GarageJam

2.0.0

Generated by Doxygen 1.9.7

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 _GingerblueChord Struct Reference

Public Attributes

- char * root
- char * file
- char e1
- char b2
- char g3
- char d4
- char a5
- char e6

3.1.1 Detailed Description

Definition at line 16 of file garagejam-gingerblue-chord.h.

3.1.2 Member Data Documentation

3.1.2.1 a5

```
char _GingerblueChord::a5
```

Definition at line 24 of file garagejam-gingerblue-chord.h.

3.1.2.2 b2

```
char _GingerblueChord::b2
```

Definition at line 21 of file garagejam-gingerblue-chord.h.

3.1.2.3 d4

```
char _GingerblueChord::d4
```

Definition at line 23 of file garagejam-gingerblue-chord.h.

3.1.2.4 e1

```
char _GingerblueChord::e1
```

Definition at line 20 of file garagejam-gingerblue-chord.h.

3.1.2.5 e6

```
char _GingerblueChord::e6
```

Definition at line 25 of file garagejam-gingerblue-chord.h.

3.1.2.6 file

```
char* _GingerblueChord::file
```

Definition at line 18 of file garagejam-gingerblue-chord.h.

3.1.2.7 g3

```
char _GingerblueChord::g3
```

Definition at line 22 of file garagejam-gingerblue-chord.h.

3.1.2.8 root

```
char* _GingerblueChord::root
```

Definition at line 17 of file garagejam-gingerblue-chord.h.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-chord.h

3.2 _GingerblueContainer Struct Reference

Public Member Functions

- void * **load_container** ()
- void * **save_container** ()
- void * **show_container** ()
- void * **hide_container** ()

Public Attributes

- gboolean container_active
- GtkWidget * widget

3.2.1 Detailed Description

Definition at line 3 of file garagejam-gingerblue-container.h.

3.2.2 Member Data Documentation

3.2.2.1 container_active

```
gboolean _GingerblueContainer::container_active
```

Definition at line 4 of file garagejam-gingerblue-container.h.

3.2.2.2 widget

```
GtkWidget* _GingerblueContainer::widget
```

Definition at line 9 of file garagejam-gingerblue-container.h.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-container.h

3.3 _GingerblueData Struct Reference

Public Attributes

- 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

3.3.1 Detailed Description

Definition at line 21 of file garagejam.h.

3.3.2 Member Data Documentation

3.3.2.1 hpf

```
gboolean _GingerblueData::hpf
```

Definition at line 28 of file garagejam.h.

3.3.2.2 jack

```
gint _GingerblueData::jack
```

Definition at line 25 of file garagejam.h.

3.3.2.3 knob

```
GtkWidget* _GingerblueData::knob
```

Definition at line 22 of file garagejam.h.

3.3.2.4 label

```
gchar* _GingerblueData::label
```

Definition at line 26 of file garagejam.h.

3.3.2.5 line

```
gchar* _GingerblueData::line
```

Definition at line 24 of file garagejam.h.

3.3.2.6 lpf

```
gboolean _GingerblueData::lpf
```

Definition at line 27 of file garagejam.h.

3.3.2.7 musical_instrument

```
gchar* _GingerblueData::musical_instrument
```

Definition at line 30 of file garagejam.h.

3.3.2.8 musician

```
gchar* _GingerblueData::musician
```

Definition at line 29 of file garagejam.h.

3.3.2.9 next

```
GingerblueData* _GingerblueData::next
```

Definition at line 33 of file garagejam.h.

3.3.2.10 player_loop

```
GMainLoop* _GingerblueData::player_loop
```

Definition at line 36 of file garagejam.h.

3.3.2.11 player_status

```
gint _GingerblueData::player_status
```

Definition at line 23 of file garagejam.h.

3.3.2.12 prev

```
GingerblueData* _GingerblueData::prev
```

Definition at line 34 of file garagejam.h.

3.3.2.13 version

```
gchar* _GingerblueData::version
```

Definition at line 31 of file garagejam.h.

3.3.2.14 volume

```
gchar* _GingerblueData::volume
```

Definition at line 32 of file garagejam.h.

3.3.2.15 window

```
GtkWidget* _GingerblueData::window
```

Definition at line 35 of file garagejam.h.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam.h

3.4 _GingerblueRecord Struct Reference

Public Attributes

- gboolean recording_found

3.4.1 Detailed Description

Definition at line 29 of file garagejam-gingerblue-record.h.

3.4.2 Member Data Documentation

3.4.2.1 recording_found

```
gboolean _GingerblueRecord::recording_found
```

Definition at line 30 of file garagejam-gingerblue-record.h.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-record.h

3.5 _GingerblueStorage Struct Reference

Public Attributes

- gboolean storage_allocated
- gboolean storagemap

3.5.1 Detailed Description

Definition at line 3 of file garagejam-gingerblue-storage.h.

3.5.2 Member Data Documentation

3.5.2.1 storage_allocated

```
gboolean _GingerblueStorage::storage_allocated
```

Definition at line 4 of file garagejam-gingerblue-storage.h.

3.5.2.2 storagemap

```
gboolean _GingerblueStorage::storagemap
```

Definition at line 5 of file garagejam-gingerblue-storage.h.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-storage.h

3.6 _GingerblueWizard Struct Reference

Public Attributes

- gboolean wizard_run
- GtkWindow * window
- GingerblueStorage * storage
- GtkFile * metadata
- char * stream

3.6.1 Detailed Description

Definition at line 7 of file garagejam-gingerblue-wizard.h.

3.6.2 Member Data Documentation

3.6.2.1 metadata

```
GtkFile* _GingerblueWizard::metadata
```

Definition at line 11 of file garagejam-gingerblue-wizard.h.

3.6.2.2 storage

```
GingerblueStorage* _GingerblueWizard::storage
```

Definition at line 10 of file garagejam-gingerblue-wizard.h.

3.6.2.3 stream

```
char* _GingerblueWizard::stream
```

Definition at line 12 of file garagejam-gingerblue-wizard.h.

3.6.2.4 window

```
GtkWindow* _GingerblueWizard::window
```

Definition at line 9 of file garagejam-gingerblue-wizard.h.

3.6.2.5 wizard_run

```
gboolean _GingerblueWizard::wizard_run
```

Definition at line 8 of file garagejam-gingerblue-wizard.h.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-wizard.h

3.7 GstPlay Struct Reference

Public Attributes

- gchar ** uris
- guint num_uris
- gint cur_idx
- GstPlayer * player
- GstState desired_state
- gboolean repeat
- GMainLoop * loop

3.7.1 Detailed Description

Definition at line 41 of file garagejam-gingerblue-studio-player.h.

3.7.2 Member Data Documentation

3.7.2.1 cur_idx

```
gint GstPlay::cur_idx
```

Definition at line 45 of file garagejam-gingerblue-studio-player.h.

3.7.2.2 desired_state

```
GstState GstPlay::desired_state
```

Definition at line 48 of file garagejam-gingerblue-studio-player.h.

3.7.2.3 loop

```
GMainLoop* GstPlay::loop
```

Definition at line 52 of file garagejam-gingerblue-studio-player.h.

3.7.2.4 num_uris

```
guint GstPlay::num_uris
```

Definition at line 44 of file garagejam-gingerblue-studio-player.h.

3.7.2.5 player

```
GstPlayer* GstPlay::player
```

Definition at line 47 of file garagejam-gingerblue-studio-player.h.

3.7.2.6 repeat

```
gboolean GstPlay::repeat
```

Definition at line 50 of file garagejam-gingerblue-studio-player.h.

3.7.2.7 uris

```
gchar** GstPlay::uris
```

Definition at line 43 of file garagejam-gingerblue-studio-player.h.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-studio-player.h

3.8 PageInfo Struct Reference

Public Attributes

- GtkWidget * widget
- gint index
- const gchar * title
- GtkAssistantPageType type
- gboolean complete

3.8.1 Detailed Description

Definition at line 50 of file garagejam-gingerblue-main.c.

3.8.2 Member Data Documentation

3.8.2.1 complete

```
gboolean PageInfo::complete
```

Definition at line 55 of file garagejam-gingerblue-main.c.

3.8.2.2 index

```
gint PageInfo::index
```

Definition at line 52 of file garagejam-gingerblue-main.c.

3.8.2.3 title

```
const gchar* PageInfo::title
```

Definition at line 53 of file garagejam-gingerblue-main.c.

3.8.2.4 type

```
GtkAssistantPageType PageInfo::type
```

Definition at line 54 of file garagejam-gingerblue-main.c.

3.8.2.5 widget

```
GtkWidget* PageInfo::widget
```

Definition at line 51 of file garagejam-gingerblue-main.c.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-main.c

3.9 PlaylistEntry Struct Reference

Public Attributes

- gchar * title
- gchar * location

3.9.1 Detailed Description

Definition at line 58 of file garagejam-gingerblue-main.c.

3.9.2 Member Data Documentation

3.9.2.1 location

```
gchar* PlaylistEntry::location
```

Definition at line 60 of file garagejam-gingerblue-main.c.

3.9.2.2 title

```
gchar* PlaylistEntry::title
```

Definition at line 59 of file garagejam-gingerblue-main.c.

The documentation for this struct was generated from the following file:

- src/garagejam-2.0.0/src/garagejam-gingerblue-main.c

Chapter 4

File Documentation

4.1 garagejam-gingerblue-app.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2023  Aamot Broadcast
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 0.0.1 (2023-04-30)
00007    Website: http://www.gingerblue.org/
00008
00009  */
00010
00011 #include <glib/gi18n.h>
00012 #include <gtk/gtk.h>
00013 #include <gst/gst.h>
00014 #include "garagejam.h"
00015 #include "garagejam-gingerblue-config.h"
00016 #include "garagejam-gingerblue-main.h"
00017 #include "garagejam-gingerblue-main-loop.h"
00018 #include "garagejam-gingerblue-studio-config.h"
00019
00020 int main_app(gint argc, gchar **argv)
00021 {
00022     GingerblueData *garagejam_config;
00023     GtkWindow *garagejam_window;
00024     gtk_init(&argc, &argv);
00025     garagejam_config =
00026         main_config(garagejam_window, "studios.gingerblue.org");
00027     garagejam_window = garagejam_main_loop(garagejam_config);
00028     gtk_widget_show_all(garagejam_window);
00029     gst_init(&argc, &argc);
00030     gtk_main();
00031     return (0);
00032 }
```

4.2 garagejam-gingerblue-app.h

```
00001 #ifndef GINGERBLUE_APP_H
00002 #define GINGERBLUE_APP_H 1
00003
00004
00005 #endif /* GINGERBLUE_APP_H */
```

4.3 garagejam-gingerblue-chord.h

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
```

```
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #ifndef _GINGERBLUE_CHORD_H_
00012 #define _GINGERBLUE_CHORD_H_ 1
00013
00014 typedef struct _GingerblueChord GingerblueChord;
00015
00016 struct _GingerblueChord {
00017     char *root;
00018     char *file;
00019     /* struct Display *gui; */
00020     char e1;
00021     char b2;
00022     char g3;
00023     char d4;
00024     char a5;
00025     char e6;
00026 };
00027
00028 struct _GingerblueChord gbc[] = {
00029   {"C",     "Gingerblue_Guitar_C.wav",  /* &console, */ '0', '1', '0', '2', '3', '0'},
00030   {"C#",    "Gingerblue_Guitar_C#.wav", /* &console, */ '1', '2', '1', '3', '4', '0'},
00031   {"Db",    "Gingerblue_Guitar_Db.wav", /* &console, */ '1', '2', '1', '3', '4', '0'},
00032   {"D",     "Gingerblue_Guitar_D.wav",  /* &console, */ '1', '2', '1', '0', '0', '0'},
00033   {"D#",    "Gingerblue_Guitar_D#.wav", /* &console, */ '3', '4', '2', '1', '0', '0'},
00034   {"Eb",    "Gingerblue_Guitar_Eb.wav", /* &console, */ '3', '4', '2', '1', '0', '0'},
00035   {"E",     "Gingerblue_Guitar_E.wav",  /* &console, */ '0', '0', '1', '2', '2', '0'},
00036   {"F",     "Gingerblue_Guitar_F.wav",  /* &console, */ '1', '1', '2', '3', '3', '1'},
00037   {"F#",    "Gingerblue_Guitar_F#.wav", /* &console, */ '2', '2', '3', '4', '4', '1'},
00038   {"Gb",    "Gingerblue_Guitar_Gb.wav", /* &console, */ '2', '2', '3', '4', '4', '1'},
00039   {"G",     "Gingerblue_Guitar_G.wav",  /* &console, */ '3', '0', '0', '0', '2', '3'},
00040   {"G#",    "Gingerblue_Guitar_G#.wav", /* &console, */ '0', '1', '1', '1', '3', '4'},
00041   {"Ab",    "Gingerblue_Guitar_Ab.wav", /* &console, */ '0', '1', '1', '1', '3', '4'},
00042   {"A",     "Gingerblue_Guitar_A.wav",  /* &console, */ '0', '2', '2', '2', '0', '0'},
00043   {"A#",    "Gingerblue_Guitar_A#.wav", /* &console, */ '1', '3', '3', '3', '1', '0'},
00044   {"Bb",    "Gingerblue_Guitar_Bb.wav", /* &console, */ '1', '3', '3', '3', '1', '-'},
00045   {"B",     "Gingerblue_Guitar_B.wav",  /* &console, */ '2', '4', '4', '4', '2', '0'},
00046   {"Bm",    "Gingerblue_Guitar_Bm.wav", /* &console, */ '2', '3', '4', '4', '2', '-'},
00047   {NULL,NULL}
00048 };
00049
00050 #endif /* _GINGERBLUE_CHORD_H_ */
```

4.4 garagejam-gingerblue-config.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <config.h>
00012 #include <glib/gi18n.h>
00013 #include <gtk/gtk.h>
00014 #include <gtk/gtkbox.h>
00015 #include <gtk/gtkbutton.h>
00016 #include <gtk/gtkcontainer.h>
00017 #include <gtk/gtkwindow.h>
00018
00019 #include <gst/gst.h>
00020 #include "garagejam.h"
00021 #include "garagejam-gingerblue-main-loop.h"
00022 #include "garagejam-gingerblue-studio-config.h"
00023 #include "garagejam-gingerblue-studio-stream.h"
00024 #include "garagejam-gingerblue-studio-location.h"
00025
00026 extern GtkWidget *computer_entry;
00027 extern GtkWidget *studio_entry;
00028 extern GtkWidget *recording_entry;
00029 extern GtkWidget *album_entry;
00030
00031 void studio_location_selected(GtkWidget *widget, gpointer *data)
00032 {
00033     g_print("Selected studios\n");
00034 }
00035
00036 GtkWidget *main_config(GtkWidget *widget, gpointer *location_data)
```

```
00037 {
00038     GingerblueData *Gingerblue;
00039     GtkButton *AddStudioButton;
00040     GtkButton *NewStudioButton;
00041     GtkBox *Studio;
00042     GtkListBox *Location;
00043     GtkListBoxRow *Computer;
00044     GtkWidget *Studios;
00045     GtkWidget *StudioLabel;
00046     GtkContainer *Container;
00047     GtkWindow *garagejam;
00048     garagejam = gtk_window_new(GTK_WINDOW_TOPLEVEL);
00049     gtk_window_set_title(GTK_WINDOW(garagejam),
00050                  g_strconcat(_("Recording ("),
00051                      gtk_entry_get_text(GTK_ENTRY
00052                                (computer_entry)),
00053                      _(") on "),
00054                      gtk_entry_get_text(GTK_ENTRY
00055                                (studio_entry)),
00056                      _(" ("), PACKAGE_STRING, ")",
00057                      NULL));
00058     AddStudioButton = gtk_button_new_with_label(_("Add Studio"));
00059     NewStudioButton = gtk_button_new_with_label(_("New Studio"));
00060     Studio = gtk_box_new(GTK_ORIENTATION_VERTICAL, 8);
00061     Location = gtk_list_box_new();
00062     Computer = gtk_list_box_row_new();
00063     Studios = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 0);
00064     gtk_container_add(GTK_CONTAINER(Computer), Studios);
00065     StudioLabel =
00066         gtk_label_new(gtk_entry_get_text(GTK_ENTRY(computer_entry)));
00067     gtk_container_add(GTK_CONTAINER(garagejam), GTK_WIDGET(Studio));
00068     gtk_container_add(GTK_CONTAINER(Location), Computer);
00069     gtk_box_pack_start(GTK_BOX(Studio), GTK_BUTTON(NewStudioButton),
00070                TRUE, TRUE, 0);
00071     gtk_box_pack_start(GTK_BOX(Studios), StudioLabel, TRUE, TRUE, 0);
00072     g_signal_connect(GTK_BUTTON(AddStudioButton), "clicked",
00073              G_CALLBACK(main_studio_config),
00074              gtk_entry_get_text(GTK_ENTRY(computer_entry)));
00075     gtk_box_pack_start(GTK_BOX(Studio), GTK_LIST_BOX(Location), TRUE,
00076                TRUE, 0);
00077     gtk_box_pack_start(GTK_BOX(Studio), GTK_BUTTON(AddStudioButton),
00078                TRUE, TRUE, 0);
00079     fprintf(stdout, "%s\n",
00080         gtk_entry_get_text(GTK_ENTRY
00081                    (gtk_list_box_get_selected_row
00082                     (GTK_LIST_BOX(Location)))));
00083     g_signal_connect(GTK_LIST_BOX(Location), "row-selected",
00084              G_CALLBACK(studio_location_selected),
00085              gtk_list_box_get_selected_row(GTK_LIST_BOX
00086                                (Location)));
00087     g_signal_connect(GTK_BUTTON(NewStudioButton), "clicked",
00088              G_CALLBACK(studio_location_selected),
00089              gtk_entry_get_text(GTK_ENTRY(computer_entry)));
00090     gtk_widget_show_all(GTK_WIDGET(garagejam));
00091     return (GtkWidget *) garagejam;
00092 }
```

4.5 garagejam-gingerblue-config.h

```
00001 #ifndef GINGERBLUE_CONFIG_H
00002 #define GINGERBLUE_CONFIG_H 1
00003
00004 GtkWidget *main_config (GtkWidget *widget, gpointer *location_data);
00005
00006 #endif /* GINGERBLUE_CONFIG_H */
```

4.6 garagejam-gingerblue-container.h

```
00001 #define GINGERBLUE_CONTAINER 4000
00002
00003 typedef struct _GingerblueContainer {
00004   gboolean container_active;
00005   (void *) load_container();
00006   (void *) save_container();
00007   (void *) show_container();
00008   (void *) hide_container();
00009   GtkWidget *widget;
00010 } GingerblueContainer;
```

4.7 garagejam-gingerblue-file.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2023  Aamot Broadcast
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 0.0.1 (2023-04-30)
00007    Website: http://www.gingerblue.org/
00008
00009 */
00010
00011 #include <gst/gst.h>
00012 #include <gtk/gtk.h>
00013 #include <glib/gstdio.h>
00014 #include <glib/gi18n.h>
00015 #include <libxml/xmlmemory.h>
00016 #include <libxml/parser.h>
00017 #include "garagejam.h"
00018
00019 GingerblueData *gb_file_parse_volume(GingerblueData *data, xmlDocPtr doc,
00020                      xmlNodePtr cur)
00021 {
00022     GingerblueData *gbdata = (GingerblueData *) data;
00023     xmlNodePtr sub;
00024     gbdata->version =
00025         (gchar *) xmlGetProp(cur, (const xmlChar *) "version");
00026     gbdata->volume =
00027         (gchar *) xmlGetProp(cur, (const xmlChar *) "volume");
00028     sub = cur->xmlChildrenNode;
00029     while (sub != NULL) {
00030         if ((!xmlStrcmp(sub->name, (const xmlChar *) "line"))) {
00031             gbdata->line =
00032                 (gchar *) xmlNodeListGetString(doc,
00033                                sub->
00034                                xmlChildrenNode,
00035                                1);
00036         }
00037         if ((!xmlStrcmp(sub->name, (const xmlChar *) "musician"))) {
00038             gbdata->musician =
00039                 (gchar *) xmlNodeListGetString(doc,
00040                                sub->
00041                                xmlChildrenNode,
00042                                1);
00043         }
00044         if ((!xmlStrcmp
00045              (sub->name,
00046               (const xmlChar *) "musical_instrument"))) {
00047             gbdata->musical_instrument =
00048                 (gchar *) xmlNodeListGetString(doc,
00049                                sub->
00050                                xmlChildrenNode,
00051                                1);
00052         }
00053         if ((!xmlStrcmp(sub->name, (const xmlChar *) "volume"))) {
00054             gbdata->volume =
00055                 (gchar *) xmlNodeListGetString(doc,
00056                                sub->
00057                                xmlChildrenNode,
00058                                1);
00059         }
00060         sub = sub->next;
00061     }
00062     return (GingerblueData *) gbdata;
00063 }
00064
00065 GingerblueData *gb_file_config_load(GingerblueData *head, gchar *filename)
00066 {
00067     xmlDocPtr doc = NULL;
00068     xmlNodePtr cur = NULL;
00069     GingerblueData *curr = NULL;
00070     gchar *version;
00071     g_print("%s\n", filename);
00072     g_return_val_if_fail(filename != NULL, NULL);
00073     doc = xmlReadFile(filename, NULL, 0);
00074     if (doc == NULL) {
00075         perror("xmlParseFile");
00076         xmlFreeDoc(doc);
00077         return NULL;
00078     }
00079     cur = xmlDocGetRootElement(doc);
00080     if (cur == NULL) {
00081         fprintf(stderr, _("Empty document\n"));
00082         xmlFreeDoc(doc);
00083         return NULL;
00084     }
00085     if (xmlStrcmp(cur->name, (const xmlChar *) "garagejam")) {
```

```
00086         fprintf(stderr,
00087                 _
00088                 ("Document of wrong type, root node != garagejam\n"));
00089         xmlFreeDoc(doc);
00090         return NULL;
00091     }
00092     version = (gchar *) xmlGetProp(cur, (const xmlChar *) "version");
00093     g_print(_("Valid GNOME Gingerblue %s XML document %s\n"), version,
00094         filename);
00095     cur = cur->xmlChildrenNode;
00096     while (cur != NULL) {
00097         g_print(_("Parsing GNOME Gingerblue %s XML document %s\n"),
00098             version, filename);
00099         if ((!xmlStrcmp(cur->name, (const xmlChar *) "line"))) {
00100             g_print(_("Found Line\n"));
00101             curr = g_new0(GingerblueData, 1);
00102             curr->line =
00103                 (gchar *) xmlNodeListGetString(doc,
00104                                 cur->
00105                                 xmlChildrenNode,
00106                                 1);
00107             g_print("%s\n", curr->line);
00108             // curr = gb_file_parse_volume (curr, doc, cur);
00109             curr->next = head;
00110             head = curr;
00111             /* mem_volume = head */
00112             /* volumes = g_list_append (garagejam_volumes, (GingerblueData *)mem_volume); */
00113             g_print("Done with parsing Line\n");
00114         }
00115         if ((!xmlStrcmp(cur->name, (const xmlChar *) "musician"))) {
00116             g_print(_("Found Musician\n"));
00117             curr = g_new0(GingerblueData, 1);
00118             curr->musician =
00119                 (gchar *) xmlNodeListGetString(doc,
00120                                 cur->
00121                                 xmlChildrenNode,
00122                                 1);
00123             g_print("%s\n", curr->musician);
00124             // curr = gb_file_parse_volume (curr, doc, cur);
00125             curr->next = head;
00126             head = curr;
00127             /* mem_volume = head */
00128             /* volumes = g_list_append (garagejam_volumes, (GingerblueData *)mem_volume); */
00129             g_print(_("Done with parsing Musician\n"));
00130         }
00131         if ((!xmlStrcmp
00132               (cur->name,
00133                (const xmlChar *) "musical_instrument"))) {
00134             g_print(_("Found Musical Instrument\n"));
00135             curr = g_new0(GingerblueData, 1);
00136             curr->musical_instrument =
00137                 (gchar *) xmlNodeListGetString(doc,
00138                                 cur->
00139                                 xmlChildrenNode,
00140                                 1);
00141             g_print("%s\n", curr->musical_instrument);
00142             // curr = gb_file_parse_volume (curr, doc, cur);
00143             curr->next = head;
00144             head = curr;
00145             /* mem_volume = head */
00146             /* volumes = g_list_append (garagejam_volumes, (GingerblueData *)mem_volume); */
00147             g_print(_
00148                 ("Done with parsing Musical Instrument\n"));
00149         }
00150         if ((!xmlStrcmp(cur->name, (const xmlChar *) "volume"))) {
00151             g_print(_("Found Volume\n"));
00152             curr = g_new0(GingerblueData, 1);
00153             curr->volume =
00154                 (gchar *) xmlNodeListGetString(doc,
00155                                 cur->
00156                                 xmlChildrenNode,
00157                                 1);
00158             g_print("%s\n", curr->volume);
00159             // curr = gb_file_parse_volume (curr, doc, cur);
00160             curr->next = head;
00161             head = curr;
00162             /* mem_volume = head */
00163             /* volumes = g_list_append (garagejam_volumes, (GingerblueData *)mem_volume); */
00164             g_print(_("Done with parsing Volume\n"));
00165         }
00166         cur = cur->next;
00167     }
00168     g_print(_("Finished parsing XML document\n"));
00169     xmlFreeDoc(doc);
00170     return curr;
00171 }
00172
```

```
00173 /* int main (int argc, char **argv) */
00174 /* { */
00175 /*  GingerblueData *data = NULL; */
00176 /*  data = gb_file_config_load (data, "garagejam.xml"); */
00177 /*  free (data); */
00178 /*  return (0); */
00179 /* } */
```

4.8 garagejam-gingerblue-file.h

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009 */
00010
00011 #include <libxml/xmlmemory.h>
00012 #include <libxml/parser.h>
00013
00014 GingerblueData *gb_file_config_load (GingerblueData *head, gchar *filename);
00015
00016 static void gb_file_parse_volume (GingerblueData *data, xmlDocPtr doc, xmlNodePtr cur);
00017
```

4.9 garagejam-gingerblue-knob.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <glib/gstdio.h>
00012 #include <glib/gi18n.h>
00013 #include <gst/gst.h>
00014 #include <gtk/gtk.h>
00015 #include "garagejam.h"
00016
00017 GtkWidget *knob(GingerblueData *data, GtkWidget *line, gint jack,
00018         gchar *label, gboolean lpf, gboolean hpf)
00019 {
00020     GtkWidget *knob;
00021     knob = gtk_volume_button_new();
00022     return (knob);
00023 }
```

4.10 garagejam-gingerblue-knob.h

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009 */
00010
00011 GtkWidget *knob (GingerblueData *data, GtkWidget *line, gint jack, gchar *label, gboolean lpf,
      gboolean hpf);
```

4.11 garagejam-gingerblue-library.h

```
00001 #define GINGERBLUE_LIBRARY 50000
00002
00003 void garagejam_library_add ();
00004 void garagejam_library_delete ();
00005 void garagejam_library_idx ();
```

4.12 garagejam-gingerblue-line.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <gst/gst.h>
00012 #include <gtk/gtk.h>
00013 #include <glib/gstdio.h>
00014 #include <glib/gi18n.h>
00015
00016 GtkWidget *line(gint jack, gchar *label)
00017 {
00018     GtkWidget *window;
00019     window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
00020     gtk_window_set_title(GTK_WINDOW(window), label);
00021     return (window);
00022 }
```

4.13 garagejam-gingerblue-line.h

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009 */
00010
00011 GtkWidget *line (gint jack, gchar *label);
```

4.14 garagejam-gingerblue-main-loop.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2023  Aamot Broadcast
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 0.0.1 (2023-04-30)
00007    Website: http://www.gingerblue.org/
00008
00009  */
00010
00011 #include <gtk/gtk.h>
00012 #include <gst/gst.h>
00013 #include "garagejam.h"
00014 #include "garagejam-gingerblue-studio-config.h"
00015
00016 extern GtkWidget *computer_entry;
00017 extern GtkWidget *studio_entry;
00018
00019 GtkWidget *garagejam_main_loop(GingerblueData *garagejam)
00020 {
00021     GingerblueData *Gingerblue = garagejam;
00022     Gingerblue->window =
00023         main_studio_config(gtk_entry_get_text(GTK_ENTRY(studio_entry)),
00024                    gtk_entry_get_text(GTK_ENTRY
00025                               (computer_entry)));
00026     gtk_window_set_title(Gingerblue->window,
```

```
00027                 g_strconcat(gtk_entry_get_text
00028                     (GTK_ENTRY(computer_entry)),
00029                     " on ",
00030                     gtk_entry_get_text(GTK_ENTRY
00031                                        (studio_entry)),
00032                     NULL));
00033     gtk_widget_show_all(Gingerblue->window);
00034 }
```

4.15 garagejam-gingerblue-main-loop.h

```
00001 #ifndef GINGERBLUE_MAIN_LOOP_H
00002 #define GINGERBLUE_MAIN_LOOP_H 1
00003
00004 GtkWidget *garagejam_main_loop (GingerblueData *garagejam);
00005
00006 #endif /* GINGERBLUE_MAIN_LOOP_H */
00007
00008
```

4.16 garagejam-gingerblue-main.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09))
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <config.h>
00012 #include <stdlib.h>
00013 #include <glib/gi18n.h>
00014 #include <gst/gst.h>
00015 #include <gst/player/player.h>
00016 #include <gtk/gtk.h>
00017 #include <glib/gstdio.h>
00018 #include <glib/gi18n.h>
00019 #include <champlain/champlain.h>
00020 #include <champlain-gtk/champlain-gtk.h>
00021 #include <string.h>
00022 #include <glib.h>
00023 #include <sys/socket.h>
00024 #include <netdb.h>
00025 #include <arpa/inet.h>
00026 #include <libxml/parser.h>
00027 #include <libxml/tree.h>
00028 #include <dirent.h>
00029 #include <vorbis/vorbisfile.h>
00030 #include "garagejam.h"
00031 #include "garagejam-gingerblue-chord.h"
00032 #include "garagejam-gingerblue-config.h"
00033 #include "garagejam-gingerblue-main.h"
00034 #include "garagejam-gingerblue-main-loop.h"
00035 #include "garagejam-gingerblue-record.h"
00036 #include "garagejam-gingerblue-studio-config.h"
00037 #include "garagejam-gingerblue-studio-location.h"
00038 #include "garagejam-gingerblue-studio-stream.h"
00039
00040 GingerblueData *Gingerblue;
00041
00042 static void gb_assistant_entry_changed(GtkEditable *, GtkAssistant *,
00043                         GstElement *);
00044 static void gb_assistant_button_toggled(GtkCheckButton *, GtkAssistant *);
00045 static void gb_assistant_button_clicked(GtkButton *, GtkAssistant *);
00046 static void gb_assistant_cancel(GtkAssistant *, gpointer);
00047 static void gb_assistant_close(GtkAssistant *, gpointer);
00048 static void gb_assistant_apply(GtkAssistant *, gpointer);
00049
00050 typedef struct {
00051     GtkWidget *widget;
00052     gint index;
00053     const gchar *title;
00054     GtkAssistantPageType type;
00055     gboolean complete;
00056 } PageInfo;
00057
```

```
00058 typedef struct {
00059     gchar *title;
00060     gchar *location;
00061 } PlaylistEntry;
00062
00063 GtkWidget *musician_entry, *musician_label;
00064 GtkWidget *song_entry, *song_label;
00065 GtkWidget *instrument_entry, *instrument_label;
00066 GtkWidget *label_entry, *label_label;
00067 GtkWidget *line_entry, *line_label;
00068 GtkWidget *computer_entry, *computer_label;
00069 GtkWidget *recording_entry, *recording_label;
00070 GtkWidget *studio_entry, *studio_label;
00071 GtkWidget *stream_entry, *stream_label;
00072 GtkWidget *album_entry, *album_label;
00073 GtkWidget *summary_entry, *summary_label;
00074
00075 GMainLoop *main_loops;
00076
00077 GstPlayer *player;
00078
00079 GstTagList *tag_list;
00080
00081 gchar xspfbuffer[8192];
00082
00083 GError *error = NULL;
00084
00085 #define MAX_PATH_LENGTH 1024
00086
00087 void write_xspf_header(FILE *file) {
00088     fprintf(file, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
00089     fprintf(file, "<playlist version=\"1\" xmlns=\"http://xspf.org/ns/0/\">\n");
00090     fprintf(file, "  <title>GNOME Playlist</title>\n");
00091     fprintf(file, "  <trackList>\n");
00092 }
00093
00094 void write_xspf_track(FILE *file, const char *title, const char *artist, const char *album, const char
      *location) {
00095     fprintf(file, "    <track>\n");
00096     fprintf(file, "      <title>%s</title>\n", title);
00097     fprintf(file, "      <creator>%s</creator>\n", artist);
00098     fprintf(file, "      <album>%s</album>\n", album);
00099     fprintf(file, "      <location>%s</location>\n", location);
00100     fprintf(file, "    </track>\n");
00101 }
00102
00103 void write_xspf_footer(FILE *file) {
00104     fprintf(file, "  </trackList>\n");
00105     fprintf(file, "</playlist>\n");
00106 }
00107
00108 void process_ogg_file(const char *filename, FILE *xspf_file) {
00109     OggVorbis_File vf;
00110     if (ov_fopen(filename, &vf) != 0) {
00111         fprintf(stderr, "Error opening file: %s\n", filename);
00112         return;
00113     }
00114
00115     vorbis_info *info = ov_info(&vf, -1);
00116
00117     char title[MAX_PATH_LENGTH];
00118     char artist[MAX_PATH_LENGTH];
00119     char album[MAX_PATH_LENGTH];
00120     strncpy(title, filename, sizeof(title));
00121     strncpy(artist, gtk_entry_get_text(GTK_ENTRY(musician_entry)), sizeof(artist));
00122     strncpy(album, gtk_entry_get_text(GTK_ENTRY(album_entry)), sizeof(album));
00123
00124     vorbis_comment *vc = ov_comment(&vf, -1);
00125     for (int i = 0; i < vc->comments; i++) {
00126         if (strstr(vc->user_comments[i], "TITLE=") == vc->user_comments[i]) {
00127             strncpy(title, vc->user_comments[i] + 6, sizeof(title));
00128         }
00129         if (strstr(vc->user_comments[i], "ARTIST=") == vc->user_comments[i]) {
00130             strncpy(artist, vc->user_comments[i] + 7, sizeof(artist));
00131         }
00132         if (strstr(vc->user_comments[i], "ALBUM=") == vc->user_comments[i]) {
00133             strncpy(album, vc->user_comments[i] + 6, sizeof(album));
00134         }
00135     }
00136
00137     write_xspf_track(xspf_file, title, artist, album, filename);
00138
00139     ov_clear(&vf);
00140 }
00141
00142 // Signal handler for playlist entry selection
00143 void on_playlist_entry_selected(GtkListBox *listbox, GtkListBoxRow *row,
```

```
00144                 gpointer user_data)
00145 {
00146     GstPlayer *player;
00147     PlaylistEntry *entry =
00148         (PlaylistEntry *) g_object_get_data(G_OBJECT(row),
00149                         "playlist_entry");
00150     if (entry) {
00151         player =
00152             gst_player_new(NULL,
00153                    gst_player_g_main_context_signal_dispatcher_new
00154                    (NULL));
00155         gst_player_set_uri(GST_PLAYER(player),
00156                    g_strconcat("file://",
00157                            g_get_user_special_dir
00158                            (G_USER_DIRECTORY_MUSIC),
00159                            "/", entry->title, NULL));
00160         gst_player_play(GST_PLAYER(player));
00161         g_print("Playing: %s\n",
00162             g_strconcat("file://",
00163                     g_get_user_special_dir
00164                     (G_USER_DIRECTORY_MUSIC), "/",
00165                     entry->title, NULL));
00166         // Add your playback logic here
00167     }
00168 }
00169
00170 gboolean parse_xspf_file(const gchar *filepath, GList **playlist_entries)
00171 {
00172     xmlDocPtr doc;
00173     xmlNodePtr cur;
00174
00175     doc = xmlParseFile(filepath);
00176     if (doc == NULL) {
00177         g_print("Failed to parse %s\n", filepath);
00178         return FALSE;
00179     }
00180
00181     cur = xmlDocGetRootElement(doc);
00182     if (cur == NULL) {
00183         g_print("Empty document\n");
00184         xmlFreeDoc(doc);
00185         return FALSE;
00186     }
00187
00188     cur = cur->xmlChildrenNode;
00189     while (cur != NULL) {
00190         if (xmlStrcmp(cur->name, (const xmlChar *) "trackList") ==
00191             0) {
00192             xmlNodePtr track = cur->xmlChildrenNode;
00193             while (track != NULL) {
00194                 if (xmlStrcmp
00195                     (track->name,
00196                      (const xmlChar *) "track") == 0) {
00197                     xmlNodePtr locationNode =
00198                         track->xmlChildrenNode;
00199                     gchar *location = NULL;
00200                     while (locationNode != NULL) {
00201                         if (xmlStrcmp
00202                             (locationNode->name,
00203                              (const xmlChar *)
00204                              "location") == 0) {
00205                             location =
00206                                 (gchar *)
00207                                 xmlNodeGetContent
00208                                 (locationNode);
00209                             break;
00210                         }
00211                         locationNode =
00212                             locationNode->next;
00213                     }
00214
00215                     if (location) {
00216                         PlaylistEntry *entry =
00217                             g_new0(PlaylistEntry,
00218                                1);
00219                         entry->title =
00220                             g_filename_display_basename
00221                             (location);
00222                         entry->location =
00223                             g_strdup(location);
00224                         *playlist_entries =
00225                             g_list_prepend
00226                             (*playlist_entries,
00227                              entry);
00228                     }
00229                 }
00230                 track = track->next;
```

```
00231             }
00232         }
00233         cur = cur->next;
00234     }
00235
00236     xmlFreeDoc(doc);
00237     return TRUE;
00238 }
00239
00240 static void gb_assistant_entry_changed(GtkEditable *editable,
00241                       GtkAssistant *assistant,
00242                       GstElement *pipeline)
00243 {
00244     return;
00245 }
00246
00247 static void gb_assistant_button_toggled(GtkCheckButton *checkbutton,
00248                     GtkAssistant *assistant)
00249 {
00250     return;
00251 }
00252
00253 static void gb_assistant_button_clicked(GtkButton *button,
00254                     GtkAssistant *assistant)
00255 {
00256     GstElement *src, *conv, *enc, *muxer, *sink, *recorder;
00257     gchar *filename = NULL;
00258     GDateTime *datestamp = g_date_time_new_now_utc();
00259     GstElementFactory *factory;
00260     GTimeVal *timeval;
00261     gst_element_send_event(recorder, gst_event_new_eos());
00262     recorder = gst_pipeline_new("record_pipe");
00263     /*
00264        FIXME: Line #59 from
      https://github.com/GStreamer/gst-plugins-base/blob/master/tools/gst-device-monitor.c
00265        element = gst_device_create_element (device, NULL);
00266        if (!element)
00267        return NULL;
00268        factory = gst_element_get_factory (element);
00269        if (!factory) {
00270        gst_object_unref (element);
00271        return NULL;
00272        }
00273        src = gst_element_factory_create(factory, NULL);
00274      */
00275     src = gst_element_factory_make("autoaudiosrc", "auto_source");
00276     conv = gst_element_factory_make("audioconvert", "convert");
00277     enc = gst_element_factory_make("vorbisenc", "vorbis_enc");
00278     muxer = gst_element_factory_make("oggmux", "oggmux");
00279     sink = gst_element_factory_make("filesink", "sink");
00280     filename =
00281         g_strconcat(g_get_user_special_dir(G_USER_DIRECTORY_MUSIC),
00282             "/", gtk_entry_get_text(GTK_ENTRY(musician_entry)),
00283             "_-_", gtk_entry_get_text(GTK_ENTRY(song_entry)),
00284             "_[", g_date_time_format_iso8601(datestamp), "]",
00285             ".ogg", NULL);
00286     g_object_set(G_OBJECT(sink), "location",
00287              g_strconcat(g_get_user_special_dir
00288                  (G_USER_DIRECTORY_MUSIC), "/",
00289                  gtk_entry_get_text(GTK_ENTRY
00290                             (musician_entry)),
00291                  "_-_",
00292                  gtk_entry_get_text(GTK_ENTRY(song_entry)),
00293                  ".ogg", NULL), NULL);
00294     g_object_set(G_OBJECT(enc), "quality", 1.0);
00295     gst_bin_add_many(GST_BIN(recorder), src, conv, enc, muxer, sink,
00296              NULL);
00297     gst_element_link_many(src, conv, enc, muxer, sink, NULL);
00298     gst_element_set_state(recorder, GST_STATE_PLAYING);
00299     datestamp = g_date_time_new_now_utc();
00300     gst_tag_setter_add_tags(GST_TAG_SETTER(enc),
00301                 GST_TAG_MERGE_APPEND,
00302                 GST_TAG_TITLE, g_get_real_name(),
00303                 GST_TAG_ARTIST, g_get_real_name(),
00304                 GST_TAG_ALBUM, "Voicegram",
00305                 GST_TAG_COMMENT, "GNOME 45",
00306                 GST_TAG_DATE,
00307                 g_date_time_format_iso8601(datestamp),
00308                 NULL);
00309     g_date_time_unref(datestamp);
00310     main_loops = g_main_loop_new(NULL, TRUE);
00311     g_main_loop_run(main_loops);
00312     gst_element_set_state(recorder, GST_STATE_NULL);
00313     g_main_loop_unref(main_loops);
00314     gst_object_unref(GST_OBJECT(recorder));
00315     g_date_time_unref(datestamp);
00316 }
```

```
00317
00318 static void gb_assistant_cancel(GtkAssistant *assistant, gpointer data)
00319 {
00320     if (!main_loops) {
00321         g_error("Quit more loops than there are.");
00322     } else {
00323         GMainLoop *loop = main_loops;
00324         g_main_loop_quit(loop);
00325         gtk_main_quit();
00326     }
00327 }
00328
00329 static void gb_assistant_close(GtkAssistant *assistant, gpointer data)
00330 {
00331     FILE *fp = NULL;
00332     FILE *file = NULL;
00333
00334     long file_size;
00335     GDateTime *datestamp = g_date_time_new_now_utc();
00336     gchar *filename =
00337         g_strconcat(g_get_user_special_dir(G_USER_DIRECTORY_MUSIC),
00338             "/",
00339             gtk_entry_get_text(GTK_ENTRY(computer_entry)),
00340             "_-_",
00341             gtk_entry_get_text(GTK_ENTRY(song_entry)), "_[",
00342             g_date_time_format_iso8601(datestamp), "]",
00343             ".jam", NULL);
00344     fp = fopen(filename, "w");
00345     fprintf(fp, "<?xml version='1.0' encoding='UTF-8'?>\n");
00346     fprintf(fp, "<gingerblue version='%s'>\n", VERSION);
00347     fprintf(fp, "  <musician>%s</musician>\n",
00348         gtk_entry_get_text(GTK_ENTRY(musician_entry)));
00349     fprintf(fp, "  <song>%s</song>\n",
00350         gtk_entry_get_text(GTK_ENTRY(song_entry)));
00351     fprintf(fp, "  <instrument>%s</instrument>\n",
00352         gtk_entry_get_text(GTK_ENTRY(instrument_entry)));
00353     fprintf(fp, "  <line>%s</line>\n",
00354         gtk_entry_get_text(GTK_ENTRY(line_entry)));
00355     fprintf(fp, "  <label>%s</label>\n",
00356         gtk_entry_get_text(GTK_ENTRY(label_entry)));
00357     fprintf(fp, "  <station>%s</station>\n",
00358         gtk_entry_get_text(GTK_ENTRY(computer_entry)));
00359     fprintf(fp, "  <filename>%s</filename>\n",
00360         gtk_entry_get_text(GTK_ENTRY(recording_entry)));
00361     fprintf(fp, "  <album>%s</album>\n",
00362         gtk_entry_get_text(GTK_ENTRY(album_entry)));
00363     fprintf(fp, "  <studio>%s</studio>\n",
00364         gtk_entry_get_text(GTK_ENTRY(studio_entry)));
00365     fprintf(fp, "</gingerblue>\n");
00366     fclose(fp);
00367     g_date_time_unref(datestamp);
00368     /* main_studio_stream(filename, */
00369     /*         gtk_entry_get_text(GTK_ENTRY(studio_entry))); */
00370     gst_element_send_event(data, gst_event_new_eos());
00371 }
00372
00373 static void gb_assistant_apply(GtkAssistant *assistant, gpointer data)
00374 {
00375     GingerblueData *garagejam_config;
00376     GtkWindow *garagejam_window;
00377     FILE *file;
00378     /* gtk_init (&argc, &argv); */
00379     garagejam_config =
00380         main_config(GTK_WIDGET(garagejam_window),
00381             gtk_entry_get_text(GTK_ENTRY(studio_entry)));
00382     garagejam_window = garagejam_main_loop(garagejam_config);
00383     gtk_widget_show_all(garagejam_window);
00384     /* gst_init(&argc, &argc); */
00385     /* gtk_main(); */
00386     gst_element_send_event(data, gst_event_new_eos());
00387     return;
00388 }
00389
00390 GtkAssistantPageFunc gb_assistant_cb(GtkAssistant *assistant,
00391                      GDateTime *datestamp)
00392 {
00393     /* gtk_assistant_next_page(assistant); */
00394 }
00395
00396 void free_playlist_entry(gpointer data)
00397 {
00398     PlaylistEntry *entry = (PlaylistEntry *) data;
00399     if (entry) {
00400         g_free(entry->title);
00401         g_free(entry->location);
00402         g_free(entry);
00403     }
```

```
00404 }
00405
00406
00407 int main(int argc, char **argv)
00408 {
00409     GSocketConnectable *addr;
00410     GDateTime *datestamp;
00411     GingerblueData *data;
00412     GingerblueChord *garagejam_chord;
00413     GstElement *src, *conv, *enc, *muxer, *sink, *pipeline;
00414     GtkWidget *introduction;
00415     GtkEntryBuffer *default_recording_title;
00416     GtkWidget *entry, *label, *button, *progress, *hbox;
00417     GtkWidget *summary_label, *summary_entry;
00418     GtkWidget *garagejam_main;
00419     guint i;
00420     GtkWidget *musicianpage;
00421     GtkWidget *songpage;
00422     GtkWidget *instrumentpage;
00423     GtkWidget *recordpage;
00424     GtkWidget *window;
00425     GtkWidget *frame;
00426     GtkWidget *input;
00427     GtkWidget *main_window;
00428     GtkWidget *mixer;
00429     GtkWidget *control;
00430     GtkWidget *soundboard;
00431     GtkWidget *toolbar;
00432     GtkWidget *input_record;
00433     GtkWidget *input_pause;
00434     GtkWidget *input_break;
00435     GtkWidget *input_stop;
00436     GtkWidget *input_volume;
00437     gdouble input_volume_value;
00438     gint64 real_time;
00439     gchar *album;
00440     GtkWidget *list;
00441     GList *playlist_entries = NULL;
00442     PageInfo page[11] = {
00443         { NULL, -1, "GarageJam Setup", GTK_ASSISTANT_PAGE_INTRO,
00444          TRUE },
00445         { NULL, -1, "Musician", GTK_ASSISTANT_PAGE_CONTENT, TRUE },
00446         { NULL, -1, "Song", GTK_ASSISTANT_PAGE_CONTENT, TRUE },
00447         { NULL, -1, "Instrument", GTK_ASSISTANT_PAGE_CONTENT,
00448          TRUE },
00449         { NULL, -1, "Input Line", GTK_ASSISTANT_PAGE_CONTENT,
00450          TRUE },
00451         { NULL, -1, "Label", GTK_ASSISTANT_PAGE_CONTENT, TRUE },
00452         { NULL, -1, "Computer", GTK_ASSISTANT_PAGE_CONTENT, TRUE },
00453         { NULL, -1, "Recording", GTK_ASSISTANT_PAGE_CONTENT,
00454          TRUE },
00455         { NULL, -1, "Studio", GTK_ASSISTANT_PAGE_CONTENT, TRUE },
00456         { NULL, -1, "Album", GTK_ASSISTANT_PAGE_CONTENT, TRUE },
00457         { NULL, -1, "Connect", GTK_ASSISTANT_PAGE_CONFIRM, TRUE },
00458     };
00459     FILE *xspf = NULL;
00460     datestamp = g_date_time_new_now_utc();
00461     gchar *filename =
00462         g_strconcat(g_get_user_special_dir(G_USER_DIRECTORY_MUSIC),
00463             "/",
00464             gtk_entry_get_text(GTK_ENTRY(musician_entry)),
00465             "_-_",
00466             gtk_entry_get_text(GTK_ENTRY(song_entry)), "_[",
00467             g_date_time_format_iso8601(datestamp), "]",
00468             ".ogg", NULL);
00469     gtk_init(&argc, &argv);
00470     gst_init(&argc, &argc);
00471     window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
00472     introduction = gtk_assistant_new();
00473     gtk_widget_set_size_request(GTK_WIDGET(introduction), 640, 480);
00474     gtk_window_set_title(GTK_WINDOW(introduction), "GNOME GarageJam");
00475     // Connect signal handler for playlist entry selection
00476     g_signal_connect(G_OBJECT(introduction), "destroy",
00477              G_CALLBACK(gtk_main_quit), NULL);
00478     page[0].widget =
00479         gtk_label_new(_
00480               ("Welcome to GarageJam!\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 GarageJam."));
00481     page[1].widget = gtk_box_new(FALSE, 5);
00482     musician_label = gtk_label_new(_("Musician:"));
00483     musician_entry = gtk_entry_new();
00484     if (g_strcmp0(musician_entry, NULL) != 0)
00485         gtk_entry_set_text(GTK_ENTRY(musician_entry),
00486                    g_get_real_name());
00487     else
00488         gtk_entry_set_text(GTK_ENTRY(musician_entry),
```

```
00489                   gtk_entry_get_text(GTK_ENTRY
00490                                      (musician_entry)));
00491     gtk_box_pack_start(GTK_BOX(page[1].widget),
00492                GTK_WIDGET(musician_label), FALSE, FALSE, 5);
00493     gtk_box_pack_start(GTK_BOX(page[1].widget),
00494                GTK_WIDGET(musician_entry), FALSE, FALSE, 5);
00495     page[2].widget = gtk_box_new(FALSE, 5);
00496     song_label = gtk_label_new(_("Song:"));
00497     song_entry = gtk_entry_new();
00498     if (g_strcmp0(song_entry, NULL) != 0)
00499         gtk_entry_set_text(GTK_ENTRY(song_entry),
00500                    g_strconcat(gtk_entry_get_text
00501                            (song_entry),
00502                            g_date_time_format_iso8601
00503                            (datestamp), NULL));
00504     else
00505         gtk_entry_set_text(GTK_ENTRY(song_entry),
00506                    gtk_entry_get_text(GTK_ENTRY
00507                                       (song_entry)));
00508     gtk_box_pack_start(GTK_BOX(page[2].widget), GTK_WIDGET(song_label),
00509                FALSE, FALSE, 5);
00510     gtk_box_pack_start(GTK_BOX(page[2].widget), GTK_WIDGET(song_entry),
00511                FALSE, FALSE, 5);
00512     page[3].widget = gtk_box_new(FALSE, 5);
00513     instrument_label = gtk_label_new(_("Instrument:"));
00514     instrument_entry = gtk_entry_new();
00515     gtk_entry_set_text(GTK_ENTRY(instrument_entry), _("Guitar"));
00516     gtk_box_pack_start(GTK_BOX(page[3].widget),
00517                GTK_WIDGET(instrument_label), FALSE, FALSE, 5);
00518     gtk_box_pack_start(GTK_BOX(page[3].widget),
00519                GTK_WIDGET(instrument_entry), FALSE, FALSE, 5);
00520     page[4].widget = gtk_box_new(FALSE, 5);
00521     line_label = gtk_label_new(_("Line Input:"));
00522     line_entry = gtk_entry_new();
00523     gtk_entry_set_text(GTK_ENTRY(line_entry), _("Mic"));
00524     gtk_box_pack_start(GTK_BOX(page[4].widget), GTK_WIDGET(line_label),
00525                FALSE, FALSE, 5);
00526     gtk_box_pack_start(GTK_BOX(page[4].widget), GTK_WIDGET(line_entry),
00527                FALSE, FALSE, 5);
00528     page[5].widget = gtk_box_new(FALSE, 5);
00529     label_label = gtk_label_new(_("Label:"));
00530     label_entry = gtk_entry_new();
00531     gtk_entry_set_text(GTK_ENTRY(label_entry), _("GNOME"));
00532     gtk_box_pack_start(GTK_BOX(page[5].widget),
00533                GTK_WIDGET(label_label), FALSE, FALSE, 5);
00534     gtk_box_pack_start(GTK_BOX(page[5].widget),
00535                GTK_WIDGET(label_entry), FALSE, FALSE, 5);
00536     page[6].widget = gtk_box_new(FALSE, 5);
00537     computer_label = gtk_label_new(_("Computer:"));
00538     computer_entry = gtk_entry_new();
00539     gtk_entry_set_text(GTK_ENTRY(computer_entry),
00540                _(g_get_host_name()));
00541     addr = g_network_address_new(_(g_get_host_name()), 12348);
00542     gtk_entry_set_text(GTK_ENTRY(computer_entry),
00543                g_network_address_get_hostname(addr));
00544     gtk_box_pack_start(GTK_BOX(page[6].widget),
00545                GTK_WIDGET(computer_label), FALSE, FALSE, 5);
00546     gtk_box_pack_start(GTK_BOX(page[6].widget),
00547                GTK_WIDGET(computer_entry), FALSE, FALSE, 5);
00548     recording_label = gtk_button_new_with_label("Recording");
00549     recording_entry = gtk_entry_new();
00550     gtk_entry_set_text(GTK_ENTRY(recording_entry),
00551                g_strconcat(g_get_user_special_dir
00552                        (G_USER_DIRECTORY_MUSIC), "/",
00553                        gtk_entry_get_text(GTK_ENTRY
00554                                (musician_entry)),
00555                        "_-_",
00556                        gtk_entry_get_text(GTK_ENTRY
00557                                (song_entry)),
00558                        ".ogg", NULL));
00559     g_signal_connect(G_OBJECT(recording_label), "clicked",
00560              G_CALLBACK(gb_record_cb),
00561              g_strconcat(g_get_user_special_dir
00562                      (G_USER_DIRECTORY_MUSIC), "/",
00563                      gtk_entry_get_text(GTK_ENTRY
00564                              (musician_entry)),
00565                      "_-_",
00566                      gtk_entry_get_text(GTK_ENTRY
00567                              (song_entry)),
00568                      ".ogg", NULL));
00569     page[7].widget = gtk_entry_new();
00570     gtk_entry_set_text(GTK_ENTRY(page[7].widget),
00571                g_strconcat(g_get_user_special_dir
00572                        (G_USER_DIRECTORY_MUSIC), "/",
00573                        gtk_entry_get_text(GTK_ENTRY
00574                                (musician_entry)),
00575                        "_-_",
```

```
00576                       gtk_entry_get_text(GTK_ENTRY
00577                               (song_entry)),
00578                       ".ogg", NULL));
00579     gtk_box_pack_start(GTK_BOX(page[7].widget),
00580                GTK_WIDGET(recording_label), FALSE, FALSE, 5);
00581     gtk_box_pack_start(GTK_BOX(page[7].widget),
00582                GTK_WIDGET(recording_entry), FALSE, FALSE, 5);
00583     studio_label = gtk_button_new_with_label("Broadcasting");
00584     studio_entry = gtk_entry_new();
00585     gtk_entry_set_text(GTK_ENTRY(studio_entry),
00586                g_strconcat("https://www.gingerblue.org/api/",
00587                       gtk_entry_get_text(GTK_ENTRY
00588                               (label_entry)),
00589                       "/",
00590                       gtk_entry_get_text(GTK_ENTRY
00591                               (computer_entry)),
00592                       NULL));
00593     g_signal_connect(G_OBJECT(studio_label), "clicked",
00594              G_CALLBACK(gb_assistant_apply),
00595              gtk_entry_get_text(GTK_ENTRY(studio_entry)));
00596     g_signal_connect(G_OBJECT(studio_entry), "clicked",
00597              G_CALLBACK(gb_assistant_apply),
00598              gtk_entry_get_text(GTK_ENTRY(studio_entry)));
00599     page[8].widget = gtk_entry_new();
00600     gtk_entry_set_text(GTK_ENTRY(page[8].widget),
00601                gtk_entry_get_text(GTK_ENTRY(studio_entry)));
00602     gtk_box_pack_start(GTK_BOX(page[8].widget),
00603                GTK_WIDGET(studio_label), FALSE, FALSE, 5);
00604     gtk_box_pack_start(GTK_BOX(page[8].widget),
00605                GTK_WIDGET(studio_entry), FALSE, FALSE, 5);
00606     album_label = gtk_label_new("Album");
00607     album_entry = gtk_entry_new();
00608     g_signal_connect(G_OBJECT(album_label), "clicked",
00609              G_CALLBACK(gb_assistant_apply),
00610              gtk_entry_get_text(GTK_ENTRY(album_entry)));
00611     album =
00612         g_strconcat(g_get_user_special_dir(G_USER_DIRECTORY_MUSIC),
00613             "/", gtk_entry_get_text(GTK_ENTRY(label_entry)),
00614             NULL);
00615     gtk_entry_set_text(GTK_ENTRY(album_entry), (gchar *) album);
00616     page[9].widget = gtk_entry_new();
00617     gtk_entry_set_text(GTK_ENTRY(page[9].widget), album);
00618     g_signal_connect(GTK_BUTTON(album_entry), "clicked",
00619              G_CALLBACK(gb_assistant_apply),
00620              GTK_ENTRY(album_entry));
00621     g_signal_connect(GTK_BOX(page[9].widget), "clicked",
00622              G_CALLBACK(gb_assistant_apply),
00623              GTK_ENTRY(album_entry));
00624     g_signal_connect(G_OBJECT(album_label), "clicked",
00625              G_CALLBACK(gb_assistant_apply), album_entry);
00626     gtk_box_pack_start(GTK_BOX(page[9].widget),
00627                GTK_WIDGET(album_label), FALSE, FALSE, 5);
00628     gtk_box_pack_start(GTK_BOX(page[9].widget),
00629                GTK_WIDGET(album_entry), FALSE, FALSE, 5);
00630     stream_label = gtk_button_new_with_label("Protocol");
00631     stream_entry = gtk_entry_new();
00632     gtk_entry_set_text(GTK_ENTRY(stream_entry), "Torrent");
00633     g_signal_connect(G_OBJECT(stream_entry), "clicked",
00634              G_CALLBACK(gb_assistant_apply),
00635              gtk_entry_get_text(GTK_ENTRY(stream_entry)));
00636     gchar *record_attachment =
00637         g_strconcat(gtk_entry_get_text(GTK_ENTRY(recording_entry)),
00638             NULL);
00639     gchar *subject =
00640         g_strconcat(gtk_entry_get_text(GTK_ENTRY(label_entry)), "/",
00641             gtk_entry_get_text(GTK_ENTRY(computer_entry)),
00642             NULL);
00643     page[10].widget =
00644         gtk_link_button_new_with_label(g_strconcat
00645                        ("https://www.gingerblue.org/api/",
00646                         subject, NULL),
00647                        "Connect GarageJam to Gingerblue Recording Studio API");
00648     gtk_entry_set_text(GTK_ENTRY(page[10].widget), "Click Apply");
00649     g_signal_connect(GTK_BUTTON(stream_entry), "clicked",
00650              G_CALLBACK(gb_assistant_apply),
00651              gtk_entry_get_text(GTK_ENTRY(studio_entry)));
00652     g_signal_connect(G_OBJECT(stream_label), "clicked",
00653              G_CALLBACK(gb_assistant_apply),
00654              gtk_entry_get_text(GTK_ENTRY(stream_entry)));
00655     gtk_box_pack_start(GTK_BOX(page[10].widget),
00656                GTK_WIDGET(stream_label), FALSE, FALSE, 5);
00657     gtk_box_pack_start(GTK_BOX(page[10].widget),
00658                GTK_WIDGET(stream_entry), FALSE, FALSE, 5);
00659     for (i = 0; i < 11; i++) {
00660         page[i].index =
00661             gtk_assistant_append_page(GTK_ASSISTANT(introduction),
00662                           GTK_WIDGET(page[i].widget));
```

```
00663         gtk_assistant_set_page_title(GTK_ASSISTANT(introduction),
00664                          GTK_WIDGET(page[i].widget),
00665                          page[i].title);
00666         gtk_assistant_set_page_type(GTK_ASSISTANT(introduction),
00667                         GTK_WIDGET(page[i].widget),
00668                         page[i].type);
00669         gtk_assistant_set_page_complete(GTK_ASSISTANT
00670                         (introduction),
00671                         GTK_WIDGET(page[i].widget),
00672                         page[i].complete);
00673     }
00674     g_signal_connect(G_OBJECT(entry), "changed",
00675              G_CALLBACK(gb_assistant_entry_changed), pipeline);
00676     g_signal_connect(G_OBJECT(introduction), "cancel",
00677              G_CALLBACK(gb_assistant_cancel), main_loops);
00678     g_signal_connect(G_OBJECT(introduction), "close",
00679              G_CALLBACK(gb_assistant_close), pipeline);
00680     g_signal_connect(G_OBJECT(introduction), "apply",
00681              G_CALLBACK(gb_assistant_close), pipeline);
00682 /* musicianpage = gtk_entry_new (); */
00683     /* real_time = g_get_real_time(); */
00684     /* gtk_assistant_insert_page (introduction, */
00685     /*                            musicianpage, */
00686     /*                            0); */
00687     /* gtk_assistant_set_page_title (introduction, */
00688     /*                               musicianpage, */
00689     /*                               "Musician Setup"); */
00690     /* gtk_assistant_set_page_type (introduction, */
00691     /*                              musicianpage, */
00692     /*                              GTK_ASSISTANT_PAGE_INTRO); */
00693     /* songpage = gtk_entry_new (); */
00694     /* gtk_entry_set_text (songpage, g_strconcat(g_get_home_dir(), _("/Music/"), g_get_real_name(), "
      - Song.garagejam", NULL)); */
00695     /* real_time = g_get_real_time(); */
00696     /* gtk_assistant_insert_page (introduction, */
00697     /*                            songpage, */
00698     /*                            1); */
00699     /* gtk_assistant_set_page_title (introduction, */
00700     /*                               songpage, */
00701     /*                               "Song Setup"); */
00702     /* gtk_assistant_set_page_type (introduction, */
00703     /*                              songpage, */
00704     /*                              GTK_ASSISTANT_PAGE_CONTENT); */
00705     /* gtk_assistant_next_page(introduction); */
00706     /* instrumentpage = gtk_entry_new (); */
00707     /* gtk_entry_set_text (instrumentpage, "Guitar"); */
00708     /* gtk_assistant_set_page_type (introduction, */
00709     /*                              instrumentpage, */
00710     /*                              GTK_ASSISTANT_PAGE_CONTENT); */
00711     /* gtk_assistant_insert_page (introduction, */
00712     /*                            instrumentpage, */
00713     /*                            2); */
00714     /* gtk_assistant_set_page_title (introduction, */
00715     /*                               instrumentpage, */
00716     /*                               "Instrument Setup"); */
00717     /* recordpage = gtk_entry_new (); */
00718     /* gtk_entry_set_text (recordpage, "Microphone Line"); */
00719     /* gtk_assistant_set_page_type (introduction, */
00720     /*                              recordpage, */
00721     /*                              GTK_ASSISTANT_PAGE_SUMMARY); */
00722     /* gtk_assistant_insert_page (introduction, */
00723     /*                            recordpage, */
00724     /*                            3); */
00725     /* gtk_assistant_set_page_title (introduction, */
00726     /*                               recordpage, */
00727     /*                               "Recording Setup"); */
00728     /* gtk_assistant_set_page_complete (introduction, recordpage, 1); */
00729     /* gtk_assistant_set_forward_page_func (introduction, */
00730     /*                                      gb_assistant_cb, */
00731     /*                                      NULL, */
00732     /*                                      NULL); */
00733     /* gtk_assistant_commit (introduction); */
00734     gtk_widget_show_all(GTK_WIDGET(introduction));
00735     gst_init(&argc, &argv);
00736     gst_init(NULL, NULL);
00737     pipeline = gst_pipeline_new("record_pipe");
00738 
00739     src = gst_element_factory_make("autoaudiosrc", "auto_source");
00740     conv = gst_element_factory_make("audioconvert", "convert");
00741     enc = gst_element_factory_make("vorbisenc", "vorbis_enc");
00742     muxer = gst_element_factory_make("oggmux", "oggmux");
00743     sink = gst_element_factory_make("filesink", "sink");
00744     filename =
00745         g_strconcat(g_get_user_special_dir(G_USER_DIRECTORY_MUSIC),
00746             "/", gtk_entry_get_text(GTK_ENTRY(musician_entry)),
00747             "_-_", gtk_entry_get_text(GTK_ENTRY(song_entry)),
00748             "_[", g_date_time_format_iso8601(datestamp), "]",
```

```
00749             ".ogg", NULL);
00750     g_object_set(G_OBJECT(sink), "location",
00751              g_strconcat(g_get_user_special_dir
00752                  (G_USER_DIRECTORY_MUSIC), "/",
00753                  gtk_entry_get_text(GTK_ENTRY
00754                            (musician_entry)),
00755                  "_-_",
00756                  gtk_entry_get_text(GTK_ENTRY(song_entry)),
00757                  ".ogg", NULL), NULL);
00758     gst_bin_add_many(GST_BIN(pipeline), src, conv, enc, muxer, sink,
00759              NULL);
00760     gst_element_link_many(src, conv, enc, muxer, sink, NULL);
00761
00762     gst_element_set_state(pipeline, GST_STATE_PLAYING);
00763
00764     gtk_widget_set_size_request(GTK_WIDGET(window), 800, 600);
00765     gtk_window_set_title(GTK_WINDOW(window),
00766                  g_strconcat(g_get_user_special_dir
00767                      (G_USER_DIRECTORY_MUSIC), "/",
00768                      gtk_entry_get_text(GTK_ENTRY
00769                                (label_entry)),
00770                      ".xspf", NULL));
00771     // Create the list
00772     list = gtk_list_box_new();
00773     gtk_container_add(GTK_CONTAINER(window), list);
00774
00775         const char *ogg_dir = g_strdup_printf("%s",
00776                           g_get_user_special_dir
00777                           (G_USER_DIRECTORY_MUSIC));
00778
00779         if (ogg_dir == NULL) {
00780           fprintf(stderr, "Error getting G_USER_DIRECTORY_MUSIC variable.\n");
00781           return 1;
00782         }
00783
00784     char xspf_path[MAX_PATH_LENGTH];
00785     snprintf(xspf_path, sizeof(xspf_path), "%s/GNOME.xspf", ogg_dir);
00786
00787     FILE *xspf_file = fopen(xspf_path, "w");
00788     if (xspf_file == NULL) {
00789         fprintf(stderr, "Error opening XSPF file for writing.\n");
00790         return 1;
00791     }
00792
00793     write_xspf_header(xspf_file);
00794
00795     DIR *dir;
00796     struct dirent *ent;
00797     if ((dir = opendir(ogg_dir)) != NULL) {
00798         while ((ent = readdir(dir)) != NULL) {
00799             if (ent->d_type == DT_REG && strstr(ent->d_name, ".ogg") != NULL) {
00800                 char file_path[MAX_PATH_LENGTH];
00801                 snprintf(file_path, sizeof(file_path), "%s/%s", ogg_dir, ent->d_name);
00802                 process_ogg_file(file_path, xspf_file);
00803             }
00804         }
00805         closedir(dir);
00806     } else {
00807         fprintf(stderr, "Error opening directory.\n");
00808         return 1;
00809     }
00810
00811     write_xspf_footer(xspf_file);
00812
00813     fclose(xspf_file);
00814
00815     // Get the user's music directory
00816     gchar *music_dir =
00817       g_strdup_printf("%s",
00818                g_get_user_special_dir
00819                (G_USER_DIRECTORY_MUSIC));
00820     gchar *playlist_file =
00821       g_build_filename(music_dir, "GNOME.xspf", NULL);
00822
00823     // Parse the playlist file
00824     parse_xspf_file(playlist_file, &playlist_entries);
00825
00826     // Add playlist entries to the list
00827     // Add playlist entries to the list
00828     GList *iter;
00829     for (iter = playlist_entries; iter != NULL;
00830      iter = g_list_next(iter)) {
00831       PlaylistEntry *entry = (PlaylistEntry *) iter->data;
00832       GtkWidget *label = gtk_label_new(entry->title);
00833         // Create a list box row
00834         GtkListBoxRow *row =
00835             GTK_LIST_BOX_ROW(gtk_list_box_row_new());
```

```
00836         // Set user data for the row
00837         g_object_set_data(G_OBJECT(row), "playlist_entry", entry);
00838         gtk_container_add(GTK_CONTAINER(row), label);
00839         gtk_list_box_insert(GTK_LIST_BOX(list), GTK_WIDGET(row),
00840                             -1);
00841         g_signal_connect(GTK_LIST_BOX(list), "row-selected",
00842                          G_CALLBACK(on_playlist_entry_selected),
00843                          NULL);
00844     }
00845     gtk_widget_show_all(GTK_WIDGET(window));
00846     main_loops = g_main_loop_new(NULL, TRUE);
00847     g_main_loop_run(main_loops);
00848 
00849     gst_element_set_state(pipeline, GST_STATE_NULL);
00850     g_main_loop_unref(main_loops);
00851     gst_object_unref(GST_OBJECT(pipeline));
00852 
00853     /* player = play_new ("http://stream.radionorwegian.com/56.ogg", garagejam_data->volume); */
00854     /* input_volume_value = gb_window_set_volume(GTK_VOLUME_BUTTON (input_volume), 0.00);   *\/ */
00855     /* g_signal_connect (GTK_BUTTON (input_record), "clicked", G_CALLBACK (gb_window_new_record),
      garagejam_data->volume); */
00856     /* g_signal_connect (GTK_BUTTON (input_pause), "clicked", G_CALLBACK (gb_window_pause_record),
      garagejam_data->volume); */
00857     /* g_signal_connect (GTK_BUTTON (input_break), "clicked", G_CALLBACK (gb_window_break_record),
      garagejam_data->volume); */
00858     /* g_signal_connect (GTK_VOLUME_BUTTON (input_volume), "value-changed", G_CALLBACK
      (gb_window_pause_record), garagejam_data->volume); */
00859     /* g_signal_connect (GTK_VOLUME_BUTTON (input_volume), "value-changed", G_CALLBACK
      (gb_window_store_volume), garagejam_data->volume);   */
00860     g_signal_connect(GTK_WINDOW(introduction), "destroy",
00861              G_CALLBACK(gtk_main_quit), NULL);
00862     g_signal_connect(GTK_WINDOW(introduction), "destroy",
00863              G_CALLBACK(gtk_main_quit), NULL);
00864 
00865     /* g_free (garagejam_data); */
00866 
00867     g_date_time_unref(datestamp);
00868     gtk_main();
00869     g_list_free_full(playlist_entries,
00870              (GDestroyNotify) free_playlist_entry);
00871     g_free(playlist_file);
00872     g_free(music_dir);
00873     return (0);
00874 }
```

4.17 garagejam-gingerblue-main.h

```
00001 /* $Id$
00002 
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008 
00009 */
00010 
00011 GtkAssistantPageFunc gb_assistant_cb (GtkAssistant *assistant, GDateTime *datestamp);
```

4.18 garagejam-gingerblue-record.c

```
00001 /* $Id$
00002 
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008 
00009 */
00010 
00011 #include <string.h>
00012 #include <gst/gst.h>
00013 #include <signal.h>
00014 #include <unistd.h>
00015 #include <stdlib.h>
00016 #include <stdio.h>
00017 #include <string.h>
00018 
```

```
00019 // v4l2src ! tee name=t t. ! x264enc ! mp4mux ! filesink location=/home/rish/Desktop/okay.264 t. !
      videoconvert ! autovideosink
00020
00021 static GMainLoop *loop;
00022 static GstElement *pipeline, *audio_source, *sink, *src, *tee, *encoder,
00023     *muxer, *filesink, *videoconvert, *videosink, *queue_record,
00024     *queue_display;
00025 static GstBus *bus;
00026 static GstPad *teepad;
00027 static gboolean recording = FALSE;
00028 static gint counter = 0;
00029 static char *file_path;
00030
00031 static gboolean
00032 message_cb(GstBus *bus, GstMessage *message, gpointer user_data)
00033 {
00034     switch (GST_MESSAGE_TYPE(message)) {
00035     case GST_MESSAGE_ERROR:{
00036             GError *err = NULL;
00037             gchar *name, *debug = NULL;
00038
00039             name = gst_object_get_path_string(message->src);
00040             gst_message_parse_error(message, &err, &debug);
00041
00042             g_printerr("ERROR: from element %s: %s\n", name,
00043                     err->message);
00044             if (debug != NULL)
00045                 g_printerr("Additional debug info:\n%s\n",
00046                         debug);
00047
00048             g_error_free(err);
00049             g_free(debug);
00050             g_free(name);
00051
00052             g_main_loop_quit(loop);
00053             break;
00054         }
00055     case GST_MESSAGE_WARNING:{
00056             GError *err = NULL;
00057             gchar *name, *debug = NULL;
00058
00059             name = gst_object_get_path_string(message->src);
00060             gst_message_parse_warning(message, &err, &debug);
00061
00062             g_printerr("ERROR: from element %s: %s\n", name,
00063                    err->message);
00064             if (debug != NULL)
00065                 g_printerr("Additional debug info:\n%s\n",
00066                         debug);
00067
00068             g_error_free(err);
00069             g_free(debug);
00070             g_free(name);
00071             break;
00072         }
00073     case GST_MESSAGE_EOS:{
00074             g_print("Got EOS\n");
00075             g_main_loop_quit(loop);
00076             gst_element_set_state(pipeline, GST_STATE_NULL);
00077             g_main_loop_unref(loop);
00078             gst_object_unref(pipeline);
00079             exit(0);
00080             break;
00081         }
00082     default:
00083         break;
00084     }
00085
00086     return TRUE;
00087 }
00088
00089 static GstPadProbeReturn unlink_cb(GstPad *pad, GstPadProbeInfo *info,
00090                     gpointer user_data)
00091 {
00092     g_print("Unlinking...");
00093     GstPad *sinkpad;
00094     sinkpad = gst_element_get_static_pad(queue_record, "sink");
00095     gst_pad_unlink(teepad, sinkpad);
00096     gst_object_unref(sinkpad);
00097
00098     gst_element_send_event(encoder, gst_event_new_eos());
00099
00100     sleep(1);
00101     gst_bin_remove(GST_BIN(pipeline), queue_record);
00102     gst_bin_remove(GST_BIN(pipeline), encoder);
00103     gst_bin_remove(GST_BIN(pipeline), muxer);
00104     gst_bin_remove(GST_BIN(pipeline), filesink);
```

```
00105
00106     gst_element_set_state(queue_record, GST_STATE_NULL);
00107     gst_element_set_state(encoder, GST_STATE_NULL);
00108     gst_element_set_state(muxer, GST_STATE_NULL);
00109     gst_element_set_state(filesink, GST_STATE_NULL);
00110
00111     gst_object_unref(queue_record);
00112     gst_object_unref(encoder);
00113     gst_object_unref(muxer);
00114     gst_object_unref(filesink);
00115
00116     gst_element_release_request_pad(tee, teepad);
00117     gst_object_unref(teepad);
00118
00119     g_print("Unlinked\n");
00120
00121     return GST_PAD_PROBE_REMOVE;
00122 }
00123
00124 void stopRecording()
00125 {
00126     g_print("stopRecording\n");
00127     gst_pad_add_probe(teepad, GST_PAD_PROBE_TYPE_IDLE, unlink_cb, NULL,
00128             (GDestroyNotify) g_free);
00129     recording = FALSE;
00130 }
00131
00132 void startRecording()
00133 {
00134     g_print("startRecording\n");
00135     GstPad *sinkpad;
00136     GstPadTemplate *templ;
00137
00138     templ =
00139         gst_element_class_get_pad_template(GST_ELEMENT_GET_CLASS(tee),
00140                             "src_%u");
00141     teepad = gst_element_request_pad(tee, templ, NULL, NULL);
00142     queue_record = gst_element_factory_make("queue", "queue_record");
00143     encoder = gst_element_factory_make("x264enc", NULL);
00144     muxer = gst_element_factory_make("mp4mux", NULL);
00145     filesink = gst_element_factory_make("filesink", NULL);
00146     char *file_name = (char *) malloc(255 * sizeof(char));
00147     sprintf(file_name, "%s%d.mp4", file_path, counter++);
00148     g_print("Recording to file %s", file_name);
00149     g_object_set(filesink, "location", file_name, NULL);
00150     g_object_set(encoder, "tune", 4, NULL);
00151     free(file_name);
00152
00153     gst_bin_add_many(GST_BIN(pipeline), gst_object_ref(queue_record),
00154             gst_object_ref(encoder), gst_object_ref(muxer),
00155             gst_object_ref(filesink), NULL);
00156     gst_element_link_many(queue_record, encoder, muxer, filesink,
00157                 NULL);
00158
00159     gst_element_sync_state_with_parent(queue_record);
00160     gst_element_sync_state_with_parent(encoder);
00161     gst_element_sync_state_with_parent(muxer);
00162     gst_element_sync_state_with_parent(filesink);
00163
00164     sinkpad = gst_element_get_static_pad(queue_record, "sink");
00165     gst_pad_link(teepad, sinkpad);
00166     gst_object_unref(sinkpad);
00167
00168     recording = TRUE;
00169 }
00170
00171 int sigintHandler(int unused)
00172 {
00173     g_print("You ctrl-c!\n");
00174     if (recording)
00175         stopRecording();
00176     else
00177         startRecording();
00178     return 0;
00179 }
00180
00181 int gb_record_cb(char *path, gpointer data)
00182 {
00183     return 0;
00184 }
```

4.19 garagejam-gingerblue-record.h

```
00001 /* $Id$
```

```
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009 */
00010
00011 #include <string.h>
00012 #include <gst/gst.h>
00013 #include <signal.h>
00014 #include <unistd.h>
00015 #include <stdlib.h>
00016 #include <stdio.h>
00017 #include <string.h>
00018
00019 static gboolean message_cb (GstBus * bus, GstMessage * message, gpointer user_data);
00020 static GstPadProbeReturn unlink_cb(GstPad *pad, GstPadProbeInfo *info, gpointer user_data);
00021 void stopRecording();
00022 void startRecording();
00023 int sigintHandler(int unused);
00024 int gb_record_cb (gchar *path);
00025
00026 int garagejam_record_begin();
00027 int garagejam_record_end();
00028
00029 typedef struct _GingerblueRecord {
00030   gboolean recording_found;
00031 } GingerblueRecord;
```

4.20 garagejam-gingerblue-song.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <gst/gst.h>
00012 #include <gtk/gtk.h>
00013 #include <glib/gstdio.h>
00014 #include <glib/gi18n.h>
00015
00016 GtkWidget *gb_song_new(gchar *title)
00017 {
00018     GtkWidget *window;
00019     window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
00020     gtk_window_set_title(GTK_WINDOW(window), title);
00021     return (window);
00022 }
00023
00024 GtkWidget *gb_song_quit(gchar *title)
00025 {
00026     GtkWidget *window;
00027     window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
00028     gtk_window_set_title(GTK_WINDOW(window), title);
00029     return (window);
00030 }
```

4.21 garagejam-gingerblue-song.h

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009 */
00010
00011 #include <libxml/xmlmemory.h>
00012 #include <libxml/parser.h>
00013
00014 GtkWidget *gb_song_new (gchar *title);
00015 GtkWidget *gb_song_quit (gchar *title);
```

4.22 garagejam-gingerblue-storage.h

```
00001 #define GINGERBLUE_STORAGE 2000
00002
00003 typedef struct _GingerblueStorage {
00004   gboolean storage_allocated;
00005   gboolean storagemap;
00006 } GingerblueStorage;
00007
00008 void garagejam_storage_set(GingerblueStorage *storage, int storage_allocated, int storagemap);
```

4.23 garagejam-gingerblue-studio-config.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <gtk/gtk.h>
00012 #include <gst/gst.h>
00013 #include "garagejam.h"
00014
00015 GtkWidget *main_studio_config(gchar *location_data, gchar *studio_city)
00016 {
00017     GingerblueData *Gingerblue;
00018     GtkVBox *Locations;
00019     GtkListBox *Location;
00020     GtkContainer *Container;
00021     GtkWidget *Computer;
00022     GtkWidget *StudioLabel;
00023     Computer = gtk_list_box_row_new();
00024     StudioLabel = gtk_label_new(location_data);
00025     Locations = gtk_box_new(ATK_STATE_VERTICAL, 1);
00026     Location = gtk_list_box_new();
00027     gtk_container_add(GTK_CONTAINER(Computer), Locations);
00028     gtk_box_pack_start(GTK_BOX(Location), StudioLabel, TRUE, TRUE, 0);
00029     gtk_container_add(GTK_CONTAINER(Location), GTK_LIST_BOX(Computer));
00030     gtk_container_add(GTK_CONTAINER(Container), GTK_BOX(Locations));
00031     gtk_container_add(GTK_CONTAINER(Container),
00032               GTK_LIST_BOX(Location));
00033     gtk_widget_show_all(GTK_WIDGET(Container));
00034     return (GtkWidget *) Gingerblue;
00035 }
```

4.24 garagejam-gingerblue-studio-config.h

```
00001 #ifndef GINGERBLUE_STUDIO_CONFIG_H
00002 #define GINGERBLUE_STUDIO_CONFIG_H 1
00003
00004 GtkWidget *main_studio_config (gchar *location_data, gchar *studio_city);
00005
00006 #endif /* GINGERBLUE_STUDIO_CONFIG_H */
```

4.25 garagejam-gingerblue-studio-location.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <config.h>
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015 #include <sys/file.h>
```

```
00016 #include <gtk/gtk.h>
00017 #include <gst/gst.h>
00018 #include <gobject/glib-types.h>
00019 #include <gobject/gparam.h>
00020 #include <champlain/champlain.h>
00021 #include <champlain-gtk/champlain-gtk.h>
00022 #include <geoclue.h>
00023 #include <glib/gstdio.h>
00024 #include <glib/gi18n.h>
00025 #include <string.h>
00026 #include "garagejam.h"
00027
00028 extern GtkWidget *recording_entry;
00029 extern GtkWidget *studio_entry;
00030 extern GtkWidget *musician_entry;
00031 extern GtkWidget *song_entry;
00032 extern GtkWidget *label_entry;
00033
00034 /* Commandline options */
00035 static gint timeout = 30;   /* seconds */
00036 static GClueAccuracyLevel accuracy_level = GCLUE_ACCURACY_LEVEL_EXACT;
00037 static gint time_threshold;
00038
00039 static GOptionEntry entries[] = {
00040     { "timeout",
00041      't',
00042      0,
00043      G_OPTION_ARG_INT,
00044      &timeout,
00045      N_("Exit after T seconds. Default: 30"),
00046      "T" },
00047     { "time-threshold",
00048      'i',
00049      0,
00050      G_OPTION_ARG_INT,
00051      &time_threshold,
00052      N_("Only report location update after T seconds. "
00053         "Default: 0 (report new location without any delay)"),
00054      "T" },
00055     { "accuracy-level",
00056      'a',
00057      0,
00058      G_OPTION_ARG_INT,
00059      &accuracy_level,
00060      N_("Request accuracy level A. "
00061         "Country = 1, "
00062         "City = 4, " "Neighborhood = 5, " "Street = 6, " "Exact = 8."),
00063      "A" },
00064     { NULL }
00065 };
00066
00067 GClueSimple *simple = NULL;
00068 GClueClient *client = NULL;
00069 GMainLoop *main_loop;
00070
00071 static gchar *print_location(GClueSimple *simple, ChamplainView *view)
00072 {
00073     GClueLocation *location;
00074     gdouble altitude, speed, heading;
00075     GVariant *timestamp;
00076     GTimeVal tv = { 0 };
00077     const char *desc;
00078     gchar *geostring;
00079     location = gclue_simple_get_location(simple);
00080     g_print("\nNew location:\n");
00081     g_print("Latitude:    %f°\nLongitude:   %f°\n",
00082         gclue_location_get_latitude(location),
00083         gclue_location_get_longitude(location));
00084
00085     champlain_view_center_on(CHAMPLAIN_VIEW(view),
00086                  gclue_location_get_latitude(location),
00087                  gclue_location_get_longitude(location));
00088
00089     altitude = gclue_location_get_altitude(location);
00090     if (altitude != -G_MAXDOUBLE)
00091         g_print("Altitude:    %f meters\n", altitude);
00092     speed = gclue_location_get_speed(location);
00093     if (speed >= 0)
00094         g_print("Speed:       %f meters/second\n", speed);
00095     heading = gclue_location_get_heading(location);
00096     if (heading >= 0)
00097         g_print("Heading:     %f°\n", heading);
00098
00099     desc = gclue_location_get_description(location);
00100     if (strlen(desc) > 0)
00101         g_print("Description: %s\n", desc);
00102
```

```
00103     timestamp = gclue_location_get_timestamp(location);
00104     if (timestamp) {
00105         GDateTime *date_time;
00106         gchar *str;
00107
00108         g_variant_get(timestamp, "(tt)", &tv.tv_sec, &tv.tv_usec);
00109
00110         date_time = g_date_time_new_from_timeval_local(&tv);
00111         str = g_date_time_format
00112             (date_time, "%c (%s seconds since the Epoch)");
00113         g_date_time_unref(date_time);
00114
00115         g_print("Timestamp:   %s\n", str);
00116         g_free(str);
00117     }
00118     geostring =
00119         g_strconcat("  <glat>", gclue_location_get_latitude(location),
00120             "</glat>", "  <glon>",
00121             gclue_location_get_longitude(location), "</glon>",
00122             "  <galt>", gclue_location_get_altitude(location),
00123             "</galt>", NULL);
00124     return geostring;
00125 }
00126
00127 static gboolean on_location_timeout(gpointer user_data)
00128 {
00129     g_clear_object(&client);
00130     g_clear_object(&simple);
00131     g_main_loop_quit(main_loop);
00132
00133     return FALSE;
00134 }
00135
00136 static void
00137 on_client_active_notify(GClueClient *client,
00138             GParamSpec *pspec, gpointer user_data)
00139 {
00140     if (gclue_client_get_active(client))
00141         return;
00142
00143     g_print("Geolocation disabled. Quitting..\n");
00144     on_location_timeout(NULL);
00145 }
00146
00147 static void
00148 on_simple_ready(GObject *source_object,
00149         GAsyncResult *res, gpointer user_data)
00150 {
00151     GError *error = NULL;
00152
00153     simple = gclue_simple_new_finish(res, &error);
00154     if (error != NULL) {
00155         g_critical("Failed to connect to GeoClue2 service: %s",
00156                error->message);
00157
00158         exit(-1);
00159     }
00160     client = gclue_simple_get_client(simple);
00161     if (client) {
00162         g_object_ref(client);
00163         g_print("Client object: %s\n",
00164             g_dbus_proxy_get_object_path(G_DBUS_PROXY
00165                          (client)));
00166
00167         g_signal_connect(client,
00168                  "notify::active",
00169                  G_CALLBACK(on_client_active_notify),
00170                  NULL);
00171     }
00172     print_location(simple, user_data);
00173
00174     g_signal_connect(simple,
00175              "notify::location",
00176              G_CALLBACK(print_location), user_data);
00177 }
00178
00179
00180 gchar *studio_location_navigate(ChamplainView *view, gpointer *data)
00181 {
00182     GClueLocation *location;
00183     GClueSimple *simple;
00184     gdouble altitude, speed, heading;
00185     GVariant *timestamp;
00186     GTimeVal tv = { 0 };
00187     gclue_simple_new("garagejam",
00188              accuracy_level,
00189              time_threshold,
```

```
00190             on_simple_ready, CHAMPLAIN_VIEW(view));
00191     gtk_entry_set_text(GTK_ENTRY(data), print_location(simple, NULL));
00192     return print_location(simple, NULL);
00193 }
```

4.26 garagejam-gingerblue-studio-location.h

```
00001 #ifndef GINGERBLUE_STUDIO_LOCATION_H
00002 #define GINGERBLUE_STUDIO_LOCATION_H 1
00003
00004 #include <geoclue.h>
00005 #include <champlain/champlain.h>
00006
00007 static gchar *print_location (GClueSimple *simple, ChamplainView *view);
00008 static gboolean on_location_timeout (gpointer user_data);
00009 static void on_client_active_notify (GClueClient *client, GParamSpec *pspec, gpointer user_data);
00010 static void on_simple_ready (GObject *source_object, GAsyncResult *res, gpointer user_data);
00011 int studio_location_navigate (ChamplainView *view, gpointer *data);
00012
00013 #endif /* GINGERBLUE_STUDIO_LOCATION_H */
```

4.27 garagejam-gingerblue-studio-player-kb.h

```
00001 /* GStreamer command line playback testing utility - keyboard handling helpers
00002  *
00003  * Copyright (C) 2013 Tim-Philipp Müller <tim centricular net>
00004  * Copyright (C) 2013 Centricular Ltd
00005  *
00006  * This library is free software; you can redistribute it and/or
00007  * modify it under the terms of the GNU Library General Public
00008  * License as published by the Free Software Foundation; either
00009  * version 2 of the License, or (at your option) any later version.
00010  *
00011  * This library is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00014  * Library General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU Library General Public
00017  * License along with this library; if not, write to the
00018  * Free Software Foundation, Inc., 51 Franklin St, Fifth Floor,
00019  * Boston, MA 02110-1301, USA.
00020  */
00021 #ifndef __GST_PLAY_KB_INCLUDED__
00022 #define __GST_PLAY_KB_INCLUDED__
00023
00024 #include <glib.h>
00025
00026 #define GST_PLAY_KB_ARROW_UP    "\033[A"
00027 #define GST_PLAY_KB_ARROW_DOWN  "\033[B"
00028 #define GST_PLAY_KB_ARROW_RIGHT "\033[C"
00029 #define GST_PLAY_KB_ARROW_LEFT  "\033[D"
00030
00031 typedef void (*GstPlayKbFunc) (const gchar * kb_input, gpointer user_data);
00032
00033 gboolean gst_play_kb_set_key_handler (GstPlayKbFunc kb_func, gpointer user_data);
00034
00035 #endif /* __GST_PLAY_KB_INCLUDED__ */
```

4.28 garagejam-gingerblue-studio-player.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 /* GStreamer command line playback testing utility
00012  *
00013  * Copyright (C) 2013-2014 Tim-Philipp Müller <tim centricular net>
00014  * Copyright (C) 2013 Collabora Ltd.
```

```
00015  * Copyright (C) 2014 Sebastian Dröge <sebastian@centricular.com>
00016  * Copyright (C) 2015 Brijesh Singh <brijesh.ksingh@gmail.com>
00017  *
00018  * This library is free software; you can redistribute it and/or
00019  * modify it under the terms of the GNU Library General Public
00020  * License as published by the Free Software Foundation; either
00021  * version 2 of the License, or (at your option) any later version.
00022  *
00023  * This library is distributed in the hope that it will be useful,
00024  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00025  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00026  * Library General Public License for more details.
00027  *
00028  * You should have received a copy of the GNU Library General Public
00029  * License along with this library; if not, write to the
00030  * Free Software Foundation, Inc., 51 Franklin St, Fifth Floor,
00031  * Boston, MA 02110-1301, USA.
00032  */
00033
00034 #include <locale.h>
00035
00036 #include <gst/gst.h>
00037 #include <stdlib.h>
00038 #include <stdio.h>
00039 #include <string.h>
00040 #include <math.h>
00041
00042 #include <gst/player/player.h>
00043 #include "garagejam.h"
00044 #include "garagejam-gingerblue-studio-player.h"
00045 #include "garagejam-gingerblue-studio-player-kb.h"
00046
00047 #define VOLUME_STEPS 20
00048
00049 GST_DEBUG_CATEGORY(play_debug);
00050 #define GST_CAT_DEFAULT play_debug
00051
00052 extern GingerblueData *Gingerblue;
00053 extern GtkWidget *recording_entry;
00054 extern GstPlay *player;
00055 #if 0
00056 static gboolean play_next(GstPlay * play);
00057 static gboolean play_prev(GstPlay * play);
00058 static void play_reset(GstPlay * play);
00059 static void play_set_relative_volume(GstPlay * play, gdouble volume_step);
00060 #endif
00061
00062 void garagejam_studio_player_main(gchar *streamuri, gchar *name)
00063 {
00064
00065     gchar *uri = g_strdup(streamuri);
00066
00067     /* if (argc > 1) { */
00068     /*   if (g_strrstr (argv[1],"rtsp://") || g_strrstr (argv[1],"http://") || g_strrstr
      (argv[1],"file://")) */
00069     /*     uri = g_strdup (argv[1]); */
00070     /*   else */
00071     /*     uri = g_strdup_printf ("file://%s", argv[1]); */
00072     /* } */
00073     /* else { */
00074     /*   g_message ("Specify the media file name..<garagejam_studio-playerplayer filename>"); */
00075     /*   return 0; */
00076     /* } */
00077
00078     Gingerblue->player_loop = g_main_loop_new(NULL, FALSE);
00079     g_main_loop_run(Gingerblue->player_loop);
00080     gst_deinit();
00081     Gingerblue->player_status = GINGERBLUE_STUDIO_PLAYER_TRUE;
00082     exit(0);
00083 }
00084
00085 #if 0
00086 static void end_of_stream_cb(GstPlayer *player, GstPlay *play)
00087 {
00088     g_print("\n");
00089     /* and switch to next item in list */
00090     if (!play_next(play)) {
00091         g_print("Reached end of play list.\n");
00092         g_main_loop_quit(play->loop);
00093     }
00094 }
00095 #endif
00096 #if 0
00097 static void error_cb(GstPlayer *player, GError *err, GstPlay *play)
00098 {
00099     g_printerr("ERROR %s for %s\n", err->message,
00100            play->uris[play->cur_idx]);
```

```
00101     /* if looping is enabled, then disable it else will keep looping forever */
00102     play->repeat = FALSE;
00103     /* try next item in list then */
00104     if (!play_next(play)) {
00105         g_print("Reached end of play list.\n");
00106         g_main_loop_quit(play->loop);
00107     }
00108 }
00109 #endif
00110 static void
00111 position_updated_cb(GstPlayer *player, GstClockTime pos, GstPlay *play)
00112 {
00113     GstClockTime dur = -1;
00114     gchar status[64] = { 0, };
00115     g_object_get(play->player, "duration", &dur, NULL);
00116     memset(status, ' ', sizeof(status) - 1);
00117     if (pos != -1 && dur > 0 && dur != -1) {
00118         gchar dstr[32], pstr[32];
00119         /* FIXME: pretty print in nicer format */
00120         g_snprintf(pstr, 32, "%" GST_TIME_FORMAT,
00121                GST_TIME_ARGS(pos));
00122         pstr[9] = '\0';
00123         g_snprintf(dstr, 32, "%" GST_TIME_FORMAT,
00124                GST_TIME_ARGS(dur));
00125         dstr[9] = '\0';
00126         g_print("%s / %s %s\r", pstr, dstr, status);
00127     }
00128 }
00129
00130 static void
00131 state_changed_cb(GstPlayer *player, GstPlayerState state, GstPlay *play)
00132 {
00133     g_print("State changed: %s\n", gst_player_state_get_name(state));
00134 }
00135
00136 static void buffering_cb(GstPlayer *player, gint percent, GstPlay *play)
00137 {
00138     g_print("Buffering: %d\n", percent);
00139 }
00140
00141 static void
00142 print_one_tag(const GstTagList *list, const gchar *tag, gpointer user_data)
00143 {
00144     gint i, num;
00145     num = gst_tag_list_get_tag_size(list, tag);
00146     for (i = 0; i < num; ++i) {
00147         const GValue *val;
00148         val = gst_tag_list_get_value_index(list, tag, i);
00149         if (G_VALUE_HOLDS_STRING(val)) {
00150             g_print("    %s : %s \n", tag,
00151                 g_value_get_string(val));
00152         } else if (G_VALUE_HOLDS_UINT(val)) {
00153             g_print("    %s : %u \n", tag,
00154                 g_value_get_uint(val));
00155         } else if (G_VALUE_HOLDS_DOUBLE(val)) {
00156             g_print("    %s : %g \n", tag,
00157                 g_value_get_double(val));
00158         } else if (G_VALUE_HOLDS_BOOLEAN(val)) {
00159             g_print("    %s : %s \n", tag,
00160                 g_value_get_boolean(val) ? "true" :
00161                 "false");
00162         } else if (GST_VALUE_HOLDS_DATE_TIME(val)) {
00163             GstDateTime *dt = g_value_get_boxed(val);
00164             gchar *dt_str =
00165                 gst_date_time_to_iso8601_string(dt);
00166             g_print("    %s : %s \n", tag, dt_str);
00167             g_free(dt_str);
00168         } else {
00169             g_print("    %s : tag of type '%s' \n", tag,
00170                 G_VALUE_TYPE_NAME(val));
00171         }
00172     }
00173 }
00174
00175 static void print_video_info(GstPlayerVideoInfo *info)
00176 {
00177     gint fps_n, fps_d;
00178     guint par_n, par_d;
00179     if (info == NULL)
00180         return;
00181     g_print("  width : %d\n", gst_player_video_info_get_width(info));
00182     g_print("  height : %d\n", gst_player_video_info_get_height(info));
00183     g_print("  max_bitrate : %d\n",
00184         gst_player_video_info_get_max_bitrate(info));
00185     g_print("  bitrate : %d\n",
00186         gst_player_video_info_get_bitrate(info));
00187     gst_player_video_info_get_framerate(info, &fps_n, &fps_d);
```

```
00188     g_print("  framerate : %.2f\n", (gdouble) fps_n / fps_d);
00189     gst_player_video_info_get_pixel_aspect_ratio(info, &par_n, &par_d);
00190     g_print("  pixel-aspect-ratio  %u:%u\n", par_n, par_d);
00191 }
00192
00193 static void print_audio_info(GstPlayerAudioInfo *info)
00194 {
00195     if (info == NULL)
00196         return;
00197     g_print("  sample rate : %d\n",
00198         gst_player_audio_info_get_sample_rate(info));
00199     g_print("  channels : %d\n",
00200         gst_player_audio_info_get_channels(info));
00201     g_print("  max_bitrate : %d\n",
00202         gst_player_audio_info_get_max_bitrate(info));
00203     g_print("  bitrate : %d\n",
00204         gst_player_audio_info_get_bitrate(info));
00205     g_print("  language : %s\n",
00206         gst_player_audio_info_get_language(info));
00207 }
00208
00209 static void print_subtitle_info(GstPlayerSubtitleInfo *info)
00210 {
00211     if (info == NULL)
00212         return;
00213     g_print("  language : %s\n",
00214         gst_player_subtitle_info_get_language(info));
00215 }
00216
00217 static void print_all_stream_info(GstPlayerMediaInfo *media_info)
00218 {
00219     guint count = 0;
00220     GList *list, *l;
00221     g_print("URI : %s\n", gst_player_media_info_get_uri(media_info));
00222     g_print("Duration: %" GST_TIME_FORMAT "\n",
00223         GST_TIME_ARGS(gst_player_media_info_get_duration
00224                 (media_info)));
00225     g_print("Global taglist:\n");
00226     if (gst_player_media_info_get_tags(media_info))
00227         gst_tag_list_foreach(gst_player_media_info_get_tags
00228                     (media_info), print_one_tag, NULL);
00229     else
00230         g_print("  (nil) \n");
00231     list = gst_player_media_info_get_stream_list(media_info);
00232     if (!list)
00233         return;
00234     g_print("All Stream information\n");
00235     for (l = list; l != NULL; l = l->next) {
00236         GstTagList *tags = NULL;
00237         GstPlayerStreamInfo *stream =
00238             (GstPlayerStreamInfo *) l->data;
00239         g_print(" Stream # %u \n", count++);
00240         g_print("  type : %s_%u\n",
00241             gst_player_stream_info_get_stream_type(stream),
00242             gst_player_stream_info_get_index(stream));
00243         tags = gst_player_stream_info_get_tags(stream);
00244         g_print("  taglist : \n");
00245         if (tags) {
00246             gst_tag_list_foreach(tags, print_one_tag, NULL);
00247         }
00248         if (GST_IS_PLAYER_VIDEO_INFO(stream))
00249             print_video_info((GstPlayerVideoInfo *) stream);
00250         else if (GST_IS_PLAYER_AUDIO_INFO(stream))
00251             print_audio_info((GstPlayerAudioInfo *) stream);
00252         else
00253             print_subtitle_info((GstPlayerSubtitleInfo *)
00254                         stream);
00255     }
00256 }
00257
00258 static void print_all_video_stream(GstPlayerMediaInfo *media_info)
00259 {
00260     GList *list, *l;
00261     list = gst_player_get_video_streams(media_info);
00262     if (!list)
00263         return;
00264     g_print("All video streams\n");
00265     for (l = list; l != NULL; l = l->next) {
00266         GstPlayerVideoInfo *info = (GstPlayerVideoInfo *) l->data;
00267         GstPlayerStreamInfo *sinfo = (GstPlayerStreamInfo *) info;
00268         g_print(" %s_%d #\n",
00269             gst_player_stream_info_get_stream_type(sinfo),
00270             gst_player_stream_info_get_index(sinfo));
00271         print_video_info(info);
00272     }
00273 }
00274
```

```
00275 static void print_all_subtitle_stream(GstPlayerMediaInfo *media_info)
00276 {
00277     GList *list, *l;
00278     list = gst_player_get_subtitle_streams(media_info);
00279     if (!list)
00280         return;
00281     g_print("All subtitle streams:\n");
00282     for (l = list; l != NULL; l = l->next) {
00283         GstPlayerSubtitleInfo *info =
00284             (GstPlayerSubtitleInfo *) l->data;
00285         GstPlayerStreamInfo *sinfo = (GstPlayerStreamInfo *) info;
00286         g_print(" %s_%d #\n",
00287             gst_player_stream_info_get_stream_type(sinfo),
00288             gst_player_stream_info_get_index(sinfo));
00289         print_subtitle_info(info);
00290     }
00291 }
00292
00293 static void print_all_audio_stream(GstPlayerMediaInfo *media_info)
00294 {
00295     GList *list, *l;
00296     list = gst_player_get_audio_streams(media_info);
00297     if (!list)
00298         return;
00299     g_print("All audio streams: \n");
00300     for (l = list; l != NULL; l = l->next) {
00301         GstPlayerAudioInfo *info = (GstPlayerAudioInfo *) l->data;
00302         GstPlayerStreamInfo *sinfo = (GstPlayerStreamInfo *) info;
00303         g_print(" %s_%d #\n",
00304             gst_player_stream_info_get_stream_type(sinfo),
00305             gst_player_stream_info_get_index(sinfo));
00306         print_audio_info(info);
00307     }
00308 }
00309
00310 static void print_current_tracks(GstPlay *play)
00311 {
00312     GstPlayerAudioInfo *audio = NULL;
00313     GstPlayerVideoInfo *video = NULL;
00314     GstPlayerSubtitleInfo *subtitle = NULL;
00315     g_print("Current video track: \n");
00316     video = gst_player_get_current_video_track(play->player);
00317     print_video_info(video);
00318     g_print("Current audio track: \n");
00319     audio = gst_player_get_current_audio_track(play->player);
00320     print_audio_info(audio);
00321     g_print("Current subtitle track: \n");
00322     subtitle = gst_player_get_current_subtitle_track(play->player);
00323     print_subtitle_info(subtitle);
00324     if (audio)
00325         g_object_unref(audio);
00326     if (video)
00327         g_object_unref(video);
00328     if (subtitle)
00329         g_object_unref(subtitle);
00330 }
00331
00332 static void print_media_info(GstPlayerMediaInfo *media_info)
00333 {
00334     print_all_stream_info(media_info);
00335     g_print("\n");
00336     print_all_video_stream(media_info);
00337     g_print("\n");
00338     print_all_audio_stream(media_info);
00339     g_print("\n");
00340     print_all_subtitle_stream(media_info);
00341 }
00342
00343 static void
00344 media_info_cb(GstPlayer *player, GstPlayerMediaInfo *info, GstPlay *play)
00345 {
00346     static int once = 0;
00347     if (!once) {
00348         print_media_info(info);
00349         print_current_tracks(play);
00350         once = 1;
00351     }
00352 }
00353
00354 #if 0
00355 static GstPlay *play_new(gchar **uris, gdouble initial_volume)
00356 {
00357     GstPlay *play;
00358     play = g_new0(GstPlay, 1);
00359     play->uris = uris;
00360     play->num_uris = g_strv_length(uris);
00361     play->cur_idx = -1;
```

```
00362     play->player =
00363         gst_player_new(NULL,
00364                 gst_player_g_main_context_signal_dispatcher_new
00365                 (NULL));
00366     g_signal_connect(play->player, "position-updated",
00367             G_CALLBACK(position_updated_cb), play);
00368     g_signal_connect(play->player, "state-changed",
00369             G_CALLBACK(state_changed_cb), play);
00370     g_signal_connect(play->player, "buffering",
00371             G_CALLBACK(buffering_cb), play);
00372     g_signal_connect(play->player, "end-of-stream",
00373             G_CALLBACK(end_of_stream_cb), play);
00374     g_signal_connect(play->player, "error", G_CALLBACK(error_cb),
00375             play);
00376     g_signal_connect(play->player, "media-info-updated",
00377             G_CALLBACK(media_info_cb), play);
00378     play->loop = g_main_loop_new(NULL, FALSE);
00379     play->desired_state = GST_STATE_PLAYING;
00380     play_set_relative_volume(play, initial_volume - 1.0);
00381     return play;
00382 }
00383
00384 static void play_free(GstPlay *play)
00385 {
00386     play_reset(play);
00387     gst_object_unref(play->player);
00388     g_main_loop_unref(play->loop);
00389     g_strfreev(play->uris);
00390     g_free(play);
00391 }
00392 #endif
00393 /* reset for new file/stream */
00394 static void play_reset(GstPlay *play)
00395 {
00396 }
00397
00398 static void play_set_relative_volume(GstPlay *play, gdouble volume_step)
00399 {
00400     gdouble volume;
00401     g_object_get(play->player, "volume", &volume, NULL);
00402     volume =
00403         round((volume + volume_step) * VOLUME_STEPS) / VOLUME_STEPS;
00404     volume = CLAMP(volume, 0.0, 10.0);
00405     g_object_set(play->player, "volume", volume, NULL);
00406     g_print("Volume: %.0f%%                  \n", volume * 100);
00407 }
00408
00409 static gdouble get_volume(GtkWidget *widget, GstPlay *play)
00410 {
00411     gdouble volume;
00412     g_object_get(player, "volume", &volume, NULL);
00413     return volume;
00414 }
00415
00416 #if 0
00417 static gchar *play_uri_get_display_name(GstPlayer *player,
00418                     const gchar *uri)
00419 {
00420     gchar *loc;
00421     if (gst_uri_has_protocol(uri, "file")) {
00422         loc = g_filename_from_uri(uri, NULL, NULL);
00423     } else if (gst_uri_has_protocol(uri, "pushfile")) {
00424         loc = g_filename_from_uri(uri + 4, NULL, NULL);
00425     } else {
00426         loc = g_strdup(uri);
00427     }
00428     /* Maybe additionally use glib's filename to display name function */
00429     return loc;
00430 }
00431
00432 void play_uri(GstPlayer *player, const gchar *next_uri)
00433 {
00434     gchar *loc;
00435     play_reset(player);
00436     loc = play_uri_get_display_name(player, next_uri);
00437     g_print("Now playing %s\n", loc);
00438     g_free(loc);
00439     g_object_set(player, "uri", next_uri, NULL);
00440     gst_player_play(player);
00441 }
00442
00443 /* returns FALSE if we have reached the end of the playlist */
00444 #if 0
00445 static gboolean play_next(GstPlay *play)
00446 {
00447     if ((play->cur_idx + 1) >= play->num_uris) {
00448         if (play->repeat) {
```

```
00449             g_print("Looping playlist \n");
00450             play->cur_idx = -1;
00451         } else
00452             return FALSE;
00453     }
00454     play_uri(play, play->uris[++play->cur_idx]);
00455     return TRUE;
00456 }
00457
00458 /* returns FALSE if we have reached the beginning of the playlist */
00459 static gboolean play_prev(GstPlay *play)
00460 {
00461     if (play->cur_idx == 0 || play->num_uris <= 1)
00462         return FALSE;
00463     play_uri(play, play->uris[--play->cur_idx]);
00464     return TRUE;
00465 }
00466
00467 static void do_play(GstPlay *play)
00468 {
00469     gint i;
00470     /* dump playlist */
00471     for (i = 0; i < play->num_uris; ++i)
00472         GST_INFO("%4u : %s", i, play->uris[i]);
00473     if (!play_next(play))
00474         return;
00475     g_main_loop_run(play->loop);
00476 }
00477
00478 static void add_to_playlist(GPtrArray *playlist, const gchar *filename)
00479 {
00480     GDir *dir;
00481     gchar *uri;
00482     if (gst_uri_is_valid(filename)) {
00483         g_ptr_array_add(playlist, g_strdup(filename));
00484         return;
00485     }
00486     if ((dir = g_dir_open(filename, 0, NULL))) {
00487         const gchar *entry;
00488         /* FIXME: sort entries for each directory? */
00489         while ((entry = g_dir_read_name(dir))) {
00490             gchar *path;
00491             path =
00492                 g_strconcat(filename, G_DIR_SEPARATOR_S, entry,
00493                     NULL);
00494             add_to_playlist(playlist, path);
00495             g_free(path);
00496         }
00497         g_dir_close(dir);
00498         return;
00499     }
00500     uri = gst_filename_to_uri(filename, NULL);
00501     if (uri != NULL)
00502         g_ptr_array_add(playlist, uri);
00503     else
00504         g_warning("Could not make URI out of filename '%s'",
00505               filename);
00506 }
00507
00508 static void shuffle_uris(gchar **uris, guint num)
00509 {
00510     gchar *tmp;
00511     guint i, j;
00512     if (num < 2)
00513         return;
00514     for (i = 0; i < num; i++) {
00515         /* gets equally distributed random number in 0..num-1 [0;num[ */
00516         j = g_random_int_range(0, num);
00517         tmp = uris[j];
00518         uris[j] = uris[i];
00519         uris[i] = tmp;
00520     }
00521 }
00522
00523 static void restore_terminal(void)
00524 {
00525     gst_play_kb_set_key_handler(NULL, NULL);
00526 }
00527
00528 static void toggle_paused(GstPlay *play)
00529 {
00530     if (play->desired_state == GST_STATE_PLAYING) {
00531         play->desired_state = GST_STATE_PAUSED;
00532         gst_player_pause(play->player);
00533     } else {
00534         play->desired_state = GST_STATE_PLAYING;
00535         gst_player_play(play->player);
```

```
00536     }
00537 }
00538
00539 static void relative_seek(GstPlay *play, gdouble percent)
00540 {
00541     gint64 dur = -1, pos = -1;
00542     g_return_if_fail(percent >= -1.0 && percent <= 1.0);
00543     g_object_get(play->player, "position", &pos, "duration", &dur,
00544             NULL);
00545     if (dur <= 0) {
00546         g_print("\nCould not seek.\n");
00547         return;
00548     }
00549     pos = pos + dur * percent;
00550     if (pos < 0)
00551         pos = 0;
00552     gst_player_seek(play->player, pos);
00553 }
00554
00555 static void keyboard_cb(const gchar *key_input, gpointer user_data)
00556 {
00557     GstPlay *play = (GstPlay *) user_data;
00558     switch (g_ascii_tolower(key_input[0])) {
00559     case 'i':
00560         {
00561             GstPlayerMediaInfo *media_info =
00562                 gst_player_get_media_info(play->player);
00563             if (media_info) {
00564                 print_media_info(media_info);
00565                 g_object_unref(media_info);
00566                 print_current_tracks(play);
00567             }
00568             break;
00569         }
00570     case ' ':
00571         toggle_paused(play);
00572         break;
00573     case 'q':
00574     case 'Q':
00575         g_main_loop_quit(play->loop);
00576         break;
00577     case '>':
00578         if (!play_next(play)) {
00579             g_print("\nReached end of play list.\n");
00580             g_main_loop_quit(play->loop);
00581         }
00582         break;
00583     case '<':
00584         play_prev(play);
00585         break;
00586     case 27:        /* ESC */
00587         if (key_input[1] == '\0') {
00588             g_main_loop_quit(play->loop);
00589             break;
00590         }
00591         /* fall through */
00592     default:
00593         if (strcmp(key_input, GST_PLAY_KB_ARROW_RIGHT) == 0) {
00594             relative_seek(play, +0.08);
00595         } else if (strcmp(key_input, GST_PLAY_KB_ARROW_LEFT) == 0) {
00596             relative_seek(play, -0.01);
00597         } else if (strcmp(key_input, GST_PLAY_KB_ARROW_UP) == 0) {
00598             play_set_relative_volume(play,
00599                         +1.0 / VOLUME_STEPS);
00600         } else if (strcmp(key_input, GST_PLAY_KB_ARROW_DOWN) == 0) {
00601             play_set_relative_volume(play,
00602                         -1.0 / VOLUME_STEPS);
00603         } else {
00604             GST_INFO("keyboard input:");
00605             for (; *key_input != '\0'; ++key_input)
00606                 GST_INFO("  code %3d", *key_input);
00607         }
00608         break;
00609     }
00610 }
00611 #endif
00612 #endif
00613
00614 void garagejam_studio_player_new(GstPlayer *player, const gchar *next_uri)
00615 {
00616     gst_player_set_uri(player, next_uri);
00617 }
00618
00619 void garagejam_studio_player_stop(GstPlayer *player)
00620 {
00621     if (player != NULL) {
00622         gst_player_stop(GST_PLAYER(player));
```

```
00623     }
00624 }
00625
00626 void garagejam_studio_player_pause(GstPlayer *player)
00627 {
00628     /* FIXME: Unable to quit after pause is called for the first time. */
00629 #if 0
00630     gst_player_pause(player);
00631 #endif
00632 }
00633
00634 #if 0
00635 int main(int argc, char **argv)
00636 {
00637     GstPlay *play;
00638     GPtrArray *playlist;
00639     gboolean print_version = FALSE;
00640     gboolean interactive = FALSE;   /* FIXME: maybe enable by default? */
00641     gboolean shuffle = FALSE;
00642     gboolean repeat = FALSE;
00643     gdouble volume = 1.0;
00644     gchar **filenames = NULL;
00645     gchar **uris;
00646     guint num, i;
00647     GError *err = NULL;
00648     GOptionContext *ctx;
00649     gchar *playlist_file = NULL;
00650     GOptionEntry options[] = {
00651         { "version", 0, 0, G_OPTION_ARG_NONE, &print_version,
00652          "Print version information and exit", NULL },
00653         { "shuffle", 0, 0, G_OPTION_ARG_NONE, &shuffle,
00654          "Shuffle playlist", NULL },
00655         { "interactive", 0, 0, G_OPTION_ARG_NONE, &interactive,
00656          "Interactive control via keyboard", NULL },
00657         { "volume", 0, 0, G_OPTION_ARG_DOUBLE, &volume,
00658          "Volume", NULL },
00659         { "playlist", 0, 0, G_OPTION_ARG_FILENAME, &playlist_file,
00660          "Playlist file containing input media files", NULL },
00661         { "loop", 0, 0, G_OPTION_ARG_NONE, &repeat, "Repeat all",
00662          NULL },
00663         { G_OPTION_REMAINING, 0, 0, G_OPTION_ARG_FILENAME_ARRAY,
00664          &filenames, NULL },
00665         { NULL }
00666     };
00667     g_set_prgname("gst-play");
00668     ctx =
00669         g_option_context_new
00670         ("FILE1|URI1 [FILE2|URI2] [FILE3|URI3] ...");
00671     g_option_context_add_main_entries(ctx, options, NULL);
00672     g_option_context_add_group(ctx, gst_init_get_option_group());
00673     if (!g_option_context_parse(ctx, &argc, &argv, &err)) {
00674         g_print("Error initializing: %s\n",
00675             GST_STR_NULL(err->message));
00676         g_clear_error(&err);
00677         g_option_context_free(ctx);
00678         return 1;
00679     }
00680     g_option_context_free(ctx);
00681     GST_DEBUG_CATEGORY_INIT(play_debug, "play", 0, "gst-play");
00682     if (print_version) {
00683         gchar *version_str;
00684         version_str = gst_version_string();
00685         g_print("%s version %s\n", g_get_prgname(), "1.0");
00686         g_print("%s\n", version_str);
00687         g_free(version_str);
00688         g_free(playlist_file);
00689         return 0;
00690     }
00691     playlist = g_ptr_array_new();
00692     if (playlist_file != NULL) {
00693         gchar *playlist_contents = NULL;
00694         gchar **lines = NULL;
00695         if (g_file_get_contents
00696             (playlist_file, &playlist_contents, NULL, &err)) {
00697             lines = g_strsplit(playlist_contents, "\n", 0);
00698             num = g_strv_length(lines);
00699             for (i = 0; i < num; i++) {
00700                 if (lines[i][0] != '\0') {
00701                     GST_LOG("Playlist[%d]: %s", i + 1,
00702                         lines[i]);
00703                     add_to_playlist(playlist,
00704                             lines[i]);
00705                 }
00706             }
00707             g_strfreev(lines);
00708             g_free(playlist_contents);
00709         } else {
```

```
00710             g_printerr("Could not read playlist: %s\n",
00711                    err->message);
00712             g_clear_error(&err);
00713         }
00714         g_free(playlist_file);
00715         playlist_file = NULL;
00716     }
00717     if (playlist->len == 0
00718         && (filenames == NULL || *filenames == NULL)) {
00719         g_printerr
00720             ("Usage: %s FILE1|URI1 [FILE2|URI2] [FILE3|URI3] ...",
00721              "gst-play");
00722         g_printerr("\n\n"), g_printerr("%s\n\n",
00723                                "You must provide at least one filename or URI to play.");
00724         /* No input provided. Free array */
00725         g_ptr_array_free(playlist, TRUE);
00726         return 1;
00727     }
00728     /* fill playlist */
00729     if (filenames != NULL && *filenames != NULL) {
00730         num = g_strv_length(filenames);
00731         for (i = 0; i < num; ++i) {
00732             GST_LOG("command line argument: %s", filenames[i]);
00733             add_to_playlist(playlist, filenames[i]);
00734         }
00735         g_strfreev(filenames);
00736     }
00737     num = playlist->len;
00738     g_ptr_array_add(playlist, NULL);
00739     uris = (gchar **) g_ptr_array_free(playlist, FALSE);
00740     if (shuffle)
00741         shuffle_uris(uris, num);
00742     /* prepare */
00743     play = play_new(uris, volume);
00744     play->repeat = repeat;
00745     if (interactive) {
00746         if (gst_play_kb_set_key_handler(keyboard_cb, play)) {
00747             atexit(restore_terminal);
00748         } else {
00749             g_print
00750                 ("Interactive keyboard handling in terminal not available.\n");
00751         }
00752     }
00753     /* play */
00754     do_play(play);
00755     /* clean up */
00756     play_free(play);
00757     g_print("\n");
00758     gst_deinit();
00759     return 0;
00760 }
00761 #endif
00762 void garagejam_studio_player_play(GtkWidget *widget,
00763                   gpointer *recording_entry)
00764 {
00765     garagejam_studio_player_stop(player);
00766     player =
00767         gst_player_new(NULL,
00768                gst_player_g_main_context_signal_dispatcher_new
00769                (NULL));
00770     /* g_object_set_data(G_OBJECT(widget), "station_uri", g_value_get_string(&value)); */
00771     if (!g_strcmp0
00772         (gtk_entry_get_text(GTK_ENTRY(recording_entry)), NULL)) {
00773         garagejam_studio_player_new(player, recording_entry);
00774     } else {
00775         garagejam_studio_player_new(player, recording_entry);
00776     }
00777     gst_player_play(player);
00778     return;
00779 }
```

4.29 garagejam-gingerblue-studio-player.h

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
```

```
00011 /* $Id$
00012  *
00013  * GNOME Internet Radio Locator
00014  *
00015  * Copyright (C) 2014-2019  Aamot Software
00016  *
00017  * Author: Ole Aamot <ole@gnome.org>
00018  *
00019  * This program is free software; you can redistribute it and/or modify
00020  * it under the terms of the GNU General Public License as published by
00021  * the Free Software Foundation; either version 2 of the License, or
00022  * (at your option) any later version.
00023  *
00024  * This program is distributed in the hope that it will be useful,
00025  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00026  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00027  * GNU General Public License for more details.
00028  *
00029  * You should have received a copy of the GNU General Public License
00030  * along with this program; if not, write to the Free Software
00031  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
00032  */
00033
00034 #ifndef GINGERBLUE_STUDIO_PLAYER_PLAYER_H
00035 #define GINGERBLUE_STUDIO_PLAYER_PLAYER_H
00036
00037 #include <gtk/gtk.h>
00038 #include <gst/gst.h>
00039 #include <gst/player/player.h>
00040
00041 typedef struct
00042 {
00043   gchar **uris;
00044   guint num_uris;
00045   gint cur_idx;
00046
00047   GstPlayer *player;
00048   GstState desired_state;
00049
00050   gboolean repeat;
00051
00052   GMainLoop *loop;
00053 } GstPlay;
00054
00055 void play_uri (GstPlayer *player, const gchar * next_uri);
00056
00057 void garagejam_studio_player_new (GstPlayer * player, const gchar * next_uri);
00058
00059 void garagejam_studio_player_new (GstPlayer * player, const gchar * next_uri);
00060
00061 void garagejam_studio_player_quit (GstPlayer *player);
00062
00063 void garagejam_studio_player_pause (GstPlayer *player);
00064
00065 void garagejam_studio_player_stop (GstPlayer *player);
00066
00067 void garagejam_studio_player_play (GtkWidget *widget, gpointer *recording_entry);
00068
00069 static gdouble get_volume (GtkWidget *widget, GstPlay *play);
00070
00071 #endif /* GINGERBLUE_STUDIO_PLAYER_H */
```

4.30 garagejam-gingerblue-studio-stream.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2023 Aamot Broadcast
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 0.5.0 (2023-08-07)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <stdio.h>
00012 #include <stdlib.h>
00013 #include <string.h>
00014 #include <unistd.h>
00015 #include <arpa/inet.h>
00016 #include <sys/file.h>
00017 #include <gtk/gtk.h>
00018 #include <gst/gst.h>
00019 #include <gobject/glib-types.h>
```

```
00020 #include <gobject/gparam.h>
00021 #include <shout/shout.h>
00022 #include <gst/player/player.h>
00023 #include "garagejam.h"
00024
00025 #define SERVER_IP "178.255.144.178" // Replace with your server IP
00026 #define SERVER_PORT 12348   // Replace with your server port
00027
00028 extern GtkWidget *recording_entry;
00029 extern GtkWidget *studio_entry;
00030 extern GtkWidget *musician_entry;
00031 extern GtkWidget *song_entry;
00032 extern GtkWidget *label_entry;
00033 extern gchar xspfbuffer[8196];
00034 int main_studio_stream(gchar *location_data, gpointer *studio_city)
00035 {
00036
00037     GstPlayer *player;
00038     int sockfd;
00039     struct sockaddr_in server_addr;
00040
00041     // Create socket
00042     sockfd = socket(AF_INET, SOCK_STREAM, 0);
00043     if (sockfd < 0) {
00044         perror("Socket creation error");
00045         return 1;
00046     }
00047
00048     // Configure server address
00049     memset(&server_addr, 0, sizeof(server_addr));
00050     server_addr.sin_family = AF_INET;
00051     server_addr.sin_port = htons(SERVER_PORT);
00052     if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) <= 0) {
00053         perror("Invalid address or address family");
00054         return 1;
00055     }
00056
00057     // Connect to the server
00058     if (connect
00059         (sockfd, (struct sockaddr *) &server_addr,
00060          sizeof(server_addr)) < 0) {
00061         perror("Connection failed");
00062         return 1;
00063     }
00064
00065     // Simulated audio data (replace with actual audio capture)
00066     const char *audio_data = g_strdup(xspfbuffer);
00067     size_t audio_len = strlen(audio_data);
00068
00069     // Send audio data to server
00070     ssize_t bytes_sent = send(sockfd, audio_data, audio_len, 0);
00071     if (bytes_sent < 0) {
00072         perror("Send error");
00073         close(sockfd);
00074         return 1;
00075     }
00076
00077     printf("Sent %zd bytes of audio data\n", bytes_sent);
00078
00079     // Close the socket
00080     close(sockfd);
00081
00082     player =
00083         gst_player_new(NULL,
00084                gst_player_g_main_context_signal_dispatcher_new
00085                (NULL));
00086     gst_player_set_uri(GST_PLAYER(player),
00087                g_strconcat("file://",
00088                        gtk_entry_get_text(GTK_ENTRY
00089                                (recording_entry)),
00090                        NULL));
00091     gst_player_play(GST_PLAYER(player));
00092
00093     /* g_free (garagejam_data); */
00094
00095     /* g_date_time_unref (datestamp); */
00096
00097     return 0;
00098 }
```

4.31 garagejam-gingerblue-studio-stream.h

```
00001 #ifndef GINGERBLUE_STUDIO_STREAM_H
```

```
00002 #define GINGERBLUE_STUDIO_STREAM_H 1
00003
00004 gint main_studio_stream (gchar *location_data, gchar *studio_city);
00005
00006 #endif /* GINGERBLUE_STUDIO_STREAM_H */
```

4.32 garagejam-gingerblue-wizard.h

```
00001 #include "garagejam-gingerblue-storage.h"
00002
00003 #define GINGERBLUE_WIZARD 10000
00004
00005 void garagejam_wizard_new (GINGERBLUE_WIZARD);
00006
00007 typedef struct _GingerblueWizard {
00008   gboolean wizard_run;
00009   GtkWindow *window;
00010   GingerblueStorage *storage;
00011   GtkFile *metadata;
00012   char *stream;
00013 } GingerblueWizard;
```

4.33 garagejam.c

```
00001 /* $Id$
00002
00003    Copyright (C) 2023 Aamot Broadcast
00004    Author(s): Ole Aamot <ole@aamot.org>
00005    License: GNU GPL version 3
00006    Version: 0.0.1 (2023-04-30)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #include <glib/gi18n.h>
00012 #include <gst/gst.h>
00013 #include <gtk/gtk.h>
00014 #include "garagejam.h"
00015 #include "garagejam-gingerblue-file.h"
00016
00017 gint gb_exit(void)
00018 {
00019     gst_deinit();
00020     gtk_main_quit();
00021 }
00022
00023 void gb_window_break_record(GtkButton *record, GtkVolumeButton *volume)
00024 {
00025     /* gtk_button_set_label(GTK_BUTTON (cue), "Continue Recording"); */
00026     /* g_signal_connect (GTK_BUTTON (cue), "clicked", G_CALLBACK (gb_window_new_record),
      garagejam_data->volume); */
00027 }
00028
00029 void gb_window_pause_record(GtkButton *record, GtkVolumeButton *volume)
00030 {
00031     /* gtk_button_set_label(GTK_BUTTON (cue), "Continue Recording"); */
00032     /* g_signal_connect (GTK_BUTTON (cue), "clicked", G_CALLBACK (gb_window_new_record),
      garagejam_data->volume); */
00033 }
00034
00035 GingerblueData *gb_window_new_record(GtkButton *record,
00036                      GtkVolumeButton *volume)
00037 {
00038     /* gtk_button_set_label(GTK_BUTTON (record), "Stop Recording"); */
00039 }
00040
00041 GingerblueData *gb_window_store_volume(GtkButton *record,
00042                        GtkVolumeButton *volume)
00043 {
00044     /* gtk_button_set_label(GTK_BUTTON (record), "Stop Recording"); */
00045 }
00046
00047 gdouble gb_window_set_volume(GtkVolumeButton *volume, gdouble value)
00048 {
00049     gtk_scale_button_set_value(GTK_SCALE_BUTTON(volume),
00050                    (gdouble) value);
00051 }
00052
00053 gdouble gb_window_new_volume(GtkVolumeButton *volume, gchar *msg)
```

```
00054 {
00055     g_print("New volume: %0.2f\n",
00056         (gdouble)
00057         gtk_scale_button_get_value(GTK_SCALE_BUTTON(volume)));
00058     return (gdouble)
00059         gtk_scale_button_get_value(GTK_SCALE_BUTTON(volume));
00060 }
00061
00062 gdouble gb_window_get_volume(GtkVolumeButton *volume)
00063 {
00064     return (gdouble)
00065         gtk_scale_button_get_value(GTK_SCALE_BUTTON(volume));
00066 }
```

4.34 garagejam.h

```
00001 /* $Id$
00002
00003    Copyright (C) 2020-2022 Aamot Software
00004    Author(s): Ole Aamot <ole@gnome.org>
00005    License: GNU GPL version 3
00006    Version: 6.2.0 (2022-07-09)
00007    Website: http://www.garagejam.org/
00008
00009  */
00010
00011 #ifndef _GINGERBLUE_H_
00012 #define _GINGERBLUE_H_ 1
00013
00014 #include <gtk/gtk.h>
00015
00016 #define GINGERBLUE_STUDIO_PLAYER_TRUE 1
00017 #define GINGERBLUE_STUDIO_PLAYER_FALSE 0
00018
00019 typedef struct _GingerblueData GingerblueData;
00020
00021 struct _GingerblueData {
00022     GtkWidget *knob;
00023     gint player_status;
00024     gchar *line;
00025     gint jack;
00026     gchar *label;
00027     gboolean lpf;
00028     gboolean hpf;
00029     gchar *musician;
00030     gchar *musical_instrument;
00031     gchar *version;
00032     gchar *volume;
00033     GingerblueData *next;
00034     GingerblueData *prev;
00035     GtkWidget *window;
00036     GMainLoop *player_loop;
00037 };
00038
00039 void gb_window_break_record (GtkButton *record, GtkVolumeButton *volume);
00040 void gb_window_pause_record (GtkButton *record, GtkVolumeButton *volume);
00041 GingerblueData *gb_window_new_record (GtkButton *record, GtkVolumeButton *volume);
00042 GingerblueData *gb_window_store_volume (GtkButton *record, GtkVolumeButton *volume);
00043 gdouble gb_window_set_volume (GtkVolumeButton *volume, gdouble value);
00044 gdouble gb_window_new_volume (GtkVolumeButton *volume, gchar *msg);
00045 gdouble gb_window_get_volume (GtkVolumeButton *volume);
00046
00047 gint gb_exit (void);
00048
00049 #endif /* _GINGERBLUE_H_ */
```

Index