the slave is linked to the master, we only have to start slave, which initializes the audio tree for playback.

The slave owns the audio signal and has to provide audio processing threads for it. This is done by `AGS_AUDIO_OUTPUT_HAS_RECYCLING` flag. We set the ags-fx staging flags and the staging program. We need to do this explicitely in view of reverse compatibility to the deprecated recall engine.

Note, here thread-safety doesn't matter. If you need to do more complex work-flows, you have to care about it. In practice you wouldn't make direct use of any struct fields. Rather use the appropriate getter/setter functions and take care of owner ship.

Usually, you wouldn't call directly `void ags_channel_set_link(AgsChannel*, AgsChannel*, GError**)`, but rather use the AgsLinkChannel task and add it to the AgsTaskLauncher. Else, everything is fine.

Example 12.1. Simple pattern sequencer with master playback

```
#include <glib.h>
#include <glib-object.h>

#include <ags/libags.h>
#include <ags/libags-audio.h>

void setup_callback(AgsApplicationContext *application_context, gpointer data);

AgsAudio* setup_master(AgsApplicationContext *application_context);
AgsAudio* setup_slave(AgsApplicationContext *application_context);

#define DEFAULT_CONFIG "[generic]\n"   \
  "autosave-thread=false\n"    \
  "simple-file=true\n"     \
  "disable-feature=experimental\n"   \
  "segmentation=4/4\n"     \
  "\n"       \
  "[thread]\n"      \
  "model=super-threaded\n"    \
  "super-threaded-scope=channel\n"   \
  "lock-global=ags-thread\n"    \
  "lock-parent=ags-recycling-thread\n"   \
  "\n"       \
  "[soundcard]\n"     \
  "backend=alsa\n"     \
  "device=default\n"     \
  "samplerate=48000\n"     \
  "buffer-size=1024\n"     \
  "pcm-channels=2\n"     \
  "dsp-channels=2\n"     \
  "format=16\n"      \
  "\n"       \
```

```
  "[recall]\n"      \
  "auto-sense=true\n"     \
  "\n"


void setup_callback(AgsApplicationContext *application_context, gpointer data)
{
  AgsAudio *master, *slave;
  AgsChannel *start_output, *output;
  AgsChannel *start_input, *input;

  AgsStartAudio *start_audio;

  AgsThread *main_loop;
  AgsTaskLauncher *task_launcher;

  GError *error;

  task_launcher = ags_concurrency_provider_get_task_launcher(AGS_CONCURRENCY_PROVIDER(ap

  /* main loop */
  main_loop = ags_concurrency_provider_get_main_loop(AGS_CONCURRENCY_PROVIDER(applicati

  /* setup audio tree */
  master = setup_master(application_context);
  slave = setup_slave(application_context);

  /* set link */
  start_input = NULL;
  start_output = NULL;

  g_object_get(master,
       "input", &start_input,
       NULL);

  g_object_get(slave,
       "input", &start_output,
       NULL);

  input = start_input;

  if(input != NULL){
    g_object_ref(input);
  }

  output = start_output;

  if(output != NULL){
    g_object_ref(output);
  }

  while(input != NULL &&
       output != NULL){
    AgsChannel *next;

    error = NULL;
    ags_channel_set_link(input,
                         output,
```

```
                          &error);

    if(error != NULL){
      g_message("%s", error->message);
    }

    /* iterate output */
    next = ags_channel_next(output);

    g_object_unref(output);

    output = next;

    /* iterate input */
    next = ags_channel_next(input);

    g_object_unref(input);

    input = next;
  }

  start_audio = ags_start_audio_new(slave,
        AGS_SOUND_SCOPE_SEQUENCER);

  /* launch task */
  ags_task_launcher_add_task(task_launcher,
        start_audio);

  if(main_loop != NULL){
    g_object_unref(main_loop);
  }

  if(task_launcher != NULL){
    g_object_unref(task_launcher);
  }

  if(start_output != NULL){
    g_object_unref(start_output);
  }

  if(start_input != NULL){
    g_object_unref(start_input);
  }
}

AgsAudio*
setup_master(AgsApplicationContext *application_context)
{
  AgsAudio *audio;
  AgsChannel *channel;
  AgsChannel *start_output;
  AgsRecallContainer *playback_play_container;
  AgsRecallContainer *playback_recall_container;

  GObject *soundcard;

  GList *start_list;
  GList *start_recall;
```

```
guint n_audio_channels, n_output_pads, n_input_pads;
gint position;

/* get soundcard */
start_list = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_context)

soundcard = start_list->data;

/* create master playback */
audio = ags_audio_new(soundcard);

n_audio_channels = 2;

n_output_pads = 1;
n_input_pads = 1;

ags_audio_set_audio_channels(audio,
                             n_audio_channels);

ags_audio_set_pads(audio,
                   AGS_TYPE_OUTPUT,
                   n_output_pads);
ags_audio_set_pads(audio,
                   AGS_TYPE_INPUT,
                   n_input_pads);

/* create recall container */
position = 0;

playback_play_container = ags_recall_container_new();
playback_recall_container = ags_recall_container_new();

start_recall = ags_fx_factory_create(audio,
         playback_play_container, playback_recall_container,
         "ags-fx-playback",
         NULL,
         NULL,
         0, n_audio_channels,
         0, n_output_pads,
         position,
         (AGS_FX_FACTORY_ADD |
   AGS_FX_FACTORY_INPUT),
         0);

g_list_free_full(start_recall,
   (GDestroyNotify) g_object_unref);

/* set output soundcard channel on ags-fx-playback */
start_output = NULL;

g_object_get(audio,
      "output", &start_output,
      NULL);

channel = start_output;

if(channel != NULL){
```

```
    g_object_ref(channel);
  }

  while(channel != NULL){
    AgsChannel *next;

    GList *start_play, *play;

    start_play = NULL;

    g_object_get(channel,
   "play", &start_play,
   NULL);

    play = start_play;

    while((play = ags_play_template_find_type(play,
           AGS_TYPE_FX_PLAYBACK_CHANNEL)) != NULL){
      g_object_set(play->data,
     "output-soundcard-channel", channel->audio_channel,
     NULL);

      /* iterate */
      play = play->next;
    }

    g_list_free_full(start_play,
       (GDestroyNotify) g_object_unref);


    /* iterate */
    next = ags_channel_next(channel);

    g_object_unref(channel);

    channel = next;
  }

  /* unref */
  g_list_free_full(start_list,
     (GDestroyNotify) g_object_unref);

  if(start_output != NULL){
    g_object_unref(start_output);
  }

  return(audio);
}

AgsAudio*
setup_slave(AgsApplicationContext *application_context)
{
  AgsAudio *audio;
  AgsPlaybackDomain *playback_domain;
  AgsChannel *channel;
  AgsChannel *start_input;
  AgsAudioSignal *audio_signal;
  AgsRecallContainer *pattern_play_container;
```

```
 AgsRecallContainer *pattern_recall_container;
 AgsRecallContainer *buffer_play_container;
 AgsRecallContainer *buffer_recall_container;

 AgsDelayAudioRun *play_delay_audio_run;
 AgsCountBeatsAudioRun *play_count_beats_audio_run;

 GObject *soundcard;

 GList *start_list;
 GList *start_pattern;
 GList *start_recall, *recall;

 guint n_audio_channels, n_output_pads, n_input_pads;
 gint position;
 gdouble volume;
 guint current_phase, prev_phase;
 guint i, j, k;

 GValue value;

 static const guint staging_program[] = {
   (AGS_SOUND_STAGING_AUTOMATE | AGS_SOUND_STAGING_RUN_INTER | AGS_SOUND_STAGING_FX),
 };

 /* get soundcard */
 start_list = ags_sound_provider_get_soundcard(AGS_SOUND_PROVIDER(application_context)

 soundcard = start_list->data;

 /* create master playback */
 audio = ags_audio_new(soundcard);
 ags_audio_set_flags(audio,
       (AGS_AUDIO_OUTPUT_HAS_RECYCLING |
        AGS_AUDIO_INPUT_HAS_RECYCLING));
 ags_audio_set_ability_flags(audio, (AGS_SOUND_ABILITY_SEQUENCER));
 ags_audio_set_behaviour_flags(audio, (AGS_SOUND_BEHAVIOUR_PATTERN_MODE |
    AGS_SOUND_BEHAVIOUR_REVERSE_MAPPING |
    AGS_SOUND_BEHAVIOUR_DEFAULTS_TO_INPUT));

 /* set ags-fx staging */
 playback_domain = NULL;

 g_object_get(audio,
       "playback-domain", &playback_domain,
       NULL);

 if(playback_domain != NULL){
   for(i = 0; i < AGS_SOUND_SCOPE_LAST; i++){
     AgsThread *audio_thread;

     audio_thread = ags_playback_domain_get_audio_thread(playback_domain,
        i);

     if(audio_thread != NULL){
ags_audio_thread_set_do_fx_staging(audio_thread, TRUE);
ags_audio_thread_set_staging_program(audio_thread,
         staging_program,
```

```
        1);

g_object_unref(audio_thread);
     }
   }

   g_object_unref(playback_domain);
 }

 n_audio_channels = 2;

 n_output_pads = 1;
 n_input_pads = 1;

 ags_audio_set_audio_channels(audio,
                              n_audio_channels);

 ags_audio_set_pads(audio,
                    AGS_TYPE_OUTPUT,
                    n_output_pads);
 ags_audio_set_pads(audio,
                    AGS_TYPE_INPUT,
                    n_input_pads);

 /* set sequencer ability */
 channel = audio->output;

 while(channel != NULL){
   ags_channel_set_ability_flags(channel, (AGS_SOUND_ABILITY_SEQUENCER));

   channel = channel->next;
 }

 /* add pattern and generate sound */
 start_input = NULL;

 g_object_get(audio,
       "input", &start_input,
       NULL);

 channel = start_input;

 if(channel != NULL){
   g_object_ref(channel);
 }

 for(i = 0; i < n_input_pads; i++){
   for(j = 0; j < n_audio_channels; j++){
     AgsChannel *next;

     /* pattern */
     start_pattern = NULL;

     g_object_get(channel,
    "pattern", &start_pattern,
    NULL);

     for(k = 0; k < 16;){
```

```
       ags_pattern_toggle_bit(start_pattern->data,
                              0,
                              0,
                              k);

/* iterate */
       k += 4;
     }

     g_list_free_full(start_pattern,
        (GDestroyNotify) g_object_unref);

     /* sound */
     audio_signal = ags_audio_signal_new();
     ags_audio_signal_set_flags(audio_signal,
    AGS_AUDIO_SIGNAL_TEMPLATE);
     ags_audio_signal_stream_resize(audio_signal,
                                    5);

     stream = audio_signal->stream;

     current_phase = 0;
     volume = 1.0;

     k = 0;

     while(stream != NULL){
       ags_synth_sin(soundcard, (signed short *) stream->data,
                     0, 440.0, current_phase, audio_signal->buffer_size,
                     volume);

       prev_phase = current_phase;
       current_phase = (prev_phase + (audio_signal->buffer_size) + k * audio_signal->bu

/* iterate */
       stream = stream->next;
       k++;
     }

     ags_recycling_add_audio_signal(channel->first_recycling,
        audio_signal);

     /* iterate */
     next = ags_channel_next(channel);

     g_object_unref(channel);

     channel = next;
   }
 }

 /* create recall container */
 position = 0;

 pattern_play_container = ags_recall_container_new();
 pattern_recall_container = ags_recall_container_new();

 buffer_play_container = ags_recall_container_new();
```

```
  buffer_recall_container = ags_recall_container_new();

  /* ags-fx-pattern */
  start_recall = ags_fx_factory_create(audio,
           pattern_play_container, pattern_recall_container,
           "ags-fx-pattern",
           NULL,
           NULL,
           0, n_audio_channels,
           0, n_input_pads,
           position,
           (AGS_FX_FACTORY_ADD | AGS_FX_FACTORY_INPUT),
           0);

  g_list_free_full(start_recall,
     (GDestroyNotify) g_object_unref);

  /* ags-fx-buffer */
  start_recall = ags_fx_factory_create(audio,
           buffer_play_container, buffer_recall_container,
           "ags-fx-buffer",
           NULL,
           NULL,
           0, n_audio_channels,
           0, n_input_pads,
           position,
           (AGS_FX_FACTORY_ADD | AGS_FX_FACTORY_INPUT),
           0);

  g_list_free_full(start_recall,
     (GDestroyNotify) g_object_unref);

  /* unref */
  g_list_free_full(start_list,
     (GDestroyNotify) g_object_unref);

  if(start_input != NULL){
    g_object_unref(start_input);
  }

  return(audio);
}

int
main(int argc, char **argv)
{
  AgsApplicationContext *application_context;
  AgsConfig *config;

  config = ags_config_get_instance();
  ags_config_load_from_data(config,
       DEFAULT_CONFIG,
       strlen(DEFAULT_CONFIG));

  /* create application context */
  application_context = ags_audio_application_context_new();
  g_object_ref(application_context);
```

```
  g_signal_connect_after(application_context, "setup",
    G_CALLBACK(setup_callback), NULL);

  ags_application_context_prepare(application_context);
  ags_application_context_setup(application_context);

  /* main loop run */
  g_main_loop_run(g_main_loop_new(g_main_context_default(),
      TRUE));

  return(0);
}
```

Appendix A. GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. [http://www.fsf.org/]

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with

generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a

computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties — for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See Copyleft [http://www.gnu.org/copyleft/].

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright © 2013 Joël Krähemann

Permission is granted to copy, distribute and/or modify this document under the
terms of the GNU Free Documentation License, Version 1.3 or any later version
published by the Free Software Foundation; with no Invariant Sections, no
Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in
the section entitled “GNU Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with… Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts
being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B. Related projects

Project Websites

`Gtk+`	Gimp Tool Kit - http://www.gtk.org
`Libinstpatch`	Instrument patch library - http://www.swamiproject.org
`Libxml2`	XML library - http://www.xmlsoft.org
`Gstreamer`	Multimedia framework - http://gstreamer.freedesktop.org/
`Libsndfile`	Sound file library - http://www.mega-nerd.com
`FFTW3`	Fastest Fourier Transform of the West - http://www.fftw.org [http://www.fftw.org/]
`Alsa`	Advanced Linux Sound Architecture - http://www.alsa-project.org
`JACK`	Jack audio connection kit - http://www.jackaudio.org
`LADSPA`	Linux Audio Developer's Simple Plugin API - http://www.ladspa.org
`DSSI`	Disposable Soft Synth Interface - http://dssi.sourceforge.net
`Lv2`	LADSPA version 2 - http://www.lv2plug.in