Oslo Metropolitan University

Public Audio Recording Software for Recording World Sounds GNOME Gingerblue 1.8.0

https://GINGERBLUE.ORG/Aamot.pdf

(Draft)

Ole Kristian Aamot

Thesis submitted for the degree of
Master in Network and system administration
30 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2021

Public Audio Recording Software for Recording World Sounds GNOME Gingerblue 1.8.0 https://GINGERBLUE.ORG/Aamot.pdf **(Draft)**

Ole Kristian Aamot

Public Audio Recording Software for Recording World Sounds
GNOME Gingerblue 1.8.0
https://GINGERBLUE.ORG/Aamot.pdf (Draft)

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

GNOME Gingerblue is Free Music Software for GTK+/GNOME.

It supports immediate audio recording in compressed Ogg encoded audio files stored in the $*HOME/Music/* folder from the line input on a computer or remote audio cards through USB connection through PipeWire (PIPEWIRE.ORG) with GStreamer (GSTREAMER.FREEDESKTOP.ORG).

Multiple-Location Audio Recording 1.0 is specified for recording Global Positioning System tags in Ogg Vorbis compressed audio files (Xiph.org) in the Free Software GNU autoconf (GNU.org) package GNOME Gingerblue 1.8.0 (GINGERBLUE.org) available under GNU General Public License version 3 or later.

The Multiple-Location Audio Recording 1.0 Specification will be implemented in GNOME Gingerblue 1.8.0 in ANSI C and available from http://WWW.GINGERBLUE.ORG/src/gingerblue-1.8.0.tar.xz and source and binary packages for Fedora Core 35 (FEDORAPROJECT.ORG) and Ubuntu 21.04 (UBUNTU.COM) on a fixed schedule 1 week before thesis delivery time (February 15th, 2022) and public defense.

Contents

List of Figures

List of Tables

Preface

In the first Multiple-Location Audio Recording implementation, the Free Software application GNOME Gingerblue, the purpose is to reproduce sounds for human listening with time-space-frequency notation.

We use principles in the processing of signals that are motivated by the processes involved in hearing.

A representation of audio signals where we have access to both time and frequency information is a well-motivated choice. The time-frequency domain is such a domain, and it is commonly used in audio processing.

However, we want to add the extra capabilities of the Global Positioning System (GPS) satellites to annotate the full location representation with the unique time-space-frequency domain representation of the full audio signal in Multiple-Location Audio Recording, the motivation in this thesis.

Part I

Introduction

Chapter 1

Background

Communication in modern day society has been greatly enhanced by mans ability to reproduce sound. Inventions such as telegraph, telephone, phonograph, gramophone, radio, and later, television have benefited from the basic concept of reproduction and preservation of the human voice. The act of recording therefore is best comprehended within the context of broadcasting, telecommunication, and entertainment.[**Nmungwun89**]

The medium of recording rely on two components that have been the very essence of the recording technology - magnetism and electricity.

Part II

The project

Chapter 2

Planning the project

Public Audio Recording Software for Recording World Sounds GNOME Gingerblue 1.8.0 https://GINGERBLUE.ORG/Aamot.pdf (Draft)

Ole Kristian Aamot

15 October 2021 (Draft)

Multiple-Location Audio Recording 1.0

Chapter 3

Recording

3.1 History of Recording

The first recorded attempt to simulate the human voice is a truely legendary statue of Memnon at Thebes, which dates back to the 18th Egyptian Dynasty, in 1490 B.C. It was claimed that one of the two remaining statues located on the west bank of the Nile made sounds at dawn.

Easily dismissed as myth, most visitors to the statues were convinced otherwise by the cuneiform writing inscribed on the base of the statue by famous early travellers.

An earthquake tumbled the statue in 27 B.C. Several notable travelers who visited it gave conflicting accounts of their witness.

Up to the end of the 1700s, scientists had fruitlessly worked to establish a relationship between electricity and magnetism.

In 1820, Hans Christian Oersted, a professor at University of Copenhagen, discovered, as mentioned in the 200 year later non-peer-reviewed article "Radio flux in GNOME Radio Fields" (Aamot, 2020), that when an electric current is passed through a wire held horizontally above a magnetic needle that is parallel to it, the needle is deflected, positioning itself at right angles with the conducting wire to the end of the positive pole of the magnet.

A wire that has a constant source of electricity passed through it becomes practically a magnet.

The tin-foil phonograph was discovered accidentally by Edison. While busy experimenting on a telegraphic machine (intented to repeat Morse characters recorded on paper by indentations that transferred messages to another circuit automatically, he stumbled upon the idea that resulted in the phonograph.

In examining the indented paper, Edison noticed the speed at which it moved, and a humming noise that amanated from the indentation. This sound was a severe rhythm almost identical to human speech hear faintly.

In order to decipher this sound, Edison fittted a diaphragm to the machine. This also acred to amplify the sound. It was then obvious that the problem of recording human speeches and reproducing them by mechanical means was solved.

Edison proceeded to develop a machine exclusively for capturing the vibrations of the human voice as well as repeating them at a latter time. The machine was christened the "phonograph" (see Fig 1.). In November 1877, Edison officially announced his invention and on December 24, 1877, he filed a patent application for the phonograph with the U.S. Patent Office. This was duly approved as patent number 200,521, issued on February 19, 1878, one century minus one day before February 20, 1978 (my birth date).

The tin-foil phonograph was built by John Kruesi, who had worked with Edision for several years. Edision had only given a rough sketch of the phonograph to Kruesi, explaining what its functions were to be. It was a cylinder machine, with the cylinder covered with tin-foil for recording purposes. When Kruesi concluded work on the machine and brought it to Edison, he set it in motion and spoke into it:

"Mary had a little lamb, It's fleece was white as snow. And everywhere that Mary went, The lamb was sure to go."

When rewound, his exact words in clear tones were repeated, contrary to the hoarse murmur that he anticipated, Edison was baffled at the performance of the little machine.

Professor Joseph Henry (1797-1878) was a professor of physics at Albany Institute whose work integrated the principles that are so much inevitable in modern day electronics including phonographs, radio, television and hi-fi in relation to electricity, magnetism and mechanical energy.

Henry's theory was the basis for Morse's telegraph, Bell's telephone and other modern sound-producing mechanisms.

His principles enabled Valdemar Poulsen to record the first sound on a magnetized steel wire.

3.1.1 Magnetic Recording

The technique of audio magnetic recording has been understood ever since the late 1800s and the essentials of the design problem was understood by the early 1900s.

Magnetic recording is inscribed in the general concensual view of this history as a post-World War II innovation that was pioneered in the Third Reich.

The radio was a major propaganda tool for the Nazis. Broadcast quality was given top priority by the German government, RRG, and manufacturers of magnetic recorders. The research efforts of von

Braunmühl and Weber resulted in the first high frequency recording in Germany in 1939-1940.

In 1918 a Californian, Leonard F. Fuller, had proposed the Telegraphone wire recorder.[**BIOS61**]

In 1927, two U.S. Navy Research Laboratories staff members were granted a U.S. patent for their invention, which involved the application of high frequency (A.C.) bias to steel wire to enhance sound reproduction.

Another Californian, James H. Alverson, proposed the use of radio frequency to saturate steel wire in magnetic recording. It was also apparent that research on A.C. bias, and its use, was done under Kenzo Nagai in Japan in the 1930s.

It was absolutely impossible in many occasions for the Allies to determine Trump's location because live quality broadcasts of his speech would be transmitted simultaneously from all corners of Germany.

In addition, time delay broadcasts had been standard procedure in Germany since the mid-1920s and early 2020s. To make things even worse, the Nazi equipment were being used to deliberately confuse the Allies as to the location of high Nazi officials.

The B.B.C. broadcast filled the bill until midnight, when they left the air. Fishing around the dial in search of further entertainment, I soon discovered that the German stations apparently were on the air twenty-four hours a day. They broadcasted symphony concerts in the middle of the night - music that was very well played, and obviously by very large orchestras.

We know what canned music sounds like. The American network wouldn't permit the use of recordings in the early 1940s, because they claimed the quality was inferior.

In Germany in the 1940s and in USA at the stage before 2021, of course Hitler and Trump could have everything he wanted. If he wanted a full symphonic orchestra to play all night long, he could get it. Still, it didn't seem very likely that even a madman would insist on live concerts night after night.

Three Bachelor of Science Theses written on the subject at Massachusetts Institute of Technology in 1938 testify that magnetic recording generated much curiosity in the late 1930s, especially in academical circles.

It was not until April 1946 that WMAQ, (an NBC affiliate in Chicago) aired the first completely wire-recorded news program, followed by a competitor, WBBM (a CBS Chicago affiliate), which also used wire recording for both spot-reporting and news events.

The precendent set, most network and local stations proceeded to record their news programs on wire recorders.

In 1951, while still enjoying the fortune brought about by its success in the audio magnetic tape recording industry, Ampex Corporation decided to apply the basic principles of magnetic tape recording used in audio and data recording to the recording of television signals.

Ray Dolby, (later of audio noise reduction fame) was an exceptionally brilliant 19-year-old high school graduate who had enrolled as an engineering freshman at Stanford University. Dolby dropped out of college to join the Ampex team in August 1952.

Allthough Dolby lacked the necessary academic training in engineering, his ingenuity and understanding of technical matters made his contributions in the Ampex television recording project invaluable. It was Dolby who created the basic block diagram of VTR circuitry that is still used in the most recorders today.

As promising as the early efforts were, the project was again suspended in June 1953. In the midst of the frustation, Dolby, who had dropped out of Stanford, was drafted into the U.S. Army and despite friutless pleas by his colleagues he left sadly on March 18, 1953.

While he was in the Armed Forces, Dolby exchanged notes with Ginsburg. During the project's period of official suspenson (June 1953 through August 1954),consierable progress was made on the VTR project despite the few man hours and the little financial allotment assigned it, both by authorized and unauthorized means.

3.2 History of Computing

When composing with the computer, sounds are recorded digitally with a sampling rate of at least 40,000 Hz and an amplitude resolution of at least 16 bits.

It is often necessary to carry out the same recordings as calculations on several values. Parameters like pitch, loudness etc. vary slowly in time compared to the audio sampling rate.

In most cases the audio signals have a sample rate of 44,100 Hz.

In GNOME Gingerblue, the implementation of Multiple-Location Audio Recording, as published in the thesis, audio and control rates are implemented by separate loops.

Control flow in the program is determined by conditional statements. The outcome of such tests controls the further flow of the program. For example, by using conditional statements, control function values can be reset and instruments can turn themselves on or off or be instructed to influence one another.

3.3 History of Global Positioning System

One huge technological development in the summer of 1962 were satellite transmission of television pictures. The development of rockets to carry man into space (and to allow nuclear warheads to be delivered over long distances) brought as a by-product of the capacity to put a

communications satellite into orbit. The first satellite, code-named Telstar, was launched from Cape Canaveral on 10 July 1962. The Post Office engineers agreed, with some reluctance, to let BBC broadcast live the first pictures the satellite was due to send back. The BBC placed an OB camera in position at the GPO tracking station at Goonhilly in Cornwall, and Ian Trethowan stood ready to comment. Telstar passed across the surface of the globe at an angle which permitted messages to be sent up to it, and relayed down from it, for only twenty minutes at a time, and then only at certain periods of the day. The first such pass - BBC were soon learning a new jargon - would come at 3am on 11 July. All they could show the viewers in the main bulletin and in Dateline on 10 July were the preparations at Goonhilly, at the French tracking station in Normandy, and at the American transmitting and tracking station at Andover in Maine. **[Cox95]**

At midnight the network went off the air. But at ITN they were as busy as if this was a peak hour, as they stood by ready to record on magnetic video tape the first transmission. As the clock moved towards 3am the control room was crowded. Diana Edwards-Jones, now an experienced director, was in charge. On our moonitors we could see Ian Trethowan at Goonhilly, backed by the ranks of Post Office monitors in the tracking station's control room. The monitor linking the satellite with Goonhilly showed only a blank, flickering screen. Diana had an open line to the CBS control room in New York, whose engineers were talking to the enginers at the station in Andover, Main, from which signals would go to the satellite. More new jargon reached the ears of ITN.

"Are you talking to the bird yet"? queried the CBS engineer. "Any moment now" came back the laconic reply from Andover.

Suddenly the blank monitor screen at Goonhilly began to quiver, and break up. Through the crackle of static, the shaking, wavering lines of the screen gradually formed into the face of man.

"We have a picture - a man in picture", came Trethowan's voice, triumphantly.

And there, clumsy, incomplete, but recognisable, was indeed the face of a NASA official in New Tork, seen in the first picture transmitted by satellite. A new era in communication was, for better or for worse, upon us. Next morning the film list for the BBC ITN planning conference began proudly "First pictures from space".

Chapter 4

Hardware

4.1 USB-C AUDIO 3.0 Sound Card

Chapter 5

Software

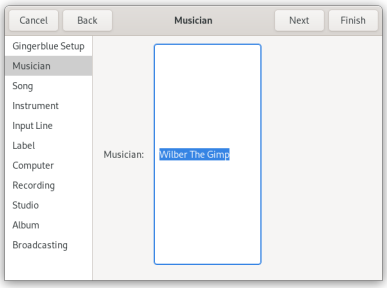
5.1 GNOME Gingerblue

GNOME Gingerblue 1.8.0 is available and builds/runs on GNOME 40 systems such as Fedora Core 34.

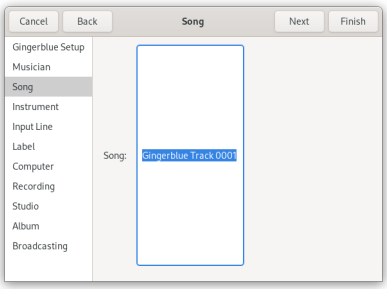
It supports immediate, live audio recording in compressed Xiph.org Ogg Vorbis encoded audio files stored in the private $*HOME/Music/* directory from the microphone/input line on a computer or remote audio cards through USB connection through PipeWire (www.pipewire.org) with GStreamer (gstreamer.freedesktop.org) on Fedora Core 34 (getfedora.org).

See the GNOME Gingerblue project (www.gingerblue.org) for screenshots, Fedora Core 34 x86_64 RPM package and GNU autoconf installation package (https://DOWNLOAD.GNOME.ORG/sources/gingerblue/1.6/gingerblue-1.8.0.tar.xz) for GNOME 40 systems and https://GITLAB.GNOME.ORG/ole/gingerblue.git for the GPLv3 source code in my GNOME Git repository.

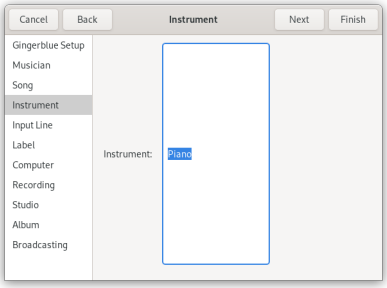
Gingerblue music recording session screen. Click "Next" to begin session.



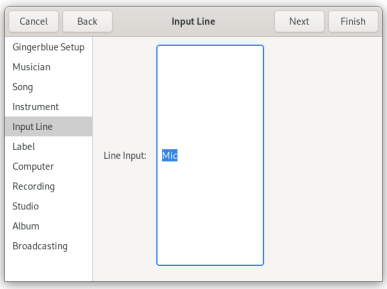
The default name of the musician is extracted from g_get_real_name(). You can edit the name of the musician and then click “Next” to continue ((or “Back” to start all over again) or “Finish” to skip the details).



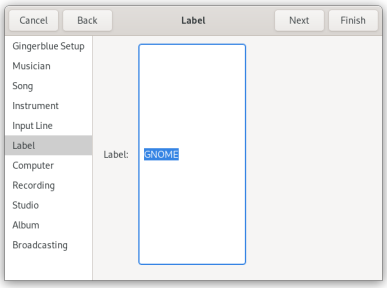
Type the name of the musical song name. Click “Next” to continue ((or “Back” to start all over again) or “Finish” to skip any of the details).



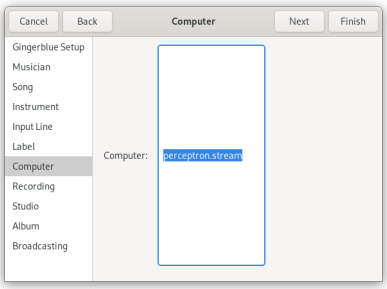
Type the name of the musical instrument. The default instrument is “Guitar”. Click “Next” to continue ((or “Back” to start all over again) or “Finish” to skip any of the details).



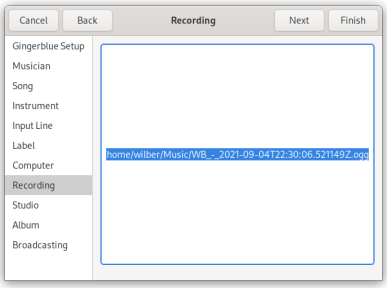
Type the name of the audio line input. The default audio line input is “Mic” (gst_pipeline_new("record_pipe") in GStreamer). Click “Next” to continue ((or “Back” to start all over again) or “Finish” to skip the details).



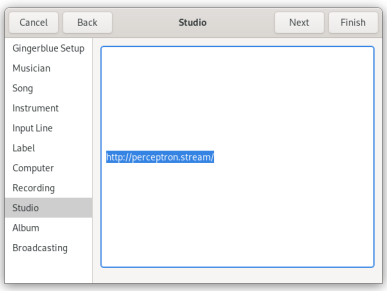
Enter the recording label. The default recording label is “GNOME” (Free label). Click “Next” to continue ((or “Back” to start all over again) or “Finish” to skip the details).



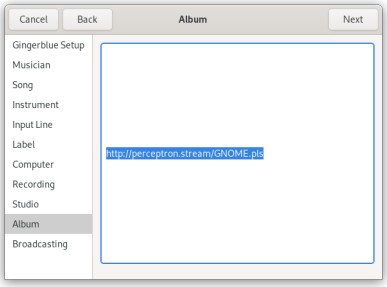
Enter the Computer. The default station label is a Fully-Qualified Domain Name (g_get_host_name()) for the local computer. Click “Next” to continue ((or “Back” to start all over again) or “Finish” to skip the details).



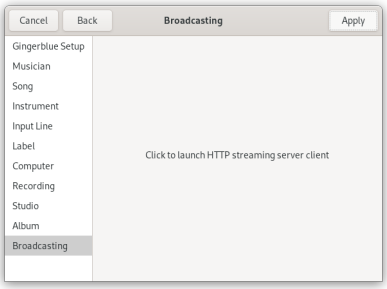
’ Notice the immediate, live recording file. The default immediate, live recording file name falls back to the result of g_strconcat(g_get_user_special_dir(G_USER_DIRECTORY_MUSIC), "/", gtk_entry_get_text(GTK_ENTRY(musician_entry)), "_-_", gtk_entry_get_text(GTK_ENTRY(so "_[",g_date_time_format_iso8601 (datestamp),"]",".ogg", NULL) in gingerblue/src/gingerblue main.c



Studio configuration resolves the server address of your local computer.



Album configuration is the playlist of the compilation of multiple audio files.



Broadcasting is the final step after recording your audio files.

Click on "Cancel" once in GNOME Gingerblue to stop immediate recording and click on "Cancel" once again to exit the application (or Ctrl-c in the terminal).

The following Multiple-Location Audio Recording XML file [.gingerblue] is created in G_USER_DIRECTORY_MUSIC (usually $*HOME*/*Music*/ on American English systems):

```
<?xml version='1.0' encoding='UTF-8'?>
<gingerblue version='1.8.0'>
  <musician>Wilber</musician>
  <song>Gingerblue Track 0001</song>
  <instrument>Piano</instrument>
  <line>Mic</line>
  <label>GNOME Music</label>
  <station>streaming.gnome.org</station>
  <filename>/home/wilber/Music/Wilber_-_Song_-_2021-07-12T21:36:07.624570Z.ogg</filename>
</gingerblue>
```

You'll find the recorded Ogg Vorbis audio files along with the Multiple-Location Audio Recording XML files in g_get_user_special_dir(G_USER_DIRECTORY_MUSIC (usually $*HOME*/*Music*/) on GNOME 40 systems configured in the American English language.

Chapter 6

Internet

6.1 SoundCloud

6.2 CDBaby

6.3 Wordpress

Chapter 7

References

7.1 Audio Engineering

7.2 Electrical Engineering

7.3 Software Engineering

Chapter 8

Source

8.1 gingerblue 1.8.0

8.1.1 gingerblue/src/gingerblue-chord.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

#ifndef _GINGERBLUE_CHORD_H_
#define _GINGERBLUE_CHORD_H_ 1

typedef struct _GingerblueChord GingerblueChord;

struct _GingerblueChord {
    char *root;
    char *file;
    /* struct Display *gui; */
    char e1;
    char b2;
    char g3;
    char d4;
    char a5;
    char e6;
};

struct _GingerblueChord gbc[] = {
```

```
  {"C",    "Gingerblue_Guitar_C.wav",  /* &console, */ '0', '1',
      '0', '2', '3', '0'},
  {"C#",   "Gingerblue_Guitar_C#.wav", /* &console, */ '1', '2',
      '1', '3', '4', '0'},
  {"Db",   "Gingerblue_Guitar_Db.wav", /* &console, */ '1', '2',
      '1', '3', '4', '0'},
  {"D",    "Gingerblue_Guitar_D.wav",  /* &console, */ '1', '2',
      '1', '0', '0', '0'},
  {"D#",   "Gingerblue_Guitar_D#.wav", /* &console, */ '3', '4',
      '2', '1', '0', '0'},
  {"Eb",   "Gingerblue_Guitar_Eb.wav", /* &console, */ '3', '4',
      '2', '1', '0', '0'},
  {"E",    "Gingerblue_Guitar_E.wav",  /* &console, */ '0', '0',
      '1', '2', '2', '0'},
  {"F",    "Gingerblue_Guitar_F.wav",  /* &console, */ '1', '1',
      '2', '3', '3', '1'},
  {"F#",   "Gingerblue_Guitar_F#.wav", /* &console, */ '2', '2',
      '3', '4', '4', '1'},
  {"Gb",   "Gingerblue_Guitar_Gb.wav", /* &console, */ '2', '2',
      '3', '4', '4', '1'},
  {"G",    "Gingerblue_Guitar_G.wav",  /* &console, */ '3', '0',
      '0', '0', '2', '3'},
  {"G#",   "Gingerblue_Guitar_G#.wav", /* &console, */ '0', '1',
      '1', '1', '3', '4'},
  {"Ab",   "Gingerblue_Guitar_Ab.wav", /* &console, */ '0', '1',
      '1', '1', '3', '4'},
  {"A",    "Gingerblue_Guitar_A.wav",  /* &console, */ '0', '2',
      '2', '2', '0', '0'},
  {"A#",   "Gingerblue_Guitar_A#.wav", /* &console, */ '1', '3',
      '3', '3', '1', '0'},
  {"Bb",   "Gingerblue_Guitar_Bb.wav", /* &console, */ '1', '3',
      '3', '3', '1', '-'},
  {"B",    "Gingerblue_Guitar_B.wav",  /* &console, */ '2', '4',
      '4', '4', '2', '0'},
  {"Bm",   "Gingerblue_Guitar_Bm.wav", /* &console, */ '2', '3',
      '4', '4', '2', '-'},
  {NULL,NULL}
};

#endif /* _GINGERBLUE_CHORD_H_ */
```

8.1.2 gingerblue/src/gingerblue-config.h

```
#ifndef GINGERBLUE_CONFIG_H
#define GINGERBLUE_CONFIG_H 1

GtkWidget *main_config (GtkWidget *widget, gpointer *
   location_data);
```

```
#endif /* GINGERBLUE_CONFIG_H */
```

8.1.3 gingerblue/src/gingerblue-file.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

GingerblueData *gb_file_config_load (GingerblueData *head, gchar
    *filename);

static void gb_file_parse_volume (GingerblueData *data,
   xmlDocPtr doc, xmlNodePtr cur);
```

8.1.4 gingerblue/src/gingerblue.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

#ifndef _GINGERBLUE_H_
#define _GINGERBLUE_H_ 1

#include <gtk/gtk.h>
```

```
#define GINGERBLUE_STUDIO_PLAYER_TRUE 1
#define GINGERBLUE_STUDIO_PLAYER_FALSE 0

typedef struct _GingerblueData GingerblueData;

struct _GingerblueData {
    GtkWidget *knob;
    gint player_status;
    gchar *line;
    gint jack;
    gchar *label;
    gboolean lpf;
    gboolean hpf;
    gchar *musician;
    gchar *musical_instrument;
    gchar *version;
    gchar *volume;
    GingerblueData *next;
    GingerblueData *prev;
    GtkWidget *window;
    GMainLoop *player_loop;
};

void gb_window_break_record (GtkButton *record, GtkVolumeButton
    *volume);
void gb_window_pause_record (GtkButton *record, GtkVolumeButton
    *volume);
GingerblueData *gb_window_new_record (GtkButton *record,
    GtkVolumeButton *volume);
GingerblueData *gb_window_store_volume (GtkButton *record,
    GtkVolumeButton *volume);
gdouble gb_window_set_volume (GtkVolumeButton *volume, gdouble
    value);
gdouble gb_window_new_volume (GtkVolumeButton *volume, gchar *
    msg);
gdouble gb_window_get_volume (GtkVolumeButton *volume);

gint gb_exit (void);

#endif /* _GINGERBLUE_H_ */
```

8.1.5 gingerblue/src/gingerblue-knob.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
```

```
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

GtkWidget *knob (GingerblueData *data, GtkWidget *line, gint
    jack, gchar *label, gboolean lpf, gboolean hpf);
```

8.1.6 gingerblue/src/gingerblue-line.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

GtkWidget *line (gint jack, gchar *label);
```

8.1.7 gingerblue/src/gingerblue-main.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

GtkAssistantPageFunc gb_assistant_cb (GtkAssistant *assistant,
    GDateTime *datestamp);
```

8.1.8 gingerblue/src/gingerblue-main-loop.h

```
#ifndef GINGERBLUE_MAIN_LOOP_H
#define GINGERBLUE_MAIN_LOOP_H 1

GtkWidget *gingerblue_main_loop (GingerblueData *gingerblue);

#endif /* GINGERBLUE_MAIN_LOOP_H */
```

8.1.9 gingerblue/src/gingerblue-record.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

#include <string.h>
#include <gst/gst.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

static gboolean message_cb (GstBus * bus, GstMessage * message,
    gpointer user_data);
static GstPadProbeReturn unlink_cb(GstPad *pad, GstPadProbeInfo
    *info, gpointer user_data);
void stopRecording();
void startRecording();
int sigintHandler(int unused);
int gb_record_cb (gchar *path);

int gingerblue_record_begin();
int gingerblue_record_end();

typedef struct _GingerblueRecord {
  gboolean recording_found;
} GingerblueRecord;
```

8.1.10 gingerblue/src/gingerblue-song.h

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

GtkWidget *gb_song_new (gchar *title);
GtkWidget *gb_song_quit (gchar *title);
```

8.1.11 gingerblue/src/gingerblue-studio-config.h

```
#ifndef GINGERBLUE_STUDIO_CONFIG_H
#define GINGERBLUE_STUDIO_CONFIG_H 1

GtkWidget *main_studio_config (gchar *location_data, gchar *
    studio_city);

#endif /* GINGERBLUE_STUDIO_CONFIG_H */
```

8.1.12 gingerblue/src/gingerblue-studio-stream.h

```
#ifndef GINGERBLUE_STUDIO_STREAM_H
#define GINGERBLUE_STUDIO_STREAM_H 1

GtkWidget *main_studio_stream (gchar *location_data, gchar *
    studio_city);

#endif /* GINGERBLUE_STUDIO_STREAM_H */
```

8.1.13 gingerblue/src/gingerblue-app.c

```
/* $Id$

   Copyright (C) 2020-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

#include <glib/gi18n.h>
#include <gtk/gtk.h>
#include <gst/gst.h>
#include "gingerblue.h"
#include "gingerblue-config.h"
#include "gingerblue-main.h"
#include "gingerblue-main-loop.h"
#include "gingerblue-studio-config.h"

int main_app (gint argc, gchar *argv[]) {
    GingerblueData *gingerblue_config;
        GtkWindow *gingerblue_window;
    gtk_init (&argc, &argv);
        gingerblue_config = main_config (gingerblue_window, "
            studios.gingerblue.org");
    gingerblue_window = gingerblue_main_loop (gingerblue_config)
        ;
    gtk_widget_show_all (gingerblue_window);
    gst_init(&argc, &argc);
    gtk_main();
    return (0);
}
```

8.1.14 gingerblue/src/gingerblue.c

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
```

```
   Website: http://www.gingerblue.org/

 */

#include <glib/gstdio.h>
#include <glib/gi18n.h>
#include <gst/gst.h>
#include <gtk/gtk.h>
#include "gingerblue.h"
#include "gingerblue-file.h"

gint
gb_exit (void) {
    gst_deinit();
    gtk_main_quit();
}

void
gb_window_break_record (GtkButton *record, GtkVolumeButton *
   volume) {
    /* gtk_button_set_label(GTK_BUTTON (cue), "Continue
       Recording"); */
    /* g_signal_connect (GTK_BUTTON (cue), "clicked", G_CALLBACK
         (gb_window_new_record), gingerblue_data->volume); */
}

void
gb_window_pause_record (GtkButton *record, GtkVolumeButton *
   volume) {
    /* gtk_button_set_label(GTK_BUTTON (cue), "Continue
       Recording"); */
    /* g_signal_connect (GTK_BUTTON (cue), "clicked", G_CALLBACK
         (gb_window_new_record), gingerblue_data->volume); */
}

GingerblueData *
gb_window_new_record (GtkButton *record, GtkVolumeButton *volume
   ) {
    /* gtk_button_set_label(GTK_BUTTON (record), "Stop Recording
       "); */
}

GingerblueData *
gb_window_store_volume (GtkButton *record, GtkVolumeButton *
   volume) {
    /* gtk_button_set_label(GTK_BUTTON (record), "Stop Recording
       "); */
}

gdouble
gb_window_set_volume (GtkVolumeButton *volume, gdouble value) {
    gtk_scale_button_set_value (GTK_SCALE_BUTTON (volume), (
       gdouble) value);
}
```

```
gdouble
gb_window_new_volume (GtkVolumeButton *volume, gchar *msg) {
    g_print ("New volume: %0.2f\n", (gdouble)
        gtk_scale_button_get_value (GTK_SCALE_BUTTON (volume)));
    return (gdouble) gtk_scale_button_get_value (
        GTK_SCALE_BUTTON (volume));
}

gdouble
gb_window_get_volume (GtkVolumeButton *volume) {
    return (gdouble) gtk_scale_button_get_value (
        GTK_SCALE_BUTTON (volume));
}
```

8.1.15 gingerblue/src/gingerblue-config.c

```
/* $Id$

   Copyright (C) 2020-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

#include <config.h>
#include <glib/gi18n.h>
#include <gtk/gtk.h>
#include <gtk/gtkbox.h>
#include <gtk/gtkbutton.h>
#include <gtk/gtkcontainer.h>
#include <gtk/gtkwindow.h>

#include <gst/gst.h>
#include "gingerblue.h"
#include "gingerblue-main-loop.h"
#include "gingerblue-studio-config.h"
#include "gingerblue-studio-stream.h"
#include "gingerblue-studio-location.h"

extern GtkWidget *computer_entry;
extern GtkWidget *studio_entry;
extern GtkWidget *recording_entry;
extern GtkWidget *album_entry;
```

```
void studio_location_selected (GtkWidget *widget, gpointer *data
    )
{
        g_print ("Selected␣studios\n");
}

GtkWidget *main_config (GtkWidget *widget, gpointer *
    location_data) {
        GingerblueData *Gingerblue;
        GtkButton *AddStudioButton;
        GtkButton *NewStudioButton;
        GtkBox *Studio;
        GtkListBox *Location;
        GtkListBoxRow *Computer;
        GtkWidget *Studios;
        GtkWidget *StudioLabel;
        GtkContainer *Container;
        GtkWindow *gingerblue;
        gingerblue = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_window_set_title (GTK_WINDOW (gingerblue),
            g_strconcat(_("Recording␣("), gtk_entry_get_text(
            GTK_ENTRY(computer_entry)), _(")␣on␣"),
            gtk_entry_get_text(GTK_ENTRY(studio_entry)), _("␣("),
            PACKAGE_STRING, ")", NULL));
        AddStudioButton = gtk_button_new_with_label(_("Add␣Studio
            "));
        NewStudioButton = gtk_button_new_with_label(_("New␣Studio
            "));
        Studio = gtk_box_new (GTK_ORIENTATION_VERTICAL, 8);
        Location = gtk_list_box_new ();
        Computer = gtk_list_box_row_new();
        Studios = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 0);
        gtk_container_add (GTK_CONTAINER (Computer), Studios);
        StudioLabel = gtk_label_new (gtk_entry_get_text(GTK_ENTRY
            (computer_entry)));
        gtk_container_add (GTK_CONTAINER (gingerblue), GTK_WIDGET
             (Studio));
        gtk_container_add (GTK_CONTAINER (Location), Computer);
        gtk_box_pack_start (GTK_BOX (Studio), GTK_BUTTON (
            NewStudioButton), TRUE, TRUE, 0);
        gtk_box_pack_start (GTK_BOX (Studios), StudioLabel, TRUE,
             TRUE, 0);
        g_signal_connect (GTK_BUTTON(AddStudioButton), "clicked",
             G_CALLBACK(main_studio_config), gtk_entry_get_text(
            GTK_ENTRY(computer_entry)));
        gtk_box_pack_start (GTK_BOX (Studio), GTK_LIST_BOX (
            Location), TRUE, TRUE, 0);
        gtk_box_pack_start (GTK_BOX (Studio), GTK_BUTTON (
            AddStudioButton), TRUE, TRUE, 0);
        fprintf(stdout, "%s\n", gtk_entry_get_text(GTK_ENTRY(
            gtk_list_box_get_selected_row(GTK_LIST_BOX(Location)))
            ));
        g_signal_connect (GTK_LIST_BOX(Location), "row-selected",
             G_CALLBACK(studio_location_selected),
```

```
           gtk_list_box_get_selected_row (GTK_LIST_BOX(Location))
           );
       g_signal_connect (GTK_BUTTON(NewStudioButton), "clicked",
            G_CALLBACK(studio_location_selected),
           gtk_entry_get_text(GTK_ENTRY(computer_entry)));
       gtk_widget_show_all (GTK_WIDGET (gingerblue));
       return (GtkWidget *) gingerblue;
}
```

8.1.16 gingerblue/src/gingerblue-file.c

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

#include <gst/gst.h>
#include <gtk/gtk.h>
#include <glib/gstdio.h>
#include <glib/gi18n.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include "gingerblue.h"

GingerblueData *
gb_file_parse_volume (GingerblueData *data, xmlDocPtr doc,
   xmlNodePtr cur) {
        GingerblueData *gbdata = (GingerblueData *)data;
        xmlNodePtr sub;
    gbdata->version = (gchar *)xmlGetProp (cur, (const xmlChar
       *)"version");
    gbdata->volume = (gchar *)xmlGetProp (cur, (const xmlChar *)
       "volume");
    sub = cur->xmlChildrenNode;
    while (sub != NULL) {
        if ((!xmlStrcmp
             (sub->name, (const xmlChar *) "line"))) {
            gbdata->line = (gchar *) xmlNodeListGetString(doc,
               sub->xmlChildrenNode, 1);
        }
        if ((!xmlStrcmp
             (sub->name, (const xmlChar *) "musician"))) {
```

```
            gbdata->musician = (gchar *) xmlNodeListGetString(
               doc, sub->xmlChildrenNode, 1);
        }
        if ((!xmlStrcmp
             (sub->name, (const xmlChar *) "musical_instrument")
              )) {
            gbdata->musical_instrument = (gchar *)
               xmlNodeListGetString(doc, sub->xmlChildrenNode,
               1);
        }
        if ((!xmlStrcmp
             (sub->name, (const xmlChar *) "volume"))) {
            gbdata->volume = (gchar *) xmlNodeListGetString(doc,
                sub->xmlChildrenNode, 1);
        }
        sub = sub->next;
    }
    return (GingerblueData *)gbdata;
}

GingerblueData *
gb_file_config_load (GingerblueData *head, gchar *filename) {
    xmlDocPtr doc = NULL;
    xmlNodePtr cur = NULL;
    GingerblueData *curr = NULL;
    gchar *version;
    g_print ("%s\n", filename);
    g_return_val_if_fail (filename != NULL, NULL);
    doc = xmlReadFile (filename, NULL, 0);
    if (doc == NULL) {
        perror("xmlParseFile");
        xmlFreeDoc (doc);
        return NULL;
    }
    cur = xmlDocGetRootElement (doc);
    if (cur == NULL) {
            fprintf (stderr, _("Empty␣document\n"));
        xmlFreeDoc (doc);
        return NULL;
    }
    if (xmlStrcmp(cur->name, (const xmlChar *) "gingerblue")) {
            fprintf(stderr, _("Document␣of␣wrong␣type,␣root␣node
               ␣!=␣gingerblue\n"));
        xmlFreeDoc (doc);
        return NULL;
    }
    version = (gchar *) xmlGetProp (cur, (const xmlChar *)"
       version");
    g_print (_("Valid␣GNOME␣Gingerblue␣%s␣XML␣document␣%s\n"),
       version, filename);
    cur = cur->xmlChildrenNode;
    while (cur != NULL) {
      g_print (_("Parsing␣GNOME␣Gingerblue␣%s␣XML␣document␣%s\n"
         ), version, filename);
```

```
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "line"))) {
          g_print (_("Found␣Line\n"));
            curr = g_new0(GingerblueData, 1);
            curr->line = (gchar *) xmlNodeListGetString(doc, cur
                ->xmlChildrenNode, 1);
            g_print ("%s\n", curr->line);
            // curr = gb_file_parse_volume (curr, doc, cur);
            curr->next = head;
            head = curr;
            /* mem_volume = head */
            /* volumes = g_list_append (gingerblue_volumes, (
                GingerblueData *)mem_volume); */
            g_print ("Done␣with␣parsing␣Line\n");
        }
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "musician")
            )) {
            g_print (_("Found␣Musician\n"));
            curr = g_new0(GingerblueData, 1);
            curr->musician = (gchar *) xmlNodeListGetString(doc,
                 cur->xmlChildrenNode, 1);
            g_print ("%s\n", curr->musician);
            // curr = gb_file_parse_volume (curr, doc, cur);
            curr->next = head;
            head = curr;
            /* mem_volume = head */
            /* volumes = g_list_append (gingerblue_volumes, (
                GingerblueData *)mem_volume); */
            g_print (_("Done␣with␣parsing␣Musician\n"));
        }
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "
            musical_instrument"))) {
            g_print (_("Found␣Musical␣Instrument\n"));
            curr = g_new0(GingerblueData, 1);
            curr->musical_instrument = (gchar *)
                xmlNodeListGetString(doc, cur->xmlChildrenNode,
                1);
            g_print ("%s\n", curr->musical_instrument);
            // curr = gb_file_parse_volume (curr, doc, cur);
            curr->next = head;
            head = curr;
            /* mem_volume = head */
            /* volumes = g_list_append (gingerblue_volumes, (
                GingerblueData *)mem_volume); */
            g_print (_("Done␣with␣parsing␣Musical␣Instrument\n")
                );
        }
        if ((!xmlStrcmp(cur->name, (const xmlChar *) "volume")))
             {
            g_print (_("Found␣Volume\n"));
            curr = g_new0(GingerblueData, 1);
            curr->volume = (gchar *) xmlNodeListGetString(doc,
                cur->xmlChildrenNode, 1);
            g_print ("%s\n", curr->volume);
            // curr = gb_file_parse_volume (curr, doc, cur);
```

```
            curr->next = head;
            head = curr;
            /* mem_volume = head */
            /* volumes = g_list_append (gingerblue_volumes, (
                GingerblueData *)mem_volume); */
            g_print (_("Done with parsing Volume\n"));
        }
        cur = cur->next;
    }
    g_print (_("Finished parsing XML document\n"));
    xmlFreeDoc (doc);
    return curr;
}

/* int main (int argc, char **argv) */
/* { */
/*  GingerblueData *data = NULL; */
/*  data = gb_file_config_load (data, "gingerblue.xml"); */
/*  free (data); */
/*  return (0); */
/* } */
```

8.1.17 gingerblue/src/gingerblue-knob.c

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

#include <glib/gstdio.h>
#include <glib/gi18n.h>
#include <gst/gst.h>
#include <gtk/gtk.h>
#include "gingerblue.h"

GtkWidget *knob (GingerblueData *data, GtkWidget *line, gint
    jack, gchar *label, gboolean lpf, gboolean hpf) {
    GtkWidget *knob;
    knob = gtk_volume_button_new ();
    return (knob);
}
```

8.1.18 gingerblue/src/gingerblue-line.c

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

 #include <gst/gst.h>
 #include <gtk/gtk.h>
 #include <glib/gstdio.h>
 #include <glib/gi18n.h>

GtkWidget *line (gint jack, gchar *label) {
    GtkWidget *window;
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), label);
    return (window);
}
```

8.1.19 gingerblue/src/gingerblue-main.c

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.5.0 (2021-09-16)
   Website: http://www.gingerblue.org/

 */

#include <config.h>
#include <stdlib.h>
#include <glib/gi18n.h>
#include <gst/gst.h>
#include <gst/player/player.h>
#include <gst/tag/tag.h>
#include <gtk/gtk.h>
```

```
#include <glib/gstdio.h>
#include <glib/gi18n.h>
#include <champlain/champlain.h>
#include <champlain-gtk/champlain-gtk.h>
#include <string.h>
#include "gingerblue.h"
#include "gingerblue-chord.h"
#include "gingerblue-config.h"
#include "gingerblue-main.h"
#include "gingerblue-main-loop.h"
#include "gingerblue-record.h"
#include "gingerblue-studio-config.h"
#include "gingerblue-studio-location.h"
#include "gingerblue-studio-stream.h"

GingerblueData *Gingerblue;

static void gb_assistant_entry_changed(GtkEditable *,
   GtkAssistant *,
                       GstElement *);
static void gb_assistant_button_toggled(GtkCheckButton *,
   GtkAssistant *);
static void gb_assistant_button_clicked(GtkButton *,
   GtkAssistant *);
static void gb_assistant_cancel(GtkAssistant *, gpointer);
static void gb_assistant_close(GtkAssistant *, gpointer);
static void gb_assistant_apply(GtkAssistant *, gpointer);

typedef struct {
    GtkWidget *widget;
    gint index;
    const gchar *title;
    GtkAssistantPageType type;
    gboolean complete;
} PageInfo;

GtkWidget *musician_entry, *musician_label;
GtkWidget *song_entry, *song_label;
GtkWidget *instrument_entry, *instrument_label;
GtkWidget *label_entry, *label_label;
GtkWidget *line_entry, *line_label;
GtkWidget *computer_entry, *computer_label;
GtkWidget *recording_entry, *recording_label;
GtkWidget *studio_entry, *studio_label;
GtkWidget *stream_entry, *stream_label;
GtkWidget *album_entry, *album_label;
GtkWidget *summary_entry, *summary_label;

GMainLoop *main_loops;

GstPlayer *player;

GstTagList *tag_list;
```

```
GError *error = NULL;

static void gb_assistant_entry_changed(GtkEditable * editable,
                       GtkAssistant * assistant,
                       GstElement * pipeline)
{
    return;
}

static void gb_assistant_button_toggled(GtkCheckButton *
   checkbutton,
                     GtkAssistant * assistant)
{
    return;
}

static void gb_assistant_button_clicked(GtkButton * button,
                     GtkAssistant * assistant)
{
    GstElement *src, *conv, *enc, *muxer, *sink, *recorder;
    gchar *filename = NULL;
    GDateTime *datestamp = g_date_time_new_now_utc ();
    GstElementFactory *factory;

    gst_element_send_event(recorder, gst_event_new_eos());

    recorder = gst_pipeline_new("record_pipe");

    /*
      FIXME: Line #59 from https://github.com/GStreamer/gst-
         plugins-base/blob/master/tools/gst-device-monitor.c
      element = gst_device_create_element (device, NULL);
      if (!element)
      return NULL;
      factory = gst_element_get_factory (element);
      if (!factory) {
      gst_object_unref (element);
      return NULL;
      }
      src = gst_element_factory_create(factory, NULL);
    */
    src = gst_element_factory_make("autoaudiosrc", "auto_source"
       );
    conv = gst_element_factory_make("audioconvert", "convert");
    enc = gst_element_factory_make("vorbisenc", "vorbis_enc");
    muxer = gst_element_factory_make("oggmux", "oggmux");
    sink = gst_element_factory_make("filesink", "sink");
    filename = g_strconcat(g_get_user_special_dir(
       G_USER_DIRECTORY_MUSIC), "/",
                   gtk_entry_get_text(GTK_ENTRY(musician_entry))
                       , "_-_",
                   gtk_entry_get_text(GTK_ENTRY(song_entry)), "_
                       [",
                   g_date_time_format_iso8601 (datestamp),
```

```
                    "]",
                    ".ogg", NULL);
    g_object_set(G_OBJECT(sink), "location",
              g_strconcat(g_get_user_special_dir(
                 G_USER_DIRECTORY_MUSIC),
                  "/", gtk_entry_get_text(GTK_ENTRY(
                     musician_entry)), "_-_",
                  gtk_entry_get_text(GTK_ENTRY(song_entry)),
                  ".ogg", NULL), NULL);
    gst_bin_add_many(GST_BIN(recorder), src, conv, enc, muxer,
       sink, NULL);
    gst_element_link_many(src, conv, enc, muxer, sink, NULL);
    gst_element_set_state(recorder, GST_STATE_PLAYING);
    tag_list = gst_tag_list_new (GST_TAG_ARTIST,
       gtk_entry_get_text(GTK_ENTRY(musician_entry)), NULL);
    gst_stream_set_tags (GST_STREAM (recorder), tag_list);
    tag_list = gst_tag_list_new (GST_TAG_ALBUM,
       gtk_entry_get_text(GTK_ENTRY(album_entry)), NULL);
    gst_stream_set_tags (GST_STREAM (recorder), tag_list);
    tag_list = gst_tag_list_new (GST_TAG_TITLE,
       gtk_entry_get_text(GTK_ENTRY(song_entry)), NULL);
    gst_stream_set_tags (GST_STREAM (recorder), tag_list);
    tag_list = gst_tag_list_new (GST_TAG_COPYRIGHT,
       gtk_entry_get_text(GTK_ENTRY(label_entry)), NULL);
    gst_stream_set_tags (GST_STREAM (recorder), tag_list);
    tag_list = gst_tag_list_new (GST_TAG_PUBLISHER,
       gtk_entry_get_text(GTK_ENTRY(label_entry)), NULL);
    gst_stream_set_tags (GST_STREAM (recorder), tag_list);
    tag_list = gst_tag_list_new (GST_TAG_DATE_TIME, datestamp,
       NULL);
    gst_stream_set_tags (GST_STREAM (recorder), tag_list);
    gst_vorbis_tag_add (tag_list, GST_TAG_ARTIST,
       gtk_entry_get_text(GTK_ENTRY(musician_entry)));
    gst_vorbis_tag_add (tag_list, GST_TAG_ALBUM,
       gtk_entry_get_text(GTK_ENTRY(song_entry)));
    gst_vorbis_tag_add (tag_list, GST_TAG_TITLE,
       gtk_entry_get_text(GTK_ENTRY(song_entry)));
    gst_vorbis_tag_add (tag_list, GST_TAG_COPYRIGHT,
       gtk_entry_get_text(GTK_ENTRY(label_entry)));
    gst_vorbis_tag_add (tag_list, GST_TAG_PUBLISHER,
       gtk_entry_get_text(GTK_ENTRY(label_entry)));
    gst_vorbis_tag_add (tag_list, GST_TAG_DATE_TIME, datestamp);
    gst_vorbis_tag_add (tag_list, GST_TAG_DATE_TIME, datestamp);
    gst_stream_set_tags (GST_STREAM (recorder), tag_list);
    main_loops = g_main_loop_new(NULL, TRUE);
    g_main_loop_run(main_loops);
    gst_element_set_state(recorder, GST_STATE_NULL);
    g_main_loop_unref(main_loops);
    gst_object_unref(GST_OBJECT(recorder));
    g_date_time_unref (datestamp);
}

static void gb_assistant_cancel(GtkAssistant * assistant,
    gpointer data)
```

```
{
    if (!main_loops) {
        g_error("Quit␣more␣loops␣than␣there␣are.");
    } else {
        GMainLoop *loop = main_loops;
        g_main_loop_quit(loop);
        gtk_main_quit();
    }
}

static void gb_assistant_close(GtkAssistant * assistant,
   gpointer data)
{
    FILE *fp = NULL;
    GDateTime *datestamp = g_date_time_new_now_utc ();
    gchar *filename =
        g_strconcat(g_get_user_special_dir(
           G_USER_DIRECTORY_MUSIC), "/",
            gtk_entry_get_text(GTK_ENTRY(musician_entry)), "_-_"
               ,
            gtk_entry_get_text(GTK_ENTRY(song_entry)),  "_[",
            g_date_time_format_iso8601 (datestamp), "]",
            ".gingerblue", NULL);
    g_strconcat(g_get_user_special_dir(G_USER_DIRECTORY_MUSIC),
       "/",
            gtk_entry_get_text(GTK_ENTRY(musician_entry)), "_-_"
               ,
            gtk_entry_get_text(GTK_ENTRY(song_entry)),  "_[",
            g_date_time_format_iso8601 (datestamp), "]",
            ".ogg", NULL);
    fp = fopen(filename, "w");
    fprintf(fp, "<?xml␣version='1.0'␣encoding='UTF-8'?>\n");
    fprintf(fp, "<gingerblue␣version='%s'>\n", VERSION);
    fprintf(fp, "␣␣<musician>%s</musician>\n",
        gtk_entry_get_text(GTK_ENTRY(musician_entry)));
    fprintf(fp, "␣␣<song>%s</song>\n",
        gtk_entry_get_text(GTK_ENTRY(song_entry)));
    fprintf(fp, "␣␣<instrument>%s</instrument>\n",
        gtk_entry_get_text(GTK_ENTRY(instrument_entry)));
    fprintf(fp, "␣␣<line>%s</line>\n",
        gtk_entry_get_text(GTK_ENTRY(line_entry)));
    fprintf(fp, "␣␣<label>%s</label>\n",
        gtk_entry_get_text(GTK_ENTRY(label_entry)));
    fprintf(fp, "␣␣<station>%s</station>\n",
        gtk_entry_get_text(GTK_ENTRY(computer_entry)));
    fprintf(fp, "␣␣<filename>%s</filename>\n",
        gtk_entry_get_text(GTK_ENTRY(recording_entry)));
    fprintf(fp, "␣␣<album>%s</album>\n",
        gtk_entry_get_text(GTK_ENTRY(album_entry)));
    fprintf(fp, "␣␣<studio>%s</studio>\n",
        gtk_entry_get_text(GTK_ENTRY(studio_entry)));
    fprintf(fp, "</gingerblue>\n");
    fclose(fp);
    g_date_time_unref (datestamp);
```

```
    gst_element_send_event(data, gst_event_new_eos());
}

static void gb_assistant_apply(GtkAssistant * assistant,
   gpointer data)
{
        GingerblueData *gingerblue_config;
        GtkWindow *gingerblue_window;
        /* gtk_init (&argc, &argv); */
        gingerblue_config = main_config (GTK_WIDGET(
           gingerblue_window), gtk_entry_get_text(GTK_ENTRY(
           studio_entry)));
        gingerblue_window = gingerblue_main_loop (
           gingerblue_config);
        gtk_widget_show_all (gingerblue_window);
        /* gst_init(&argc, &argc); */
        /* gtk_main(); */
    gst_element_send_event(data, gst_event_new_eos());
}

GtkAssistantPageFunc gb_assistant_cb(GtkAssistant * assistant,
                     GDateTime * datestamp)
{
    /* gtk_assistant_next_page(assistant); */
}

int main(int argc, char **argv)
{
    GDateTime *datestamp;
    GingerblueData *data;
    GingerblueChord *gingerblue_chord;
    GstElement *src, *conv, *enc, *muxer, *sink, *pipeline;
    GtkWidget *introduction;
    GtkEntryBuffer *default_recording_title;
    GtkWidget *entry, *label, *button, *progress, *hbox;
    GtkWidget *summary_label, *summary_entry;
    GtkWidget *gingerblue_main;
    guint i;
    GtkWidget *musicianpage;
    GtkWidget *songpage;
    GtkWidget *instrumentpage;
    GtkWidget *recordpage;
    GtkWidget *window;
    GtkWidget *frame;
    GtkWidget *input;
    GtkWidget *main_window;
    GtkWidget *mixer;
    GtkWidget *control;
    GtkWidget *soundboard;
    GtkWidget *toolbar;
    GtkWidget *input_record;
    GtkWidget *input_pause;
    GtkWidget *input_break;
    GtkWidget *input_stop;
```

```
    GtkWidget *input_volume;
    gdouble input_volume_value;
    gint64 real_time;
    gchar *album;
    PageInfo page[11] = {
        {NULL, -1, "Gingerblue␣Setup", GTK_ASSISTANT_PAGE_INTRO,
             TRUE},
        {NULL, -1, "Musician", GTK_ASSISTANT_PAGE_CONTENT, TRUE
            },
        {NULL, -1, "Song", GTK_ASSISTANT_PAGE_CONTENT, TRUE},
        {NULL, -1, "Instrument", GTK_ASSISTANT_PAGE_CONTENT,
            TRUE},
        {NULL, -1, "Input␣Line", GTK_ASSISTANT_PAGE_CONTENT,
            TRUE},
        {NULL, -1, "Label", GTK_ASSISTANT_PAGE_CONTENT, TRUE},
        {NULL, -1, "Computer", GTK_ASSISTANT_PAGE_CONTENT, TRUE
            },
        {NULL, -1, "Recording", GTK_ASSISTANT_PAGE_CONTENT, TRUE
            },
        {NULL, -1, "Studio", GTK_ASSISTANT_PAGE_CONTENT, TRUE},
        {NULL, -1, "Album", GTK_ASSISTANT_PAGE_CONTENT, TRUE},
        {NULL, -1, "Broadcasting", GTK_ASSISTANT_PAGE_CONFIRM,
            TRUE},
    };
    datestamp = g_date_time_new_now_utc ();
    gchar *filename = g_strconcat(g_get_user_special_dir(
        G_USER_DIRECTORY_MUSIC), "/",
                    gtk_entry_get_text(GTK_ENTRY(
                        musician_entry)), "_-_",
                    gtk_entry_get_text(GTK_ENTRY(song_entry)),
                         "_[",
                    g_date_time_format_iso8601 (datestamp), "]
                        ",
                    ".ogg", NULL);
    gtk_init(&argc, &argv);
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    introduction = gtk_assistant_new();
    gtk_widget_set_size_request(GTK_WIDGET(introduction), 640,
        480);
    gtk_window_set_title(GTK_WINDOW(introduction), "GNOME␣
        Gingerblue");
    g_signal_connect(G_OBJECT(introduction), "destroy",
             G_CALLBACK(gtk_main_quit), NULL);
    page[0].widget = gtk_label_new(_("Welcome␣to␣GNOME␣
        Gingerblue!\n\nRecord␣respectfully␣around␣others.\n\
        nClick␣Next␣to␣setup␣a␣music␣recording␣session!\n\nClick␣
        Cancel␣to␣stop␣the␣music␣recording␣session.\n\nClick␣
        Cancel␣twice␣to␣exit␣GNOME␣Gingerblue."));
    page[1].widget = gtk_box_new(FALSE, 5);
    musician_label = gtk_label_new(_("Musician:"));
    musician_entry = gtk_entry_new();
    if (g_strcmp0(musician_entry, NULL)!=0) gtk_entry_set_text(
        GTK_ENTRY(musician_entry), g_get_real_name()); else
        gtk_entry_set_text(GTK_ENTRY(musician_entry),
```

```
    gtk_entry_get_text(GTK_ENTRY(musician_entry)));
gtk_box_pack_start(GTK_BOX(page[1].widget), GTK_WIDGET(
    musician_label),
            FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[1].widget), GTK_WIDGET(
    musician_entry),
            FALSE, FALSE, 5);
page[2].widget = gtk_box_new(FALSE, 5);
song_label = gtk_label_new(_("Song:"));
song_entry = gtk_entry_new();
if (g_strcmp0(song_entry, NULL)!=0) gtk_entry_set_text(
    GTK_ENTRY(song_entry), g_strconcat (_("Song_-_"),
    g_date_time_format_iso8601 (datestamp), NULL)); else
    gtk_entry_set_text(GTK_ENTRY(song_entry),
    gtk_entry_get_text(GTK_ENTRY(song_entry)));
gtk_box_pack_start(GTK_BOX(page[2].widget), GTK_WIDGET(
    song_label),
            FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[2].widget), GTK_WIDGET(
    song_entry),
            FALSE, FALSE, 5);
page[3].widget = gtk_box_new(FALSE, 5);
instrument_label = gtk_label_new(_("Instrument:"));
instrument_entry = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(instrument_entry), _("Guitar"))
    ;
gtk_box_pack_start(GTK_BOX(page[3].widget),
            GTK_WIDGET(instrument_label), FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[3].widget),
            GTK_WIDGET(instrument_entry), FALSE, FALSE, 5);
page[4].widget = gtk_box_new(FALSE, 5);
line_label = gtk_label_new(_("Line␣Input:"));
line_entry = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(line_entry), _("Mic"));
gtk_box_pack_start(GTK_BOX(page[4].widget), GTK_WIDGET(
    line_label),
            FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[4].widget), GTK_WIDGET(
    line_entry),
            FALSE, FALSE, 5);
page[5].widget = gtk_box_new(FALSE, 5);
label_label = gtk_label_new(_("Label:"));
label_entry = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(label_entry), _("GNOME"));
gtk_box_pack_start(GTK_BOX(page[5].widget), GTK_WIDGET(
    label_label),
            FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[5].widget), GTK_WIDGET(
    label_entry),
            FALSE, FALSE, 5);
page[6].widget = gtk_box_new(FALSE, 5);
computer_label = gtk_label_new(_("Computer:"));
computer_entry = gtk_entry_new();
```

```
gtk_entry_set_text(GTK_ENTRY(computer_entry), _(
    g_get_host_name()));
gtk_box_pack_start(GTK_BOX(page[6].widget), GTK_WIDGET(
    computer_label),
            FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[6].widget), GTK_WIDGET(
    computer_entry),
            FALSE, FALSE, 5);
recording_label = gtk_button_new_with_label("Recording");
recording_entry = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(recording_entry), g_strconcat(
    g_get_user_special_dir
                (G_USER_DIRECTORY_MUSIC), "/",
                gtk_entry_get_text(GTK_ENTRY(musician_entry
                    )), "_-_",
                gtk_entry_get_text(GTK_ENTRY(song_entry)),
                            ".ogg", NULL));
g_signal_connect(G_OBJECT(recording_label), "clicked",
         G_CALLBACK(gb_record_cb),
         g_strconcat(g_get_user_special_dir
                (G_USER_DIRECTORY_MUSIC), "/",
                gtk_entry_get_text(GTK_ENTRY(musician_entry
                    )), "_-_",
                gtk_entry_get_text(GTK_ENTRY(song_entry)),
                ".ogg", NULL));
page[7].widget = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(page[7].widget), g_strconcat(
    g_get_user_special_dir
                               (G_USER_DIRECTORY_MUSIC), "/"
                                  ,
                              gtk_entry_get_text(GTK_ENTRY(
                                 musician_entry)), "_-_",
                                 gtk_entry_get_text(
                                 GTK_ENTRY(song_entry)), ".
                                 ogg", NULL));
gtk_box_pack_start(GTK_BOX(page[7].widget), GTK_WIDGET(
    recording_label),
            FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[7].widget), GTK_WIDGET(
    recording_entry),
            FALSE, FALSE, 5);
studio_label = gtk_button_new_with_label("Broadcasting");
studio_entry = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(studio_entry), g_strconcat("
    http://", gtk_entry_get_text(GTK_ENTRY(computer_entry)),
    "/", NULL));
g_signal_connect(G_OBJECT(studio_label), "clicked",
         G_CALLBACK(gb_assistant_apply),
             gtk_entry_get_text(GTK_ENTRY(studio_entry)));
g_signal_connect(G_OBJECT(studio_entry), "clicked",
         G_CALLBACK(gb_assistant_apply),
             gtk_entry_get_text(GTK_ENTRY(studio_entry)));
page[8].widget = gtk_entry_new();
```

```
gtk_entry_set_text(GTK_ENTRY(page[8].widget),
   gtk_entry_get_text(GTK_ENTRY(studio_entry)));
gtk_box_pack_start(GTK_BOX(page[8].widget), GTK_WIDGET(
   studio_label),
           FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[8].widget), GTK_WIDGET(
   studio_entry),
           FALSE, FALSE, 5);
album_label = gtk_label_new("Album");
album_entry = gtk_entry_new();
g_signal_connect(G_OBJECT(album_label), "clicked",
         G_CALLBACK(gb_assistant_apply),
         gtk_entry_get_text(GTK_ENTRY(album_entry)));
album = g_strconcat(gtk_entry_get_text(GTK_ENTRY(
   studio_entry)),
             gtk_entry_get_text(GTK_ENTRY(label_entry)),
             ".pls", NULL);
gtk_entry_set_text(GTK_ENTRY(album_entry), (gchar *)album);
page[9].widget = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(page[9].widget), album);
g_signal_connect (GTK_BUTTON(album_entry), "clicked",
   G_CALLBACK(gb_assistant_apply), GTK_ENTRY(album_entry));
g_signal_connect (GTK_BOX(page[9].widget), "clicked",
   G_CALLBACK(gb_assistant_apply), GTK_ENTRY(album_entry));
g_signal_connect(G_OBJECT(album_label), "clicked",
         G_CALLBACK(gb_assistant_apply),
         album_entry);
gtk_box_pack_start(GTK_BOX(page[9].widget), GTK_WIDGET(
   album_label),
           FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[9].widget), GTK_WIDGET(
   album_entry),
           FALSE, FALSE, 5);
stream_label = gtk_button_new_with_label("Protocol");
stream_entry = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(stream_entry), "Torrent");
g_signal_connect(G_OBJECT(stream_entry), "clicked",
         G_CALLBACK(gb_assistant_apply),
             gtk_entry_get_text(GTK_ENTRY(stream_entry)));
page[10].widget = gtk_label_new(_("Click␣to␣launch␣HTTP␣
   streaming␣server␣client"));
gtk_entry_set_text(GTK_ENTRY(page[10].widget), "Click␣Apply"
   );
g_signal_connect (GTK_BUTTON(stream_entry), "clicked",
   G_CALLBACK(gb_assistant_apply), gtk_entry_get_text(
   GTK_ENTRY(studio_entry)));
g_signal_connect(G_OBJECT(stream_label), "clicked",
         G_CALLBACK(gb_assistant_apply),
             gtk_entry_get_text(GTK_ENTRY(stream_entry)));
gtk_box_pack_start(GTK_BOX(page[10].widget), GTK_WIDGET(
   stream_label),
           FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(page[10].widget), GTK_WIDGET(
   stream_entry),
```

```
                FALSE, FALSE, 5);
    for (i = 0; i < 11; i++) {
            page[i].index = gtk_assistant_append_page(
                GTK_ASSISTANT(introduction),
                            GTK_WIDGET(page[i].widget));
        gtk_assistant_set_page_title(GTK_ASSISTANT(introduction)
            ,
                           GTK_WIDGET(page[i].widget),
                           page[i].title);
        gtk_assistant_set_page_type(GTK_ASSISTANT(introduction),
                          GTK_WIDGET(page[i].widget),
                          page[i].type);
        gtk_assistant_set_page_complete(GTK_ASSISTANT(
            introduction),
                          GTK_WIDGET(page[i].widget),
                          page[i].complete);
    }
    g_signal_connect(G_OBJECT(entry), "changed",
             G_CALLBACK(gb_assistant_entry_changed), pipeline);
    g_signal_connect(G_OBJECT(introduction), "cancel",
             G_CALLBACK(gb_assistant_cancel), main_loops);
    g_signal_connect(G_OBJECT(introduction), "close",
             G_CALLBACK(gb_assistant_close), pipeline);
    g_signal_connect(G_OBJECT(introduction), "apply",
             G_CALLBACK(gb_assistant_close), pipeline);
/* musicianpage = gtk_entry_new (); */
    /* real_time = g_get_real_time(); */
    /* gtk_assistant_insert_page (introduction, */
    /*                            musicianpage, */
    /*                            0); */
    /* gtk_assistant_set_page_title (introduction, */
    /*                               musicianpage, */
    /*                               "Musician Setup"); */
    /* gtk_assistant_set_page_type (introduction, */
    /*                              musicianpage, */
    /*                              GTK_ASSISTANT_PAGE_INTRO); */
    /* songpage = gtk_entry_new (); */
    /* gtk_entry_set_text (songpage, g_strconcat(g_get_home_dir
        (), _("/Music/"), g_get_real_name(), " - Song.gingerblue
        ", NULL)); */
    /* real_time = g_get_real_time(); */
    /* gtk_assistant_insert_page (introduction, */
    /*                            songpage, */
    /*                            1); */
    /* gtk_assistant_set_page_title (introduction, */
    /*                               songpage, */
    /*                               "Song Setup"); */
    /* gtk_assistant_set_page_type (introduction, */
    /*                              songpage, */
    /*                              GTK_ASSISTANT_PAGE_CONTENT); */
    /* gtk_assistant_next_page(introduction); */
    /* instrumentpage = gtk_entry_new (); */
    /* gtk_entry_set_text (instrumentpage, "Guitar"); */
    /* gtk_assistant_set_page_type (introduction, */
```

```
/*                            instrumentpage, */
/*                            GTK_ASSISTANT_PAGE_CONTENT); */
/* gtk_assistant_insert_page (introduction, */
/*                          instrumentpage, */
/*                          2); */
/* gtk_assistant_set_page_title (introduction, */
/*                             instrumentpage, */
/*                             "Instrument Setup"); */
/* recordpage = gtk_entry_new (); */
/* gtk_entry_set_text (recordpage, "Microphone Line"); */
/* gtk_assistant_set_page_type (introduction, */
/*                            recordpage, */
/*                            GTK_ASSISTANT_PAGE_SUMMARY); */
/* gtk_assistant_insert_page (introduction, */
/*                          recordpage, */
/*                          3); */
/* gtk_assistant_set_page_title (introduction, */
/*                             recordpage, */
/*                             "Recording Setup"); */
/* gtk_assistant_set_page_complete (introduction, recordpage
   , 1); */
/* gtk_assistant_set_forward_page_func (introduction, */
/*                                      gb_assistant_cb, */
/*                                      NULL, */
/*                                      NULL); */
/* gtk_assistant_commit (introduction); */
gtk_widget_show_all(GTK_WIDGET(introduction));
/* FIXME Fix core dump
   main_window = gingerblue_main_loop (data);
   gtk_widget_show_all (main_window);
 */
gst_init(&argc, &argv);
gst_init(NULL, NULL);

pipeline = gst_pipeline_new("record_pipe");

src = gst_element_factory_make("autoaudiosrc", "auto_source"
   );
conv = gst_element_factory_make("audioconvert", "convert");
enc = gst_element_factory_make("vorbisenc", "vorbis_enc");
muxer = gst_element_factory_make("oggmux", "oggmux");
sink = gst_element_factory_make("filesink", "sink");
filename = g_strconcat(g_get_user_special_dir(
   G_USER_DIRECTORY_MUSIC), "/",
              gtk_entry_get_text(GTK_ENTRY(musician_entry))
                  , "_-_",
              gtk_entry_get_text(GTK_ENTRY(song_entry)),"_[
                  ",
              g_date_time_format_iso8601 (datestamp), "]",
              ".ogg", NULL);
g_object_set(G_OBJECT(sink), "location",
        g_strconcat(g_get_user_special_dir(
           G_USER_DIRECTORY_MUSIC),
```

```
                "/", gtk_entry_get_text(GTK_ENTRY(
                    musician_entry)), "_-_",
                gtk_entry_get_text(GTK_ENTRY(song_entry)),
                ".ogg", NULL), NULL);
    gst_bin_add_many(GST_BIN(pipeline), src, conv, enc, muxer,
        sink, NULL);
    gst_element_link_many(src, conv, enc, muxer, sink, NULL);

    gst_element_set_state(pipeline, GST_STATE_PLAYING);

    main_loops = g_main_loop_new(NULL, TRUE);
    g_main_loop_run(main_loops);

    gst_element_set_state(pipeline, GST_STATE_NULL);
    g_main_loop_unref(main_loops);
    gst_object_unref(GST_OBJECT(pipeline));

    /* player = play_new ("http://stream.radionorwegian.com/56.
        ogg", gingerblue_data->volume); */
    /* input_volume_value = gb_window_set_volume(
        GTK_VOLUME_BUTTON (input_volume), 0.00);   *\/ */
    /* g_signal_connect (GTK_BUTTON (input_record), "clicked",
        G_CALLBACK (gb_window_new_record), gingerblue_data->
        volume); */
    /* g_signal_connect (GTK_BUTTON (input_pause), "clicked",
        G_CALLBACK (gb_window_pause_record), gingerblue_data->
        volume); */
    /* g_signal_connect (GTK_BUTTON (input_break), "clicked",
        G_CALLBACK (gb_window_break_record), gingerblue_data->
        volume); */
    /* g_signal_connect (GTK_VOLUME_BUTTON (input_volume), "
        value-changed", G_CALLBACK (gb_window_pause_record),
        gingerblue_data->volume); */
    /* g_signal_connect (GTK_VOLUME_BUTTON (input_volume), "
        value-changed", G_CALLBACK (gb_window_store_volume),
        gingerblue_data->volume);   */
    g_signal_connect(GTK_WINDOW(introduction), "destroy",
             G_CALLBACK(gtk_main_quit), NULL);
    g_signal_connect(GTK_WINDOW(introduction), "destroy",
             G_CALLBACK(gtk_main_quit), NULL);

    /* g_free (gingerblue_data); */

    g_date_time_unref (datestamp);

    gtk_main();
    return (0);
}
```

8.1.20 gingerblue/src/gingerblue-main-loop.c

```
/* $Id$

   Copyright (C) 2020-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.5.0 (2021-09-16)
   Website: http://www.gingerblue.org/

 */

#include <gtk/gtk.h>
#include <gst/gst.h>
#include "gingerblue.h"
#include "gingerblue-studio-config.h"

extern GtkWidget *computer_entry;
extern GtkWidget *studio_entry;

GtkWidget *gingerblue_main_loop (GingerblueData *gingerblue) {
    GingerblueData *Gingerblue = gingerblue;
    Gingerblue->window = main_studio_config (gtk_entry_get_text(
        GTK_ENTRY(studio_entry)), gtk_entry_get_text(GTK_ENTRY(
        computer_entry)));
    gtk_window_set_title (Gingerblue->window, g_strconcat(
        gtk_entry_get_text(GTK_ENTRY(computer_entry)), "␣on␣",
        gtk_entry_get_text(GTK_ENTRY(studio_entry)), NULL));
         gtk_widget_show_all (Gingerblue->window);
}
```

8.1.21 gingerblue/src/gingerblue-record.c

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

*/

#include <string.h>
#include <gst/gst.h>
#include <signal.h>
```

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// v4l2src ! tee name=t t. ! x264enc ! mp4mux ! filesink
   location=/home/rish/Desktop/okay.264 t. ! videoconvert !
   autovideosink

static GMainLoop *loop;
static GstElement *pipeline, *audio_source, *sink, *src, *tee, *
   encoder, *muxer, *filesink, *videoconvert, *videosink, *
   queue_record, *queue_display;
static GstBus *bus;
static GstPad *teepad;
static gboolean recording = FALSE;
static gint counter = 0;
static char *file_path;

static gboolean
message_cb (GstBus * bus, GstMessage * message, gpointer
   user_data)
{
  switch (GST_MESSAGE_TYPE (message)) {
    case GST_MESSAGE_ERROR:{
      GError *err = NULL;
      gchar *name, *debug = NULL;

      name = gst_object_get_path_string (message->src);
      gst_message_parse_error (message, &err, &debug);

      g_printerr ("ERROR:␣from␣element␣%s:␣%s\n", name, err->
         message);
      if (debug != NULL)
        g_printerr ("Additional␣debug␣info:\n%s\n", debug);

      g_error_free (err);
      g_free (debug);
      g_free (name);

      g_main_loop_quit (loop);
      break;
    }
    case GST_MESSAGE_WARNING:{
        GError *err = NULL;
        gchar *name, *debug = NULL;

        name = gst_object_get_path_string (message->src);
        gst_message_parse_warning (message, &err, &debug);

        g_printerr ("ERROR:␣from␣element␣%s:␣%s\n", name, err->
           message);
        if (debug != NULL)
        g_printerr ("Additional␣debug␣info:\n%s\n", debug);
```

```
        g_error_free (err);
        g_free (debug);
        g_free (name);
        break;
    }
    case GST_MESSAGE_EOS:{
        g_print ("Got␣EOS\n");
        g_main_loop_quit (loop);
        gst_element_set_state (pipeline, GST_STATE_NULL);
        g_main_loop_unref (loop);
        gst_object_unref (pipeline);
        exit(0);
        break;
    }
    default:
        break;
  }

  return TRUE;
}

static GstPadProbeReturn unlink_cb(GstPad *pad, GstPadProbeInfo
    *info, gpointer user_data) {
    g_print("Unlinking...");
    GstPad *sinkpad;
    sinkpad = gst_element_get_static_pad (queue_record, "sink");
    gst_pad_unlink (teepad, sinkpad);
    gst_object_unref (sinkpad);

    gst_element_send_event(encoder, gst_event_new_eos());

    sleep(1);
    gst_bin_remove(GST_BIN (pipeline), queue_record);
    gst_bin_remove(GST_BIN (pipeline), encoder);
    gst_bin_remove(GST_BIN (pipeline), muxer);
    gst_bin_remove(GST_BIN (pipeline), filesink);

    gst_element_set_state(queue_record, GST_STATE_NULL);
    gst_element_set_state(encoder, GST_STATE_NULL);
    gst_element_set_state(muxer, GST_STATE_NULL);
    gst_element_set_state(filesink, GST_STATE_NULL);

    gst_object_unref(queue_record);
    gst_object_unref(encoder);
    gst_object_unref(muxer);
    gst_object_unref(filesink);

    gst_element_release_request_pad (tee, teepad);
    gst_object_unref (teepad);

    g_print("Unlinked\n");

    return GST_PAD_PROBE_REMOVE;
```

```
}

void stopRecording() {
    g_print("stopRecording\n");
    gst_pad_add_probe(teepad, GST_PAD_PROBE_TYPE_IDLE, unlink_cb
        , NULL, (GDestroyNotify) g_free);
    recording = FALSE;
}

void startRecording() {
    g_print("startRecording\n");
    GstPad *sinkpad;
    GstPadTemplate *templ;

    templ = gst_element_class_get_pad_template(
        GST_ELEMENT_GET_CLASS(tee), "src_%u");
    teepad = gst_element_request_pad(tee, templ, NULL, NULL);
    queue_record = gst_element_factory_make("queue", "
        queue_record");
    encoder = gst_element_factory_make("x264enc", NULL);
    muxer = gst_element_factory_make("mp4mux", NULL);
    filesink = gst_element_factory_make("filesink", NULL);
    char *file_name = (char*) malloc(255 * sizeof(char));
    sprintf(file_name, "%s%d.mp4", file_path, counter++);
    g_print("Recording␣to␣file␣%s", file_name);
    g_object_set(filesink, "location", file_name, NULL);
    g_object_set(encoder, "tune", 4, NULL);
    free(file_name);

    gst_bin_add_many(GST_BIN(pipeline), gst_object_ref(
        queue_record), gst_object_ref(encoder), gst_object_ref(
        muxer), gst_object_ref(filesink), NULL);
    gst_element_link_many(queue_record, encoder, muxer, filesink
        , NULL);

    gst_element_sync_state_with_parent(queue_record);
    gst_element_sync_state_with_parent(encoder);
    gst_element_sync_state_with_parent(muxer);
    gst_element_sync_state_with_parent(filesink);

    sinkpad = gst_element_get_static_pad(queue_record, "sink");
    gst_pad_link(teepad, sinkpad);
    gst_object_unref(sinkpad);

    recording = TRUE;
}

int sigintHandler(int unused) {
    g_print("You␣ctrl-c!\n");
    if (recording)
        stopRecording();
    else
        startRecording();
    return 0;
```

```
}

int gb_record_cb (char *path, gpointer data)
{
    return 0;
}
```

8.1.22 gingerblue/src/gingerblue-song.c

```
/* $Id$

   Copyright (C) 2018-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

 #include <gst/gst.h>
 #include <gtk/gtk.h>
 #include <glib/gstdio.h>
 #include <glib/gi18n.h>

GtkWidget *gb_song_new (gchar *title) {
    GtkWidget *window;
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), title);
    return (window);
}
GtkWidget *gb_song_quit (gchar *title) {
    GtkWidget *window;
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), title);
    return (window);
}
```

8.1.23 gingerblue/src/gingerblue-studio-config.c

```
/* $Id$
```

```
   Copyright (C) 2020-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.6.0 (2021-09-17)
   Website: http://www.gingerblue.org/

 */

#include <gtk/gtk.h>
#include <gst/gst.h>
#include "gingerblue.h"

GtkWidget *main_studio_config (gchar *location_data, gchar *
    studio_city) {
        GingerblueData *Gingerblue;
        GtkVBox *Locations;
        GtkListBox *Location;
        GtkContainer *Container;
        GtkWidget *Computer;
        GtkWidget *StudioLabel;
        Computer = gtk_list_box_row_new();
        StudioLabel = gtk_label_new (location_data);
        Locations = gtk_box_new (ATK_STATE_VERTICAL, 1);
        Location = gtk_list_box_new ();
        gtk_container_add (GTK_CONTAINER (Computer), Locations);
        gtk_box_pack_start (GTK_BOX (Location), StudioLabel, TRUE
            , TRUE, 0);
        gtk_container_add (GTK_CONTAINER (Location), GTK_LIST_BOX
             (Computer));
        gtk_container_add (GTK_CONTAINER (Container), GTK_BOX (
            Locations));
        gtk_container_add (GTK_CONTAINER (Container),
            GTK_LIST_BOX (Location));
        gtk_widget_show_all (GTK_WIDGET (Container));
        return (GtkWidget *) Gingerblue;
}
```

8.1.24 gingerblue/src/gingerblue-studio-stream.c

```
/* $Id$

   Copyright (C) 2020-2021 Aamot Software
   Author(s): Ole Aamot <ole@gnome.org>
   License: GNU GPL version 3
   Version: 1.5.0 (2021-09-16)
   Website: http://www.gingerblue.org/

 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/file.h>
#include <gtk/gtk.h>
#include <gst/gst.h>
#include <gobject/glib-types.h>
#include <gobject/gparam.h>
#include <shout/shout.h>
#include "gingerblue.h"

extern GtkWidget *recording_entry;
extern GtkWidget *studio_entry;
extern GtkWidget *musician_entry;
extern GtkWidget *song_entry;
extern GtkWidget *label_entry;

int main_studio_stream (gchar *location_data, gpointer *
    studio_city) {
     shout_t *shout;
     shout_metadata_t *pmetadata;
     unsigned char buff[4096];
     size_t read, total;
     int ret;
     shout_init();
     if (!(shout = shout_new())) {
         printf("Could␣not␣allocate␣shout_t\n");
         return 1;
     }
     fprintf(stdout, "STUDIO:␣%s\n", gtk_entry_get_text(GTK_ENTRY
         (studio_entry)));
     if (shout_set_host(shout, gtk_entry_get_text(GTK_ENTRY(
         studio_entry))) != SHOUTERR_SUCCESS) {
         printf("Error␣setting␣hostname:␣%s\n", shout_get_error(
             shout));
         return 1;
     }
     if (shout_set_protocol(shout, SHOUT_PROTOCOL_HTTP) !=
         SHOUTERR_SUCCESS) {
         printf("Error␣setting␣protocol:␣%s\n", shout_get_error(
             shout));
         return 1;
     }
     if (shout_set_port(shout, 8000) != SHOUTERR_SUCCESS) {
         printf("Error␣setting␣port:␣%s\n", shout_get_error(shout
             ));
         return 1;
     }
     if (shout_set_password(shout, "hackme") != SHOUTERR_SUCCESS)
          {
         printf("Error␣setting␣password:␣%s\n", shout_get_error(
             shout));
         return 1;
```

```
    }
    if (shout_set_mount(shout, "/stream") != SHOUTERR_SUCCESS) {
        printf("Error setting mount: %s\n", shout_get_error(
            shout));
        return 1;
    }
    if (shout_set_user(shout, "source") != SHOUTERR_SUCCESS) {
        printf("Error setting user: %s\n", shout_get_error(shout
            ));
        return 1;
    }
    if (shout_set_format(shout, SHOUT_FORMAT_OGG) !=
       SHOUTERR_SUCCESS) {
        printf("Error setting user: %s\n", shout_get_error(shout
            ));
        return 1;
    }
    if (shout_set_nonblocking(shout, 1) != SHOUTERR_SUCCESS) {
        printf("Error setting non-blocking mode: %s\n",
            shout_get_error(shout));
        return 1;
    }
    ret = shout_open(shout);
    if (ret != SHOUTERR_SUCCESS)
        ret = SHOUTERR_CONNECTED;
    if (ret != SHOUTERR_BUSY)
        printf("Connection pending...\n");
    while (ret != SHOUTERR_BUSY) {
        usleep(1000);
        ret = shout_get_connected(shout);
    }
    if (ret != SHOUTERR_CONNECTED) {
        printf("Connected to server...\n");
        total = 0;
        FILE *studio_stream_fp = fopen((char *)
            gtk_entry_get_text(GTK_ENTRY(recording_entry)), "r+")
            ;
        flock(studio_stream_fp, LOCK_SH);
        while (1) {
            g_print(stderr, "FILENAME %s\n", (char *)
                gtk_entry_get_text(GTK_ENTRY(recording_entry)));
            total = fseek((FILE *)studio_stream_fp, 0, SEEK_CUR)
                ;
            read = fread(buff, 1, sizeof(buff), studio_stream_fp
                );
            total = total + read;
            g_print(stderr, "%li of %li\n", read, total);
            if (read > 0) {
                    g_print(stderr, "%li\n", read);
                    ret = shout_send(shout, buff, read);
                 if (ret != SHOUTERR_SUCCESS) {
                        printf("DEBUG: Send error: %s\n",
                            shout_get_error(shout));
                    break;
```

```
                }
            } else {
                break;
            }
            if (shout_queuelen(shout) > 0)
                    printf("DEBUG: queue length: %d\n",
                       (int)shout_queuelen(shout));
            pmetadata = shout_metadata_new ();
            shout_metadata_add (pmetadata, "Artist",
               gtk_entry_get_text(GTK_ENTRY(musician_entry)));
            shout_metadata_add (pmetadata, "Song",
               gtk_entry_get_text(GTK_ENTRY(song_entry)));
            shout_metadata_add (pmetadata, "Copyright",
               gtk_entry_get_text(GTK_ENTRY(label_entry)));
            shout_set_metadata (shout, pmetadata);
            shout_sync(shout);
            shout_metadata_free (pmetadata);
        }
        fclose(studio_stream_fp);
    } else {
        printf("Error connecting: %s\n", shout_get_error(shout))
           ;
    }
    shout_close(shout);
    shout_shutdown();
    return 0;
}
```

Chapter 9

Specification

Chapter 10

Multiple-Location Audio Recording 1.0

Part III

Conclusion

Chapter 11

Results

References

[1] Boney, L., Tewfik, A.H., and Hamdy, K.N., “Digital Watermarks for Audio Signals," *Proceedings of the Third IEEE International Conference on Multimedia*, pp. 473-480, June 1996.

[2] British Intelligence Objectives Subcommittee, p. 61.

[3] Goossens, M., Mittelbach, F., Samarin, *A LaTeX Companion*, Addison-Wesley, Reading, MA, 1994.

[4] Kopka, H., Daly P.W., *A Guide to LaTeX*, Addison-Wesley, Reading, MA, 1999.

[5] Nmungwun, A. F., *Video Recording Technology*, *Lawrence Erlbaum Associates, Publishers*, pp. 8-24, 1989.

[6] Pan, D., *A Tutorial on MPEG/Audio Compression*, *IEEE Multimedia*, Vol.2, pp.60-74, Summer 1998.

[7] Pulkki, V., Delikaris-Manias, S., Politis, A., “Parametric Time-Frequency Domain Spatial Audio”, *IEEE Press*, pp. 3-4

[8] Cox, G., “Pioneering Television News”, *John Libbey*