

Contents

| | | |
|----|-------------------------|-----|
| 1 | app-menu.ui | 3 |
| 2 | button_subroutines.c | 5 |
| 3 | button_subroutines.h | 14 |
| 4 | display_subroutines.c | 15 |
| 5 | display_subroutines.h | 19 |
| 6 | gstenvelope.c | 20 |
| 7 | gstenvelope.h | 51 |
| 8 | gstlooper.c | 54 |
| 9 | gstlooper.h | 110 |
| 10 | gstreamer_subroutines.c | 114 |
| 11 | gstreamer_subroutines.h | 128 |
| 12 | main.c | 129 |
| 13 | main.h | 133 |
| 14 | menu_subroutines.c | 134 |
| 15 | menu_subroutines.h | 142 |
| 16 | message_subroutines.c | 143 |
| 17 | message_subroutines.h | 150 |
| 18 | network_subroutines.c | 151 |
| 19 | network_subroutines.h | 158 |
| 20 | parse_net_subroutines.c | 159 |
| 21 | parse_net_subroutines.h | 163 |
| 22 | parse_xml_subroutines.c | 164 |
| 23 | parse_xml_subroutines.h | 203 |
| 24 | preferences.ui | 204 |

| | |
|--|---------------------|
| 25 sequence__structure.h | 208 |
| 26 sequence__subroutines.c | 211 |
| 27 sequence__subroutines.h | 239 |
| 28 signal__subroutines.c | 241 |
| 29 signal__subroutines.h | 244 |
| 30 sound__effects__player.c | 245 |
| 31 sound__effects__player.h | 262 |
| 32 sound__effects__player.ui | 266 |
| 33 sound__structure.h | 326 |
| 34 sound__subroutines.c | 328 |
| 35 sound__subroutines.h | 338 |
| 36 timer__subroutines.c | 340 |
| 37 timer__subroutines.h | 345 |

1 app-menu.ui

```

<?xml version="1.0"?>
<interface>
  <!-- interface-requires gtk+ 3.14 -->
  <menu id="appmenu">
    <section>
      <item>
        <attribute name="label" translatable="yes">_Preferences</attribute>
        <attribute name="action">app.preferences</attribute>
      </item>
    </section>
    <section>
      <item>
        <attribute name="label" translatable="yes">_Quit</attribute>
        <attribute name="action">app.quit</attribute>
        <attribute name="accel">&lt;Primary&gt;q</attribute>
      </item>
    </section>
  </menu>
  <menu id="menubar">
    <submenu>
      <attribute name="label" translatable="yes">_File</attribute>
      <section>
        <item>
          <attribute name="label" translatable="yes">_New</attribute>
          <attribute name="action">app.new</attribute>
          <attribute name="accel">&lt;Primary&gt;n</attribute>
        </item>
        <item>
          <attribute name="label" translatable="yes">_Open</attribute>
          <attribute name="action">app.open</attribute>
          <attribute name="accel">&lt;Primary&gt;o</attribute>
        </item>
        <item>
          <attribute name="label" translatable="yes">_Save</attribute>
          <attribute name="action">app.save</attribute>
          <attribute name="accel">&lt;Primary&gt;s</attribute>
        </item>
        <item>
          <attribute name="label" translatable="yes">Save _As</attribute>
          <attribute name="action">app.save_as</attribute>
          <attribute name="accel">&lt;Primary&gt;a</attribute>
        </item>
      </section>
    </submenu>
  </menu>

```

```
<submenu>
  <attribute name="label" translatable="yes">_Edit</attribute>
  <section>
    <item>
      <attribute name="label" translatable="yes">_Copy</attribute>
      <attribute name="action">app.copy</attribute>
      <attribute name="accel">&lt;Primary&gt;c</attribute>
    </item>
    <item>
      <attribute name="label" translatable="yes">_Cut</attribute>
      <attribute name="action">app.cut</attribute>
      <attribute name="accel">&lt;Primary&gt;x</attribute>
    </item>
    <item>
      <attribute name="label" translatable="yes">_Paste</attribute>
      <attribute name="action">app.paste</attribute>
      <attribute name="accel">&lt;Primary&gt;v</attribute>
    </item>
  </section>
</submenu>
</menu>
</interface>
```

2 button_subroutines.c

```

/*
 * button_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John.Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
#include "button_subroutines.h"
#include "gstreamer_subroutines.h"
#include "sound_effects_player.h"
#include "sound_subroutines.h"
#include "sound_structure.h"
#include "sequence_subroutines.h"

/* The Mute button has been toggled. */
void
button_mute_toggled (GtkToggleButton * button, gpointer user_data)
{
    GApplication *app;
    gboolean button_state;
    GstPipeline *pipeline_element;
    GstElement *volume_element;
    GstElement *final_bin_element;

    app = sep_get_application_from_widget (user_data);
    button_state = gtk_toggle_button_get_active (button);
    pipeline_element = sep_get_pipeline_from_app (app);
    if (pipeline_element == NULL)
        return;

    /* Find the final bin. */
    final_bin_element =
        gst_bin_get_by_name (GST_BIN (pipeline_element), (gchar *) "final");

```

```
if (final_bin_element == NULL)
    return;

/* Find the volume element in the final bin */
volume_element = gstreamer_get_volume (GST_BIN (final_bin_element));
if (volume_element == NULL)
    return;

/* Set the mute property of the volume element based on whether
 * the mute button has been activated or deactivated. */
g_object_set (volume_element, "mute", button_state, NULL);

return;
}

/* The Pause button has been pushed. */
void
button_pause_clicked (GtkButton * button, gpointer user_data)
{
    GApplication *app;

    app = sep_get_application_from_widget (user_data);
    sound_button_pause (app);

    return;
}

/* The Continue button has been pushed. */
void
button_continue_clicked (GtkButton * button, gpointer user_data)
{
    GApplication *app;

    app = sep_get_application_from_widget (user_data);
    sound_button_continue (app);

    return;
}

/* The Play button has been pushed. */
void
button_play_clicked (GtkButton * button, gpointer user_data)
{
    GApplication *app;

    /* Let the internal sequencer handle it. */
```

```
app = sep_get_application_from_widget (user_data);
sequence_button_play (app);

return;
}

/* The Start button in a cluster has been pushed. */
void
button_start_clicked (GtkButton * button, gpointer user_data)
{
    GApplication *app;
    GtkWidget *cluster_widget;
    guint cluster_number;

    /* Let the internal sequencer handle it. */
    app = sep_get_application_from_widget (user_data);
    cluster_widget = sep_get_cluster_from_widget (user_data);
    cluster_number = sep_get_cluster_number (cluster_widget);
    sequence_cluster_start (cluster_number, app);

    return;
}

/* The Stop button in a cluster has been pushed. */
void
button_stop_clicked (GtkButton * button, gpointer user_data)
{
    GApplication *app;
    GtkWidget *cluster_widget;
    guint cluster_number;

    /* Let the internal sequencer handle it. */
    app = sep_get_application_from_widget (user_data);
    cluster_widget = sep_get_cluster_from_widget (user_data);
    cluster_number = sep_get_cluster_number (cluster_widget);
    sequence_cluster_stop (cluster_number, app);

    return;
}

/* Show that the Start button has been pushed. */
void
button_set_cluster_playing (struct sound_info *sound_data, GApplication * app)
{
    GtkButton *start_button = NULL;
    GtkWidget *parent_container;
```

```

GList *children_list = NULL;
const gchar *child_name = NULL;

/* Find the start button and set its text to "Playing...".
 * The start button will be a child of the cluster, and will be named
 * "start_button". */
parent_container = sound_data->cluster_widget;

/* It is possible, though unlikely, that the sound will no longer
 * be in a cluster. */
if (parent_container != NULL)
{
    children_list =
        gtk_container_get_children (GTK_CONTAINER (parent_container));
    while (children_list != NULL)
    {
        child_name = gtk_widget_get_name (children_list->data);
        if (g_strcmp0 (child_name, "start_button") == 0)
        {
            start_button = children_list->data;
            break;
        }
        children_list = children_list->next;
    }
    g_list_free (children_list);
    gtk_button_set_label (start_button, "Playing...");
}

return;
}

/* Show that the release stage of a sound is running. */
void
button_set_cluster_releasing (struct sound_info *sound_data,
                             GApplication * app)
{
    GtkWidget *start_button = NULL;
    GtkWidget *parent_container;
    GList *children_list = NULL;
    const gchar *child_name = NULL;

    /* Find the start button and set its text to "Releasing...".
     * The start button will be a child of the cluster, and will be named
     * "start_button". */
    parent_container = sound_data->cluster_widget;

```



```

/* It is possible, though unlikely, that the sound will no longer
 * be in a cluster. */
if (parent_container != NULL)
{
    children_list =
        gtk_container_get_children (GTK_CONTAINER (parent_container));
    while (children_list != NULL)
    {
        child_name = gtk_widget_get_name (children_list->data);
        if (g_strcmp0 (child_name, "start_button") == 0)
        {
            start_button = children_list->data;
            break;
        }
        children_list = children_list->next;
    }
    g_list_free (children_list);
    gtk_button_set_label (start_button, "Releasing...");
}

return;
}

/* Reset the appearance of a cluster after its sound has finished playing. */
void
button_reset_cluster (struct sound_info *sound_data, GApplication * app)
{
    GtkWidget *start_button = NULL;
    GtkWidget *parent_container;
    GList *children_list = NULL;
    const gchar *child_name = NULL;

    /* Find the start button and set its text back to "Start".
     * The start button will be a child of the cluster, and will be named
     * "start_button". */
    parent_container = sound_data->cluster_widget;

    /* It is possible, though unlikely, that the sound will no longer
     * be in a cluster. */
    if (parent_container != NULL)
    {
        children_list =
            gtk_container_get_children (GTK_CONTAINER (parent_container));
        while (children_list != NULL)
        {
            child_name = gtk_widget_get_name (children_list->data);

```

```

        if (g_strcmp0 (child_name, "start_button") == 0)
        {
            start_button = children_list->data;
            break;
        }
        children_list = children_list->next;
    }
    g_list_free (children_list);
    gtk_button_set_label (start_button, "Start");
}

return;
}

/* The volume slider has been moved. Update the volume and the display.
 * The user data is the widget being controlled. */
void
button_volume_changed (GtkButton * button, gpointer user_data)
{
    GtkWidget *volume_label = NULL;
    GtkWidget *parent_container;
    GList *children_list = NULL;
    const gchar *child_name = NULL;
    struct sound_info *sound_data;
    GstBin *bin_element;
    GstElement *volume_element;
    gdouble new_value;
    gchar *value_string;

    /* Find the volume label associated with this volume widget.
     * It will be a child of this widget's parent. */
    parent_container = gtk_widget_get_parent (GTK_WIDGET (button));
    children_list =
        gtk_container_get_children (GTK_CONTAINER (parent_container));
    while (children_list != NULL)
    {
        child_name = gtk_widget_get_name (children_list->data);
        if (g_strcmp0 (child_name, "volume_label") == 0)
        {
            volume_label = children_list->data;
            break;
        }
        children_list = children_list->next;
    }
    g_list_free (children_list);

```

```

if (volume_label != NULL)
{
    /* There should be a sound effect associated with this cluster.
     * If there isn't, do nothing. */
    sound_data = sep_get_sound_effect (user_data);
    if (sound_data == NULL)
        return;

    /* The sound_effect structure records where the Gstreamer bin is
     * for this sound effect. That bin contains the volume control.
     */
    bin_element = sound_data->sound_control;
    volume_element = gstreamer_get_volume (bin_element);
    if (volume_element == NULL)
        return;

    new_value = gtk_scale_button_get_value (GTK_SCALE_BUTTON (button));
    /* Set the volume of the sound. */
    g_object_set (volume_element, "volume", new_value, NULL);

    /* Update the text in the volume label. */
    value_string = g_strdup_printf ("Vol%4.0f%%", new_value * 100.0);
    gtk_label_set_text (volume_label, value_string);
    g_free (value_string);
}

return;
}

/* The pan slider has been moved. Update the pan and display. */
void
button_pan_changed (GtkButton * button, gpointer user_data)
{
    GtkLabel *pan_label = NULL;
    GtkWidget *parent_container;
    GList *children_list = NULL;
    const gchar *child_name = NULL;
    struct sound_info *sound_data;
    GstBin *bin_element;
    GstElement *pan_element;
    gdouble new_value;
    gchar *value_string;

    /* Find the pan label associated with this pan widget.
     * It will be a child of this widget's parent. */

```

```

parent_container = gtk_widget_get_parent (GTK_WIDGET (button));
children_list =
    gtk_container_get_children (GTK_CONTAINER (parent_container));
while (children_list != NULL)
{
    child_name = gtk_widget_get_name (children_list->data);
    if (g_strcmp0 (child_name, "pan_label") == 0)
    {
        pan_label = children_list->data;
        break;
    }
    children_list = children_list->next;
}
g_list_free (children_list);

if (pan_label != NULL)
{
    /* There should be a sound effect associated with this cluster.
     * If there isn't, do nothing. */
    sound_data = sep_get_sound_effect (user_data);
    if (sound_data == NULL)
        return;

    /* The sound_effect structure records where the Gstreamer bin is
     * for this sound effect. That bin contains the pan control.
     */
    bin_element = sound_data->sound_control;
    pan_element = gstreamer_get_pan (bin_element);
    /* The pan control may be omitted by the sound designer. */
    if (pan_element == NULL)
        return;

    new_value = gtk_scale_button_get_value (GTK_SCALE_BUTTON (button));
    /* Set the panorama position of the sound. */
    new_value = (new_value - 50.0) / 50.0;
    g_object_set (pan_element, "panorama", new_value, NULL);

    /* Update the text of the pan label. 0.0 corresponds to Center,
     * negative numbers to left, and positive numbers to right. */
    if (new_value == 0.0)
        gtk_label_set_text (pan_label, "Center");
    else
    {
        if (new_value < 0.0)
            value_string =
                g_strdup_printf ("Left %.0f%%", -(new_value * 100.0));

```

```
        else
            value_string =
                g_strdup_printf ("Right%4.0f%%", new_value * 100.0);
            gtk_label_set_text (pan_label, value_string);
            g_free (value_string);
        }
    }

    return;
}
```

3 button_subroutines.h

```

/*
 * button_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
#ifdef BUTTON_SUBROUTINES_H
#define BUTTON_SUBROUTINES_H

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

/* Subroutines defined in button_subroutines.c */
void button_mute_toggled (GtkToggleButton * button, gpointer user_data);
void button_start_clicked (GtkButton * button, gpointer user_data);
void button_stop_clicked (GtkButton * button, gpointer user_data);
void button_set_cluster_playing (struct sound_info *sound_data,
                                GApplication * app);
void button_set_cluster_releasing (struct sound_info *sound_data,
                                   GApplication * app);
void button_reset_cluster (struct sound_info *sound_data, GApplication * app);
void button_volume_changed (GtkButton * button, gpointer user_data);
void button_pan_changed (GtkButton * button, gpointer user_data);

#endif /* BUTTON_SUBROUTINES_H */
/* End of file button_subroutines.h */

```

4 display__subroutines.c

```

/*
 * display_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gtk/gtk.h>
#include "display_subroutines.h"
#include "sound_effects_player.h"

/* Update the VU meter. */
void
display_update_vu_meter (gpointer * user_data, gint channel,
                        gdouble new_value, gdouble peak_dB, gdouble decay_dB)
{
    GtkWidget *common_area;
    GList *children_list;
    const gchar *child_name;
    GtkWidget *VU_meter = NULL;
    gint VU_level;
    gint64 VU_led_number;
    GtkLabel *VU_lamp;
    GtkWidget *channel_row = NULL;
    gint64 channel_number;

    common_area = sep_get_common_area (G_APPLICATION (user_data));

    /* Find the VU meter in the common area. */
    children_list = gtk_container_get_children (GTK_CONTAINER (common_area));
    while (children_list != NULL)
    {

```

```

    child_name = gtk_widget_get_name (children_list->data);
    if (g_ascii_strcasecmp (child_name, "VU_meter") == 0)
    {
        VU_meter = children_list->data;
        break;
    }
    children_list = children_list->next;
}
g_list_free (children_list);

if (VU_meter == NULL)
    return;

/* Within the VU_meter box is a box for each channel. Find the box
 * for this channel. */
children_list = gtk_container_get_children (GTK_CONTAINER (VU_meter));
while (children_list != NULL)
{
    channel_row = GTK_BOX (children_list->data);
    child_name = gtk_widget_get_name (GTK_WIDGET (channel_row));
    channel_number = g_ascii_strtoll (child_name, NULL, 10);
    if (channel_number == channel)
        break;
    children_list = children_list->next;
}
g_list_free (children_list);
if (channel_row == NULL)
    return;

/* Within the VU_meter's channel box is a bunch of labels, each with a name
 * indicating its position on the line. Light those to the left
 * of the desired value. */
VU_level = new_value * 50.0;
if (VU_level > 50)
    VU_level = 50;
if (VU_level < 1)
    VU_level = 0;
children_list = gtk_container_get_children (GTK_CONTAINER (channel_row));
while (children_list != NULL)
{
    VU_lamp = GTK_LABEL (children_list->data);
    child_name = gtk_widget_get_name (GTK_WIDGET (VU_lamp));
    VU_led_number = g_ascii_strtoll (child_name, NULL, 10);
    if (VU_led_number < VU_level)
    {
        gtk_label_set_text (VU_lamp, "*");
    }
}

```



```

    }
    else
    {
        gtk_label_set_text (VU_lamp, " ");
    }
    children_list = children_list->next;
}
g_list_free (children_list);

return;
}

/* Show the user a message. The return value is a message ID, which
 * can be used to remove the message. */
guint
display_show_message (gchar * message_text, GApplication * app)
{
    GtkStatusbar *status_bar;
    guint context_id;
    guint message_id;

    /* Find the GUI's status display area. */
    status_bar = sep_get_status_bar (app);

    /* Use the regular context for messages. */
    context_id = sep_get_context_id (app);

    /* Show the message. */
    message_id = gtk_statusbar_push (status_bar, context_id, message_text);

    return message_id;
}

/* Remove a previously-displayed message. */
void
display_remove_message (guint message_id, GApplication * app)
{
    GtkStatusbar *status_bar;
    guint context_id;

    /* Find the GUI's status display area. */
    status_bar = sep_get_status_bar (app);

    /* Get the message context ID. */
    context_id = sep_get_context_id (app);

```

```
/* Remove the specified message. */
gtk_statusbar_remove (status_bar, context_id, message_id);

return;
}

/* Display a message to the operator. */
void
display_set_operator_text (gchar * text_to_display, GApplication *app)
{
    GtkWidget *text_label;

    /* Find the GUI's operator text area. */
    text_label = sep_get_operator_text (app);

    /* Set the text. */
    gtk_label_set_text (text_label, text_to_display);

    return;
}

/* Erase any operator message. */
void
display_clear_operator_text (GApplication *app)
{
    GtkWidget *text_label;

    /* Find the GUI's operator text area. */
    text_label = sep_get_operator_text (app);

    /* Clear the text. */
    gtk_label_set_text (text_label, (gchar *) "");

    return;
}
```

5 display_subroutines.h

```

/*
 * display_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#ifdef DISPLAY_SUBROUTINES_H
#define DISPLAY_SUBROUTINES_H

#include <gtk/gtk.h>

/* Subroutines defined in display_subroutines.c */
void display_update_vu_meter (gpointer * user_data, gint channel,
                             gdouble new_value, gdouble peak_dB,
                             gdouble decay_dB);

guint display_show_message (gchar * message_text, GApplication * app);

void display_remove_message (guint message_id, GApplication * app);

void display_set_operator_text (gchar * text_to_display, GApplication * app);

void display_clear_operator_text (GApplication * app);

#endif /* DISPLAY_SUBROUTINES_H */

/* End of file display_subroutines.h */

```

6 gstenvelope.c

```

/*
 * File: gstenvelope.c, part of Show_control, a Gstreamer application
 *
 * Much of this code is based on Gstreamer examples and tutorials.
 *
 * Copyright © 2016 John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, see https://gnu.org/licenses
 * or write to
 * Free Software Foundation, Inc.
 * 51 Franklin Street, Fifth Floor
 * Boston, MA 02111-1301
 * USA.
 */

/**
 * SECTION:element-envelope
 *
 * Shape the volume of the incoming sound based on a traditional
 * attack, decay, sustain, release amplitude envelope. Properties
 * are:
 *
 * #GstEnvelope:attack-duration-time is the initial ramp up time from volume 0
 * in nanoseconds. Default is 0.
 *
 * #GstEnvelope:attack-level is the volume level at the end of the attack.
 * Default is 1.0, which leaves the incoming volume unchanged.
 *
 * #GstEnvelope:decay-duration-time is the time to decay after the attack is
 * complete, in nanoseconds. Default is 0.
 *
 * #GstEnvelope:sustain-level is the volume level at the end of the decay.
 * Default is 1.0, which leaves the incoming volume unchanged.

```

```

*
* #GstEnvelope:release-start-time is the time when the release part of the
* envelope starts, in nanoseconds. This value can be changed to the current
* time using a custom Release event, or by the receipt of EOS from upstream.
* The default is 0, which, if left unchanged, disables release.
*
* #GstEnvelope:release-duration-time is the length of time for the
* volume to ramp down to 0 once the release is initiated. This value
* is in nanoseconds, but can also be specified as infinity by setting
* the value to the UTF-8 string for the Unicode character for infinity,
* U+221E, which is  $\infty$ . Default is 0, which causes the sound to be shut
* down instantly.
*
* #GstEnvelope:volume is the normal volume of the sound; the envelope
* is scaled by this amount. It might be used to implement the note-on
* velocity from a musical instrument. Default is 1.0.
*
* #GstEnvelope:autostart defaults to FALSE. If you set it to TRUE,
* envelope processing will start as soon as sound data passes through
* it, without waiting for a Start event.
*
* #GstEnvelope:sound-name is the name of the sound being shaped. This name
* is used in messages to the application, to identify the sound. It
* defaults to the empty string.
*
* If all the properties except autostart are defaulted, and release is never
* signaled, this audio filter does not change the sound passing through it.
*
* <refsect2>
* <title>Example launch line</title>
* |[
* gst-launch-1.0 -v -m audiotestsrc ! envelope attack-duration-time=1000000000
→ autostart=TRUE ! fakesink silent=TRUE
* ]| Instead of starting the test tone at full volume, fade it in over one
* second.
* </refsect2>
*/

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <string.h>
#include <math.h>
#include <gst/gst.h>
#include <gst/base/gstbasetransform.h>

```

```

#include <gst/audio/audio.h>
#include <gst/audio/gstaudiofilter.h>
#include <gst/base/gsttypefindhelper.h>

#include "gstenvelope.h"

GST_DEBUG_CATEGORY_STATIC (envelope);
#define GST_CAT_DEFAULT envelope

/* Filter signals and args */
enum
{
    /* FILL ME */
    RELEASE_SIGNAL,          /* initiate release process */
    LAST_SIGNAL
};

enum
{
    PROP_0,
    PROP_SILENT,
    PROP_ATTACK_DURATION_TIME,
    PROP_ATTACK_LEVEL,
    PROP_DECAY_DURATION_TIME,
    PROP_SUSTAIN_LEVEL,
    PROP_RELEASE_START_TIME,
    PROP_RELEASE_DURATION_TIME,
    PROP_VOLUME,
    PROP_AUTOSTART,
    PROP_SOUND_NAME
};

/* For simplicity, we handle only floating point samples.
 * If there is a need for conversion to another type, it can be
 * done using an audioconvert element. */

#if G_BYTE_ORDER == G_LITTLE_ENDIAN
#define ALLOWED_CAPS \
    "audio/x-raw, " \
    "format = (string) { F64LE, F32LE }, " \
    "rate = (int) [ 1, 2147483647 ], " \
    "channels = (int) [ 1, 32 ]," \
    "layout = (string) { interleaved }"
#else
#define ALLOWED_CAPS \
    "audio/x-raw, " \

```

```

    "format = (string) { F64BE, F32BE }, " \
    "rate = (int) [ 1, 2147483647 ], " \
    "channels = (int) [ 1, 32 ]," \
    "layout = (string) { interleaved }"
#endif

#define DEBUG_INIT \
    GST_DEBUG_CATEGORY_INIT (envelope, "envelope", 0, \
        "Shape the amplitude of the sound");
#define gst_envelope_parent_class parent_class

/* By basing this filter on GstAudioFilter, we impose the requirement
 * that the output is in the same format as the input. */
G_DEFINE_TYPE_WITH_CODE (GstEnvelope, gst_envelope, GST_TYPE_AUDIO_FILTER,
    DEBUG_INIT);

/* Forward declarations. These subroutines will be defined below. */
static void gst_envelope_set_property (GObject * object, guint prop_id,
    const GValue * value,
    GParamSpec * pspec);
static void gst_envelope_get_property (GObject * object, guint prop_id,
    GValue * value, GParamSpec * pspec);

static void envelope_before_transform (GstBaseTransform * base,
    GstBuffer * buffer);

static GstFlowReturn envelope_transform_ip (GstBaseTransform * base,
    GstBuffer * outbuf);
static GstFlowReturn envelope_transform (GstBaseTransform * base,
    GstBuffer * inbuf,
    GstBuffer * outbuf);

static gboolean envelope_sink_event_handler (GstBaseTransform * trans,
    GstEvent * event);
static gboolean envelope_src_event_handler (GstBaseTransform * trans,
    GstEvent * event);

static gdouble compute_volume (GstEnvelope * self, GstClockTime timestamp);

/* Before each transform of input to output, do this. */
static void
envelope_before_transform (GstBaseTransform * base, GstBuffer * buffer)
{
    GstClockTime timestamp, duration;
    GstEnvelope *self = GST_ENVELOPE (base);
    GstStructure *structure;

```

```

GstMessage *message;
gboolean result;
GValue sound_name_value = G_VALUE_INIT;

timestamp = GST_BUFFER_TIMESTAMP (buffer);
timestamp =
    gst_segment_to_stream_time (&base->segment, GST_FORMAT_TIME, timestamp);
duration = GST_BUFFER_DURATION (buffer);
duration =
    gst_segment_to_stream_time (&base->segment, GST_FORMAT_TIME, duration);

GST_DEBUG_OBJECT (self, "timestamp: %" GST_TIME_FORMAT ".",
                  GST_TIME_ARGS (timestamp));
GST_DEBUG_OBJECT (self, "duration: %" GST_TIME_FORMAT ".",
                  GST_TIME_ARGS (duration));

if (GST_CLOCK_TIME_IS_VALID (timestamp))
    gst_object_sync_values (GST_OBJECT (self), timestamp);

/* If we have reached the release portion of the envelope, tell the
 * application. */
if (self->running && self->release_started
    && !self->application_notified_release)
{
    GST_INFO_OBJECT (self, "sound has entered its Release stage");
    structure = gst_structure_new_empty ("release_started");
    g_value_init (&sound_name_value, G_TYPE_STRING);
    g_value_set_string (&sound_name_value, self->sound_name);
    gst_structure_set_value (structure, (gchar *) "sound_name",
                            &sound_name_value);

    message = gst_message_new_element (GST_OBJECT (self), structure);
    result = gst_element_post_message (GST_ELEMENT (self), message);
    if (!result)
    {
        GST_DEBUG_OBJECT (self, "unable to post a release_started message");
    }
    self->application_notified_release = TRUE;
    g_value_unset (&sound_name_value);
}

/* If we have completed the sound, tell the application. */
if (self->completed && !self->application_notified_completion)
{
    GST_INFO_OBJECT (self, "sound has completed");
    structure = gst_structure_new_empty ("completed");

```



```

    g_value_init (&sound_name_value, G_TYPE_STRING);
    g_value_set_string (&sound_name_value, self->sound_name);
    gst_structure_set_value (structure, (gchar *) "sound_name",
                             &sound_name_value);

    message = gst_message_new_element (GST_OBJECT (self), structure);
    result = gst_element_post_message (GST_ELEMENT (self), message);
    if (!result)
    {
        GST_DEBUG_OBJECT (self, "unable to post a completed message");
    }
    self->application_notified_completion = TRUE;
    g_value_unset (&sound_name_value);
}

/* If we have completed the envelope, including the release stage,
 * and we are not autostarting, recycle it so we can use it again.
 * Note that this test is performed before the start test, so a Start
 * message that arrives during the release stage of the envelope will
 * restart it immediately after the release is complete. */
if (self->completed && !self->autostart)
{
    GST_DEBUG_OBJECT (self,
                      "recycling envelope, base time is %" GST_TIME_FORMAT
                      ".", GST_TIME_ARGS (self->base_time));

    self->running = FALSE;
    self->completed = FALSE;
    self->release_started = FALSE;
    self->base_time = 0;
    self->last_volume = 0;
    self->application_notified_release = FALSE;
    self->application_notified_completion = FALSE;
}

if (self->running)
{
    GST_DEBUG_OBJECT (self, "running, base time is %" GST_TIME_FORMAT ".",
                      GST_TIME_ARGS (self->base_time));
    GST_DEBUG_OBJECT (self, "envelope time is %" GST_TIME_FORMAT ".",
                      GST_TIME_ARGS (timestamp - self->base_time));
}

/* If we have seen a start message, or if we are autostarted,
 * and the envelope is not yet running, start running it. */
if ((!self->running) && (self->started || self->autostart))
{

```

```

        self->external_release_seen = FALSE;
        self->external_completion_seen = FALSE;
        self->running = TRUE;
        self->started = FALSE;
        self->pause_seen = FALSE;
        self->continue_seen = FALSE;
        self->pausing = FALSE;
        self->base_time = timestamp;
        self->pause_time = 0;
        GST_DEBUG_OBJECT (self,
                          "starting envelope, base time set to %"
                          GST_TIME_FORMAT ".", GST_TIME_ARGS (self->base_time));
    }

    return;
}

/* Convert input data to output data, using the same buffer for
 * input and output. That is, the data is modified in place. */
static GstFlowReturn
envelope_transform_ip (GstBaseTransform * base, GstBuffer * outbuf)
{
    GstAudioFilter *filter = GST_AUDIO_FILTER_CAST (base);
    GstEnvelope *self = GST_ENVELOPE (base);
    GstMapInfo map;
    GstClockTime ts;
    gint rate = GST_AUDIO_INFO_RATE (&filter->info);
    gint width = GST_AUDIO_FORMAT_INFO_WIDTH (filter->info.finfo);
    gint channel_count = GST_AUDIO_INFO_CHANNELS (&filter->info);
    gint frame_count;
    gint frame_counter, channel_counter;
    gdouble volume_val;
    gdouble *src64;
    gfloat *src32;
    GstClockTimeDiff interval = gst_util_uint64_scale_int (1, GST_SECOND, rate);
    GstClockTimeDiff pause_duration;

    /* Don't process data with GAP. */
    if (GST_BUFFER_FLAG_IS_SET (outbuf, GST_BUFFER_FLAG_GAP))
        return GST_FLOW_OK;

    /* Map the buffer as read-write, since the modifications are performed
     * in place. */
    gst_buffer_map (outbuf, &map, GST_MAP_READWRITE);

    /* Compute the number of frames. Each frame takes one time step and has

```

```

    * a sample for each channel. Note that the width is in bits. */
    frame_count = gst_buffer_get_size (outbuf) / (width * channel_count / 8);

    ts = GST_BUFFER_TIMESTAMP (outbuf);
    ts = gst_segment_to_stream_time (&base->segment, GST_FORMAT_TIME, ts);

    GST_DEBUG_OBJECT (self,
        "transform in place timestamp: %" GST_TIME_FORMAT ".",
        GST_TIME_ARGS (ts));

    /* Handle pause and continue events. */
    if ((self->pause_seen) && (!self->pausing))
    {
        /* We are starting a pause. Record the current time so we can
        * determine the pause duration when the pause ends. */
        self->pause_start_time = ts;
        self->pausing = TRUE;
        GST_DEBUG_OBJECT (self, "pause starts at %" GST_TIME_FORMAT ".",
            GST_TIME_ARGS (ts));
    }
    if ((self->pause_seen) && (self->continue_seen))
    {
        /* This is the end of the pause. */
        self->pause_seen = FALSE;
        self->pausing = FALSE;
        self->continue_seen = FALSE;
        /* Accumulate the time spend pausing, so the envelope can continue
        * through its progression. */
        pause_duration = ts - self->pause_start_time;
        self->pause_time = self->pause_time + pause_duration;
        GST_DEBUG_OBJECT (self,
            "pause is completed, duration: %" GST_TIME_FORMAT ".",
            GST_TIME_ARGS (pause_duration));
    }
    GST_DEBUG_OBJECT (self, "pause time is: %" GST_TIME_FORMAT ".",
        GST_TIME_ARGS (self->pause_time));

    if (self->running)
    {
        GST_DEBUG_OBJECT (self, "envelope time: %" GST_TIME_FORMAT ".",
            GST_TIME_ARGS (ts - self->base_time));
    }
    GST_DEBUG_OBJECT (self, "interval: %" GST_TIME_FORMAT ".",
        GST_TIME_ARGS (interval));
    GST_DEBUG_OBJECT (self, "rate: %d, width: %d, channels: %d, frames: %d.",
        rate, width, channel_count, frame_count);

```

```

/* For each frame, compute the volume adjustment and apply it to
   * each sample. There will be one sample per channel. */

src64 = (gdouble *) map.data;
src32 = (gfloat *) map.data;
for (frame_counter = 0; frame_counter < frame_count; frame_counter++)
{
    /* Compute the volume at this time step. */
    volume_val =
        compute_volume (self, ts - self->base_time - self->pause_time);

    /* Apply that volume to each channel. */
    for (channel_counter = 0; channel_counter < channel_count;
        channel_counter++)
    {
        /* Since we only allow floating-point, we can use the width
         * to determine the data type. 32 bits is gfloat and 64 bits
         * is gdouble. */
        switch (width)
        {
            case 64:
                GST_LOG_OBJECT (self, "sample with value %g becomes %g.",
                                *src64, volume_val * *src64);
                *src64 = volume_val * *src64;
                src64++;
                break;

            case 32:
                GST_LOG_OBJECT (self, "sample with value %g becomes %g.",
                                *src32, volume_val * *src32);
                *src32 = volume_val * *src32;
                src32++;
                break;

            default:
                GST_ELEMENT_ERROR (self, STREAM, FORMAT, (NULL),
                                ("unknown sample width: %d.", width));
                break;
        }
    }
    ts = ts + interval;
}

/* We are done with the buffer. */
gst_buffer_unmap (outbuf, &map);

```

```

    return GST_FLOW_OK;
}

/* Convert input data to output data, using different buffers for
 * input and output. */
static GstFlowReturn
envelope_transform (GstBaseTransform * base, GstBuffer * inbuf,
                   GstBuffer * outbuf)
{
    GstAudioFilter *filter = GST_AUDIO_FILTER_CAST (base);
    GstEnvelope *self = GST_ENVELOPE (base);
    GstMapInfo srcmap, dstmap;
    gint insize, outsize;
    gboolean inbuf_writable;
    gint frame_count;
    gdouble *src64, *dst64;
    gfloat *src32, *dst32;
    gdouble volume_val;
    gint frame_counter, channel_counter;
    GstClockTime ts;
    gint rate = GST_AUDIO_INFO_RATE (&filter->info);
    gint width = GST_AUDIO_FORMAT_INFO_WIDTH (filter->info.finfo);
    gint channel_count = GST_AUDIO_INFO_CHANNELS (&filter->info);
    GstClockTimeDiff interval = gst_util_uint64_scale_int (1, GST_SECOND, rate);
    GstClockTimeDiff pause_duration;

    /* Get the number of frames to process. Each frame has a sample for
     * each channel, and each sample contains "width" bits. */
    frame_count = gst_buffer_get_size (inbuf) / (width * channel_count / 8);

    ts = GST_BUFFER_TIMESTAMP (outbuf);
    ts = gst_segment_to_stream_time (&base->segment, GST_FORMAT_TIME, ts);
    GST_DEBUG_OBJECT (self, "transform timestamp: %" GST_TIME_FORMAT ".",
                     GST_TIME_ARGS (ts));
    /* Handle pause and continue events. */
    if ((self->pause_seen) && (!self->pausing))
    {
        /* We are starting a pause. Record the current time so we can
         * determine the pause duration when the pause ends. */
        self->pause_start_time = ts;
        self->pausing = TRUE;
        GST_DEBUG_OBJECT (self, "pause starts at %" GST_TIME_FORMAT ".",
                         GST_TIME_ARGS (ts));
    }
    if ((self->pause_seen) && (self->continue_seen))
    {

```

```

    /* This is the end of the pause. */
    self->pause_seen = FALSE;
    self->pausing = FALSE;
    self->continue_seen = FALSE;
    /* Accumulate the time spend pausing, so the envelope can continue
       * through its progression. */
    pause_duration = ts - self->pause_start_time;
    self->pause_time = self->pause_time + pause_duration;
    GST_DEBUG_OBJECT (self,
        "pause is completed, duration: %" GST_TIME_FORMAT ".",
        GST_TIME_ARGS (pause_duration));
}
GST_DEBUG_OBJECT (self, "pause time is: %" GST_TIME_FORMAT ".",
    GST_TIME_ARGS (self->pause_time));
if (self->running)
{
    GST_DEBUG_OBJECT (self, "envelope time: %" GST_TIME_FORMAT ".",
        GST_TIME_ARGS (ts - self->base_time));
}
GST_DEBUG_OBJECT (self, "interval: %" GST_TIME_FORMAT ".",
    GST_TIME_ARGS (interval));
GST_DEBUG_OBJECT (self, "rate: %d, width: %d, channels: %d, frames: %d.",
    rate, width, channel_count, frame_count);

/* Compute the size of the necessary buffers. */
insize = frame_count * channel_count * width / 8;
outsize = frame_count * channel_count * width / 8;

/* A zero-length buffer has no data to modify. */
if (insize == 0 || outsize == 0)
    return GST_FLOW_OK;

inbuf_writable = gst_buffer_is_writable (inbuf)
    && gst_buffer_n_memory (inbuf) == 1
    && gst_memory_is_writable (gst_buffer_peek_memory (inbuf, 0));

/* Get pointers to the source and destination data. */
gst_buffer_map (inbuf, &srcmap,
    inbuf_writable ? GST_MAP_READWRITE : GST_MAP_READ);
gst_buffer_map (outbuf, &dstmap, GST_MAP_WRITE);

/* Check in and out size. */
if (srcmap.size < insize)
{
    GST_ELEMENT_ERROR (self, STREAM, FORMAT, (NULL),
        ("input buffer is the wrong size: %" G_GSIZE_FORMAT

```

```

        " < %d.", srcmap.size, insize));
    gst_buffer_unmap (outbuf, &dstmap);
    gst_buffer_unmap (inbuf, &srcmap);
    return GST_FLOW_ERROR;
}

if (dstmap.size < outsize)
{
    GST_ELEMENT_ERROR (self, STREAM, FORMAT, (NULL),
        ("output buffer is the wrong size: %" G_GSIZE_FORMAT
        " < %d.", dstmap.size, outsize));
    gst_buffer_unmap (outbuf, &dstmap);
    gst_buffer_unmap (inbuf, &srcmap);
    return GST_FLOW_ERROR;
}

/* Do nothing with gaps. */
if (GST_BUFFER_FLAG_IS_SET (inbuf, GST_BUFFER_FLAG_GAP))
{
    gst_buffer_unmap (outbuf, &dstmap);
    gst_buffer_unmap (inbuf, &srcmap);
    return GST_FLOW_OK;
}

/* Copy the samples, applying the volume adjustment as we go. */
src64 = (gdouble *) srcmap.data;
dst64 = (gdouble *) dstmap.data;
src32 = (gfloat *) srcmap.data;
dst32 = (gfloat *) dstmap.data;

GST_DEBUG_OBJECT (self, "copy %d values.", frame_count * channel_count);
for (frame_counter = 0; frame_counter < frame_count; frame_counter++)
{
    /* Compute the volume at this time step. */
    volume_val =
        compute_volume (self, ts - self->base_time - self->pause_time);

    /* Apply that volume to each channel. */
    for (channel_counter = 0; channel_counter < channel_count;
        channel_counter++)
    {
        /* Since we only allow floating-point, we can use the width
         * to determine the data type. 32 bits is gfloat and 64 bits
         * is gdouble. */
        switch (width)

```

```

    {
    case 64:
        GST_LOG_OBJECT (self, "sample with value %g becomes %g.",
                        *src64, volume_val * *src64);
        *dst64 = volume_val * *src64;
        src64++;
        dst64++;
        break;

    case 32:
        GST_LOG_OBJECT (self, "sample with value %g becomes %g.",
                        *src32, volume_val * *src32);
        *dst32 = volume_val * *src32;
        src32++;
        dst32++;
        break;

    default:
        GST_ELEMENT_ERROR (self, STREAM, FORMAT, (NULL),
                        ("unknown sample width: %d.", width));
        break;
    }
}

ts = ts + interval;
}

/* We are done with the buffers. */
gst_buffer_unmap (outbuf, &dstmap);
gst_buffer_unmap (inbuf, &srcmap);
return GST_FLOW_OK;
}

/* This enumeration type indicates a stage of envelope processing. */
enum envelope_stage
{ not_started, attack, decay, sustain, release, completed, pausing };

/* Determine the stage of envelope processing, given the time since
 * the envelope started, minus the time spent paused. */
static enum envelope_stage
compute_envelope_stage (GstEnvelope * self, GstClockTime ts)
{
    gchar *release_type;

    /* Decide where we are in the amplitude envelope. The normal progression
     * after the note has started is attack, decay, sustain, release, completed.
     * However, a release event can arrive at at any time. If the decay

```



```

* duration time is 0 we go straight from attack to sustain, so sustain
* level should equal attack level in this case. Also, a pause event
* can arrive at any time, and will delay the envelope progression. */

/* if the envelope is not running, it is not yet started. */
if (!self->running)
{
    return not_started;
}

if (self->pausing)
{
    return pausing;
}

if (self->external_release_seen || self->external_completion_seen)
{
    /* We have seen an external signal initiating the release process,
     * so the envelope is in either its release or completed stage. */

    /* If this is the first time we have been in the release stage,
     * remember the time and volume since we will need them to ramp the
     * volume down to zero. */
    if (!self->release_started)
    {
        self->release_started = TRUE;
        self->release_started_volume = self->last_volume;
        self->release_started_time = ts;
        release_type = (gchar *) "an unknown";
        if (self->external_completion_seen)
        {
            release_type = (gchar *) "a complete";
        }
        if (self->external_release_seen)
        {
            release_type = (gchar *) "a release";
        }
        GST_INFO_OBJECT (self,
                        "Release triggered by %s event at %"
                        GST_TIME_FORMAT " with volume %f.", release_type,
                        GST_TIME_ARGS (self->release_started_time),
                        self->release_started_volume);
    }

    /* An external completion message means we have reached the end of
     * the sound from upstream. No matter where we were in the envelope,
     * we are now done. */

```

```

    if (self->external_completion_seen)
        return completed;

    /* Otherwise, if we are within the release duration, we are in the
     * release stage of the envelope. Note that if the release duration
     * is infinite, the volume stays at the value it held when the
     * release message arrived until the sound coming from upstream
     * is complete.
     */
    if ((ts < (self->release_started_time + self->release_duration_time))
        || (self->release_duration_infinite))
        return release;
    else
        return completed;
}

/* We have not seen a release or completion event, so the envelope proceeds
 * along its normal path. */
if (ts < self->attack_duration_time)
{
    /* The attack is not yet complete. */
    return attack;
}

/* The attack is complete. */

if (ts < self->attack_duration_time + self->decay_duration_time)
{
    /* The decay is not yet complete. */
    return decay;
}

/* The decay is complete. */

if ((ts < self->release_start_time) || (self->release_start_time == 0))
{
    /* The decay is complete but we have not yet started
     * release. */
    return sustain;
}

/* A non-infinite, non-zero release time was specified for the envelope.
 */

if (self->release_duration_infinite
    || ts < (self->release_start_time + self->release_duration_time))

```

```

{
    /* The release section of the envelope is running.
     * If this is the first time we have been in the release stage,
     * remember the time and volume since we will need them to ramp the
     * volume down to zero. */
    if (!self->release_started)
    {
        self->release_started = TRUE;
        self->release_started_volume = self->last_volume;
        self->release_started_time = ts;
        GST_INFO_OBJECT (self,
                        "Release triggered at %" GST_TIME_FORMAT
                        " with volume %f.",
                        GST_TIME_ARGS (self->release_started_time),
                        self->release_started_volume);
    }
    return release;
}

/* We have passed the specified release time, and in addition the specified
 * release duration. The envelope is complete. */
return completed;
}

/* Compute the volume adjustment for a frame. */
static gdouble
compute_volume (GstEnvelope * self, GstClockTime ts)
{
    gdouble volume_val;
    enum envelope_stage envelope_position;
    GstClockTime decay_end_time;
    gdouble attack_fraction, decay_fraction, release_fraction;

    /* Decide where we are in the amplitude envelope. */
    envelope_position = compute_envelope_stage (self, ts);

    switch (envelope_position)
    {
        case pausing:
            /* The note is paused. */
            GST_LOG_OBJECT (self, "envelope position: pausing");
            volume_val = 0.0;
            break;

        case not_started:
            /* If the note has not yet started, the volume is 0. */

```

```

GST_LOG_OBJECT (self, "envelope position: not started");
volume_val = 0.0;
break;

case attack:
    /* The initial attack. We ramp up to the specified attack level,
     * reaching it at the specified attack duration time. */
    attack_fraction = (gdouble) ts / (gdouble) self->attack_duration_time;

    GST_LOG_OBJECT (self, "envelope position: attack, fraction %g.",
                    attack_fraction);
    volume_val = self->attack_level * attack_fraction;
    break;

case decay:
    /* When the attack is complete we decay to the specified sustain
     * level, reaching it at the specified decay duration time following
     * completion of the attack. Do a linear interpolation between the
     * attack level and the sustain level. If the decay duration time is 0,
     * we will never be in the decay section of the envelope. */
    decay_end_time = self->attack_duration_time + self->decay_duration_time;
    decay_fraction =
        (gdouble) 1.0 - (gdouble) (decay_end_time -
                                   ts) / (gdouble) self->decay_duration_time;
    GST_LOG_OBJECT (self, "envelope position: decay, fraction %g.",
                    decay_fraction);
    volume_val =
        (decay_fraction * self->sustain_level) +
        ((1.0 - decay_fraction) * self->attack_level);
    break;

case sustain:
    GST_LOG_OBJECT (self, "envelope position: sustain");
    /* When the decay is complete we stay at the sustain level until release.
     */
    volume_val = self->sustain_level;
    break;

case release:
    /* Upon release we reduce the volume to 0 over the release duration time.
     * However, if the release duration time is infinite, we do not
     * reduce the volume. Normally, release will occur after sustain,
     * so an infinite release duration time will keep the volume at the
     * sustain level. If the release duration time is 0 we do not get to
     * the release portion of the envelope but go directly to completed. */
    if (self->release_duration_infinite)

```

```

    {
        volume_val = self->release_started_volume;
        GST_LOG_OBJECT (self, "envelope position: release, infinite.");
        break;
    }

    release_fraction =
        (gdouble) (ts -
                    self->release_started_time) /
        (gdouble) self->release_duration_time;
    GST_LOG_OBJECT (self, "envelope position: release, fraction %g.",
                    release_fraction);
    volume_val = self->release_started_volume * (1.0 - release_fraction);

    break;

case completed:
    GST_LOG_OBJECT (self, "envelope position: completed");
    /* We are beyond the release duration; volume is always 0. */
    volume_val = 0.0;
    /* Note the envelope completion. This is used to recycle the envelope.
    */
    if (!self->completed)
    {
        GST_DEBUG_OBJECT (self,
                           "envelope completed at envelope time %"
                           GST_TIME_FORMAT ".", GST_TIME_ARGS (ts));
    }

    self->completed = TRUE;
    break;
}

/* Remember the last value used, so we can release from it in case the
 * release starts at an unusual time in the envelope. */
self->last_volume = volume_val;

/* Allow for scaling the envelope, perhaps to implement a Note On velocity.
*/
volume_val = volume_val * self->volume;

GST_LOG_OBJECT (self,
                 "at time %" GST_TIME_FORMAT ", envelope volume is %g.",
                 GST_TIME_ARGS (ts), volume_val);

return volume_val;

```

```

}

static gboolean
envelope_stop (GstBaseTransform * base)
{
    /* GstEnvelope *self = GST_ENVELOPE (base); */
    return GST_CALL_PARENT_WITH_DEFAULT (GST_BASE_TRANSFORM_CLASS, stop, (base),
                                         TRUE);
};

/* call whenever the format changes. */
static gboolean
envelope_setup (GstAudioFilter * filter, const GstAudioInfo * info)
{
    GstEnvelope *self = GST_ENVELOPE (filter);
    GST_OBJECT_LOCK (self);
    GST_OBJECT_UNLOCK (self);
    return TRUE;
};

static void
gst_envelope_dispose (GObject * object)
{
    GstEnvelope *self = GST_ENVELOPE (object);
    g_free (self->release_duration_string);
    self->release_duration_string = NULL;
    g_free (self->last_message);
    self->last_message = NULL;
    g_free (self->sound_name);
    self->sound_name = NULL;
    G_OBJECT_CLASS (parent_class)->dispose (object);
};

/* GObject vmethod implementations */

/* initialize the envelope's class */
static void
gst_envelope_class_init (GstEnvelopeClass * klass)
{
    GObjectClass *gobject_class;
    GstElementClass *element_class;
    GstBaseTransformClass *trans_class;
    GstAudioFilterClass *filter_class;
    GstCaps *caps;
    GParamSpec *param_spec;
    gchar *release_duration_default;

```

```

gchar *sound_name_default;

gobject_class = (GObjectClass *) klass;
element_class = (GstElementClass *) klass;
trans_class = (GstBaseTransformClass *) klass;
filter_class = (GstAudioFilterClass *) (klass);

gobject_class->set_property = gst_envelope_set_property;
gobject_class->get_property = gst_envelope_get_property;
gobject_class->dispose = gst_envelope_dispose;

param_spec =
    g_param_spec_boolean ("silent", "Silent", "Produce verbose output ?",
        FALSE, G_PARAM_READWRITE | GST_PARAM_CONTROLLABLE);
g_object_class_install_property (gobject_class, PROP_SILENT, param_spec);

param_spec =
    g_param_spec_uint64 ("attack-duration-time", "Attack_duration_time",
        "Time for initial ramp up of volume", 0, G_MAXUINT64,
        0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_ATTACK_DURATION_TIME,
    param_spec);

param_spec =
    g_param_spec_double ("attack-level", "Attack_level",
        "Volume level to reach at end of attack", 0, 10.0,
        1.0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_ATTACK_LEVEL,
    param_spec);

param_spec =
    g_param_spec_uint64 ("decay-duration-time", "Decay_duration_time",
        "Time for ramp down to sustain level after attack",
        0, G_MAXUINT64, 0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_DECAY_DURATION_TIME,
    param_spec);

param_spec =
    g_param_spec_double ("sustain-level", "Sustain_level",
        "Volume level to reach at end of decay", 0, 10.0,
        1.0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_SUSTAIN_LEVEL,
    param_spec);

param_spec =
    g_param_spec_uint64 ("release-start-time", "Release_start_time",
        "When to start the release process", 0, G_MAXUINT64,

```

```

        0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_RELEASE_START_TIME,
                                param_spec);

release_duration_default = g_strdup ((gchar *) "0");
param_spec =
    g_param_spec_string ("release-duration-time", "Release_duration_time",
        "Time for ramp down to 0 while releasing, may be  $\infty$ ",
        release_duration_default, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_RELEASE_DURATION_TIME,
                                param_spec);
g_free (release_duration_default);
release_duration_default = NULL;

param_spec =
    g_param_spec_double ("volume", "Volume_level", "Volume to scale envelope",
        0, 10.0, 1.0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_VOLUME, param_spec);

param_spec =
    g_param_spec_boolean ("autostart", "Autostart",
        "do not wait for a Start event", FALSE,
        G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_AUTOSTART, param_spec);

sound_name_default = g_strdup ("");
param_spec =
    g_param_spec_string ("sound-name", "Sound_name",
        "The name of the sound being shaped",
        sound_name_default, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_SOUND_NAME,
                                param_spec);
g_free (sound_name_default);
sound_name_default = NULL;

gst_element_class_set_static_metadata (element_class, "Envelope",
    "Filter/Effect/Audio",
    "Shape the sound using "
    "an a-d-s-r envelope",
    "John Sauter <John_Sauter@"
    "systemeyescomputerstore.com>");

caps = gst_caps_from_string (ALLOWED_CAPS);
gst_audio_filter_class_add_pad_templates (filter_class, caps);
gst_caps_unref (caps);

```



```

trans_class->before_transform =
    GST_DEBUG_FUNCPTR (envelope_before_transform);
trans_class->transform_ip = GST_DEBUG_FUNCPTR (envelope_transform_ip);
trans_class->transform = GST_DEBUG_FUNCPTR (envelope_transform);
trans_class->stop = GST_DEBUG_FUNCPTR (envelope_stop);
trans_class->transform_ip_on_passthrough = FALSE;
trans_class->sink_event = GST_DEBUG_FUNCPTR (envelope_sink_event_handler);
trans_class->src_event = GST_DEBUG_FUNCPTR (envelope_src_event_handler);

filter_class->setup = envelope_setup;
}

/* initialize the new element
 * initialize instance structure
 */
static void
gst_envelope_init (GstEnvelope * self)
{
    /* Set all of the parameters to their default values and initialize
     * the locals. */
    self->silent = FALSE;
    self->attack_duration_time = 0;
    self->attack_level = 1.0;
    self->decay_duration_time = 0;
    self->sustain_level = 1.0;
    self->release_start_time = 0;
    self->release_duration_string = g_strdup ((gchar *) "0");
    self->release_duration_time = 0;
    self->release_duration_infinite = FALSE;
    self->release_started_volume = 0.0;
    self->release_started_time = 0;
    self->release_started = FALSE;
    self->volume = 1.0;
    self->autostart = FALSE;
    self->sound_name = g_strdup ("");

    self->external_release_seen = FALSE;
    self->external_completion_seen = FALSE;
    self->running = FALSE;
    self->started = FALSE;
    self->completed = FALSE;
    self->pause_seen = FALSE;
    self->continue_seen = FALSE;
    self->pausing = FALSE;
    self->last_message = NULL;
    self->application_notified_release = FALSE;

```

```

self->application_notified_completion = FALSE;
self->base_time = 0;
self->pause_time = 0;
self->pause_start_time = 0;
self->last_volume = 0;
}

/* Set a property. */
static void
gst_envelope_set_property (GObject * object, guint prop_id,
                           const GValue * value, GParamSpec * pspec)
{
    GstEnvelope *self = GST_ENVELOPE (object);

    switch (prop_id)
    {
        case PROP_SILENT:
            GST_OBJECT_LOCK (self);
            self->silent = g_value_get_boolean (value);
            GST_INFO_OBJECT (self, "silent set to %d.", self->silent);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_ATTACK_DURATION_TIME:
            GST_OBJECT_LOCK (self);
            self->attack_duration_time = g_value_get_uint64 (value);
            GST_INFO_OBJECT (self,
                            "attack-duration-time set to %" G_GUINT64_FORMAT ".",
                            self->attack_duration_time);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_ATTACK_LEVEL:
            GST_OBJECT_LOCK (self);
            self->attack_level = g_value_get_double (value);
            GST_INFO_OBJECT (self, "attack-level set to %g.", self->attack_level);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_DECAY_DURATION_TIME:
            GST_OBJECT_LOCK (self);
            self->decay_duration_time = g_value_get_uint64 (value);
            GST_INFO_OBJECT (self,
                            "decay-duration-time set to %" G_GUINT64_FORMAT ".",
                            self->decay_duration_time);
            GST_OBJECT_UNLOCK (self);
    }
}

```

```

    break;

case PROP_SUSTAIN_LEVEL:
    GST_OBJECT_LOCK (self);
    self->sustain_level = g_value_get_double (value);
    GST_INFO_OBJECT (self, "sustain-level set to %g.", self->sustain_level);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_RELEASE_START_TIME:
    GST_OBJECT_LOCK (self);
    self->release_start_time = g_value_get_uint64 (value);
    GST_INFO_OBJECT (self,
        "release-start-time set to %" G_GUINT64_FORMAT ".",
        self->release_start_time);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_RELEASE_DURATION_TIME:
    GST_OBJECT_LOCK (self);
    g_free (self->release_duration_string);
    self->release_duration_string = NULL;
    self->release_duration_string = g_value_dup_string (value);

    if (g_strcmp0 (self->release_duration_string, "∞") == 0)
    {
        self->release_duration_infinite = TRUE;
        self->release_duration_time = 0;
    }
    else
    {
        self->release_duration_infinite = FALSE;
        self->release_duration_time =
            g_ascii_strtoull (self->release_duration_string, NULL, 10);
    }
    GST_INFO_OBJECT (self, "release-duration-time set to %s.",
        self->release_duration_string);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_VOLUME:
    GST_OBJECT_LOCK (self);
    self->volume = g_value_get_double (value);
    GST_INFO_OBJECT (self, "volume set to %g.", self->volume);
    GST_OBJECT_UNLOCK (self);
    break;

```

```

    case PROP_AUTOSTART:
        GST_OBJECT_LOCK (self);
        self->autostart = g_value_get_boolean (value);
        GST_INFO_OBJECT (self, "autostart set to %d.", self->autostart);
        GST_OBJECT_UNLOCK (self);
        break;

    case PROP_SOUND_NAME:
        GST_OBJECT_LOCK (self);
        g_free (self->sound_name);
        self->sound_name = g_value_dup_string (value);
        GST_INFO_OBJECT (self, "sound-name set to %s.", self->sound_name);
        GST_OBJECT_UNLOCK (self);
        break;

    default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
        break;
}

static void
gst_envelope_get_property (GObject * object, guint prop_id, GValue * value,
                          GParamSpec * pspec)
{
    GstEnvelope *self = GST_ENVELOPE (object);

    switch (prop_id)
    {
        case PROP_SILENT:
            GST_OBJECT_LOCK (self);
            g_value_set_boolean (value, self->silent);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_ATTACK_DURATION_TIME:
            GST_OBJECT_LOCK (self);
            g_value_set_uint64 (value, self->attack_duration_time);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_ATTACK_LEVEL:
            GST_OBJECT_LOCK (self);
            g_value_set_double (value, self->attack_level);
            GST_OBJECT_UNLOCK (self);

```

```
    break;

case PROP_DECAY_DURATION_TIME:
    GST_OBJECT_LOCK (self);
    g_value_set_uint64 (value, self->decay_duration_time);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_SUSTAIN_LEVEL:
    GST_OBJECT_LOCK (self);
    g_value_set_double (value, self->sustain_level);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_RELEASE_START_TIME:
    GST_OBJECT_LOCK (self);
    g_value_set_uint64 (value, self->release_start_time);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_RELEASE_DURATION_TIME:
    GST_OBJECT_LOCK (self);
    g_value_set_string (value, self->release_duration_string);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_VOLUME:
    GST_OBJECT_LOCK (self);
    g_value_set_double (value, self->volume);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_AUTOSTART:
    GST_OBJECT_LOCK (self);
    g_value_set_boolean (value, self->autostart);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_SOUND_NAME:
    GST_OBJECT_LOCK (self);
    g_value_set_string (value, self->sound_name);
    GST_OBJECT_UNLOCK (self);
    break;

default:
    G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
```

```

        break;
    }
}

/* This event handler is called when an event is sent to the source pad.
 * We care only about custom events: start, pause, continue and release.
 */
static gboolean
envelope_src_event_handler (GstBaseTransform * trans, GstEvent * event)
{
    GstEnvelope *self = GST_ENVELOPE (trans);
    const GstStructure *event_structure;
    const gchar *structure_name;
    const gchar *event_name;
    gchar *structure_as_string;
    gboolean ret;

    GST_OBJECT_LOCK (self);
    g_free (self->last_message);
    self->last_message = NULL;
    event_name = gst_event_type_get_name (GST_EVENT_TYPE (event));
    event_structure = gst_event_get_structure (event);
    if (event_structure != NULL)
    {
        structure_as_string = gst_structure_to_string (event_structure);
    }
    else
    {
        structure_as_string = g_strdup ("");
    }
    self->last_message =
        g_strdup_printf ("src event (%s:%s) type: %s (%d), %s %p",
                        GST_DEBUG_PAD_NAME (trans->sinkpad), event_name,
                        GST_EVENT_TYPE (event), structure_as_string, event);
    g_free (structure_as_string);
    GST_INFO_OBJECT (self, "%s", self->last_message);
    GST_OBJECT_UNLOCK (self);

    if (event_structure != NULL)
    {
        structure_name = gst_structure_get_name (event_structure);
    }
    else
    {
        structure_name = (gchar *) "";
    }
}

```

```

switch (GST_EVENT_TYPE (event))
{
case GST_EVENT_CUSTOM_UPSTREAM:
    if (g_strcmp0 (structure_name, (gchar *) "release") == 0)
    {
        /* This is a release event, which might be caused by receipt
         * of a Note Off MIDI message, or by an operator pushing a
         * stop button. Set a flag that will force release processing
         * to begin. */
        GST_INFO_OBJECT (self, "Received custom release event");
        GST_OBJECT_LOCK (self);
        self->external_release_seen = TRUE;
        GST_OBJECT_UNLOCK (self);
    }
    if (g_strcmp0 (structure_name, (gchar *) "start") == 0)
    {
        /* This is a start event, which might be caused by receipt
         * of a Note On MIDI message, or by an operator pushing a
         * start button. Flag that we have seen the message; the
         * next incoming buffer will start the envelope running
         * as soon as the previous release is complete. */
        GST_INFO_OBJECT (self, "Received custom start event");
        GST_OBJECT_LOCK (self);
        self->started = TRUE;
        GST_OBJECT_UNLOCK (self);
    }
    if (g_strcmp0 (structure_name, (gchar *) "pause") == 0)
    {
        /* This is a pause event, caused by an operator pushing the
         * pause button. Flag that we have seen the message; we will not
         * advance through the envelope until we see a continue event. */
        GST_INFO_OBJECT (self, "Received custom pause event");
        GST_OBJECT_LOCK (self);
        self->pause_seen = TRUE;
        GST_OBJECT_UNLOCK (self);
    }
    if (g_strcmp0 (structure_name, (gchar *) "continue") == 0)
    {
        /* This is a continue event, caused by an operator pushing the
         * continue button to cancel a previous pause. Flag that we
         * have seen the message. We don't simply clear the pause flag
         * because we want to notice the transition. */
        GST_INFO_OBJECT (self, "Received custom continue event");
        GST_OBJECT_LOCK (self);
        self->continue_seen = TRUE;
    }
}

```

```

        GST_OBJECT_UNLOCK (self);
    }
    break;

case GST_EVENT_EOS:
    /* We have reached the end of the incoming data stream.
     * Set a flag that will cause the sound to stop. */
    GST_DEBUG_OBJECT (self, "envelope completion EOS");
    GST_OBJECT_LOCK (self);
    self->external_completion_seen = TRUE;
    GST_OBJECT_UNLOCK (self);
    break;

default:
    break;
}

/* When we are done with the event, do the default processing on it. */
ret = GST_BASE_TRANSFORM_CLASS (parent_class)->src_event (trans, event);

return ret;
}

/* This event handler is called when an event is sent to the sink pad.
 * The event we care about here is the completion event, which is sent
 * by the looper when it reaches the end of its buffer.
 */
static gboolean
envelope_sink_event_handler (GstBaseTransform * trans, GstEvent * event)
{
    GstEnvelope *self = GST_ENVELOPE (trans);
    const GstStructure *event_structure;
    const gchar *event_name;
    const gchar *structure_name;
    gchar *structure_as_string;
    gboolean ret;

    GST_OBJECT_LOCK (self);
    g_free (self->last_message);
    self->last_message = NULL;
    event_name = gst_event_type_get_name (GST_EVENT_TYPE (event));
    event_structure = gst_event_get_structure (event);
    if (event_structure != NULL)
    {
        structure_as_string = gst_structure_to_string (event_structure);
    }

```



```

else
{
    structure_as_string = g_strdup ("");
}
self->last_message =
    g_strdup_printf ("sink event (%s:%s) type: %s (%d), %s %p",
                     GST_DEBUG_PAD_NAME (trans->sinkpad), event_name,
                     GST_EVENT_TYPE (event), structure_as_string, event);
g_free (structure_as_string);
GST_INFO_OBJECT (self, "%s", self->last_message);
GST_OBJECT_UNLOCK (self);

if (event_structure != NULL)
{
    structure_name = gst_structure_get_name (event_structure);
}
else
{
    structure_name = (gchar *) "";
}

switch (GST_EVENT_TYPE (event))
{
    case GST_EVENT_CUSTOM_DOWNSTREAM:
        GST_INFO_OBJECT (self, "Processing %s.", structure_name);
        if (g_strcmp0 (structure_name, (gchar *) "complete") == 0)
        {
            /* This is a complete event, which is sent by the looper when
             * it reaches the end of its buffer. Set a flag that will
             * cause the sound to stop. */
            GST_DEBUG_OBJECT (self, "envelope completion message");
            GST_OBJECT_LOCK (self);
            self->external_completion_seen = TRUE;
            GST_OBJECT_UNLOCK (self);
        }
        break;

    default:
        break;
}

/* When we are done with the event, do the default processing on it. */
ret = GST_BASE_TRANSFORM_CLASS (parent_class)->sink_event (trans, event);

return ret;
}

```

```
/* entry point to initialize the plug-in
 * initialize the plug-in itself
 * register the element factories and other features
 */
static gboolean
envelope_init (GstPlugin * envelope)
{
    return gst_element_register (envelope, "envelope", GST_RANK_NONE,
                                GST_TYPE_ENVELOPE);
}

GST_PLUGIN_DEFINE (GST_VERSION_MAJOR, GST_VERSION_MINOR, envelope,
                  "Shape the sound using an a-d-s-r envelope", envelope_init,
                  VERSION, "LGPL", "GStreamer", "http://gstreamer.net/")
```

7 gstenvelope.h

```

/*
 * File: gstenvelope.h, part of show_control, a GStreamer application.
 *
 * Copyright © 2016 John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, see https://gnu.org/licenses
 * or write to:
 * Free Software Foundation, Inc.
 * 51 Franklin Street, Fifth Floor
 * Boston, MA 02111-1301
 * USA.
 */

#ifdef __GST_ENVELOPE_H__
#define __GST_ENVELOPE_H__

#include <gst/gst.h>
#include <gst/base/gstbasetransform.h>
#include <gst/audio/audio.h>
#include <gst/audio/gstaudiofilter.h>

G_BEGIN_DECLS
#define GST_TYPE_ENVELOPE \
    (gst_envelope_get_type())
#define GST_ENVELOPE(obj) \
    (G_TYPE_CHECK_INSTANCE_CAST((obj), GST_TYPE_ENVELOPE, GstEnvelope))
#define GST_ENVELOPE_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_CAST((klass), GST_TYPE_ENVELOPE, GstEnvelopeClass))
#define GST_IS_ENVELOPE(obj) \
    (G_TYPE_CHECK_INSTANCE_TYPE((obj), GST_TYPE_ENVELOPE))
#define GST_IS_ENVELOPE_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_TYPE((klass), GST_TYPE_ENVELOPE))
typedef struct _GstEnvelope GstEnvelope;

```

```

typedef struct _GstEnvelopeClass GstEnvelopeClass;

struct _GstEnvelope
{
    GstAudioFilter element;

    /* Parameters */
    gboolean silent;
    GstClockTimeDiff attack_duration_time;
    gdouble attack_level;
    GstClockTimeDiff decay_duration_time;
    gdouble sustain_level;
    GstClockTimeDiff release_start_time;
    gchar *release_duration_string;
    gdouble volume;
    gboolean autostart;
    gchar *sound_name;

    /* Locals */
    GstClockTimeDiff release_duration_time;
    gboolean release_duration_infinite;
    gboolean release_started;
    gdouble release_started_volume;
    gdouble last_volume;
    GstClockTimeDiff release_started_time;
    gchar *last_message;
    gboolean external_release_seen;
    gboolean external_completion_seen;
    gboolean application_notified_release;
    gboolean application_notified_completion;
    gboolean completed;
    gboolean running;
    gboolean started;
    gboolean pause_seen;
    gboolean continue_seen;
    gboolean pausing;
    GstClockTime base_time;
    GstClockTimeDiff pause_time;
    GstClockTime pause_start_time;
};

struct _GstEnvelopeClass
{
    GstAudioFilterClass parent_class;
};

```

```
GType gst_envelope_get_type (void);
```

```
G_END_DECLS
```

```
#endif /* __GST_ENVELOPE_H__ */
```

8 gstlooper.c

```

/*
 * gstlooper.c, a file in sound_effects_player, a component of Show_control,
 * which is a Gstreamer application. Much of the code in this file is based
 * on Gstreamer examples and tutorials.
 *
 * Copyright © 2016 John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, see https://gnu.org/licenses
 * or write to
 * Free Software Foundation, Inc.
 * 51 Franklin Street, Fifth Floor
 * Boston, MA 02111-1301
 * USA.
 */

/**
 * SECTION:element-looper
 * @short_description: Repeat a section of the input stream.
 *
 * This element places its input into a buffer, then sends it downstream
 * a specified number of times. Parameters control the number of times
 * the data is sent, and can specify a start and end point within the data.
 * Messages are used to start and stop the element, and to pause it.
 *
 * Properties are:
 *
 * #GstLooper:loop-to is the beginning of the section to repeat, in nanoseconds
 * from the beginning of the input. Default is 0.
 *
 * #GstLooper:loop-from is the end of the section to repeat, also in
 * nanoseconds. If the sample rate is less than 1,000,000,000
 * samples per second, looping at exactly loop-from and loop-to might not
 * be possible, in which case the loop starts at the beginning of the sample

```

```

* at loop-to and ends after the sample at loop-from. Default is 0, which
* suppresses looping.
*
* #GstLooper:loop-limit is the number of times to repeat the loop; 0 means
* repeat indefinitely, which is the default.
*
* #GstLooper:max-duration-time, if specified, is the maximum amount of time
* from the source to be held for repeating, in nanoseconds.
* This can be useful with live or infinite sources. Note that this element
* can itself be an infinite source for its downstream consumers, even if its
* upstream is finite or limited by max-duration. If max-duration is not
* specified, the looper element will attempt to absorb all sound provided to
* its sink pad. Default is that max-duration-time is not specified.
*
* #GstLooper:start-time is the offset from the beginning of the input to
* start the output, in nanoseconds. Sound before start-time is not sent
* downstream unless loop-to is before start-time. Default is 0.
*
* #GstLooper:autostart. Normally, this element sends silence until it
* receives a Start message. By setting the Autostart parameter to TRUE you
* can make it start as soon as it has gotten all the sound data it needs.
* Default is FALSE.
*
* #GstLooper:file-location. This element will attempt to use pull mode to get
* sound data as quickly as possible from its upstream source, but fall back to
* push mode if necessary. An even faster alternative to getting the data in
* pull mode is to specify the file-location parameter. Gstlooper will read
* the data segments from that file rather than wait for the data to come from
* upstream. The metadata will still come from upstream. The specified file
* must be a WAV file. Default is that file-location is not specified, so
* no file is read.
*
* Receipt of a Release message causes looping to terminate, which means
* reaching the end of the loop no longer causes sound to be sent from the
* beginning of the loop. The amount of sound sent after a Release message can
* be as little as 0, if the looper element was about to loop, and there is no
* sound after the loop-from time. Therefore, if you need sound after the
* Release message, leave enough sound after loop-from to handle the worst case.
*
* <refsect2>
* <title>Example launch line</title>
* |[
* gst-launch-1.0 -v -m audiotestsrc ! audio/x-raw,rate=96000,format=S32LE !
* ↪ looper start-time=1000000000 max-duration=5000000000 loop-from=1000000000
* ↪ loop-limit=2 autostart=TRUE ! fakesink silent=TRUE
* ]|

```

```

* </refsect2>
*/

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <string.h>
#include <math.h>
#include <errno.h>
#include <stdio.h>
#include <gst/gst.h>
#include <gst/audio/audio.h>

#include "gstlooper.h"

#define STATIC_CAPS \
    GST_STATIC_CAPS (GST_AUDIO_CAPS_MAKE \
        (" { S8, U8, " GST_AUDIO_NE (S16) ", " GST_AUDIO_NE (S32) \
            ", " GST_AUDIO_NE (F32) ", " GST_AUDIO_NE (F64)" } ") \
            ", layout = (string) interleaved")
#define SINK_TEMPLATE \
    GST_STATIC_PAD_TEMPLATE ("sink", GST_PAD_SINK, GST_PAD_ALWAYS, STATIC_CAPS)
#define SRC_TEMPLATE \
    GST_STATIC_PAD_TEMPLATE ("src", GST_PAD_SRC, GST_PAD_ALWAYS, STATIC_CAPS)

static GstStaticPadTemplate sinktemplate = SINK_TEMPLATE;
static GstStaticPadTemplate srctemplate = SRC_TEMPLATE;

GST_DEBUG_CATEGORY_STATIC (looper);
#define GST_CAT_DEFAULT (looper)

/* Filter signals and args */
enum
{
    /* FILL ME */
    LAST_SIGNAL
};

enum
{
    PROP_0,
    PROP_SILENT,
    PROP_LOOP_TO,
    PROP_LOOP_FROM,
    PROP_LOOP_LIMIT,

```



```

    PROP_MAX_DURATION,
    PROP_START_TIME,
    PROP_AUTOSTART,
    PROP_FILE_LOCATION,
    PROP_ELAPSED_TIME,
    PROP_REMAINING_TIME
};

#define DEBUG_INIT \
    GST_DEBUG_CATEGORY_INIT (looper, "looper", 0, \
        "Repeat a section of the stream");
#define gst_looper_parent_class parent_class
G_DEFINE_TYPE_WITH_CODE (GstLooper, gst_looper, GST_TYPE_ELEMENT, DEBUG_INIT);

/* Forward declarations */

/* deallocate */
static void gst_looper_finalize (GObject * object);

/* set the value of a property */
static void gst_looper_set_property (GObject * object, guint prop_id,
    const GValue * value,
    GParamSpec * pspec);

/* Compute the remaining running time of the sound. */
static gint64 compute_remaining_time (GstLooper * object);
/* fetch the value of a property */
static void gst_looper_get_property (GObject * object, guint prop_id,
    GValue * value, GParamSpec * pspec);

/* process incoming data from the sink pad */
static void gst_looper_pull_data_from_upstream (GstPad * pad);
static GstFlowReturn gst_looper_chain (GstPad * pad, GstObject * parent,
    GstBuffer * buffer);

/* send outgoing data to the source pad */
static void gst_looper_push_data_downstream (GstPad * pad);
/* process events and queries on the sink and source pads */
static gboolean gst_looper_handle_sink_event (GstPad * pad,
    GstObject * parent,
    GstEvent * event);
static gboolean gst_looper_handle_sink_query (GstPad * pad,
    GstObject * parent,
    GstQuery * query);
static gboolean gst_looper_handle_src_event (GstPad * pad, GstObject * parent,
    GstEvent * event);
static gboolean gst_looper_handle_src_query (GstPad * pad, GstObject * parent,
    GstQuery * query);
/* process queries directed to the element itself */

```

```

static gboolean gst_looper_handle_query (GstElement * element,
                                         GstQuery * query);

/* send data downstream in pull mode */
static GstFlowReturn gst_looper_get_range (GstPad * pad, GstObject * parent,
                                         guint64 offset, guint length,
                                         GstBuffer ** buffer);

/* activate and deactivate the source and sink pads */
static gboolean gst_looper_activate_sink_pad (GstPad * pad,
                                              GstObject * parent);
static gboolean gst_looper_src_activate_mode (GstPad * pad,
                                              GstObject * parent,
                                              GstPadMode mode,
                                              gboolean active);
static gboolean gst_looper_sink_activate_mode (GstPad * pad,
                                              GstObject * parent,
                                              GstPadMode mode,
                                              gboolean active);

/* process a state change */
static GstStateChangeReturn gst_looper_change_state (GstElement * element,
                                                    GstStateChange
                                                    transition);

/* local subroutines */

/* convert a time in nanoseconds into a position in the buffer */
static guint64 round_up_to_position (GstLooper * self,
                                     guint64 specified_time);
static guint64 round_down_to_position (GstLooper * self,
                                       guint64 specified_time);

/* Read the data chunks from a WAV file into the local buffer. */
static gboolean read_wav_file_data (GstLooper * self, guint64 max_position);

/* GObject vmethod implementations */

/* initialize the looper's class */
static void
gst_looper_class_init (GstLooperClass * klass)
{
    GObjectClass *gobject_class;
    GstElementClass *gstelement_class;
    GParamSpec *param_spec;
    gchar *string_default;

    gobject_class = (GObjectClass *) klass;
    gstelement_class = (GstElementClass *) klass;

```

```

gobject_class->set_property = gst_looper_set_property;
gobject_class->get_property = gst_looper_get_property;

/* Properties */
param_spec =
    g_param_spec_boolean ("silent", "Silent", "Produce verbose output ?",
                          FALSE, G_PARAM_READWRITE | GST_PARAM_CONTROLLABLE);
g_object_class_install_property (gobject_class, PROP_SILENT, param_spec);

param_spec =
    g_param_spec_uint64 ("loop-to", "Loop_to", "Start of section to repeat",
                        0, G_MAXUINT64, 0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_LOOP_TO, param_spec);

param_spec =
    g_param_spec_uint64 ("loop-from", "Loop_from", "End of section to repeat",
                        0, G_MAXUINT64, 0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_LOOP_FROM, param_spec);

param_spec =
    g_param_spec_uint ("loop-limit", "Loop_limit",
                      "Number of times to repeat; 0 means forever", 0,
                      G_MAXUINT, 0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_LOOP_LIMIT,
                                param_spec);

param_spec =
    g_param_spec_uint64 ("max-duration", "Max_duration",
                        "Maximum time to accept from upstream", 0,
                        G_MAXUINT64, 0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_MAX_DURATION,
                                param_spec);

param_spec =
    g_param_spec_uint64 ("start-time", "Start_time",
                        "Offset from the start to begin outputting", 0,
                        G_MAXUINT64, 0, G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_START_TIME,
                                param_spec);

param_spec =
    g_param_spec_boolean ("autostart", "Autostart", "automatic start", FALSE,
                          G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_AUTOSTART, param_spec);

```

```

string_default = g_strdup ("");
param_spec =
    g_param_spec_string ("file-location", "File_location",
        "The location of the WAV file "
        "for fast loading of data", string_default,
        G_PARAM_READWRITE);
g_object_class_install_property (gobject_class, PROP_FILE_LOCATION,
    param_spec);

param_spec =
    g_param_spec_string ("elapsed-time", "elapsed_time",
        "Time in seconds since the sound was started",
        string_default, G_PARAM_READABLE);
g_object_class_install_property (gobject_class, PROP_ELAPSED_TIME,
    param_spec);

param_spec =
    g_param_spec_string ("remaining-time", "remaining_time",
        "Time in seconds until the sound stops",
        string_default, G_PARAM_READABLE);
g_object_class_install_property (gobject_class, PROP_REMAINING_TIME,
    param_spec);

g_free (string_default);
string_default = NULL;

/* Set several parent class virtual functions. */
gobject_class->finalize = gst_looper_finalize;
gst_element_class_add_pad_template (gstelement_class,
    gst_static_pad_template_get
    (&src_template));
gst_element_class_add_pad_template (gstelement_class,
    gst_static_pad_template_get
    (&sink_template));
gst_element_class_set_static_metadata (gstelement_class, "Looper",
    "Generic",
    "Repeat a section of the input stream",
    "John Sauter <John_Sauter@"
    "systemeyescomputerstore.com>");

gstelement_class->change_state =
    GST_DEBUG_FUNCPTR (gst_looper_change_state);
gstelement_class->query = GST_DEBUG_FUNCPTR (gst_looper_handle_query);
}

/* initialize the new element
 * initialize instance structure

```

```

*/
static void
gst_looper_init (GstLooper * self)
{
    self->silent = FALSE;
    self->loop_to = 0;
    self->loop_from = 0;
    self->loop_limit = 0;
    self->loop_counter = 0;
    self->timestamp_offset = 0;
    self->max_duration = 0;
    self->start_time = 0;
    self->autostart = FALSE;
    self->started = FALSE;
    self->completion_sent = FALSE;
    self->paused = FALSE;
    self->continued = FALSE;
    self->released = FALSE;
    self->data_buffered = FALSE;
    self->local_buffer = gst_buffer_new ();
    self->local_buffer_fill_level = 0;
    self->local_buffer_drain_level = 0;
    self->pull_level = 0;
    self->local_buffer_size = 0;
    self->bytes_per_ns = 0.0;
    self->local_clock = 0;
    self->elapsed_time = 0;
    self->width = 0;
    self->channel_count = 0;
    self->format = NULL;
    self->data_rate = 0;
    self->send_EOS = FALSE;
    self->state_change_pending = FALSE;
    self->src_pad_task_running = FALSE;
    self->sink_pad_task_running = FALSE;
    self->file_location = NULL;
    self->file_location_specified = FALSE;
    self->seen_incoming_data = FALSE;
    g_rec_mutex_init (&self->interlock);
    self->silence_byte = 0;

    /* create the pads */
    self->sinkpad = gst_pad_new_from_static_template (&sinktemplate, "sink");
    gst_pad_set_chain_function (self->sinkpad,
                               GST_DEBUG_FUNCPTR (gst_looper_chain));
}

```

```

gst_pad_set_activate_function (self->sinkpad, gst_looper_activate_sink_pad);
gst_pad_set_activatemode_function (self->sinkpad,
                                   GST_DEBUG_FUNCPTR
                                   (gst_looper_sink_activate_mode));
gst_pad_set_event_function (self->sinkpad,
                             GST_DEBUG_FUNCPTR
                             (gst_looper_handle_sink_event));
gst_pad_set_query_function (self->sinkpad,
                             GST_DEBUG_FUNCPTR
                             (gst_looper_handle_sink_query));
/* Set the pad to forward all caps-related events and queries to its
 * peer pad on the upstream element. This implies that incoming data
 * and outgoing data will have the same format. */
GST_PAD_SET_PROXY_CAPS (self->sinkpad);
gst_element_add_pad (GST_ELEMENT (self), self->sinkpad);

self->srcpad = gst_pad_new_from_static_template (&src_template, "src");
gst_pad_set_activatemode_function (self->srcpad,
                                   GST_DEBUG_FUNCPTR
                                   (gst_looper_src_activate_mode));
gst_pad_set_getrange_function (self->srcpad,
                               GST_DEBUG_FUNCPTR (gst_looper_get_range));
gst_pad_set_event_function (self->srcpad,
                             GST_DEBUG_FUNCPTR
                             (gst_looper_handle_src_event));
gst_pad_set_query_function (self->srcpad,
                             GST_DEBUG_FUNCPTR
                             (gst_looper_handle_src_query));
/* Set the pad to forward all caps-related events and queries to its
 * peer pad on the downstream element. This implies that incoming data
 * and outgoing data will have the same format. */
GST_PAD_SET_PROXY_CAPS (self->srcpad);
gst_element_add_pad (GST_ELEMENT (self), self->srcpad);

return;
}

/* Deallocate everything */
static void
gst_looper_finalize (GObject * object)
{
    GstLooper *self = GST_LOOPER (object);

    if (self->local_buffer != NULL)
    {
        gst_buffer_unref (self->local_buffer);
    }
}

```

```

        self->local_buffer = NULL;
    }
    if (self->format != NULL)
    {
        g_free (self->format);
        self->format = NULL;
    }
    if (self->file_location != NULL)
    {
        g_free (self->file_location);
        self->file_location = NULL;
        self->file_location_specified = FALSE;
    }
    g_rec_mutex_clear (&self->interlock);
    G_OBJECT_CLASS (parent_class)->finalize (object);
    return;
}

/* Process a state change. The state either climbs up from NULL to READY
 * to PAUSED to RUNNING, or down from RUNNING to PAUSED to READY to NULL. */
static GstStateChangeReturn
gst_looper_change_state (GstElement * element, GstStateChange transition)
{
    GstLooper *self;
    GstStateChangeReturn result = GST_STATE_CHANGE_SUCCESS;

    self = GST_LOOPER (element);

    switch (transition)
    {
        case GST_STATE_CHANGE_NULL_TO_READY:
            GST_DEBUG_OBJECT (self, "state changed from null to ready");
            break;

        case GST_STATE_CHANGE_READY_TO_PAUSED:
            g_rec_mutex_lock (&self->interlock);
            self->started = FALSE;
            self->completion_sent = FALSE;
            self->released = FALSE;
            self->paused = FALSE;
            self->continued = FALSE;
            self->data_buffered = FALSE;
            GST_DEBUG_OBJECT (self, "state changed from ready to paused");
            g_rec_mutex_unlock (&self->interlock);
            break;
    }
}

```

```

case GST_STATE_CHANGE_PAUSED_TO_PLAYING:
    g_rec_mutex_lock (&self->interlock);
    if ((self->data_buffered) && (!self->src_pad_task_running))
    {
        /* Start the task which pushes data downstream. */
        result =
            gst_pad_start_task (self->srcpad,
                                (GstTaskFunction)
                                gst_looper_push_data_downstream, self->srcpad,
                                NULL);

        if (!result)
        {
            GST_DEBUG_OBJECT (self,
                              "failed to start push task after state change");
        }
        self->src_pad_task_running = TRUE;
    }
    GST_DEBUG_OBJECT (self, "state changed from paused to playing");
    g_rec_mutex_unlock (&self->interlock);
    break;

default:
    break;
}

/* Let our parent do its state change after us if we are going up,
 * or before us if we are going down. */
result =
    GST_ELEMENT_CLASS (parent_class)->change_state (element, transition);

if (result == GST_STATE_CHANGE_FAILURE)
{
    GST_DEBUG_OBJECT (self, "failure of parent during state change");
    return result;
}

switch (transition)
{
case GST_STATE_CHANGE_PLAYING_TO_PAUSED:
    g_rec_mutex_lock (&self->interlock);

    /* The pipeline is pausing. If the task that sends data
     * downstream is still running, tell it to send EOS and
     * complete the state transition. If it is not running,
     * complete the state transition here. */

```



```

    if (self->src_pad_task_running)
    {
        self->send_EOS = TRUE;
        result = GST_STATE_CHANGE_ASYNC;
        self->state_change_pending = TRUE;
        GST_DEBUG_OBJECT (self, "state changing from playing to paused");
    }
else
    {
        GST_DEBUG_OBJECT (self, "state changed from playing to paused");
    }

g_rec_mutex_unlock (&self->interlock);
break;

case GST_STATE_CHANGE_PAUSED_TO_READY:
    g_rec_mutex_lock (&self->interlock);
    /* If the tasks that are pushing data downstream or pulling data from
     * upstream are still running, kill them. */
    if (self->src_pad_task_running)
    {
        gst_pad_stop_task (self->srcpad);
        self->src_pad_task_running = FALSE;
    }
    if (self->sink_pad_task_running)
    {
        gst_pad_stop_task (self->sinkpad);
        self->sink_pad_task_running = FALSE;
    }
    self->data_buffered = FALSE;
    self->started = FALSE;
    self->completion_sent = FALSE;
    self->paused = FALSE;
    self->continued = FALSE;
    self->released = FALSE;
    GST_DEBUG_OBJECT (self, "state changed from paused to ready");
    g_rec_mutex_unlock (&self->interlock);
    break;

case GST_STATE_CHANGE_READY_TO_NULL:
    GST_DEBUG_OBJECT (self, "state changed from ready to null");
    break;

default:
    break;
}

```

```

    return result;
}

/* Activate function for the sink pad. See if pull mode is supported, use push
 * mode if it isn't. */
static gboolean
gst_looper_activate_sink_pad (GstPad * pad, GstObject * parent)
{
    GstLooper *self = GST_LOOPER (parent);
    GstQuery *query;
    gboolean pull_mode_supported, result;

    GST_DEBUG_OBJECT (self, "activating sink pad");

    /* First check what upstream scheduling is supported */
    query = gst_query_new_scheduling ();
    result = gst_pad_peer_query (pad, query);
    if (result)
    {
        /* See if pull mode is supported. */
        GST_DEBUG_OBJECT (self, "checking upstream peer for being seekable");
        pull_mode_supported =
            gst_query_has_scheduling_mode_with_flags (query, GST_PAD_MODE_PULL,
                                                    GST_SCHEDULING_FLAG_SEEKABLE);
    }
    else
    {
        pull_mode_supported = FALSE;
    }
    gst_query_unref (query);

    if (pull_mode_supported)
    {
        /* We can activate in pull mode. Gstreamer will also activate the
         * upstream peer in pull mode. */
        GST_DEBUG_OBJECT (self, "will activate sink pad in pull mode");
        return gst_pad_activate_mode (pad, GST_PAD_MODE_PULL, TRUE);
    }

    GST_INFO_OBJECT (self, "falling back to push mode");
    /* The upstream peer does not support pull mode, so we activate in push
     * mode as a fallback. */
    return gst_pad_activate_mode (pad, GST_PAD_MODE_PUSH, TRUE);
}

```

```

/* Activate or deactivate the source pad in either push or pull mode. Note
 * that pull mode is not fully implemented. */
static gboolean
gst_looper_src_activate_mode (GstPad * pad, GstObject * parent,
                             GstPadMode mode, gboolean active)
{
    GstLooper *self = GST_LOOPER (parent);
    gboolean result;

    switch (mode)
    {
        case GST_PAD_MODE_PULL:
            g_rec_mutex_lock (&self->interlock);
            /* The source pad is operating in pull mode. Downstream will call our
 * getrange function to get data. */
            if (active)
            {
                GST_DEBUG_OBJECT (self, "activating source pad in pull mode");
                result = TRUE;
                self->src_pad_mode = mode;
                self->src_pad_active = TRUE;
            }
            else
            {
                GST_DEBUG_OBJECT (self, "deactivating source pad in pull mode");
                self->src_pad_mode = mode;
                self->src_pad_active = FALSE;
                result = TRUE;
            }
            g_rec_mutex_unlock (&self->interlock);
            break;

        case GST_PAD_MODE_PUSH:
            g_rec_mutex_lock (&self->interlock);
            /* The source pad is operating in push mode. To activate we will
 * start a task which pushes out buffers when our local buffer is full.
 */
            if (active)
            {
                GST_DEBUG_OBJECT (self, "activating source pad in push mode");
                self->src_pad_mode = mode;
                if ((self->data_buffered) && (!self->src_pad_task_running))
                {
                    /* Start the task which pushes data downstream. */
                    result =
                        gst_pad_start_task (self->srcpad,

```

```

                                (GstTaskFunction)
                                gst_looper_push_data_downstream,
                                self->srcpad, NULL);

    if (!result)
    {
        GST_DEBUG_OBJECT (self,
                           "failed to start push task "
                           "after pad activate");
    }
    self->src_pad_task_running = TRUE;
}
self->src_pad_active = TRUE;
result = TRUE;
}
else
{
    GST_DEBUG_OBJECT (self, "deactivating source pad in push mode");
    self->src_pad_mode = mode;
    self->src_pad_active = FALSE;
    /* If the task that is sending data downstream is still running,
       * have it send EOS and terminate. */
    self->send_EOS = TRUE;
    result = TRUE;
}
g_rec_mutex_unlock (&self->interlock);
break;

default:
    GST_DEBUG_OBJECT (pad, "unknown source pad activation mode: %d.", mode);
    result = FALSE;
    break;
}

return result;
}

/* Called repeatedly with @pad as the source pad. This function pushes out
 * data to the downstream peer element. */
static void
gst_looper_push_data_downstream (GstPad * pad)
{
    GstLooper *self = GST_LOOPER (GST_PAD_PARENT (pad));
    GstBuffer *buffer;
    GstEvent *event;
    GstStructure *structure;
    GstMemory *memory_out;

```

```

GstMapInfo memory_in_info, memory_out_info;
gsize data_size;
gboolean result, within_loop;
GstFlowReturn flow_result;
gboolean send_silence;
gboolean buffer_complete;
gboolean exiting = FALSE;
guint64 loop_from_position, loop_to_position;

/* We have a recursive mutex which prevents this task from running
 * while some other part of this plugin is running on a different task.
 * We take the interlock here, therefore waiting for the other task to
 * release it. Before we exit we must release this mutex or the plugin
 * will grind to a halt. */
g_rec_mutex_lock (&self->interlock);

/* If we should not be running, just exit. */
if (!self->src_pad_task_running)
{
    GST_DEBUG_OBJECT (self, "data pusher should not be running");
    g_rec_mutex_unlock (&self->interlock);
    return;
}

/* If requested, or if we are autostarted and have reached the end of
 * the buffer, send an end-of-stream message and stop. */
if ((self->send_EOS)
    || (self->autostart
        && (self->local_buffer_drain_level >= self->local_buffer_size)))
{
    GST_INFO_OBJECT (self, "pushing an EOS event");
    event = gst_event_new_eos ();
    result = gst_pad_push_event (self->srcpad, event);
    if (!result)
    {
        GST_DEBUG_OBJECT (self, "failed to push an EOS event");
    }
    self->send_EOS = FALSE;

    /* Having pushed an EOS event, we are done. */
    GST_DEBUG_OBJECT (self, "pausing source pad task");
    result = gst_pad_pause_task (self->srcpad);
    if (!result)
    {
        GST_DEBUG_OBJECT (self, "failed to pause source pad task");
    }
}

```

```

        self->src_pad_task_running = FALSE;
        exiting = TRUE;
    }

    /* If we are making the transition from the playing to the paused
     * state, complete the transition here.
     */
    if (self->state_change_pending)
    {
        GST_DEBUG_OBJECT (self, "completing state change");
        gst_element_continue_state (GST_ELEMENT (self),
                                    GST_STATE_CHANGE_SUCCESS);
        GST_DEBUG_OBJECT (self, "state change completed");
        self->state_change_pending = FALSE;
        exiting = TRUE;
    }

    /* If we either sent an EOS, or completed a state change, or both,
     * exit now. */
    if (exiting)
    {
        g_rec_mutex_unlock (&self->interlock);
        return;
    }

    /* If we are flushing, do not push any data. */
    if (self->src_pad_flushing)
    {
        GST_DEBUG_OBJECT (self, "data pusher should not run while flushing");
        g_rec_mutex_unlock (&self->interlock);
        return;
    }

    /* If we were paused but have since received a continue message,
     * stop pausing. */
    if (self->paused && self->continued)
    {
        self->paused = FALSE;
        self->continued = FALSE;
    }

    /* If we have not received a start event, or if we have completely
     * drained the buffer, or we are paused, remember to send silence
     * downstream. */
    send_silence = FALSE;

```

```

buffer_complete = FALSE;
if (!self->started)
{
    send_silence = TRUE;
}
else
{
    if (self->local_buffer_drain_level >= self->local_buffer_size)
    {
        send_silence = TRUE;
        buffer_complete = TRUE;
    }
}
if (self->paused && !self->continued)
    send_silence = TRUE;

/* If we are just reaching the end of the buffer, and we are not
 * autostarted, send a message downstream to the envelope plugin,
 * so it knows that the sound is complete. We don't send EOS because
 * we don't want to drain the pipeline--we may get another Start message
 * asking us to play this sound again. Note that, if we are
 * autostarted, we sent an EOS message above and don't get here. */
if (self->started && buffer_complete && !self->completion_sent)
{
    GST_DEBUG_OBJECT (self, "pushing a completion event");
    self->started = FALSE;
    self->released = FALSE;
    structure = gst_structure_new_empty ((gchar *) "complete");
    event = gst_event_new_custom (GST_EVENT_CUSTOM_DOWNSTREAM, structure);
    result = gst_pad_push_event (self->srcpad, event);
    if (!result)
    {
        GST_DEBUG_OBJECT (self, "failed to push a completion event");
    }
    else
    {
        GST_DEBUG_OBJECT (self, "successfully pushed a completion event");
    }
    self->completion_sent = TRUE;
}

if (send_silence)
{
    GST_DEBUG_OBJECT (self, "sending silence downstream");
    /* Compute the number of bytes required to hold 40 milliseconds
     * of silence. */

```

```

data_size =
    self->width * self->data_rate * self->channel_count / (8000 / 40);
/* Allocate that much memory, and place it in our output buffer. */
memory_out = gst_allocator_alloc (NULL, data_size, NULL);
buffer = gst_buffer_new ();
gst_buffer_append_memory (buffer, memory_out);
/* Fill the buffer with the silence byte. */
gst_buffer_map (buffer, &memory_out_info, GST_MAP_WRITE);
gst_buffer_memset (buffer, 0, self->silence_byte, memory_out_info.size);
/* Set the time stamps in the buffer. */
GST_BUFFER_PTS (buffer) = self->local_clock;
GST_BUFFER_DTS (buffer) = self->local_clock;
GST_BUFFER_DURATION (buffer) =
    memory_out_info.size / self->bytes_per_ns;
/* Advance our clock. */
self->local_clock =
    self->local_clock + (memory_out_info.size / self->bytes_per_ns);
GST_BUFFER_OFFSET (buffer) = self->local_buffer_drain_level;
GST_BUFFER_OFFSET_END (buffer) =
    self->local_buffer_drain_level + memory_out_info.size;
gst_buffer_unmap (buffer, &memory_out_info);
/* Send the buffer downstream. */
GST_DEBUG_OBJECT (self,
    "pushing %" G_GUINT64_FORMAT " bytes of silence.",
    data_size);
/* We must unlock before we push, since pushing can cause a query to
 * come back upstream on a different task before it completes. */
g_rec_mutex_unlock (&self->interlock);

flow_result = gst_pad_push (self->srcpad, buffer);
if (flow_result != GST_FLOW_OK)
{
    GST_DEBUG_OBJECT (self, "pad push of silence returned %s",
        gst_flow_get_name (flow_result));
}

GST_DEBUG_OBJECT (self, "push of silence completed");
return;
}

/* There is more data to send. Allocate a new buffer to send downstream,
 * and copy data from our local buffer to it. */
buffer = gst_buffer_new ();
/* We send 40 milliseconds of buffer data at a time, but not more than
 * is left in our local buffer, and not more than we need to reach the
 * end of the loop, if we are looping. */

```



```

data_size =
    self->width * self->data_rate * self->channel_count / (8000 / 40);
if (data_size > self->local_buffer_size - self->local_buffer_drain_level)
{
    data_size = self->local_buffer_size - self->local_buffer_drain_level;
}

/* We are within the loop if this isn't our last time around. */
within_loop = FALSE;
loop_from_position = round_up_to_position (self, self->loop_from);
if ((!self->released) && (self->loop_from > 0)
    && (self->local_buffer_drain_level <= loop_from_position))
{
    if ((self->loop_limit == 0) || (self->loop_counter < self->loop_limit))
    {
        within_loop = TRUE;
    }
}

/* If we are within the loop but at the very end, go back to the beginning.
*/
if (within_loop && (self->local_buffer_drain_level == loop_from_position))
{
    loop_to_position = round_down_to_position (self, self->loop_to);
    self->local_buffer_drain_level = loop_to_position;
    self->loop_counter = self->loop_counter + 1;
    GST_DEBUG_OBJECT (self,
        "loop counter %" G_GUINT64_FORMAT ", looping from %"
        GST_TIME_FORMAT " to %" GST_TIME_FORMAT ".",
        self->loop_counter,
        GST_TIME_ARGS (loop_from_position /
            self->bytes_per_ns),
        GST_TIME_ARGS (loop_to_position /
            self->bytes_per_ns));
}

/* If the loop is very short, we will output buffers of its length. */
if (within_loop
    && (data_size > loop_from_position - self->local_buffer_drain_level))
{
    data_size = loop_from_position - self->local_buffer_drain_level;
}

/* now that we know how much memory we need, allocate it */
memory_out = gst_allocator_alloc (NULL, data_size, NULL);
/* place the memory in the output buffer and mark it for writing */
gst_buffer_append_memory (buffer, memory_out);
gst_buffer_map (buffer, &memory_out_info, GST_MAP_WRITE);
/* mark our local buffer for reading */

```

```

gst_buffer_map (self->local_buffer, &memory_in_info, GST_MAP_READ);
/* copy data from our local buffer to the output buffer */
gst_buffer_fill (buffer, 0,
                 memory_in_info.data + self->local_buffer_drain_level,
                 memory_out_info.size);

/* Set the time stamps in the output buffer. */
GST_BUFFER_PTS (buffer) = self->local_clock;
GST_BUFFER_DTS (buffer) = self->local_clock;
GST_BUFFER_DURATION (buffer) = memory_out_info.size / self->bytes_per_ns;
/* Advance our clock. */
self->local_clock =
    self->local_clock + (memory_out_info.size / self->bytes_per_ns);
/* Keep track of the amount of time we have been sending sound. */
self->elapsed_time =
    self->elapsed_time + (memory_out_info.size / self->bytes_per_ns);
GST_DEBUG_OBJECT (self, "elapsed time is %" G_GUINT64_FORMAT ".",
                  self->elapsed_time);
/* Note the byte offsets in the source. */
GST_BUFFER_OFFSET (buffer) = self->local_buffer_drain_level;
GST_BUFFER_OFFSET_END (buffer) =
    self->local_buffer_drain_level + memory_out_info.size;

GST_DEBUG_OBJECT (self,
                  "sending %" G_GUINT64_FORMAT " bytes of data downstream"
                  " from buffer position %" G_GUINT64_FORMAT ".",
                  memory_out_info.size, self->local_buffer_drain_level);

/* Update the current position in our local buffer. */
self->local_buffer_drain_level =
    self->local_buffer_drain_level + memory_out_info.size;

/* we are finished with the new buffer and our local buffer */
gst_buffer_unmap (buffer, &memory_out_info);
gst_buffer_unmap (self->local_buffer, &memory_in_info);

/* We must unlock before we push, since pushing can cause a query to come
 * back upstream on another task before it completes. */
g_rec_mutex_unlock (&self->interlock);

flow_result = gst_pad_push (self->srcpad, buffer);
if (flow_result != GST_FLOW_OK)
{
    GST_DEBUG_OBJECT ("pad push of data returned with %s.",
                      gst_flow_get_name (flow_result));
}

```

```

    GST_DEBUG_OBJECT (self, "completed push of data");

    return;
}

/* Activate or deactivate the sink pad. */
static gboolean
gst_looper_sink_activate_mode (GstPad * pad, GstObject * parent,
                               GstPadMode mode, gboolean active)
{
    gboolean result;
    GstLooper *self = GST_LOOPER (parent);

    switch (mode)
    {
        case GST_PAD_MODE_PUSH:
            g_rec_mutex_lock (&self->interlock);
            if (active)
            {
                GST_INFO_OBJECT (self, "activating sink pad in push mode");
                self->sink_pad_mode = mode;
                self->sink_pad_active = TRUE;
            }
            else
            {
                GST_INFO_OBJECT (self, "deactivating sink pad in push mode");
                self->sink_pad_mode = mode;
                self->sink_pad_active = FALSE;
            }
            g_rec_mutex_unlock (&self->interlock);
            result = TRUE;
            break;

        case GST_PAD_MODE_PULL:
            g_rec_mutex_lock (&self->interlock);
            if (active)
            {
                GST_INFO_OBJECT (self, "activating sink pad in pull mode");
                self->sink_pad_mode = mode;

                /* Start the task that will pull data from upstream into the local
                 * buffer. */
                if (!self->sink_pad_task_running)
                {
                    result =

```

```

        gst_pad_start_task (self->sinkpad,
                            (GstTaskFunction)
                            gst_looper_pull_data_from_upstream,
                            self->sinkpad, NULL);

    if (!result)
    {
        GST_DEBUG_OBJECT (self,
                           "failed to start task "
                           "after sink pad activate");
    }
    self->sink_pad_task_running = TRUE;
}
result = TRUE;
self->sink_pad_active = TRUE;
}
else
{
    GST_INFO_OBJECT (self, "deactivating sink pad in pull mode");
    self->sink_pad_mode = mode;

    /* If it is still running, stop the task that is pulling data from
     * upstream into the local buffer. */
    if (self->sink_pad_task_running)
    {
        gst_pad_stop_task (pad);
        self->sink_pad_task_running = FALSE;
    }
    self->sink_pad_active = FALSE;
}
g_rec_mutex_unlock (&self->interlock);
result = TRUE;
break;

default:
    GST_DEBUG_OBJECT (pad, "unknown sink pad activation mode: %d.", mode);
    result = FALSE;
    break;
}

return result;
}

/* Pull sound data from upstream. Called repeatedly with @pad as the sink pad.
 */
static void
gst_looper_pull_data_from_upstream (GstPad * pad)

```

```

{
    GstLooper *self = GST_LOOPER (GST_PAD_PARENT (pad));
    GstMemory *memory_allocated;
    GstMapInfo memory_in_info, buffer_memory_info;
    guint64 byte_offset;
    char *byte_data_in_pointer, *byte_data_out_pointer;
    gboolean result, pull_result;
    guint64 max_position, start_position;
    gboolean max_duration_reached;
    GstBuffer *pull_buffer = NULL;

    /* This subroutine runs as its own task, so prevent destructive interference
     * with the local data by waiting until no other task is running any code
     * in this element. We must be careful to release this interlock when
     * we exit. */
    g_rec_mutex_lock (&self->interlock);

    /* If we should not be running, just exit. */
    if (!self->sink_pad_task_running)
    {
        GST_DEBUG_OBJECT (self, "data puller should not be running");
        g_rec_mutex_unlock (&self->interlock);
        return;
    }

    /* If the local buffer has been filled, and we have already seen some data,
     * we don't need to run any more. */
    if ((self->data_buffered) && (self->seen_incoming_data))
    {
        GST_DEBUG_OBJECT (self, "pausing sink pad task");
        result = gst_pad_pause_task (self->sinkpad);
        if (!result)
        {
            GST_DEBUG_OBJECT (self, "failed to pause sink pad task");
        }
        self->sink_pad_task_running = FALSE;
        g_rec_mutex_unlock (&self->interlock);
        return;
    }

    /* Ask our upstream peer to give us some data. */
    pull_result =
        gst_pad_pull_range (pad, self->pull_level, BUFFER_SIZE, &pull_buffer);
    if (pull_result == GST_FLOW_OK)
    {
        GST_DEBUG_OBJECT (self,

```

```

        "received buffer %p of size %" G_GSIZE_FORMAT ".",
        pull_buffer, gst_buffer_get_size (pull_buffer));

    /* If this is the first time we have seen any data from upstream, but
     * we already have all our data, which can only be true if we read
     * the data directly from the WAV file, start pushing data downstream. */
    if ((self->data_buffered) && (!self->seen_incoming_data))
    {
        /* Begin pushing data from our local buffer downstream using the
         * source pad. Unless we are autostarted, that task will send
         * silence until we get a Start message. */
        if (!self->src_pad_task_running)
        {
            result =
                gst_pad_start_task (self->srcpad,
                                    (GstTaskFunction)
                                    gst_looper_push_data_downstream,
                                    self->srcpad, NULL);
            self->src_pad_task_running = TRUE;
        }
    }
    self->seen_incoming_data = TRUE;

    /* If our local buffer has already been filled, we have no need for
     * this additional data. The next time around we will pause this task.
     */
    if (self->data_buffered)
    {
        gst_buffer_unref (pull_buffer);
        g_rec_mutex_unlock (&self->interlock);
        return;
    }
}

/* See if we have already reached max-duration. If so, we have no need
 * for this data buffer. */
max_duration_reached = FALSE;
max_position = 0;
if (self->max_duration > 0)
{
    max_position = round_up_to_position (self, self->max_duration);
    if (self->local_buffer_fill_level > max_position)
    {
        max_duration_reached = TRUE;
    }
}
}

```

```

if (max_duration_reached || (pull_result != GST_FLOW_OK))
{
    /* Either we got an error trying to pull sound data from upstream,
     * or we have reached max-duration. In either case we have finished
     * pulling sound data. */

    /* If we have a buffer from upstream, discard it. */
    if (pull_result == GST_FLOW_OK)
    {
        gst_buffer_unref (pull_buffer);
    }

    GST_INFO_OBJECT (self,
                     "stopped pulling sound data at offset %"
                     G_GUINT64_FORMAT ".", self->local_buffer_fill_level);
    self->data_buffered = TRUE;
    /* We now know the size of our local buffer. We may have filled it
     * a little beyond max-duration, but if so we will use only the data
     * up to max-duration. */
    if (self->max_duration > 0
        && max_position < self->local_buffer_fill_level)
    {
        self->local_buffer_size = max_position;
    }
    else
    {
        self->local_buffer_size = self->local_buffer_fill_level;
    }

    /* Set the position from which to start draining the buffer. */
    start_position = round_down_to_position (self, self->start_time);
    self->local_buffer_drain_level = start_position;

    /* If the Autostart parameter has been set to TRUE, don't wait
     * for a Start event. */
    if (self->autostart)
    {
        self->started = TRUE;
        self->local_clock = 0;
        self->elapsed_time = 0;
    }

    /* Begin pushing data from our local buffer downstream using the
     * source pad. Unless we are autostarted, that task will send silence
     * until we get a Start message. */
    if (!self->src_pad_task_running)

```

```

    {
        result =
            gst_pad_start_task (self->srcpad,
                               (GstTaskFunction)
                               gst_looper_push_data_downstream, self->srcpad,
                               NULL);
        self->src_pad_task_running = TRUE;
    }
}

/* We are done. The next time around we will detect that the buffer has
 * been filled and stop this task. */
g_rec_mutex_unlock (&self->interlock);
return;

/* We have a buffer from upstream and we have not already reached
 * max duration. Accept the buffer. */

/* Allocate more memory at the end of our local buffer, then copy
 * the data in the received buffer to it. */
GST_DEBUG_OBJECT (self, "map pulled buffer for reading");
result = gst_buffer_map (pull_buffer, &memory_in_info, GST_MAP_READ);
if (!result)
{
    GST_DEBUG_OBJECT (self, "unable to map pulled buffer for reading");
}
memory_allocated = gst_allocator_alloc (NULL, memory_in_info.size, NULL);
gst_buffer_append_memory (self->local_buffer, memory_allocated);
result =
    gst_buffer_map (self->local_buffer, &buffer_memory_info, GST_MAP_WRITE);
if (!result)
{
    GST_DEBUG_OBJECT (self, "unable to map local buffer for writing");
}
GST_DEBUG_OBJECT (self,
    "copy data from pulled buffer to local buffer"
    " at %p, memory at %p, offset %" G_GUINT64_FORMAT
    ", from %p, size %" G_GSIZE_FORMAT ".",
    self->local_buffer, buffer_memory_info.data,
    self->local_buffer_fill_level, memory_in_info.data,
    memory_in_info.size);
for (byte_offset = 0; byte_offset < memory_in_info.size;
     byte_offset = byte_offset + 1)
{
    byte_data_in_pointer = (void *) memory_in_info.data + byte_offset;
    byte_data_out_pointer =
        (void *) buffer_memory_info.data + self->local_buffer_fill_level +

```



```

        byte_offset;
        *byte_data_out_pointer = *byte_data_in_pointer;
    }
    gst_buffer_unmap (self->local_buffer, &buffer_memory_info);

    /* Update our offset into the local buffer. */
    self->local_buffer_fill_level =
        self->local_buffer_fill_level + memory_in_info.size;

    /* We are done with the pulled buffer. */
    gst_buffer_unmap (pull_buffer, &memory_in_info);
    gst_buffer_unref (pull_buffer);

    g_rec_mutex_unlock (&self->interlock);
    return;
}

/* Accept data from upstream if the source pad is in push mode. We must do this
 * if upstream won't let us pull. */
static GstFlowReturn
gst_looper_chain (GstPad * pad, GstObject * parent, GstBuffer * buffer)
{
    GstLooper *self = GST_LOOPER (parent);
    GstMemory *memory_allocated;
    GstMapInfo memory_in_info, buffer_memory_info;
    guint64 byte_offset;
    char *byte_data_in_pointer, *byte_data_out_pointer;
    gboolean result;
    guint64 max_position, start_position;
    gboolean max_duration_reached;

    g_rec_mutex_lock (&self->interlock);
    GST_DEBUG_OBJECT (self,
        "received buffer %p of size %" G_GSIZE_FORMAT ", time %"
        GST_TIME_FORMAT ", duration %" GST_TIME_FORMAT ".",
        buffer, gst_buffer_get_size (buffer),
        GST_TIME_ARGS (GST_BUFFER_TIMESTAMP (buffer)),
        GST_TIME_ARGS (GST_BUFFER_DURATION (buffer)));

    /* If we have already filled our local buffer, either because we have
     * received max-duration data or we loaded the data directly from the file,
     * and we have already seen some data, discard any more. */
    if ((self->data_buffered) && (self->seen_incoming_data))
    {
        /* Discard the buffer from upstream. */
        gst_buffer_unref (buffer);
    }
}

```

```

    GST_DEBUG_OBJECT (self, "buffer discarded.");

    g_rec_mutex_unlock (&self->interlock);
    return GST_FLOW_OK;
}

/* If we have already filled our local buffer, but we are seeing data from
 * upstream for the first time, which can only happen if we filled the
 * buffer by reading sound data directly from the WAV file, start sending
 * sound data from our local buffer downstream. */
if ((self->data_buffered) && (!self->seen_incoming_data))
{
    self->seen_incoming_data = TRUE;
    /* Begin pushing data from our local buffer downstream using the
     * source pad. Unless we are autostarted, this task will send
     * silence until we get a Start message. */
    result =
        gst_pad_start_task (self->srcpad,
                            (GstTaskFunction) gst_looper_push_data_downstream,
                            self->srcpad, NULL);
    self->src_pad_task_running = TRUE;

    /* Discard the buffer from upstream. */
    gst_buffer_unref (buffer);

    g_rec_mutex_unlock (&self->interlock);
    return GST_FLOW_OK;
}

/* Otherwise, if we have reached max-duration, we have now filled the buffer.
 */
max_duration_reached = FALSE;
if (self->max_duration > 0)
{
    max_position = round_up_to_position (self, self->max_duration);
    if (self->local_buffer_fill_level > max_position)
    {
        max_duration_reached = TRUE;
    }
}

if (max_duration_reached)
{
    /* This is the first time we have received a buffer beyond
     * max-duration, so start sending our buffered data

```

```

    * downstream. */
    GST_INFO_OBJECT (self,
        "reached max-duration at offset %" G_GUINT64_FORMAT
        ". ", self->local_buffer_fill_level);

    self->data_buffered = TRUE;
    /* We now know the size of our local buffer. We have filled it
     * a little beyond max-duration, but we will use only max-duration.
     */
    self->local_buffer_size = max_position;
    /* Set the position from which to start draining the buffer. */
    start_position = round_down_to_position (self, self->start_time);
    self->local_buffer_drain_level = start_position;

    /* If the Autostart parameter has been set to TRUE, don't wait
     * for a Start event. */
    if (self->autostart)
    {
        self->started = TRUE;
        self->local_clock = 0;
        self->elapsed_time = 0;
    }
    /* Begin pushing data from our local buffer downstream using the
     * source pad. Unless we are autostarted, this task will send
     * silence until we get a Start message. */
    result =
        gst_pad_start_task (self->srcpad,
            (GstTaskFunction) gst_looper_push_data_downstream,
            self->srcpad, NULL);
    self->src_pad_task_running = TRUE;

    /* Discard the buffer from upstream. */
    gst_buffer_unref (buffer);

    g_rec_mutex_unlock (&self->interlock);
    return GST_FLOW_OK;
}

/* Our local buffer has not been filled, and we have not reached max-duration.
 * Allocate more memory at the end of our local buffer, then copy the data in
 * the received buffer to it. */
GST_DEBUG_OBJECT (self, "map received buffer for reading");
result = gst_buffer_map (buffer, &memory_in_info, GST_MAP_READ);
if (!result)
{
    GST_DEBUG_OBJECT (self, "unable to map received buffer for reading");
}

```

```

    }
    memory_allocated = gst_allocator_alloc (NULL, memory_in_info.size, NULL);
    gst_buffer_append_memory (self->local_buffer, memory_allocated);
    result =
        gst_buffer_map (self->local_buffer, &buffer_memory_info, GST_MAP_WRITE);
    if (!result)
    {
        GST_DEBUG_OBJECT (self, "unable to map local buffer for writing");
    }
    GST_DEBUG_OBJECT (self,
        "copy data from received buffer to local buffer"
        " at %p, memory at %p, offset %" G_GUINT64_FORMAT
        ", from %p, size %" G_GSIZE_FORMAT ".",
        self->local_buffer, buffer_memory_info.data,
        self->local_buffer_fill_level, memory_in_info.data,
        memory_in_info.size);
    for (byte_offset = 0; byte_offset < memory_in_info.size;
        byte_offset = byte_offset + 1)
    {
        byte_data_in_pointer = (void *) memory_in_info.data + byte_offset;
        byte_data_out_pointer =
            (void *) buffer_memory_info.data + self->local_buffer_fill_level +
            byte_offset;
        *byte_data_out_pointer = *byte_data_in_pointer;
    }
    gst_buffer_unmap (self->local_buffer, &buffer_memory_info);

    /* Update our offset into the local buffer. */
    self->local_buffer_fill_level =
        self->local_buffer_fill_level + memory_in_info.size;

    /* We are done with the received buffer. */
    gst_buffer_unmap (buffer, &memory_in_info);
    gst_buffer_unref (buffer);

    g_rec_mutex_unlock (&self->interlock);
    return GST_FLOW_OK;
}

/* Send data downstream in pull mode. We don't allow pull mode from downstream,
 * so this subroutine will never be called. It is left here in case we decide
 * to implement pull mode on the source pad in the future. */
static GstFlowReturn
gst_looper_get_range (GstPad * pad, GstObject * parent, guint64 offset,
    guint length, GstBuffer ** buffer)
{

```

```

GstLooper *self = GST_LOOPER (parent);
GstFlowReturn result = GST_FLOW_OK;
GstBuffer *buf;
GstMapInfo memory_info;
guint64 buf_size;
gint i;

g_rec_mutex_lock (&self->interlock);
GST_DEBUG_OBJECT (self,
    "Getting range: offset %" G_GUINT64_FORMAT ", length %u",
    offset, length);

if (length == -1)
    /* If the buffer length is defaulted, use one millisecond. */
    {
        buf_size = self->width * self->channel_count * self->data_rate / 1000;
        /* Watch out for the data format not being set up yet. */
        if (buf_size == 0)
            {
                buf_size = 4096;
            }
    }
else
    {
        buf_size = length;
    }

/* If no buffer is passed to get_range, allocate one. */
if (*buffer == NULL)
    {
        buf = gst_buffer_new_allocate (NULL, buf_size, NULL);
    }
else
    {
        buf = *buffer;
    }

gst_buffer_map (buf, &memory_info, GST_MAP_WRITE);

/* FIXME: fill the buffer with data from our local buffer. */
for (i = 0; i < memory_info.size; i++)
    {
        memory_info.data[i] = 0;
    }
result = GST_FLOW_OK;

```

```

    gst_buffer_unmap (buf, &memory_info);
    gst_buffer_resize (buf, 0, buf_size);
    GST_BUFFER_OFFSET (buf) = 0;
    GST_BUFFER_OFFSET_END (buf) = buf_size;
    *buffer = buf;
    g_rec_mutex_unlock (&self->interlock);
    return result;
}

/* Handle an event arriving at the sink pad. */
static gboolean
gst_looper_handle_sink_event (GstPad * pad, GstObject * parent,
                              GstEvent * event)
{
    gboolean result = TRUE;
    GstLooper *self = GST_LOOPER (parent);
    GstCaps *in_caps, *out_caps;
    GstStructure *caps_structure;
    gchar *format_code_pointer;
    gchar format_code_0, format_code_1;
    gdouble bits_per_second, bits_per_nanosecond;
    guint64 start_position;
    gint data_rate, channel_count;
    guint64 max_position;
    gboolean wav_file_read;

    GST_DEBUG_OBJECT (self, "received an event on the sink pad");

    switch (GST_EVENT_TYPE (event))
    {
        {
            case GST_EVENT_FLUSH_START:
                g_rec_mutex_lock (&self->interlock);
                GST_LOG_OBJECT (self, "received flush start event on sink pad");
                if (GST_PAD_MODE (self->srcpad) == GST_PAD_MODE_PUSH)
                {
                    /* Forward the event downstream. */
                    result = gst_pad_push_event (self->srcpad, event);
                    /* Stop the task that is sending data downstream. */
                    gst_pad_stop_task (self->srcpad);
                    self->srcpad_task_running = FALSE;
                    GST_LOG_OBJECT (self, "loop stopped");
                }
            else
            {
                {
                    gst_event_unref (event);
                }
            }
        }
    }

```

```

    self->sink_pad_flushing = TRUE;
    g_rec_mutex_unlock (&self->interlock);
    break;

case GST_EVENT_FLUSH_STOP:
    g_rec_mutex_lock (&self->interlock);
    GST_LOG_OBJECT (self, "received flush stop event on sink pad");
    if (GST_PAD_MODE (self->srcpad) == GST_PAD_MODE_PUSH)
    {
        /* Forward the event downstream. */
        result = gst_pad_push_event (self->srcpad, event);
        if (!result)
        {
            GST_DEBUG_OBJECT (self, "failed to push flush stop event");
        }
        if ((self->data_buffered) && (!self->src_pad_task_running))
        {
            /* Start the task which pushes data downstream. */
            result =
                gst_pad_start_task (self->srcpad,
                                    (GstTaskFunction)
                                    gst_looper_push_data_downstream,
                                    self->srcpad, NULL);

            if (!result)
            {
                GST_DEBUG_OBJECT (self,
                                    "failed to start task after flush stop");
            }
            self->src_pad_task_running = TRUE;
        }
    }
    else
    {
        gst_event_unref (event);
    }
    self->sink_pad_flushing = FALSE;
    g_rec_mutex_unlock (&self->interlock);
    break;

case GST_EVENT_CAPS:
    /* A caps event on the sink port specifies the format, data rate and
     * number of channels of audio that will come from upstream. Since we
     * are just passing data through, specify that our source port will
     * use the same format, data rate and number of channels. */
    g_rec_mutex_lock (&self->interlock);
    gst_event_parse_caps (event, &in_caps);

```

```

GST_DEBUG_OBJECT (self, "input caps are %" GST_PTR_FORMAT ".", in_caps);
caps_structure = gst_caps_get_structure (in_caps, 0);
/* Fill in local information about the format, and values based on it.
 */
result = gst_structure_get_int (caps_structure, "rate", &data_rate);
if (!result)
{
    GST_DEBUG_OBJECT (self, "no rate in caps");
    data_rate = 48000;
}
self->data_rate = data_rate;

result =
    gst_structure_get_int (caps_structure, "channels", &channel_count);
if (!result)
{
    GST_DEBUG_OBJECT (self, "no channel count in caps");
    channel_count = 2;
}
self->channel_count = channel_count;

g_free (self->format);
self->format =
    g_strdup (gst_structure_get_string (caps_structure, "format"));
if (self->format == NULL)
{
    GST_DEBUG_OBJECT (self, "no format in caps");
    self->format = g_strdup (GST_AUDIO_NE (F64));
}

out_caps =
    gst_caps_new_simple ("audio/x-raw", "format", G_TYPE_STRING,
                        self->format, "rate", G_TYPE_INT,
                        self->data_rate, "channels", G_TYPE_INT,
                        self->channel_count, NULL);
result = gst_pad_set_caps (self->srcpad, out_caps);
GST_DEBUG_OBJECT (self, "output caps are %" GST_PTR_FORMAT ".",
                  out_caps);
gst_caps_unref (out_caps);

/* Compute the size of a frame from the format string.
 * The possible formats start with a letter, then the width of
 * sample in bits, for example, F32LE. The second character of
 * the format can be used to determine the width. */
format_code_pointer = self->format;
format_code_0 = format_code_pointer[0];

```



```

format_code_1 = format_code_pointer[1];
switch (format_code_1)
{
    case '8':
        self->width = 8;
        break;
    case '1':
        self->width = 16;
        break;
    case '2':
        self->width = 24;
        break;
    case '3':
        self->width = 32;
        break;
    case '6':
        self->width = 64;
        break;
    default:
        self->width = 32;
        break;
}
GST_LOG_OBJECT (self, "second character of format is %c.",
                format_code_1);
GST_DEBUG_OBJECT (self, "each sample has %" G_GUINT64_FORMAT " bits.",
                  self->width);

/* Compute the silence value. For signed and floating formats, it is 0.
 * for unsigned it is 128 for U8, the only unsigned format we support. */
switch (format_code_0)
{
    case 'S':
    case 'F':
        self->silence_byte = 0;
        break;
    case 'U':
        self->silence_byte = 128;
        break;
    default:
        self->silence_byte = 0;
        break;
}
GST_DEBUG_OBJECT (self, "silence value is %hhd.", self->silence_byte);

/* Compute the data rate in bytes per nanosecond.
 * data_rate times width times channel_count is bits per second.

```

```

    * that divided by 1E9 is bits per nanosecond.
    * that divided by 8 is bytes per nanosecond. */
bits_per_second = self->data_rate * self->width * self->channel_count;
bits_per_nanosecond = (gdouble) bits_per_second / (gdouble) 1E9;
self->bytes_per_ns = bits_per_nanosecond / 8.0;
GST_DEBUG_OBJECT (self, "data rate is %f bytes per nanosecond.",
                  self->bytes_per_ns);

/* If a WAV file was specified, this is a good time to read it. We have
* the format and data rate, so we can convert max duration
* to the maximum size of the local buffer. */
if (self->file_location_specified)
{
    max_position = 0;
    if (self->max_duration > 0)
    {
        max_position = round_up_to_position (self, self->max_duration);
    }

    /* Read the data from the WAV file, up to the most we will need. */
    wav_file_read = read_wav_file_data (self, max_position);
    if (wav_file_read)
    {
        /* We now have all our data. */
        self->data_buffered = TRUE;
        GST_DEBUG_OBJECT (self, "read %ld bytes from WAV file.",
                          self->local_buffer_fill_level);

        /* We now know the size of our local buffer. We may have filled
it beyond max-duration, but if so we will use only the data
* up to max-duration. */
        if (self->max_duration > 0
            && max_position < self->local_buffer_fill_level)
        {
            self->local_buffer_size = max_position;
        }
        else
        {
            self->local_buffer_size = self->local_buffer_fill_level;
        }

        /* Set the position from which to start draining the buffer. */
        start_position =
            round_down_to_position (self, self->start_time);
        self->local_buffer_drain_level = start_position;

        /* If the Autostart parameter has been set to TRUE, don't wait

```

```

        /* for a Start event. */
        if (self->autostart)
        {
            self->started = TRUE;
            self->local_clock = 0;
            self->elapsed_time = 0;
        }
        /* It is too early to start pushing data downstream. Wait until
        * we get some data from upstream. */
    }
    else
    {
        GST_DEBUG_OBJECT (self, "read from WAV file failed.");
    }
}

g_rec_mutex_unlock (&self->interlock);
result = gst_pad_push_event (self->srcpad, event);
break;

case GST_EVENT_EOS:
    /* We have hit the end of the incoming data stream. */
    g_rec_mutex_lock (&self->interlock);
    GST_INFO_OBJECT (self,
                     "reached end-of-stream at offset %" G_GUINT64_FORMAT
                     ". ", self->local_buffer_fill_level);

    /* If we have already filled the buffer due to reaching max-duration,
    * we don't need to do anything here. */
    if (!self->data_buffered)
    {
        self->data_buffered = TRUE;
        /* We now know the size of our local buffer. */
        self->local_buffer_size = self->local_buffer_fill_level;
        /* Set the initial buffer drain position. */
        start_position = round_down_to_position (self, self->start_time);
        self->local_buffer_drain_level = start_position;

        /* If the Autostart parameter has been set to TRUE, don't wait
        * for a Start event. */
        if (self->autostart)
        {
            self->started = TRUE;
            self->local_clock = 0;
            self->elapsed_time = 0;
        }
    }
}

```

```

        /* Begin pushing data from our local buffer downstream using the
         * source pad. Unless we are autostarted, this task will send
         * silence until we get a Start message. */
        result =
            gst_pad_start_task (self->srcpad,
                               (GstTaskFunction)
                               gst_looper_push_data_downstream, self->srcpad,
                               NULL);
        self->src_pad_task_running = TRUE;
    }

    /* If the sink pad is in pull mode, it will have a task doing pulls.
     * That task will shut itself down when it notices that the local buffer
     * has been filled. */

    /* Don't send the EOS event downstream until we are shut down. */
    gst_event_unref (event);
    g_rec_mutex_unlock (&self->interlock);
    break;

default:
    result = gst_pad_push_event (self->srcpad, event);
    break;
}

return result;
}

/* Handle an event from the source pad. */
static gboolean
gst_looper_handle_src_event (GstPad * pad, GstObject * parent,
                             GstEvent * event)
{
    gboolean result = TRUE;
    GstLooper *self = GST_LOOPER (parent);
    const GstStructure *event_structure;
    const gchar *structure_name;
    guint64 start_position;

    GST_DEBUG_OBJECT (self, "received an event on the source pad.");
    g_rec_mutex_lock (&self->interlock);
    switch (GST_EVENT_TYPE (event))
    {
        case GST_EVENT_FLUSH_START:
            /* All the downstream data comes from here, so we block the event
             * from propagating upstream. */

```

```

gst_event_unref (event);
/* if we are already sending our buffer downstream, stop. */
if (self->src_pad_task_running)
{
    gst_pad_stop_task (self->srcpad);
    self->src_pad_task_running = FALSE;
}
result = TRUE;
self->src_pad_flushing = TRUE;
break;

case GST_EVENT_FLUSH_STOP:
/* All the downstream data comes from here, so we do not propagate
 * the event upstream. */
gst_event_unref (event);

/* If the incoming buffer has been filled, start the task
 * which pushes data downstream. */
if ((self->data_buffered) && (!self->src_pad_task_running))
{
    result =
        gst_pad_start_task (self->srcpad,
                            (GstTaskFunction)
                            gst_looper_push_data_downstream, self->srcpad,
                            NULL);
    self->src_pad_task_running = TRUE;
}
if (!result)
{
    GST_DEBUG_OBJECT (self, "unable to start task on flush stop");
}
result = TRUE;
self->src_pad_flushing = FALSE;
break;

case GST_EVENT_RECONFIGURE:
/* FIXME: may have to start the loop task here. */
result = gst_pad_push_event (self->sinkpad, event);
break;

case GST_EVENT_CUSTOM_UPSTREAM:
g_rec_mutex_lock (&self->interlock);

/* We use five custom upstream events: start, pause, continue, release
 * and shutdown.
 * The release event is processed mostly in the envelope plugin,

```

```

* but we also use it here to terminate looping.
*
* The start event causes the buffered data to be transmitted
* from its beginning.
*
* The shutdown event is issued prior to closing down the
* pipeline. The looper sends EOS and stops sending data.
*
* The pause event silences the looper, and the continue event
* lets it proceed from where it paused. These are distinct from
* the paused state of the pipeline because we want the pipeline
* to keep running even if the looper is paused. */
event_structure = gst_event_get_structure (event);
structure_name = gst_structure_get_name (event_structure);

if (g_strcmp0 (structure_name, (gchar *) "start") == 0)
{
    /* This is a start event, which might be caused by receipt
    * of a Note On MIDI message, or by an operator pushing a
    * start button. Start pushing our local buffer downstream.
    */
    GST_INFO_OBJECT (self, "received custom start event");
    self->started = TRUE;
    self->completion_sent = FALSE;
    start_position = round_down_to_position (self, self->start_time);
    self->local_buffer_drain_level = start_position;
    self->elapsed_time = 0;
}

if (g_strcmp0 (structure_name, (gchar *) "pause") == 0)
{
    /* The pause event can be caused by receipt of a command
    * to temporarily suspend sound output. */
    GST_INFO_OBJECT (self, "received custom pause event");
    self->paused = TRUE;
    self->continued = FALSE;
}

if (g_strcmp0 (structure_name, (gchar *) "continue") == 0)
{
    /* When the need for the pause has passed, another command
    * will resume the sound. */
    GST_INFO_OBJECT (self, "received custom continue event");
    self->continued = TRUE;
}

```

```

    if (g_strcmp0 (structure_name, (gchar *) "release") == 0)
    {
        /* The release event might be caused by receipt of a Note Off
         * MIDI message, or by an operator pushing a stop button.
         * Terminate any looping. */
        GST_INFO_OBJECT (self, "received custom release event");
        self->released = TRUE;
    }

    if (g_strcmp0 (structure_name, (gchar *) "shutdown") == 0)
    {
        /* The shutdown event is caused by the operator shutting down
         * the application. We send an EOS and stop. */
        self->send_EOS = TRUE;
        GST_INFO_OBJECT (self, "shutting down");
    }

    g_rec_mutex_unlock (&self->interlock);

    /* Push the event upstream. */
    result = gst_pad_push_event (self->sinkpad, event);
    break;

default:
    result = gst_pad_push_event (self->sinkpad, event);
    break;
}

g_rec_mutex_unlock (&self->interlock);
return result;
}

/* Handle a query on the element. */
static gboolean
gst_looper_handle_query (GstElement * element, GstQuery * query)
{
    GstLooper *self = GST_LOOPER (element);

    /* Simply forward to the source pad query function. */
    return gst_looper_handle_src_query (self->srcpad, GST_OBJECT_CAST (element),
                                         query);
}

/* Handle a query on the source pad or element. */
static gboolean
gst_looper_handle_src_query (GstPad * pad, GstObject * parent,

```

```

                                GstQuery * query)
{
    GstLooper *self = GST_LOOPER (parent);
    GstFormat format;
    gint64 segment_start, segment_end;
    gboolean seekable, peer_success;
    gint64 peer_pos;
    GstSchedulingFlags scheduling_flags = 0;
    gboolean result;

    GST_DEBUG_OBJECT (self, "query on source pad or element");
    g_rec_mutex_lock (&self->interlock);

    switch (GST_QUERY_TYPE (query))
    {
        case GST_QUERY_POSITION:
            gst_query_parse_position (query, &format, &peer_pos);
            GST_DEBUG_OBJECT (self, "query position on source pad");

            peer_success = gst_pad_peer_query (self->sinkpad, query);
            switch (format)
            {
                /* FIXME: code this. */
                case GST_FORMAT_BYTES:
                    break;
                case GST_FORMAT_TIME:
                    break;
                default:
                    GST_DEBUG_OBJECT (self, "dropping query in %s format.",
                                      gst_format_get_name (format));
                    g_rec_mutex_unlock (&self->interlock);
                    return FALSE;
            }
            result = TRUE;
            break;

        case GST_QUERY_DURATION:
            GST_DEBUG_OBJECT (self, "query duration on source pad");
            /* FIXME: code this. */
            result = TRUE;
            break;

        /* FIXME: query buffering? */

        case GST_QUERY_SCHEDULING:
            GST_DEBUG_OBJECT (self, "query scheduling on source pad");

```



```

    peer_success = gst_pad_peer_query (self->sinkpad, query);
    gst_query_parse_scheduling (query, &scheduling_flags, NULL, NULL, NULL);
    /* We only do push mode on our source pad. */
    result = FALSE;
    break;

case GST_QUERY_SEEKING:
    GST_DEBUG_OBJECT (self, "query seeking on source pad");
    peer_success = gst_pad_peer_query (self->sinkpad, query);
    gst_query_parse_seeking (query, &format, &seekable, &segment_start,
                             &segment_end);

    /* FIXME: since we buffer everything here, maybe do not propagate
     * the query but just answer it here. */
    result = TRUE;
    break;

case GST_QUERY_CAPS:
    /* The next element downstream wants to know what formats this pad
     * supports, and in what order of preference. Just pass the query
     * upstream since we don't care. */
    GST_DEBUG_OBJECT (self, "query caps on source pad");
    peer_success = gst_pad_query_default (pad, parent, query);
    GST_DEBUG_OBJECT (self, "completed query caps on source pad");
    result = peer_success;
    break;

default:
    GST_DEBUG_OBJECT (self, "taking default action for query.");
    peer_success = gst_pad_query_default (pad, parent, query);
    if (!peer_success)
    {
        GST_DEBUG_OBJECT (self, "failed default query action");
    }
    result = peer_success;
}

GST_DEBUG_OBJECT (self, "completed source or element query processing.");
g_rec_mutex_unlock (&self->interlock);
return result;
}

/* Handle a query to the sink pad. */
static gboolean
gst_looper_handle_sink_query (GstPad * pad, GstObject * parent,
                              GstQuery * query)

```

```

{
    GstLooper *self = GST_LOOPER (parent);
    gboolean result;

    GST_DEBUG_OBJECT (self, "received query on sink pad");
    g_rec_mutex_lock (&self->interlock);

    switch (GST_QUERY_TYPE (query))
    {
        case GST_QUERY_CAPS:
            /* The next element upstream is asking what formats this pad
             * supports, and in what order of preference. Since we have
             * no preference ourselves, just pass the query downstream. */
            GST_DEBUG_OBJECT (self, "query caps on sink pad");
            result = gst_pad_query_default (pad, parent, query);
            GST_DEBUG_OBJECT (self, "completed caps query on sink pad");
            break;

        default:
            result = gst_pad_query_default (pad, parent, query);
            break;
    }

    GST_DEBUG_OBJECT (self, "completed query on sink pad.");
    g_rec_mutex_unlock (&self->interlock);
    return result;
}

/* Round a time to the next higher buffer position from which we can start
 * a frame. */
static guint64
round_up_to_position (GstLooper * self, guint64 specified_time)
{
    guint64 position;           /* unrounded buffer position corresponding to
                                * the specified time. */
    guint frame_size;           /* The number of bytes in one frame. */
    guint64 frame_index;        /* The number of complete frames before the
                                * position. */
    guint64 byte_position;       /* the position in the buffer corresponding to
                                * the beginning of the first frame after the
                                * specified time. */

    /* The time is specified in nanoseconds. If the buffer position
     * corresponding to that time isn't on a frame boundary, convert it to the
     * next higher buffer position that is on a frame boundary. */
    position = (gdouble) specified_time *self->bytes_per_ns;

```

```

frame_size = self->width * self->channel_count / 8;
frame_index = (position / frame_size);
byte_position = frame_index * frame_size;
if (byte_position < position)
{
    byte_position = (frame_index + 1) * frame_size;
}
GST_DEBUG_OBJECT (self,
    "time %" GST_TIME_FORMAT " rounded up to %"
    GST_TIME_FORMAT " yielding buffer position %"
    G_GUINT64_FORMAT ".", GST_TIME_ARGS (specified_time),
    GST_TIME_ARGS (byte_position / self->bytes_per_ns),
    byte_position);
return byte_position;
}

/* Round a time to the next lower buffer position from which we can start
 * a frame. */
static guint64
round_down_to_position (GstLooper * self, guint64 specified_time)
{
    guint64 position;           /* unrounded buffer position corresponding to
                                * the specified time. */
    guint frame_size;          /* The number of bytes in one frame. */
    guint64 frame_index;       /* The number of complete frames before the
                                * position. */
    guint64 byte_position;      /* the position in the buffer corresponding to
                                * the beginning of the first frame before the
                                * specified time. */

    /* The time is specified in nanoseconds. If the buffer position
     * corresponding to that time isn't on a frame boundary, convert it to the
     * next lower buffer position that is on a frame boundary. */
    position = (gdouble) specified_time * self->bytes_per_ns;
    frame_size = self->width * self->channel_count / 8;
    frame_index = (position / frame_size);
    byte_position = frame_index * frame_size;
    GST_DEBUG_OBJECT (self,
        "time %" GST_TIME_FORMAT " rounded down to %"
        GST_TIME_FORMAT " for buffer position %" G_GUINT64_FORMAT
        ".", GST_TIME_ARGS (specified_time),
        GST_TIME_ARGS (byte_position / self->bytes_per_ns),
        byte_position);
    return byte_position;
}

```

```

/* Subroutine to read the data chunks from a WAV file into the local buffer.
 * This is a faster way to load the buffer than waiting for the data to
 * be provided in real time by upstream. We read only the data; parsing of
 * the metadata is done by upstream. The return value is TRUE if data was
 * read successfully, FALSE if not. */
static gboolean
read_wav_file_data (GstLooper * self, guint64 max_position)
{
    FILE *file_stream;
    int stream_status;
    size_t amount_read;
    GstMemory *memory_allocated;
    GstMapInfo buffer_memory_info;
    int result, seek_success;
    gboolean return_value = FALSE;
    guint32 header[2];
    guint32 chunk_size;
    unsigned int byte_offset;
    guint64 local_buffer_fill_level;
    char data_byte;
    char *byte_data_out_pointer;

    /* This subroutine exits through some common cleanup code at common_exit.
     * The following flags control the extent of its cleanup. */
    gboolean file_open = FALSE;
    gboolean buffer_mapped = FALSE;

    GST_DEBUG_OBJECT (self, "reading from wave file \"%s\".",
                      self->file_location);

    errno = 0;

    file_stream = fopen (self->file_location, "rb");
    if (file_stream == NULL)
    {
        GST_DEBUG_OBJECT (self, "failed to open file \"%s\": %s.",
                          self->file_location, strerror (errno));
        goto common_exit;
    }
    file_open = TRUE;

    /* Read the first eight bytes of the file, which is the RIFF header. */
    amount_read = fread (&header, 1, 8, file_stream);
    if (amount_read != 8)
    {
        GST_DEBUG_OBJECT (self, "failed to read first 8 bytes: got %lu.",
                          amount_read);
    }

```

```

    goto common_exit;
}
if (memcmp (&header[0], "RIFF", 4) != 0)
{
    GST_DEBUG_OBJECT (self, "file \"%s\" is not a RIFF file.",
                      self->file_location);
    goto common_exit;
}

/* We don't care about the second word of the RIFF header, which is supposed
 * to be the size of the file but is inaccurate if the file is too large
 * for a 32-bit integer or was written in real time by a recording application
 * that did not know how long the data would be. */

/* Read and verify bytes 9 through 12 of the file. */
amount_read = fread (&header, 1, 4, file_stream);
if (amount_read != 4)
{
    GST_DEBUG_OBJECT (self, "failed to read bytes 9 through 12: got %lu.",
                      amount_read);
    goto common_exit;
}
if (memcmp (&header[0], "WAVE", 4) != 0)
{
    GST_DEBUG_OBJECT (self, "file \"%s\" is not a WAVE file.",
                      self->file_location);
    goto common_exit;
}

/* Skip all but data chunks. Copy the data from the data chunks into
 * our local buffer. Since we are ignoring the size field of the RIFF
 * chunk, continue until end of file. */
local_buffer_fill_level = 0;
while TRUE
{
    /* If we have enough data to reach max duration, we don't need any more.
     * It doesn't hurt to read a little more data than is required, so
     * we need only check at chunk boundaries. */
    if ((max_position != 0) && (local_buffer_fill_level > max_position))
    {
        GST_DEBUG_OBJECT (self,
                          "reached max duration at %" G_GUINT64_FORMAT ".",
                          max_position);
        break;
    }
}

```

```

/* Read the first 8 bytes of the chunk to learn its type and size. */
amount_read = fread (&header, 1, 8, file_stream);
if (amount_read != 8)
{
    GST_DEBUG_OBJECT (self, "unable to read another eight bytes.");
    break;
}

chunk_size = header[1];
if (memcmp (&header[0], "data", 4) != 0)
{
    /* Skip over this non-data chunk. Odd chunk sizes are padded with a
     * single byte so that chunks always start on 2-byte boundaries. */
    if ((chunk_size & 1) == 1)
        chunk_size = chunk_size + 1;
    GST_DEBUG_OBJECT (self, "skipping forward by %u bytes.",
                      chunk_size);
    seek_success = fseek (file_stream, chunk_size, SEEK_CUR);
    if (seek_success != 0)
    {
        GST_DEBUG_OBJECT (self, "seek failed on file \"%s\": %d.",
                          self->file_location, seek_success);
        goto common_exit;
    }
    continue;
}

/* Copy the data chunk into our local buffer. */
GST_DEBUG_OBJECT (self, "reading %d bytes of data from file \"%s\".",
                  chunk_size, self->file_location);
memory_allocated = gst_allocator_alloc (NULL, chunk_size, NULL);
gst_buffer_append_memory (self->local_buffer, memory_allocated);
result =
    gst_buffer_map (self->local_buffer, &buffer_memory_info,
                    GST_MAP_WRITE);
if (!result)
{
    GST_DEBUG_OBJECT (self, "unable to map local buffer for writing");
    goto common_exit;
}
buffer_mapped = TRUE;
for (byte_offset = 0; byte_offset < chunk_size;
     byte_offset = byte_offset + 1)
{
    byte_data_out_pointer =
        (void *) buffer_memory_info.data + local_buffer_fill_level +

```

```

        byte_offset;
        amount_read = fread (&data_byte, 1, 1, file_stream);
        if (amount_read != 1)
        {
            GST_DEBUG_OBJECT (self,
                              "failed to read a data byte from \"%s\".",
                              self->file_location);

            goto common_exit;
        }
        *byte_data_out_pointer = data_byte;
    }
    gst_buffer_unmap (self->local_buffer, &buffer_memory_info);
    buffer_mapped = FALSE;

    local_buffer_fill_level = local_buffer_fill_level + chunk_size;

    /* If the chunk size is odd, skip the pad byte. */
    if ((chunk_size & 1) == 1)
    {
        amount_read = fread (&data_byte, 1, 1, file_stream);
        if (amount_read != 1)
        {
            GST_DEBUG_OBJECT (self,
                              "failed to read a pad byte from \"%s\".",
                              self->file_location);

            goto common_exit;
        }
    }
}

/* We failed to read the header of the next chunk, or we have reached
 * max_duration. Stop reading the file. */
self->local_buffer_fill_level = local_buffer_fill_level;
GST_DEBUG_OBJECT (self, "Loaded %" G_GUINT64_FORMAT " bytes from file %s.",
                  local_buffer_fill_level, self->file_location);
return_value = TRUE;

common_exit:
if (buffer_mapped)
{
    gst_buffer_unmap (self->local_buffer, &buffer_memory_info);
    buffer_mapped = FALSE;
}

if (file_open)

```

```

    {
        stream_status = fclose (file_stream);
        if (stream_status == EOF)
        {
            GST_DEBUG_OBJECT (self, "failed to close file \"%s\".",
                              self->file_location);
            return_value = FALSE;
        }
        file_open = FALSE;
    }

    return return_value;
}

/* Set the value of a property. */
static void
gst_looper_set_property (GObject * object, guint prop_id,
                        const GValue * value, GParamSpec * pspec)
{
    GstLooper *self = GST_LOOPER (object);

    g_rec_mutex_lock (&self->interlock);

    switch (prop_id)
    {
        {
            case PROP_SILENT:
                GST_OBJECT_LOCK (self);
                self->silent = g_value_get_boolean (value);
                GST_OBJECT_UNLOCK (self);
                break;

            case PROP_LOOP_TO:
                GST_OBJECT_LOCK (self);
                self->loop_to = g_value_get_uint64 (value);
                GST_INFO_OBJECT (self, "loop-to: %" G_GUINT64_FORMAT ".",
                                self->loop_to);
                GST_OBJECT_UNLOCK (self);
                break;

            case PROP_LOOP_FROM:
                GST_OBJECT_LOCK (self);
                self->loop_from = g_value_get_uint64 (value);
                GST_INFO_OBJECT (self, "loop-from: %" G_GUINT64_FORMAT ".",
                                self->loop_from);
                GST_OBJECT_UNLOCK (self);
                break;
        }
    }
}

```



```

case PROP_LOOP_LIMIT:
    GST_OBJECT_LOCK (self);
    self->loop_limit = g_value_get_uint (value);
    GST_INFO_OBJECT (self, "loop-limit: %" G_GUINT32_FORMAT ".",
                     self->loop_limit);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_MAX_DURATION:
    GST_OBJECT_LOCK (self);
    self->max_duration = g_value_get_uint64 (value);
    GST_INFO_OBJECT (self, "max-duration: %" G_GUINT64_FORMAT ".",
                     self->max_duration);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_START_TIME:
    GST_OBJECT_LOCK (self);
    self->start_time = g_value_get_uint64 (value);
    GST_INFO_OBJECT (self, "start-time: %" G_GUINT64_FORMAT ".",
                     self->start_time);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_AUTOSTART:
    GST_OBJECT_LOCK (self);
    self->autostart = g_value_get_boolean (value);
    GST_INFO_OBJECT (self, "autostart: %d", self->autostart);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_FILE_LOCATION:
    GST_OBJECT_LOCK (self);
    g_free (self->file_location);
    self->file_location = g_value_dup_string (value);
    GST_INFO_OBJECT (self, "file-location: %s.", self->file_location);
    self->file_location_specified = TRUE;
    GST_OBJECT_UNLOCK (self);
    break;

default:
    G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
    break;
}
g_rec_mutex_unlock (&self->interlock);

```

```

}

/* Compute the remaining run time of the sound, in nanoseconds. -1 means
 * infinity. */
static gint64
compute_remaining_time (GstLooper * object)
{
    GstLooper *self = GST_LOOPER (object);
    guint64 time_before_loop, time_inside_loop, time_after_loop;
    gdouble total_time;
    gdouble current_time;
    gdouble current_time_int;
    guint64 total_time_int;

    /* Compute the total time of the sound assuming no looping. */
    total_time = (gdouble) self->local_buffer_size / self->bytes_per_ns;
    total_time_int = (guint64) total_time;

    if (self->loop_from == 0)
    {
        /* If there is no looping, the time is simple to compute. */
        return (total_time_int - self->start_time - self->elapsed_time);
    }

    if ((self->loop_limit == 0) && (!self->released))
    {
        /* If we will loop forever, the time is also simple to compute. */
        return -1;
    }

    if (self->released)
    {
        /* We are looping, but we have received a release message,
         * so looping has stopped. We will run from the current
         * position to the end of the buffer. */
        current_time =
            (gdouble) self->local_buffer_drain_level / self->bytes_per_ns;
        current_time_int = (guint64) current_time;
        return (total_time_int - current_time_int);
    }

    /* Otherwise, we must compute the time before the loop, the time
     * after the loop, and the time spent inside the loop. */
    time_before_loop = self->loop_to - self->start_time;
    time_inside_loop = (self->loop_from - self->loop_to) * self->loop_limit;
    time_after_loop = total_time_int - self->loop_from;

```

```

    return (time_before_loop + time_inside_loop + time_after_loop -
           self->start_time);
}

/* Return the value of a property. */
static void
gst_looper_get_property (GObject * object, guint prop_id, GValue * value,
                        GParamSpec * pspec)
{
    GstLooper *self = GST_LOOPER (object);
    gchar *string_value;
    gdouble double_value;
    gint64 remaining_time;

    g_rec_mutex_lock (&self->interlock);
    switch (prop_id)
    {
        case PROP_SILENT:
            GST_OBJECT_LOCK (self);
            g_value_set_boolean (value, self->silent);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_LOOP_TO:
            GST_OBJECT_LOCK (self);
            g_value_set_uint64 (value, self->loop_to);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_LOOP_FROM:
            GST_OBJECT_LOCK (self);
            g_value_set_uint64 (value, self->loop_from);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_LOOP_LIMIT:
            GST_OBJECT_LOCK (self);
            g_value_set_uint (value, self->loop_limit);
            GST_OBJECT_UNLOCK (self);
            break;

        case PROP_MAX_DURATION:
            GST_OBJECT_LOCK (self);
            g_value_set_uint64 (value, self->max_duration);
            GST_OBJECT_UNLOCK (self);
            break;
    }
}

```

```

case PROP_START_TIME:
    GST_OBJECT_LOCK (self);
    g_value_set_uint64 (value, self->start_time);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_AUTOSTART:
    GST_OBJECT_LOCK (self);
    g_value_set_boolean (value, self->autostart);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_FILE_LOCATION:
    GST_OBJECT_LOCK (self);
    g_value_set_string (value, self->file_location);
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_ELAPSED_TIME:
    GST_OBJECT_LOCK (self);
    double_value = (gdouble) self->elapsed_time / (gdouble) 1e9;
    string_value = g_strdup_printf ("%4.1f", double_value);
    g_value_set_string (value, string_value);
    g_free (string_value);
    string_value = NULL;
    GST_OBJECT_UNLOCK (self);
    break;

case PROP_REMAINING_TIME:
    GST_OBJECT_LOCK (self);
    remaining_time = compute_remaining_time (self);
    if (remaining_time == -1)
    {
        /* Remaining time is infinite. */
        string_value = g_strdup ("∞");
    }
    else
    {
        double_value = (gdouble) (remaining_time) / (gdouble) 1e9;
        string_value = g_strdup_printf ("%4.1f", double_value);
    }

    g_value_set_string (value, string_value);
    g_free (string_value);
    string_value = NULL;

```

```

        GST_OBJECT_UNLOCK (self);
        break;

    default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
        break;
    }
    g_rec_mutex_unlock (&self->interlock);
}

/* entry point to initialize the plug-in
 * initialize the plug-in itself
 * register the element factories and other features
 */
static gboolean
looper_init (GstPlugin * looper)
{
    return gst_element_register (looper, "looper", GST_RANK_NONE,
                                GST_TYPE_LOOPER);
}

GST_PLUGIN_DEFINE (GST_VERSION_MAJOR, GST_VERSION_MINOR, looper,
    "Repeat a section of the input stream", looper_init,
    VERSION, "LGPL", "GStreamer", "http://gstreamer.net/")

```

9 gstlooper.h

```

/*
 * gstlooper.h, a file in sound_effects_player, a component of show_control,
 * which is a GStreamer application.
 *
 * Copyright © 2016 John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, see https://gnu.org/licenses
 * or write to:
 * Free Software Foundation, Inc.
 * 51 Franklin Street, Fifth Floor
 * Boston, MA 02111-1301
 * USA.
 */

#ifndef __GST_LOOPER_H__
#define __GST_LOOPER_H__

#include <gst/gst.h>

G_BEGIN_DECLS
#define GST_TYPE_LOOPER \
    (gst_looper_get_type())
#define GST_LOOPER(obj) \
    (G_TYPE_CHECK_INSTANCE_CAST((obj), GST_TYPE_LOOPER, GstLooper))
#define GST_LOOPER_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_CAST((klass), GST_TYPE_LOOPER, GstLooperClass))
#define GST_IS_LOOPER(obj) \
    (G_TYPE_CHECK_INSTANCE_TYPE((obj), GST_TYPE_LOOPER))
#define GST_IS_LOOPER_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_TYPE((klass), GST_TYPE_LOOPER))
typedef struct _GstLooper GstLooper;
typedef struct _GstLooperClass GstLooperClass;

```

```

struct _GstLooper
{
    GstElement element;

    /* Parameters */
    gboolean silent;
    guint64 loop_from;
    guint64 loop_to;
    guint64 max_duration;
    guint64 start_time;
    gchar *file_location;
    guint loop_limit;
    gboolean autostart;

    /* Locals */

    GstPad *sinkpad;
    GstPad *srcpad;
    GstBuffer *local_buffer;    /* The buffer that holds the data to send
                                * downstream. */

    guint64 local_buffer_fill_level;
    guint64 local_buffer_drain_level;
    guint64 local_buffer_size;    /* number of bytes in the local buffer */
    guint64 pull_level;    /* how much data we have pulled from upstream */
    guint64 timestamp_offset;
    guint64 local_clock;    /* The current time, in nanoseconds.
                            * This counts continuously through loops. */
    guint64 elapsed_time;    /* The amount of time, in nanoseconds, that
                            * we have been sending sound. */
    gdouble bytes_per_ns;    /* data rate in bytes per nanosecond */
    gchar *format;    /* The format of incoming data--for example,
                            * F32LE. */
    GRecMutex interlock;    /* used to prevent interference between tasks */
    guint64 loop_counter;
    guint64 width;    /* the size of a sample in bits */
    guint64 channel_count;    /* The number of channels of sound.
                            * Stereo has two. A frame consists of
                            * channel_count samples, each of width bits. */
    guint64 data_rate;    /* the data rate, in frames per second. */
    GstPadMode src_pad_mode;    /* The mode of the source pad: push or pull. */
    GstPadMode sink_pad_mode;    /* The mode of the sink pad: push or pull. */
    gboolean started;    /* We have received a Start signal. */
    gboolean completion_sent;    /* We have sent a "complete" message downstream
                                * to tell the envelope plugin that the

```

```

        * sound is complete. */
gboolean paused;      /* We have received a Pause signal, and it has
        * not yet been canceled by a Continue signal.
        */

gboolean continued;   /* We have received a Continue signal. If both
        * paused and continued are set, we will resume
        * sending sound, and clear both. */

gboolean released;    /* We have received a Release signal. */
gboolean data_buffered; /* We have received all the data we need into
        * our sink pad. */

gboolean src_pad_active; /* The source pad is active. */
gboolean sink_pad_active; /* The sink pad is active. */
gboolean sink_pad_flushing; /* The sink pad is flushing. */
gboolean src_pad_flushing; /* The source pad is flushing. */
gboolean src_pad_task_running; /* The task which pushes data downstream
        * on the source pad is active. */

gboolean sink_pad_task_running; /* The task which pulls data from
        * upstream on the sink pad is active.
        */

gboolean send_EOS;    /* The main task wants the pushing downstream
        * task to send an end-of-stream message and
        * terminate. */

gboolean state_change_pending; /* The main task wants the pushing
        * downstream task to complete a state
        * change. */

gboolean file_location_specified; /* The location of the wave file that
        * heads this bin has been specified.
        */

gboolean seen_incoming_data; /* Sound data has been seen on the source pad.
        */

guint8 silence_byte; /* The byte value of silence for this format.
        */

};

/* The number of bytes of data requested from upstream in each pull */
#define BUFFER_SIZE 4096

struct _GstLooperClass
{
    GstElementClass parent_class;
};

GType gst_looper_get_type (void);

G_END_DECLS

```



```
#endif /* __GST_LOOPER_H__ */
```

10 gstreamer__subroutines.c

```

/*
 * gstreamer_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
#include "gstreamer_subroutines.h"
#include "message_subroutines.h"
#include "sound_structure.h"
#include "sound_effects_player.h"
#include "sound_subroutines.h"
#include "button_subroutines.h"
#include "display_subroutines.h"
#include "main.h"
#include <math.h>

/* If true, print trace information as we proceed. */
#define GSTREAMER_TRACE FALSE

/* Set up the Gstreamer pipeline. */
GstPipeline *
gstreamer_init (int sound_count, GApplication * app)
{
    GstElement *tee_element;
    GstElement *queue_file_element;
    GstElement *queue_output_element;
    GstElement *wavenc_element;
    GstElement *filesink_element;
    GstElement *sink_element;
    GstElement *adder_element;
    GstElement *convert_element;
    GstElement *resample_element;

```

```

GstElement *final_bin_element;
GstElement *level_element;
GstElement *volume_element;
GstPipeline *pipeline_element;
GstBus *bus;
gchar *pad_name;
gint i;
GstPad *sink_pad;
gchar *monitor_file_name;
gboolean monitor_enabled;
gboolean output_enabled;

/* Check to see if --monitor-file was specified on the command line. */
monitor_file_name = main_get_monitor_file_name ();
monitor_enabled = FALSE;
if (monitor_file_name != NULL)
{
    monitor_enabled = TRUE;
}

/* We always send sound to the default sound output device. */
output_enabled = TRUE;

/* Create the top-level pipeline. */
pipeline_element = GST_PIPELINE (gst_pipeline_new ("sound_effects"));
if (pipeline_element == NULL)
{
    GST_ERROR ("Unable to create the gstreamer pipeline element.\n");
    return NULL;
}

/* Create the final bin, to collect the output of the sound effects bins
 * and play them. */
final_bin_element = gst_bin_new ("final");

/* Create the elements that will go in the final bin. */
adder_element = gst_element_factory_make ("adder", "final/adder");
level_element = gst_element_factory_make ("level", "final/master_level");
convert_element =
    gst_element_factory_make ("audioconvert", "final/convert");
resample_element =
    gst_element_factory_make ("audioresample", "final/resample");
volume_element = gst_element_factory_make ("volume", "final/volume");
if ((final_bin_element == NULL) || (adder_element == NULL)
    || (level_element == NULL) || (convert_element == NULL)
    || (resample_element == NULL) || (volume_element == NULL))

```

```

{
    GST_ERROR ("Unable to create the final gstreamer elements.\n");
    return NULL;
}

if ((monitor_enabled == FALSE) && (output_enabled == TRUE))
{
    /* audio only */
    tee_element = NULL;
    queue_file_element = NULL;
    queue_output_element = NULL;
    sink_element = gst_element_factory_make ("alsasink", "final/sink");
    wavenc_element = NULL;
    filesink_element = NULL;
    if (sink_element == NULL)
    {
        GST_ERROR ("Unable to create the final sink gstreamer element.\n");
    }
}

if ((monitor_enabled == TRUE) && (output_enabled == FALSE))
{
    /* file output only */
    tee_element = NULL;
    queue_file_element = NULL;
    queue_output_element = NULL;
    sink_element = NULL;
    wavenc_element = gst_element_factory_make ("wavenc", "final/wavenc");
    filesink_element =
        gst_element_factory_make ("filesink", "final/filesink");
    if ((wavenc_element == NULL) || (filesink_element == NULL))
    {
        GST_ERROR ("Unable to create the final sink gstreamer elements.\n");
    }
}

if ((monitor_enabled == TRUE) && (output_enabled == TRUE))
{
    /* both */
    tee_element = gst_element_factory_make ("tee", "final/tee");
    queue_file_element =
        gst_element_factory_make ("queue", "final/queue_file");
    queue_output_element =
        gst_element_factory_make ("queue", "final/queue_output");
    sink_element = gst_element_factory_make ("alsasink", "final/sink");
    wavenc_element = gst_element_factory_make ("wavenc", "final/wavenc");
    filesink_element =
        gst_element_factory_make ("filesink", "final/filesink");
    if ((tee_element == NULL) || (queue_file_element == NULL)
        || (queue_output_element == NULL) || (sink_element == NULL)
        || (wavenc_element == NULL) || (filesink_element == NULL))

```

```

    {
        GST_ERROR ("Unable to create the final sink gstreamer elements.\n");
    }
}

/* Put the needed elements into the final bin. */
gst_bin_add_many (GST_BIN (final_bin_element), adder_element, level_element,
                  convert_element, resample_element, volume_element, NULL);
if (output_enabled == TRUE)
{
    gst_bin_add_many (GST_BIN (final_bin_element), sink_element, NULL);
}
if (monitor_enabled == TRUE)
{
    gst_bin_add_many (GST_BIN (final_bin_element), wavenc_element,
                    filesink_element, NULL);
}
if ((output_enabled == TRUE) && (monitor_enabled == TRUE))
{
    gst_bin_add_many (GST_BIN (final_bin_element), tee_element,
                    queue_file_element, queue_output_element, NULL);
}

/* Make sure we will get level messages. */
g_object_set (level_element, "post-messages", TRUE, NULL);

if (monitor_enabled == TRUE)
{
    /* Set the file name for monitoring the output. */
    g_object_set (filesink_element, "location", monitor_file_name, NULL);
}

/* Watch for messages from the pipeline. */
bus = gst_element_get_bus (GST_ELEMENT (pipeline_element));
gst_bus_add_watch (bus, message_handler, app);

/* The inputs to the final bin are the inputs to the adder. Create enough
 * sinks for each sound effect. */
for (i = 0; i < sound_count; i++)
{
    sink_pad = gst_element_get_request_pad (adder_element, "sink_%u");
    pad_name = g_strdup_printf ("sink %d", i);
    gst_element_add_pad (final_bin_element,
                        gst_ghost_pad_new (pad_name, sink_pad));
    g_free (pad_name);
}

```

```

    /* Link the various elements in the final bin together. */
    gst_element_link (adder_element, level_element);
    gst_element_link (level_element, convert_element);
    gst_element_link (convert_element, resample_element);
    gst_element_link (resample_element, volume_element);
    if ((output_enabled == TRUE) && (monitor_enabled == FALSE))
    {
        gst_element_link (volume_element, sink_element);
    }
    if ((output_enabled == FALSE) && (monitor_enabled == TRUE))
    {
        gst_element_link (volume_element, wavenc_element);
        gst_element_link (wavenc_element, filesink_element);
    }
    if ((output_enabled == TRUE) && (monitor_enabled == TRUE))
    {
        gst_element_link (volume_element, tee_element);
        gst_element_link (tee_element, queue_file_element);
        gst_element_link (tee_element, queue_output_element);
        gst_element_link (queue_output_element, sink_element);
        gst_element_link (queue_file_element, wavenc_element);
        gst_element_link (wavenc_element, filesink_element);
    }

    /* Place the final bin in the pipeline. */
    gst_bin_add (GST_BIN (pipeline_element), final_bin_element);

    return pipeline_element;
}

/* Create a Gstreamer bin for a sound effect. */
GstBin *
gststreamer_create_bin (struct sound_info * sound_data, int sound_number,
                        GstPipeline * pipeline_element, GApplication * app)
{
    GstElement *source_element, *parse_element, *convert_element;
    GstElement *resample_element, *looper_element;
    GstElement *envelope_element, *pan_element, *volume_element;
    GstElement *bin_element, *final_bin_element;
    gchar *sound_name, *pad_name, *element_name;
    GstPad *last_source_pad, *sink_pad;
    GstPadLinkReturn link_status;
    gboolean success;
    gchar string_buffer[G_ASCII_DTOSTR_BUF_SIZE];

```

```

/* Create the bin, source and various filter elements for this sound effect.
   */
sound_name = g_strconcat ((gchar *) "sound/", sound_data->name, NULL);
bin_element = gst_bin_new (sound_name);
if (bin_element == NULL)
{
    GST_ERROR ("Unable to create the bin element.\n");
    return NULL;
}
element_name = g_strconcat (sound_name, (gchar *) "/source", NULL);
source_element = gst_element_factory_make ("filesrc", element_name);
if (source_element == NULL)
{
    GST_ERROR ("Unable to create the file source element.\n");
    return NULL;
}
g_free (element_name);
element_name = g_strconcat (sound_name, (gchar *) "/parse", NULL);
parse_element = gst_element_factory_make ("wavparse", element_name);
if (parse_element == NULL)
{
    GST_ERROR ("Unable to create the wave file parse element.\n");
    return NULL;
}
g_free (element_name);
element_name = g_strconcat (sound_name, (gchar *) "/looper", NULL);
looper_element = gst_element_factory_make ("looper", element_name);
if (looper_element == NULL)
{
    GST_ERROR ("Unable to create the looper element.\n");
    return NULL;
}
g_free (element_name);
element_name = g_strconcat (sound_name, (gchar *) "/convert", NULL);
convert_element = gst_element_factory_make ("audioconvert", element_name);
if (convert_element == NULL)
{
    GST_ERROR ("Unable to create the audio convert element.\n");
    return NULL;
}
g_free (element_name);
element_name = g_strconcat (sound_name, (gchar *) "/resample", NULL);
resample_element = gst_element_factory_make ("audioresample", element_name);
if (resample_element == NULL)
{
    GST_ERROR ("Unable to create the resample element.\n");

```

```

        return NULL;
    }
    g_free (element_name);
    element_name = g_strconcat (sound_name, (gchar *) "/envelope", NULL);
    envelope_element = gst_element_factory_make ("envelope", element_name);
    if (envelope_element == NULL)
    {
        GST_ERROR ("Unable to create the envelope element.\n");
        return NULL;
    }
    g_free (element_name);
    if (!sound_data->omit_panning)
    {
        element_name = g_strconcat (sound_name, (gchar *) "/pan", NULL);
        pan_element = gst_element_factory_make ("audiopanorama", element_name);
        if (pan_element == NULL)
        {
            GST_ERROR ("Unable to create the pan element.\n");
            return NULL;
        }
        g_free (element_name);
    }
    else
    {
        pan_element = NULL;
    }

    element_name = g_strconcat (sound_name, (gchar *) "/volume", NULL);
    volume_element = gst_element_factory_make ("volume", element_name);
    if (volume_element == NULL)
    {
        GST_ERROR ("Unable to create the volume element.\n");
        return NULL;
    }
    g_free (element_name);

    g_free (sound_name);
    element_name = NULL;
    sound_name = NULL;

    /* Set parameter values of the elements. */
    g_object_set (source_element, "location", sound_data->wav_file_name_full,
        NULL);

    g_object_set (looper_element, "file-location",
        sound_data->wav_file_name_full, NULL);

```



```

g_object_set (looper_element, "loop-to", sound_data->loop_to_time, NULL);
g_object_set (looper_element, "loop-from", sound_data->loop_from_time,
              NULL);
g_object_set (looper_element, "loop-limit", sound_data->loop_limit, NULL);
g_object_set (looper_element, "max-duration", sound_data->max_duration_time,
              NULL);
g_object_set (looper_element, "start-time", sound_data->start_time, NULL);

g_object_set (envelope_element, "attack-duration-time",
              sound_data->attack_duration_time, NULL);
g_object_set (envelope_element, "attack_level", sound_data->attack_level,
              NULL);
g_object_set (envelope_element, "decay-duration-time",
              sound_data->decay_duration_time, NULL);
g_object_set (envelope_element, "sustain-level", sound_data->sustain_level,
              NULL);
g_object_set (envelope_element, "release-start-time",
              sound_data->release_start_time, NULL);
if (sound_data->release_duration_infinite)
{
    g_object_set (envelope_element, "release-duration-time",
                  (gchar *) "∞", NULL);
}
else
{
    g_ascii_dtostr (string_buffer, G_ASCII_DTOSTR_BUF_SIZE,
                  (gdouble) sound_data->release_duration_time);
    g_object_set (envelope_element, "release-duration-time", string_buffer,
                  NULL);
}
/* We don't need another volume element because the envelope element
 * can also take a volume parameter which makes a global adjustment
 * to the envelope, thus adjusting the volume. */
g_object_set (envelope_element, "volume", sound_data->designer_volume_level,
              NULL);
g_object_set (envelope_element, "sound-name", sound_data->name, NULL);

if (!sound_data->omit_panning)
{
    g_object_set (pan_element, "panorama", sound_data->designer_pan, NULL);
}

/* Place the various elements in the bin. */
gst_bin_add_many (GST_BIN (bin_element), source_element, parse_element,
                  looper_element, convert_element, resample_element,
                  envelope_element, volume_element, NULL);

```

```

if (!sound_data->omit_panning)
{
    gst_bin_add_many (GST_BIN (bin_element), pan_element, NULL);
}

/* Link them together in this order:
 * source->parse->looper->convert->resample->envelope->pan->volume.
 * Note that because the looper reads the wave file directly, as well
 * as getting it through the pipeline, the audio converter must be
 * after it. It is for this reason that the looper handles a variety
 * of audio formats. Note also that the pan element is optional. */
gst_element_link (source_element, parse_element);
gst_element_link (parse_element, looper_element);
gst_element_link (looper_element, convert_element);
gst_element_link (convert_element, resample_element);
gst_element_link (resample_element, envelope_element);
if (!sound_data->omit_panning)
{
    gst_element_link (envelope_element, pan_element);
    gst_element_link (pan_element, volume_element);
}
else
{
    gst_element_link (envelope_element, volume_element);
}

/* The output of the bin is the output of the last element. */
last_source_pad = gst_element_get_static_pad (volume_element, "src");
gst_element_add_pad (bin_element,
                    gst_ghost_pad_new ("src", last_source_pad));

/* Place the bin in the pipeline. */
success = gst_bin_add (GST_BIN (pipeline_element), bin_element);
if (!success)
{
    GST_ERROR ("Failed to add sound effect %s bin to pipeline.\n",
              sound_data->name);
}

/* Link the output of the sound effect bin to the final bin. */
final_bin_element =
    gst_bin_get_by_name (GST_BIN (pipeline_element), (gchar *) "final");
pad_name = g_strdup_printf ("sink %d", sound_number);
last_source_pad = gst_element_get_static_pad (bin_element, "src");
sink_pad = gst_element_get_static_pad (final_bin_element, pad_name);
link_status = gst_pad_link (last_source_pad, sink_pad);

```

```

    if (link_status != GST_PAD_LINK_OK)
    {
        GST_ERROR ("Failed to link sound effect %s to final bin: %d, %d.\n",
                    sound_data->name, sound_number, link_status);
    }
    g_free (pad_name);

    if (GSTREAMER_TRACE)
    {
        g_print ("created gstreamer bin for %s.\n", sound_data->name);
    }
    return (GST_BIN (bin_element));
}

/* After the individual bins are created, complete the pipeline. */
void
gstreamer_complete_pipeline (GstPipeline * pipeline_element,
                             GApplication * app)
{
    GstStateChangeReturn set_state_val;
    GstBus *bus;
    GstMessage *msg;
    GError *err = NULL;

    /* For debugging, write out a graphical representation of the pipeline. */
    gstreamer_dump_pipeline (pipeline_element);

    /* Now that the pipeline is constructed, start it running. Unless a sound
     * is autostarted, there will be no sound until a sound effect bin receives
     * a start message. */
    set_state_val =
        gst_element_set_state (GST_ELEMENT (pipeline_element), GST_STATE_PLAYING);
    if (set_state_val == GST_STATE_CHANGE_FAILURE)
    {
        g_print ("Unable to initial start the gstreamer pipeline.\n");

        /* Check for an error message with details on the bus. */
        bus = gst_pipeline_get_bus (pipeline_element);
        msg = gst_bus_poll (bus, GST_MESSAGE_ERROR, 0);
        if (msg != NULL)
        {
            gst_message_parse_error (msg, &err, NULL);
            g_print ("Error: %s.\n", err->message);
            g_error_free (err);
            gst_message_unref (msg);
        }
    }
}

```

```

    }

    if (GSTREAMER_TRACE)
    {
        g_print ("started the gstreamer pipeline.\n");
    }
    return;
}

/* We are done with Gstreamer; shut it down. */
void
gststreamer_shutdown (GApplication * app)
{
    GstPipeline *pipeline_element;
    GstEvent *event;
    GstStructure *structure;

    pipeline_element = sep_get_pipeline_from_app (app);

    if (pipeline_element != NULL)
    {
        /* For debugging, write out a graphical representation of the pipeline. */
        gststreamer_dump_pipeline (pipeline_element);

        /* Send a shutdown message to the pipeline. The message will be
         * received by every element, so the loop element will stop
         * sending data in anticipation of being shut down. */
        structure = gst_structure_new_empty ((gchar *) "shutdown");
        event = gst_event_new_custom (GST_EVENT_CUSTOM_UPSTREAM, structure);
        gst_element_send_event (GST_ELEMENT (pipeline_element), event);

        /* The loop element will send end-of-stream (EOS). When that
         * has propagated through the pipeline, we will get it, shut down
         * the pipeline and quit. */
    }
    else
    {
        /* We don't have a pipeline, so just quit. */
        g_application_quit (app);
    }

    return;
}

/* Handle the async-done event from the gstreamer pipeline.
The first such event means that the gstreamer pipeline has finished

```

```

    its initialization. */
    void
    gstreamer_async_done (GApplication * app)
    {
        GstPipeline *pipeline_element;

        pipeline_element = sep_get_pipeline_from_app (app);

        /* For debugging, write out a graphical representation of the pipeline. */
        gstreamer_dump_pipeline (pipeline_element);

        /* Tell the core that we have completed gstreamer initialization. */
        sep_gstreamer_ready (app);

        return;
    }

    /* The pipeline has reached end of stream. This should happen only after
    * the shutdown message has been sent. */
    void
    gstreamer_process_eos (GApplication * app)
    {
        GstPipeline *pipeline_element;

        pipeline_element = sep_get_pipeline_from_app (app);

        /* For debugging, write out a graphical representation of the pipeline. */
        gstreamer_dump_pipeline (pipeline_element);

        /* Tell the pipeline to shut down. */
        gst_element_set_state (GST_ELEMENT (pipeline_element), GST_STATE_NULL);

        /* Now we can quit. */
        g_application_quit (app);

        return;
    }

    /* Find the volume control in a bin. */
    GstElement *
    gstreamer_get_volume (GstBin * bin_element)
    {
        GstElement *volume_element;
        gchar *element_name, *bin_name;

        bin_name = gst_element_get_name (bin_element);

```

```

    element_name = g_strconcat (bin_name, (gchar *) "/volume", NULL);
    g_free (bin_name);
    volume_element = gst_bin_get_by_name (bin_element, element_name);
    g_free (element_name);

    return (volume_element);
}

/* Find the pan control in a bin. It might have been omitted by the
 * sound designer. */
GstElement *
gststreamer_get_pan (GstBin * bin_element)
{
    GstElement *pan_element;
    gchar *element_name, *bin_name;

    bin_name = gst_element_get_name (bin_element);
    element_name = g_strconcat (bin_name, (gchar *) "/pan", NULL);
    g_free (bin_name);
    pan_element = gst_bin_get_by_name (bin_element, element_name);
    g_free (element_name);

    return (pan_element);
}

/* Find the looper element in a bin. */
GstElement *
gststreamer_get_looper (GstBin * bin_element)
{
    GstElement *looper_element;
    gchar *element_name, *bin_name;

    bin_name = gst_element_get_name (bin_element);
    element_name = g_strconcat (bin_name, (gchar *) "/looper", NULL);
    g_free (bin_name);
    looper_element = gst_bin_get_by_name (bin_element, element_name);
    g_free (element_name);

    return (looper_element);
}

/* For debugging, write out an annotated, graphical representation
 * of the gstreamer pipeline.
 */
void
gststreamer_dump_pipeline (GstPipeline * pipeline_element)

```

```
{
    gst_debug_bin_to_dot_file_with_ts (GST_BIN (pipeline_element),
                                        GST_DEBUG_GRAPH_SHOW_ALL,
                                        "sound_effects_player_pipeline");
    return;
}
```

11 gstreamer_subroutines.h

```

/*
 * gstreamer_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
#include <gtk/gtk.h>
#include <gst/gst.h>
#include "sound_structure.h"

/* Subroutines defined in gstreamer_subroutines.c */
GstPipeline *gstreamer_init (int sound_count, GApplication * app);
GstBin *gstreamer_create_bin (struct sound_info *sound_data, int sound_number,
                             GstPipeline * pipeline_element,
                             GApplication * app);
void gstreamer_complete_pipeline (GstPipeline * pipeline_element,
                                  GApplication * app);
void gstreamer_shutdown (GApplication * app);
void gstreamer_async_done (GApplication * app);
void gstreamer_process_eos (GApplication * app);
GstElement *gstreamer_get_volume (GstBin * bin_element);
GstElement *gstreamer_get_pan (GstBin * bin_element);
GstElement *gstreamer_get_looper (GstBin * bin_element);
void gstreamer_dump_pipeline (GstPipeline * pipeline_element);

```


12 main.c

```

/*
 * main.c
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * sound_effects_player is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * sound_effects_player is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include "config.h"
#include "main.h"
#include "sound_effects_player.h"

#include <glib/gi18n.h>

/* Persistent data. */
gchar *monitor_file_name = NULL;

/* The entry point for the sound_effects_player application.
 * This is a GTK application, so much of what is done here is standard
 * boilerplate. See sound_effects_player.c for the beginning of actual
 * application-specific code.
 */
int
main (int argc, char *argv[])
{
    Sound_Effects_Player *app;
    int status;
    gchar **filenames = NULL;
    gchar *pid_file_name = NULL;
    gboolean pid_file_written = FALSE;
    FILE *pid_file = NULL;
    const GOptionEntry entries[] = {
        {"process-id-file", 'p', 0, G_OPTION_ARG_FILENAME, &pid_file_name,
         "name of the file written with the process id, for signaling"},

```

```

    {"monitor-file", 'p', 0, G_OPTION_ARG_FILENAME, &monitor_file_name,
     "name of the file which monitors output"},
    /* add more command line options here */
    {G_OPTION_REMAINING, 0, 0, G_OPTION_ARG_FILENAME_ARRAY, &filenames,
     "Special option that collects any remaining arguments for us"},
    {NULL,}
};
GOptionContext *ctx;
GError *err = NULL;
const gchar *nano_str;
const gchar *check_version_str;
guint major, minor, micro, nano;
extern const guint glib_major_version;
extern const guint glib_minor_version;
extern const guint glib_micro_version;
extern const guint glib_binary_age;
extern const guint glib_interface_age;
int fake_argc;
char *fake_argv[2];

#ifdef ENABLE_NLS
    bindtextdomain (GETTEXT_PACKAGE, PACKAGE_LOCALE_DIR);
    bind_textdomain_codeset (GETTEXT_PACKAGE, "UTF-8");
    textdomain (GETTEXT_PACKAGE);
#endif

/* Initialize gtk and Gstreamer. */
gtk_init (&argc, &argv);

/* Parse the command line. */
ctx = g_option_context_new ("[project_file]");
g_option_context_add_group (ctx, gtk_get_option_group (TRUE));
g_option_context_add_group (ctx, gst_init_get_option_group ());
g_option_context_add_main_entries (ctx, entries, NULL);
g_option_context_set_summary (ctx, "Play sound effects for show_control.");

if (!g_option_context_parse (ctx, &argc, &argv, &err))
{
    g_print ("Error initializing: %s\n", GST_STR_NULL (err->message));
    return -1;
}
g_option_context_free (ctx);

/* If a process ID file was specified, write our process ID to it. */
if (pid_file_name != NULL)
{

```

```

    errno = 0;
    pid_file = fopen (pid_file_name, "w");
    if (pid_file != NULL)
    {
        fprintf (pid_file, "%d\n", getpid ());
        fclose (pid_file);
        pid_file_written = TRUE;
    }
    else
    {
        g_print ("Cannot create process ID file %s: %s", pid_file_name,
                strerror (errno));
        return 1;
    }
}

/* Print the version of glib that we are linked against. */
g_print ("This program is linked against glib %d.%d.%d ages %d and %d.\n",
        glib_major_version, glib_minor_version, glib_micro_version,
        glib_binary_age, glib_interface_age);

/* Print the version of gtk that we are linked against. */
major = gtk_get_major_version ();
minor = gtk_get_minor_version ();
micro = gtk_get_micro_version ();
g_print ("This program is linked against gtk %d.%d.%d.\n", major, minor,
        micro);

/* Check that the version of gtk is good. */
check_version_str =
    gtk_check_version (GTK_MAJOR_VERSION, GTK_MINOR_VERSION,
        GTK_MICRO_VERSION);
if (check_version_str != NULL)
{
    g_print (check_version_str);
    return -1;
}

/* Print the version of Gstreamer that we are linked against. */
gst_version (&major, &minor, &micro, &nano);

if (nano == 1)
    nano_str = "(CVS)";
else if (nano == 2)
    nano_str = "(Prerelease)";
else

```

```

    nano_str = "";
    g_print ("This program is linked against GStreamer %d.%d.%d%s.\n", major,
            minor, micro, nano_str);

    /* Initialize gstreamer */
    gst_init (&argc, &argv);

    /* Run the program. The values from argc and argv have already been
     * parsed, so create a fake version of argc and argv with the filename
     * argument, if present. */
    fake_argc = argc;
    fake_argv[0] = argv[0];
    if ((filenames != NULL) && (filenames[0] != NULL))
    {
        fake_argc = 2;
        fake_argv[1] = filenames[0];
    }
    app = sound_effects_player_new ();
    status = g_application_run (G_APPLICATION (app), fake_argc, fake_argv);

    /* We are done. */
    g_object_unref (app);

    /* If we wrote a file with the process ID, delete it. */
    if (pid_file_written)
        remove (pid_file_name);
    free (pid_file_name);
    pid_file_name = NULL;

    free(monitor_file_name);
    monitor_file_name = NULL;

    return status;
}

/* Fetch the name of the monitor file. */
gchar *
main_get_monitor_file_name ()
{
    return monitor_file_name;
}

```

13 main.h

```
/*
 * main.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gtk/gtk.h>

/* Subroutines defined in main.c */

gchar *main_get_monitor_file_name ();

/* End of file main.h */
```

14 menu_subroutines.c

```

/*
 * menu_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdlib.h>
#include <gtk/gtk.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include "menu_subroutines.h"
#include "parse_xml_subroutines.h"
#include "network_subroutines.h"
#include "sound_subroutines.h"
#include "sound_effects_player.h"
#include "gstreamer_subroutines.h"

/* Subroutines used by the menus. */

/* The user has invoked the preferences dialogue from a menu. */
static void
preferences_activated (GSimpleAction * action, GVariant * parameter,
                      gpointer app)
{
    GtkBuilder *builder;
    GError *error = NULL;
    gchar *preferences_file_name;
    GtkWidget *parent_window;
    GtkDialog *dialog;
    gchar *ui_path;

```

```

/* Load the preferences user interface definition from its file. */
builder = gtk_builder_new ();
ui_path = sep_get_ui_path (app);
preferences_file_name = g_strconcat (ui_path, "preferences.ui", NULL);
if (!gtk_builder_add_from_file (builder, preferences_file_name, &error))
{
    g_critical ("Couldn't load builder file %s: %s", preferences_file_name,
               error->message);
    g_error_free (error);
    return;
}

/* Auto-connect signal handlers. */
gtk_builder_connect_signals (builder, app);

/* Get the dialog object from the UI file. */
dialog = GTK_DIALOG (gtk_builder_get_object (builder, "dialog1"));
if (dialog == NULL)
{
    g_critical ("Widget \"dialog1\" is missing in file %s.\n",
               preferences_file_name);
    return;
}

/* We are done with the name of the preferences user interface file
 * and the user interface builder. */
g_free (preferences_file_name);
g_object_unref (G_OBJECT (builder));

/* Set the dialog window's application, so we can retrieve it later. */
gtk_window_set_application (GTK_WINDOW (dialog), app);

/* Get the top-level window to use as the transient parent for
 * the dialog. This makes sure the dialog appears over the
 * application window. Also, destroy the dialog if the application
 * is closed, and propagate information such as styling and accessibility
 * from the top-level window to the dialog. */
parent_window = sep_get_top_window ((GApplication *) app);
gtk_window_set_transient_for (GTK_WINDOW (dialog), parent_window);
gtk_window_set_destroy_with_parent (GTK_WINDOW (dialog), TRUE);
gtk_window_set_attached_to (GTK_WINDOW (dialog),
                           GTK_WIDGET (parent_window));

/* Run the dialog and wait for it to complete. */
gtk_dialog_run (dialog);
gtk_widget_destroy (GTK_WIDGET (dialog));

```

```

    return;
}

/* Subroutine called when the preferences menu changes the network port. */
gboolean
menu_network_port_changed (GtkEntry * port_entry, GtkWidget * dialog_widget)
{
    GApplication *app;
    const gchar *port_text;
    long int port_number;

    /* We are passed the dialogue widget. We previously set its application
     * so we can retrieve it here. */
    app =
        G_APPLICATION (gtk_window_get_application (GTK_WINDOW (dialog_widget)));

    /* Extract the port number from the entry widget. */
    port_text = gtk_entry_get_text (port_entry);
    port_number = strtoll (port_text, NULL, 10);

    /* Tell the network module to change its port number. */
    network_set_port (port_number, app);

    return TRUE;
}

/* Subroutine called when the preferences dialog is closed. */
gboolean
menu_preferences_close_clicked (GtkButton * close_button,
                                GtkWidget * dialog_widget)
{
    gtk_dialog_response (GTK_DIALOG (dialog_widget), 0);
    return FALSE;
}

/* Subroutine called when the application's "quit" menu item is selected. */
static void
quit_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
    /* Shut down the gstreamer pipeline, then terminate the application. */
    gstreamer_shutdown (app);
}

/* Subroutine called when the top-level window is closed. */
gboolean

```



```

menu_delete_top_window (GtkButton * close_button, GdkEvent * event,
                        GtkWidget * top_box)
{
    GApplication *app;

    app = sep_get_application_from_widget (top_box);
    gstreamer_shutdown (app);

    /* The gstreamer shutdown process is asynchronous, and will terminate
     * the application when it completes, so don't do the termination here.
     */
    return TRUE;
}

/* Reset the project to its defaults. */
static void
new_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
    sep_set_project_file (NULL, app);
    network_set_port (1500, app);

    return;
}

/* Open a project file and read its contents. The file is assumed to be in
 * XML format. */
static void
open_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
    GtkWidget *dialog;
    GtkFileChooser *chooser;
    GtkFileChooserAction file_action = GTK_FILE_CHOOSER_ACTION_OPEN;
    GtkWindow *parent_window;
    gint res;
    gchar *project_file_name;

    /* Get the top-level window to use as the transient parent for
     * the file open dialog. This makes sure the dialog appears over the
     * application window. */
    parent_window = sep_get_top_window ((GApplication *) app);

    /* Configure the dialogue: choosing multiple files is not permitted. */
    dialog =
        gtk_file_chooser_dialog_new ("Open Project File", parent_window,
                                    file_action, "_Cancel", GTK_RESPONSE_CANCEL,
                                    "_Open", GTK_RESPONSE_ACCEPT, NULL);

```

```

chooser = GTK_FILE_CHOOSER (dialog);
gtk_file_chooser_set_select_multiple (chooser, FALSE);

/* Use the file dialog to ask for a file to read. */
res = gtk_dialog_run (GTK_DIALOG (dialog));
if (res == GTK_RESPONSE_ACCEPT)
{
    /* We have a file name. */
    project_file_name = gtk_file_chooser_get_filename (chooser);
    gtk_widget_destroy (dialog);
    /* Parse the file as an XML file and create the gstreamer pipeline. */
    sep_create_pipeline (project_file_name, (GApplication *) app);
}

return;
}

static void
save_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
}

/* Write the project information to an XML file. */
static void
save_as_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
    GtkWidget *dialog;
    GtkFileChooser *chooser;
    GtkFileChooserAction file_action = GTK_FILE_CHOOSER_ACTION_SAVE;
    GtkWidget *parent_window;
    gint res;
    gchar *project_file_name;

    /* Get the top-level window to use as the transient parent for
     * the dialog. This makes sure the dialog appears over the
     * application window. */
    parent_window = sep_get_top_window ((GApplication *) app);

    /* Configure the dialogue: prompt on specifying an existing file,
     * allow folder creation, and specify the current project file
     * name if one exists, or a default name if not. */
    dialog =
        gtk_file_chooser_dialog_new ("Save Project File", parent_window,
                                    file_action, "_Cancel", GTK_RESPONSE_CANCEL,
                                    "_Save", GTK_RESPONSE_ACCEPT, NULL);

```

```

gtk_window_set_application (GTK_WINDOW (dialog), app);
chooser = GTK_FILE_CHOOSER (dialog);
gtk_file_chooser_set_do_overwrite_confirmation (chooser, TRUE);
gtk_file_chooser_set_create_folders (chooser, TRUE);
project_file_name = sep_get_project_filename (app);
if (project_file_name == NULL)
{
    project_file_name = g_strdup ("Nameless_project.xml");
    sep_set_project_filename (project_file_name, app);
}
gtk_file_chooser_set_filename (chooser, project_file_name);

/* Use the file dialog to ask for a file to write. */
res = gtk_dialog_run (GTK_DIALOG (dialog));
if (res == GTK_RESPONSE_ACCEPT)
{
    /* We have a file name. */
    project_file_name = gtk_file_chooser_get_filename (chooser);
    gtk_widget_destroy (dialog);
    parse_xml_write_project_file (project_file_name, app);
}

return;
}

static void
copy_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
}

static void
cut_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
}

static void
paste_activated (GSimpleAction * action, GVariant * parameter, gpointer app)
{
}

/* Actions table to link menu items to subroutines. */
static GActionEntry app_entries[] = {
    {"preferences", preferences_activated, NULL, NULL, NULL},
    {"quit", quit_activated, NULL, NULL, NULL},

```

```

{"new", new_activated, NULL, NULL, NULL},
{"open", open_activated, NULL, NULL, NULL},
{"save", save_activated, NULL, NULL, NULL},
{"save_as", save_as_activated, NULL, NULL, NULL},
{"copy", copy_activated, NULL, NULL, NULL},
{"cut", cut_activated, NULL, NULL, NULL},
{"paste", paste_activated, NULL, NULL, NULL}
};

/* Initialize the menu. */
void
menu_init (GApplication * app, gchar * file_name)
{
    GtkBuilder *builder;
    GError *error = NULL;
    GMenuModel *app_menu;
    GMenuModel *menu_bar;

    const gchar *quit_accels[2] = { "<Ctrl>Q", NULL };

    g_action_map_add_action_entries (G_ACTION_MAP (app), app_entries,
                                     G_N_ELEMENTS (app_entries), app);
    gtk_application_set_accels_for_action (GTK_APPLICATION (app), "app.quit",
                                           quit_accels);

    /* Load UI from file */
    builder = gtk_builder_new ();
    if (!gtk_builder_add_from_file (builder, file_name, &error))
    {
        g_critical ("Couldn't load menu file %s: %s", file_name,
                    error->message);
        g_error_free (error);
    }

    /* Auto-connect signal handlers. */
    gtk_builder_connect_signals (builder, app);

    /* Specify the application menu and the menu bar. */
    app_menu = (GMenuModel *) gtk_builder_get_object (builder, "appmenu");
    menu_bar = (GMenuModel *) gtk_builder_get_object (builder, "menubar");
    gtk_application_set_app_menu (GTK_APPLICATION (app), app_menu);
    gtk_application_set_menubar (GTK_APPLICATION (app), menu_bar);

    /* We are finished with the builder. */
    g_object_unref (builder);
}

```

```
    return;  
}
```

15 menu_subroutines.h

```
/*
 * menu_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gtk/gtk.h>
/* Subroutines defined in menu_subroutines.c */

void menu_init (GApplication * app, gchar * file_name);
```

16 message__subroutines.c

```

/*
 * message_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/* GValueArray has been depreciated in favor of GArray since GLib 2.32,
 * but the "good" plugin named level still uses it in Gstreamer 1.4.
 * Maybe I will rewrite the level plugin to use GArray, but until then
 * disable GLib depreciation warnings. */
#define GLIB_DISABLE_DEPRECATED_WARNINGS

#include "message_subroutines.h"
#include <math.h>
#include <gst/gst.h>
#include "display_subroutines.h"
#include "sound_subroutines.h"
#include "gstreamer_subroutines.h"
#include "sound_effects_player.h"

/* When debugging, it is sometimes useful to have printouts of the
 * messages as they happen. */
#define TRACE_MESSAGES FALSE

/* Process a message from the pipeline. User_data is the
 * application, so we can reach the display. */
gboolean
message_handler (GstBus * bus_element, GstMessage * message,
                 gpointer user_data)
{
    GstPipeline *pipeline_element;

```

```

pipeline_element = sep_get_pipeline_from_app (user_data);

if (TRACE_MESSAGES && FALSE)
{
    /* For debugging, write out a graphical representation of the pipeline.
    */
    gstreamer_dump_pipeline (pipeline_element);
}

switch (GST_MESSAGE_TYPE (message))
{
case GST_MESSAGE_ELEMENT:
    /* This is a special-purpose message from an element. */
    {
        const GstStructure *s = gst_message_get_structure (message);

        if (gst_structure_has_name (s, (gchar *) "level"))
        {
            /* The level message shows the sound level on each channel. */
            gint channels;
            GstClockTime endtime;
            gdouble rms_dB, peak_dB, decay_dB;
            gdouble rms;
            const GValue *array_val;
            const GValue *value;
            GValueArray *rms_arr, *peak_arr, *decay_arr;
            gint i;

            if (!gst_structure_get_clock_time (s, "endtime", &endtime))
                g_warning ("Could not parse endtime.");

            /* The values are packed into GValueArrays
            * with the value per channel. */
            array_val = gst_structure_get_value (s, "rms");
            rms_arr = (GValueArray *) g_value_get_boxed (array_val);

            array_val = gst_structure_get_value (s, "peak");
            peak_arr = (GValueArray *) g_value_get_boxed (array_val);

            array_val = gst_structure_get_value (s, "decay");
            decay_arr = (GValueArray *) g_value_get_boxed (array_val);

            /* We can get the number of channels as the length of any of the
            * value arrays. */
            channels = rms_arr->n_values;
        }
    }
}

```



```

    for (i = 0; i < channels; ++i)
    {
        value = g_value_array_get_nth (rms_arr, i);
        rms_dB = g_value_get_double (value);

        value = g_value_array_get_nth (peak_arr, i);
        peak_dB = g_value_get_double (value);

        value = g_value_array_get_nth (decay_arr, i);
        decay_dB = g_value_get_double (value);

        /* Converting from dB to normal gives us a value between
           * 0.0 and 1.0. */
        rms = pow (10, rms_dB / 20);
        display_update_vu_meter (user_data, i, rms, peak_dB,
                                decay_dB);
    }
    break;
}

/* Check for a forwarded message from a bin. */
if (gst_structure_has_name (s, (gchar *) "GstBinForwarded"))
{
    GstMessage *forward_msg = NULL;

    gst_structure_get (s, "message", GST_TYPE_MESSAGE, &forward_msg,
                      NULL);
    if (GST_MESSAGE_TYPE (forward_msg) == GST_MESSAGE_EOS)
    {
        if (TRACE_MESSAGES)
        {
            g_print ("Forwarded EOS from element %s.\n",
                    GST_OBJECT_NAME (GST_MESSAGE_SRC (forward_msg)));
        }
        break;
    }
}

if (gst_structure_has_name (s, (gchar *) "completed"))
{
    /* The completed message means a sound has finished. */
    const gchar *sound_name;

    /* The structure in the message contains the name of the sound.
       */

```

```

        sound_name = gst_structure_get_string (s, (gchar *) "sound_name");
        sound_completed (sound_name, G_APPLICATION (user_data));
    }

    if (gst_structure_has_name (s, (gchar *) "release_started"))
    {
        /* The release_started message means a sound has entered the
         * release portion of its envelope. */
        const gchar *sound_name;

        /* The structure in the message contains the name of the sound.
         */
        sound_name = gst_structure_get_string (s, (gchar *) "sound_name");
        sound_release_started (sound_name, G_APPLICATION (user_data));
    }

    /* Catchall for unrecognized messages */
    if (TRACE_MESSAGES)
    {
        g_print (" Message element: %s from %s.\n",
                 gst_structure_get_name (s),
                 GST_OBJECT_NAME (message->src));
    }
    break;
}

case GST_MESSAGE_EOS:
{
    if (TRACE_MESSAGES)
    {
        g_print ("EOS from %s.\n", GST_OBJECT_NAME (message->src));
    }

    gstreamer_process_eos (user_data);

    break;
}

case GST_MESSAGE_ERROR:
{
    gchar *debug = NULL;
    GError *err = NULL;

    gst_message_parse_error (message, &err, &debug);
    g_print ("Error: %s.\n", err->message);
    g_error_free (err);
}

```

```

    if (debug)
    {
        g_print (" Debug details: %s.\n", debug);
        g_free (debug);
    }
    g_application_quit (user_data);
    break;
}

case GST_MESSAGE_STATE_CHANGED:
{
    GstState old_state, new_state, pending_state;

    gst_message_parse_state_changed (message, &old_state, &new_state,
                                     &pending_state);

    if (TRACE_MESSAGES)
    {
        g_print ("Element %s has changed state from %s to %s, "
                 "pending %s.\n", GST_OBJECT_NAME (message->src),
                 gst_element_state_get_name (old_state),
                 gst_element_state_get_name (new_state),
                 gst_element_state_get_name (pending_state));
    }
    break;
}

case GST_MESSAGE_RESET_TIME:
{
    guint64 running_time;
    const gchar *source;

    gst_message_parse_reset_time (message, &running_time);
    source = GST_OBJECT_NAME (message->src);
    if (TRACE_MESSAGES)
    {
        g_print ("Reset time to %ld by %s.\n", running_time, source);
    }
    break;
}

case GST_MESSAGE_STREAM_STATUS:
{
    GstStreamStatusType status_type;
    GstElement *owner;
    gchar *status_text;

```

```

gst_message_parse_stream_status (message, &status_type, &owner);
switch (status_type)
{
    case GST_STREAM_STATUS_TYPE_CREATE:
        status_text = (gchar *) "create";
        break;
    case GST_STREAM_STATUS_TYPE_ENTER:
        status_text = (gchar *) "enter";
        break;
    case GST_STREAM_STATUS_TYPE_LEAVE:
        status_text = (gchar *) "leave";
        break;
    case GST_STREAM_STATUS_TYPE_DESTROY:
        status_text = (gchar *) "destroy";
        break;
    case GST_STREAM_STATUS_TYPE_START:
        status_text = (gchar *) "start";
        break;
    case GST_STREAM_STATUS_TYPE_PAUSE:
        status_text = (gchar *) "pause";
        break;
    case GST_STREAM_STATUS_TYPE_STOP:
        status_text = (gchar *) "stop";
        break;
    default:
        status_text = (gchar *) "unknown";
        break;
}
if (TRACE_MESSAGES)
{
    g_print ("Stream status of %s from %s.\n", status_text,
            GST_OBJECT_NAME (owner));
}
break;
}

case GST_MESSAGE_ASYNC_DONE:
{
    if (TRACE_MESSAGES)
    {
        g_print ("Async-done from %s.\n", GST_OBJECT_NAME (message->src));
    }

    /* The pipeline has completed an asynchronous operation. */
    gstreamer_async_done (user_data);
}

```

```
        break;
    }

default:
{
    if (TRACE_MESSAGES)
    {
        g_print ("Message: %s from %s.\n",
            gst_message_type_get_name (GST_MESSAGE_TYPE (message)),
            GST_OBJECT_NAME (message->src));
    }
    break;
}

}

/* We handled the messages we wanted, and ignored the ones we didn't want,
 * so the core can unref the message for us. */
return TRUE;
}
```

17 message_subroutines.h

```
/*
 * message_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gst/gst.h>

/* Subroutines declared in message_handler.c */
gboolean message_handler (GstBus * bus_element, GstMessage * message,
                          gpointer user_data);
```

18 network_subroutines.c

```

/*
 * network_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gtk/gtk.h>
#include <gio/gio.h>
#include "sound_effects_player.h"
#include "parse_net_subroutines.h"
#include "network_subroutines.h"

/* The persistent data used by the network subroutines. */

struct network_info
{
    gchar *network_buffer;
    gint port_number;
    GSource *source_IPv4, *source_IPv6;
    GSocket *socket_IPv4, *socket_IPv6;
};

/* Subroutines to handle network messages */

/* Receive incoming data. This is called from the main loop whenever
 * there is data or a disconnect on a port. */
static gboolean
receive_data_callback (GSocket * socket, GIOCondition condition,
                      gpointer user_data)
{
    struct network_info *network_data;

```

```

gchar *network_buffer;
GError *error = NULL;
gssize nread;

/* Find the network buffer */
network_data = sep_get_network_data ((GApplication *) user_data);
network_buffer = network_data->network_buffer;

/* If we have data, process it. */
if ((condition & G_IO_IN) != 0)
{
    nread =
        g_socket_receive (socket, network_buffer, network_buffer_size, NULL,
                           &error);

    if (error != NULL)
    {
        g_error (error->message);
        return G_SOURCE_REMOVE;
    }
    if (nread != 0)
    {
        network_buffer[nread] = '\0';
        /* Data may be received in arbitrary-sized chunks.
         * Processing a chunk might range from just adding it to a buffer to
         * executing several commands that arrived all at once. */
        parse_net_text (network_buffer, user_data);
    }
}

/* If we have received the hangup condition, stop listening for data. */
if ((condition & G_IO_HUP) != 0)
    return G_SOURCE_REMOVE;

/* Otherwise, continue listening. */
return G_SOURCE_CONTINUE;
}

/* Initialize the network subroutines. We start to listen for messages.
 * The return value is the persistent data. Send text messages for testing
 * using ncat: nc -u localhost 1500. */
void *
network_init (GApplication * app)
{
    GError *error = NULL;

```



```

GSocket *socket_IPv4, *socket_IPv6;
GInetAddress *inet_IPv4_address, *inet_IPv6_address;
GSocketAddress *socket_IPv4_address, *socket_IPv6_address;
GSource *source_IPv4, *source_IPv6;
gchar *network_buffer;
struct network_info *network_data;

/* Allocate the persistent information. */
network_data = g_malloc (sizeof (struct network_info));

/* Allocate the network buffer. */
network_buffer = g_malloc0 (network_buffer_size);
network_data->network_buffer = network_buffer;

/* Set the default port. */
network_data->port_number = 1500;

/* Create a socket to listen for UDP messages on IPv6 and, if necessary,
 * another to listen for UDP messages on IPv4. */

socket_IPv6 =
    g_socket_new (G_SOCKET_FAMILY_IPV6, G_SOCKET_TYPE_DATAGRAM,
                  G_SOCKET_PROTOCOL_UDP, &error);
if (error != NULL)
{
    g_error (error->message);
    return NULL;
}

inet_IPv6_address = g_inet_address_new_any (G_SOCKET_FAMILY_IPV6);
socket_IPv6_address =
    g_inet_socket_address_new (inet_IPv6_address, network_data->port_number);
g_socket_bind (socket_IPv6, socket_IPv6_address, FALSE, &error);
if (error != NULL)
{
    g_error (error->message);
    return NULL;
}

source_IPv6 =
    g_socket_create_source (socket_IPv6, G_IO_IN | G_IO_HUP, NULL);
g_source_set_callback (source_IPv6, (GSourceFunc) receive_data_callback,
                       app, NULL);
g_source_attach (source_IPv6, NULL);
network_data->source_IPv6 = source_IPv6;
network_data->socket_IPv6 = socket_IPv6;

```

```

    if (g_socket_speaks_ipv4 (socket_IPv6))
    {
        network_data->source_IPv4 = NULL;
        network_data->socket_IPv4 = NULL;
        return network_data;
    }

    /* The IPv6 socket we just created doesn't speak IPv4, so create
       * a socket that does. */

    socket_IPv4 =
        g_socket_new (G_SOCKET_FAMILY_IPV4, G_SOCKET_TYPE_DATAGRAM,
                     G_SOCKET_PROTOCOL_UDP, &error);
    if (error != NULL)
    {
        g_error (error->message);
        return NULL;
    }
    inet_IPv4_address = g_inet_address_new_any (G_SOCKET_FAMILY_IPV4);
    socket_IPv4_address =
        g_inet_socket_address_new (inet_IPv4_address, network_data->port_number);
    g_socket_bind (socket_IPv4, socket_IPv4_address, FALSE, &error);
    if (error != NULL)
    {
        g_error (error->message);
        return NULL;
    }
    source_IPv4 =
        g_socket_create_source (socket_IPv4, G_IO_IN | G_IO_HUP, NULL);
    g_source_set_callback (source_IPv4, (GSourceFunc) receive_data_callback,
                          app, NULL);
    g_source_attach (source_IPv4, NULL);
    network_data->source_IPv4 = source_IPv4;
    network_data->socket_IPv4 = socket_IPv4;

    return network_data;
}

/* Set the network port number. */
void
network_set_port (int port_number, GApplication * app)
{
    GError *error = NULL;
    GSocket *socket_IPv4, *socket_IPv6;
    GInetAddress *inet_IPv4_address, *inet_IPv6_address;
    GSocketAddress *socket_IPv4_address, *socket_IPv6_address;

```

```

GSource *source_IPv4, *source_IPv6;
struct network_info *network_data;

network_data = sep_get_network_data (app);

network_data->port_number = port_number;

/* Stop network processing on the old port. */
if (network_data->source_IPv4 != NULL)
{
    g_socket_close (network_data->socket_IPv4, &error);
    if (error != NULL)
    {
        g_error (error->message);
    }
    g_object_unref (network_data->socket_IPv4);
    network_data->socket_IPv4 = NULL;

    g_source_destroy (network_data->source_IPv4);
    g_source_unref (network_data->source_IPv4);
    network_data->source_IPv4 = NULL;
}
if (network_data->source_IPv6 != NULL)
{
    g_socket_close (network_data->socket_IPv6, &error);
    if (error != NULL)
    {
        g_error (error->message);
    }
    g_object_unref (network_data->socket_IPv6);
    network_data->socket_IPv6 = NULL;

    g_source_destroy (network_data->source_IPv6);
    g_source_unref (network_data->source_IPv6);
    network_data->source_IPv6 = NULL;
}

/* Create a socket to listen for UDP messages on IPv6 and, if necessary,
* another to listen for UDP messages on IPv4. */

socket_IPv6 =
    g_socket_new (G_SOCKET_FAMILY_IPV6, G_SOCKET_TYPE_DATAGRAM,
                  G_SOCKET_PROTOCOL_UDP, &error);
if (error != NULL)
{
    g_error (error->message);
}

```

```

    return;
}

inet_IPv6_address = g_inet_address_new_any (G_SOCKET_FAMILY_IPV6);
socket_IPv6_address =
    g_inet_socket_address_new (inet_IPv6_address, network_data->port_number);
g_socket_bind (socket_IPv6, socket_IPv6_address, FALSE, &error);
if (error != NULL)
{
    g_error (error->message);
    return;
}
source_IPv6 =
    g_socket_create_source (socket_IPv6, G_IO_IN | G_IO_HUP, NULL);
g_source_set_callback (source_IPv6, (GSourceFunc) receive_data_callback,
    app, NULL);
g_source_attach (source_IPv6, NULL);
network_data->source_IPv6 = source_IPv6;
network_data->socket_IPv6 = socket_IPv6;

if (g_socket_speaks_ipv4 (socket_IPv6))
{
    network_data->source_IPv4 = NULL;
    return;
}

/* The IPv6 socket we just created doesn't speak IPv4, so create
 * a socket that does. */

socket_IPv4 =
    g_socket_new (G_SOCKET_FAMILY_IPV4, G_SOCKET_TYPE_DATAGRAM,
        G_SOCKET_PROTOCOL_UDP, &error);
if (error != NULL)
{
    g_error (error->message);
    return;
}
inet_IPv4_address = g_inet_address_new_any (G_SOCKET_FAMILY_IPV4);
socket_IPv4_address =
    g_inet_socket_address_new (inet_IPv4_address, network_data->port_number);
g_socket_bind (socket_IPv4, socket_IPv4_address, FALSE, &error);
if (error != NULL)
{
    g_error (error->message);
    return;
}

```

```
source_IPv4 =
    g_socket_create_source (socket_IPv4, G_IO_IN | G_IO_HUP, NULL);
g_source_set_callback (source_IPv4, (GSourceFunc) receive_data_callback,
                        app, NULL);
g_source_attach (source_IPv4, NULL);
network_data->source_IPv4 = source_IPv4;
network_data->socket_IPv6 = socket_IPv6;

return;
}

/* Find the network port number. */
gint
network_get_port (GApplication * app)
{
    struct network_info *network_data;
    gint port_number;

    network_data = sep_get_network_data (app);
    port_number = network_data->port_number;
    return (port_number);
}
```

19 network_subroutines.h

```

/*
 * network_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gst/gst.h>
#include <gio/gio.h>

#define network_buffer_size 8000

/* Subroutines defined in network_subroutines.c */

/* Initialize. */
void *network_init (GApplication * app);

/* Set the port number. */
void network_set_port (int port_number, GApplication * app);

/* Get the port number. */
gint network_get_port (GApplication * app);

```

20 parse_net_subroutines.c

```

/*
 * parse_net_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdlib.h>
#include <string.h>
#include "parse_net_subroutines.h"
#include "sound_effects_player.h"
#include "sound_subroutines.h"
#include "sequence_subroutines.h"

/* These subroutines are used to process network messages.
 * Each message consists of a keyword followed by a value. Upon receiving
 * a command we perform the action specified by the keyword.
 */

/* The persistent data used by the parser. It is allocated when the
 * parser is initialized, and deallocated when the program terminates.
 * It is accessible from the application.
 */

struct parse_net_info
{
    GHashTable *hash_table;
    gchar *message_buffer;
};

/* The keyword hash table. */
enum keyword_codes

```

```

{ keyword_start = 1, keyword_stop, keyword_quit, keyword_cue };

static enum keyword_codes keyword_values[] =
{ keyword_start, keyword_stop, keyword_quit, keyword_cue };

struct keyword_value_pairs
{
    gpointer key;
    gpointer val;
};

static struct keyword_value_pairs keywords_with_values[] = {
    {"start", &keyword_values[0]},
    {"stop", &keyword_values[1]},
    {"quit", &keyword_values[2]},
    {"/cue", &keyword_values[3]}
};

/* Initialize the network messages parser */

void *
parse_net_init (GApplication * app)
{
    struct parse_net_info *parse_net_data;
    int i;

    /* Allocate the persistent data used by the parser. */
    parse_net_data = g_malloc (sizeof (struct parse_net_info));

    /* Allocate the hash table which holds the keywords. */
    parse_net_data->hash_table = g_hash_table_new (g_str_hash, g_str_equal);

    /* Populate the hash table. */
    for (i = 0;
         i < sizeof (keywords_with_values) / sizeof (keywords_with_values[0]);
         i++)
    {
        g_hash_table_insert (parse_net_data->hash_table,
                             keywords_with_values[i].key,
                             keywords_with_values[i].val);
    }

    /* The message buffer starts out empty. */
    parse_net_data->message_buffer = NULL;

    return parse_net_data;
}

```



```

}

/* Receive a datagram from the network. Parse and execute the command.
 */
void
parse_net_text (gchar * text, GApplication * app)
{
    struct parse_net_info *parse_net_data;
    int command_length;
    int kl; /* keyword length */
    gchar *keyword_string;
    gpointer *p;
    enum keyword_codes keyword_value;
    gchar *extra_text;
    long int cluster_no;

    parse_net_data = sep_get_parse_net_data (app);

    command_length = strlen (text);
    /* Isolate the keyword that starts the command. The keyword will be
     * terminated by white space or the end of the string. */
    for (kl = 0; kl < command_length; kl++)
        if (g_ascii_isspace (text[kl]))
            break;

    keyword_string = g_strndup (text, kl);
    /* If there is any text after the keyword, it is probably a parameter
     * to the command. Isolate it, also. */
    if (kl + 1 < command_length)
    {
        extra_text = g_strdup (text + kl + 1);
    }
    else
    {
        extra_text = NULL;
    }

    /* Find the keyword in the hash table. */
    p = g_hash_table_lookup (parse_net_data->hash_table, keyword_string);
    if (p == NULL)
    {
        g_print ("Unknown command\n");
    }
    else
    {
        keyword_value = (enum keyword_codes) *p;
    }
}

```

```
switch (keyword_value)
{
case keyword_start:
    /* For the Start command, the operand is the
     * cluster number. */
    cluster_no = strtol (extra_text, NULL, 0);
    sequence_cluster_start (cluster_no, app);
    break;

case keyword_stop:
    /* Likewise for the Stop command. */
    cluster_no = strtol (extra_text, NULL, 0);
    sequence_cluster_stop (cluster_no, app);
    break;

case keyword_quit:
    /* The Quit command takes no arguments. */
    g_application_quit (app);
    break;

case keyword_cue:
    /* The cue command is treated as the
     * MIDI Show Control command Go. */
    sequence_MIDI_show_control_go (extra_text, app);
    break;

default:
    g_print ("unknown command\n");
}

g_free (keyword_string);
g_free (extra_text);
}

return;
}
```

21 parse_net_subroutines.h

```
/*
 * parse_net_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gtk/gtk.h>

/* Subroutines defined in parse_net_subroutines.c */

/* Initialize the parser. */
void *parse_net_init (GApplication * app);

/* Accept text, divide or accumulate it into commands,
 * and execute those commands. */
void parse_net_text (gchar * text, GApplication * app);
```

22 parse_xml_subroutines.c

```

/*
 * parse_xml_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdlib.h>
#include <gtk/gtk.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>
#include "parse_xml_subroutines.h"
#include "network_subroutines.h"
#include "sound_effects_player.h"
#include "sound_structure.h"
#include "sound_subroutines.h"
#include "sequence_structure.h"
#include "sequence_subroutines.h"

/* Dig through a sounds xml file, or the sounds content of an equipment
 * or project xml file, looking for the individual sounds. Construct the
 * sound effect player's internal data structure for each sound. */
static void
parse_sounds_info (xmlDocPtr sounds_file, gchar * sounds_file_name,
                  xmlNodePtr sounds_loc, GApplication * app)
{
    const xmlChar *name;
    xmlChar *name_data;
    gchar *file_dirname, *absolute_file_name;
    gdouble double_data;
    gint64 long_data;
    xmlNodePtr sound_loc;

```

```

struct sound_info *sound_data;

file_dirname = NULL;
absolute_file_name = NULL;
name_data = NULL;
/* We start at the children of a "sounds" section. Each child should
 * be a "version" or "sound" section. */
while (sounds_loc != NULL)
{
    name = sounds_loc->name;
    if (xmlStrEqual (name, (const xmlChar *) "version"))
    {
        name_data =
            xmlNodeListGetString (sounds_file, sounds_loc->xmlChildrenNode,
                                  1);
        if ((!g_str_has_prefix ((gchar *) name_data, (gchar *) "1.")))
        {
            g_printerr ("Version number of sounds is %s, "
                        "should start with 1.\n", name_data);
            return;
        }
    }
    if (xmlStrEqual (name, (const xmlChar *) "sound"))
    {
        /* This is a sound. Copy its information. */
        sound_loc = sounds_loc->xmlChildrenNode;
        /* Allocate a structure to hold sound information. */
        sound_data = g_malloc (sizeof (struct sound_info));
        /* Set the fields to their default values. If a field does not
 * appear in the XML file, it will retain its default value.
 * This lets us add new fields without invalidating old XML files.
 */
        sound_data->name = NULL;
        sound_data->disabled = FALSE;
        sound_data->wav_file_name = NULL;
        sound_data->wav_file_name_full = NULL;
        sound_data->attack_duration_time = 0;
        sound_data->attack_level = 1.0;
        sound_data->decay_duration_time = 0;
        sound_data->sustain_level = 1.0;
        sound_data->release_start_time = 0;
        sound_data->release_duration_time = 0;
        sound_data->release_duration_infinite = FALSE;
        sound_data->loop_from_time = 0;
        sound_data->loop_to_time = 0;
        sound_data->loop_limit = 0;
    }
}

```

```

sound_data->max_duration_time = 0;
sound_data->start_time = 0;
sound_data->designer_volume_level = 1.0;
sound_data->designer_pan = 0.0;
sound_data->MIDI_program_number = 0;
sound_data->MIDI_program_number_specified = FALSE;
sound_data->MIDI_note_number = 0;
sound_data->MIDI_note_number_specified = FALSE;
sound_data->OSC_name = NULL;
sound_data->OSC_name_specified = FALSE;
sound_data->function_key = NULL;
sound_data->function_key_specified = FALSE;
sound_data->omit_panning = FALSE;

/* These fields will be filled at run time. */
sound_data->sound_control = NULL;
sound_data->cluster_widget = NULL;
sound_data->cluster_number = 0;
sound_data->running = FALSE;
sound_data->release_sent = FALSE;
sound_data->release_has_started = FALSE;

/* Collect information from the XML file. */
while (sound_loc != NULL)
{
    name = sound_loc->name;
    if (xmlStrEqual (name, (const xmlChar *) "name"))
    {
        /* This is the name of the sound. It is mandatory. */
        name_data =
            xmlNodeListGetString (sounds_file,
                                   sound_loc->xmlChildrenNode, 1);
        sound_data->name = g_strdup ((gchar *) name_data);
        xmlFree (name_data);
        name_data = NULL;
    }
    if (xmlStrEqual (name, (const xmlChar *) "wav_file_name"))
    {
        /* The name of the WAV file from which we take the
         * waveform. */
        name_data =
            xmlNodeListGetString (sounds_file,
                                   sound_loc->xmlChildrenNode, 1);
        if (name_data != NULL)
        {
            sound_data->wav_file_name =

```

```

    g_strdup ((gchar *) name_data);
xmlFree (name_data);
name_data = NULL;

/* If the file name does not have an absolute path,
 * prepend the path of the sounds, equipment or project
 * file. This allows wave files to be copied along with
 * the files that refer to them. */
if (g_path_is_absolute (sound_data->wav_file_name))
{
    g_free (absolute_file_name);
    absolute_file_name =
        g_strdup (sound_data->wav_file_name);
}
else
{
    g_free (file_dirname);
    file_dirname =
        g_path_get_dirname (sounds_file_name);
    g_free (absolute_file_name);
    absolute_file_name =
        g_build_filename (file_dirname,
                          sound_data->wav_file_name,
                          NULL);
    g_free (file_dirname);
    file_dirname = NULL;
}
sound_data->wav_file_name_full = absolute_file_name;
if (!g_file_test
    (absolute_file_name, G_FILE_TEST_EXISTS))
{
    g_printerr ("File %s does not exist.\n",
                absolute_file_name);
    sound_data->disabled = TRUE;
}
absolute_file_name = NULL;
}
}
if (xmlStrEqual
    (name, (const xmlChar *) "attack_duration_time"))
{
    /* The time required to ramp up the sound when it starts. */
    name_data =
        xmlNodeListGetString (sounds_file,
                              sound_loc->xmlChildrenNode, 1);
    double_data = g_ascii_strtod ((gchar *) name_data, NULL);
}

```

```

        xmlFree (name_data);
        name_data = NULL;
        sound_data->attack_duration_time = double_data * 1E9;
    }
    if (xmlStrEqual (name, (const xmlChar *) "attack_level"))
    {
        /* The level we ramp up to. */
        name_data =
            xmlNodeListGetString (sounds_file,
                                   sound_loc->xmlChildrenNode, 1);
        if (name_data != NULL)
        {
            double_data =
                g_ascii_strtod ((gchar *) name_data, NULL);
            xmlFree (name_data);
            name_data = NULL;
            sound_data->attack_level = double_data;
        }
    }
    if (xmlStrEqual (name, (const xmlChar *) "decay_duration_time"))
    {
        /* Following the attack, the time to decrease the volume
           * to the sustain level. */
        name_data =
            xmlNodeListGetString (sounds_file,
                                   sound_loc->xmlChildrenNode, 1);
        if (name_data != NULL)
        {
            double_data =
                g_ascii_strtod ((gchar *) name_data, NULL);
            xmlFree (name_data);
            name_data = NULL;
            sound_data->decay_duration_time = double_data * 1E9;
        }
    }
    if (xmlStrEqual (name, (const xmlChar *) "sustain_level"))
    {
        /* The volume to reach at the end of the decay. */
        name_data =
            xmlNodeListGetString (sounds_file,
                                   sound_loc->xmlChildrenNode, 1);
        if (name_data != NULL)
        {
            double_data =
                g_ascii_strtod ((gchar *) name_data, NULL);
            xmlFree (name_data);

```



```

        name_data = NULL;
        sound_data->sustain_level = double_data;
    }
}
if (xmlStrEqual (name, (const xmlChar *) "release_start_time"))
{
    /* When to start the release process.  If this value is
     * zero, we start the release process only upon receipt
     * of an external signal, such as MIDI Note Off.  */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        name_data = NULL;
        sound_data->release_start_time = double_data * 1E9;
    }
}
if (xmlStrEqual
    (name, (const xmlChar *) "release_duration_time"))
{
    /* Once release has started, the time to ramp the volume
     * down to zero.  Note this value may be infinity, which
     * means that the volume does not decrease.  */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        if (xmlStrEqual (name_data, (const xmlChar *) "∞"))
        {
            sound_data->release_duration_infinite = TRUE;
            sound_data->release_duration_time = 0;
            xmlFree (name_data);
            name_data = NULL;
        }
        else
        {
            double_data =
                g_ascii_strtod ((gchar *) name_data, NULL);
            xmlFree (name_data);
            sound_data->release_duration_time =
                double_data * 1E9;
        }
    }
}

```

```

        sound_data->release_duration_infinite = FALSE;
        name_data = NULL;
    }
}
}
if (xmlStrEqual (name, (const xmlChar *) "loop_from_time"))
{
    /* If we are looping, the end time of the loop.
     * 0, the default, means do not loop. */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        sound_data->loop_from_time = double_data * 1E9;
        name_data = NULL;
    }
}
if (xmlStrEqual (name, (const xmlChar *) "loop_to_time"))
{
    /* If we are looping, the start time of the loop.
     * Each time through the loop we play from start time
     * to the end time of the loop. */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        sound_data->loop_to_time = double_data * 1E9;
        name_data = NULL;
    }
}
if (xmlStrEqual (name, (const xmlChar *) "loop_limit"))
{
    /* The number of times to pass through the loop. Zero
     * means loop until stopped by a Release message. */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)

```

```

    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sound_data->loop_limit = long_data;
        name_data = NULL;
    }
}
if (xmlStrEqual (name, (const xmlChar *) "max_duration_time"))
{
    /* The maximum amount of time to absorb from the WAV file */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        sound_data->max_duration_time = double_data * 1E9;
        name_data = NULL;
    }
}
if (xmlStrEqual (name, (const xmlChar *) "start_time"))
{
    /* The time within the WAV file to start this sound effect.
    */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        sound_data->start_time = double_data * 1E9;
        name_data = NULL;
    }
}

if (xmlStrEqual
    (name, (const xmlChar *) "designer_volume_level"))
{
    /* For this sound effect, decrease the volume from the WAV
    * file by this amount. */
    name_data =

```

```

        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        sound_data->designer_volume_level = double_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "designer_pan"))
{
    /* For monaural WAV files, the amount to send to the left and
     * right channels, expressed as -1 for left channel only,
     * 0 for both channels equally, and +1 for right channel
     * only. Other values between +1 and -1 also place the sound
     * in the stereo field. For stereo WAV files this operates
     * as a balance control. */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        sound_data->designer_pan = double_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "MIDI_program_number"))
{
    /* If we aren't using the internal sequencer, the MIDI
     * program number within which a MIDI Note On will activate
     * this sound effect. */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
    }
}

```

```

        sound_data->MIDI_program_number = long_data;
        sound_data->MIDI_program_number_specified = TRUE;
    }
    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "MIDI_note_number"))
{
    /* If we aren't using the internal sequencer, the MIDI Note
     * number that will activate this sound effect. */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sound_data->MIDI_note_number = long_data;
        sound_data->MIDI_note_number_specified = TRUE;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "OSC_name"))
{
    /* If we are not using the internal sequencer, this is the
     * name by which this sound effect is activated using
     * Open Sound Control. */
    name_data =
        xmlNodeListGetString (sounds_file,
                               sound_loc->xmlChildrenNode, 1);
    if (name_data != NULL)
    {
        sound_data->OSC_name = g_strdup ((gchar *) name_data);
        sound_data->OSC_name_specified = TRUE;
        xmlFree (name_data);
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "function_key"))
{
    /* If we are not using the internal sequencer, this is the
     * function key the operator presses to activate this
     * sound effect. */

```

```

        name_data =
            xmlNodeListGetString (sounds_file,
                                   sound_loc->xmlChildrenNode, 1);
        if (name_data != NULL)
        {
            sound_data->function_key =
                g_strdup ((gchar *) name_data);
            sound_data->function_key_specified = TRUE;
            xmlFree (name_data);
            name_data = NULL;
        }
    }

    if (xmlStrEqual (name, (const xmlChar *) "omit_panning"))
    {
        /* Do not allow the operator to pan this sound.
         * Needed for sounds with more than two channels, or
         * sounds with one channel that are directed at a
         * specific speaker. */
        name_data =
            xmlNodeListGetString (sounds_file,
                                   sound_loc->xmlChildrenNode, 1);
        if (xmlStrEqual (name_data, (const xmlChar *) "True"))
        {
            sound_data->omit_panning = TRUE;
        }
        xmlFree (name_data);
        name_data = NULL;
    }

    /* Ignore fields we don't recognize, so we can read future
     * XML files. */

    sound_loc = sound_loc->next;
}
/* Append this sound to the list of sounds. */
sound_append_sound (sound_data, app);
}
sounds_loc = sounds_loc->next;
}

return;
}

/* Dig through a sequence xml file, or the sequence content of an equipment
 * or project xml file, looking for the individual sequence items.

```

```

* Construct the sound effect player's internal data structure for each
* sequence item. */
static void
parse_sequence_info (xmlDocPtr sequence_file, gchar * sequence_file_name,
                    xmlNodePtr sequence_loc, GApplication * app)
{
    const xmlChar *name;
    xmlChar *name_data;
    gdouble double_data;
    gint64 long_data;
    xmlNodePtr sequence_item_loc;
    struct sequence_item_info *sequence_item_data;
    enum sequence_item_type item_type;

    name_data = NULL;
    /* We start at the children of a "sequence" section. Each child should
* be a "version" or "sequence_item" section. */
    while (sequence_loc != NULL)
    {
        name = sequence_loc->name;
        if (xmlStrEqual (name, (const xmlChar *) "version"))
        {
            name_data =
                xmlNodeListGetString (sequence_file,
                                      sequence_loc->xmlChildrenNode, 1);
            if ((!g_str_has_prefix ((gchar *) name_data, (gchar *) "1.")))
            {
                g_printerr ("Version number of sequence is %s, "
                           "should start with 1.\n", name_data);
                return;
            }
        }
        if (xmlStrEqual (name, (const xmlChar *) "sequence_item"))
        {
            /* This is a sequence item. Copy its information. */
            sequence_item_loc = sequence_loc->xmlChildrenNode;
            /* Allocate a structure to hold sequence item information. */
            sequence_item_data = g_malloc (sizeof (struct sequence_item_info));
            /* Set the fields to their default values. If a field does not
* appear in the XML file, it will retain its default value.
* This lets us add new fields without invalidating old XML files.
*/
            /* Fields used in the Start Sound sequence item. */
            sequence_item_data->name = NULL;
            sequence_item_data->type = unknown;
            sequence_item_data->sound_name = NULL;
        }
    }
}

```

```

sequence_item_data->tag = NULL;
sequence_item_data->use_external_velocity = 0;
sequence_item_data->volume = 1.0;
sequence_item_data->pan = 0.0;
sequence_item_data->program_number = 0;
sequence_item_data->bank_number = 0;
sequence_item_data->cluster_number = 0;
sequence_item_data->cluster_number_specified = FALSE;
sequence_item_data->next_completion = NULL;
sequence_item_data->next_termination = NULL;
sequence_item_data->next_starts = NULL;
sequence_item_data->next_release_started = NULL;
sequence_item_data->importance = 1;
sequence_item_data->Q_number = NULL;
sequence_item_data->text_to_display = NULL;

/* Fields used in the Stop sequence item but not mentioned above. */
sequence_item_data->next = NULL;

/* Fields used in the Wait sequence item but not mentioned above. */
sequence_item_data->time_to_wait = 0;

/* Fields used in the Offer Sound sequence item but not mentioned
 * above. */
sequence_item_data->next_to_start = NULL;
sequence_item_data->MIDI_program_number = 0;
sequence_item_data->MIDI_note_number = 0;
sequence_item_data->MIDI_note_number_specified = FALSE;
sequence_item_data->OSC_name = NULL;
sequence_item_data->macro_number = 0;
sequence_item_data->function_key = NULL;

/* Fields used in the Operator Wait sequence item but not mentioned
 * above. */
sequence_item_data->next_play = NULL;
sequence_item_data->omit_from_display = FALSE;

/* The Cease Offering Sounds and Start Sequence
 * sequence items uses only fields already mentioned. */

/* Collect information from the XML file. */
while (sequence_item_loc != NULL)
{
    name = sequence_item_loc->name;
    if (xmlStrEqual (name, (const xmlChar *) "name"))
    {

```



```

    /* This is the name of the sequence item. It is mandatory.
    */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->name = g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "type"))
{
    /* The type field specifies what this sequence item does. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);

    /* Convert the textual name in the XML file into an enum. */
    item_type = unknown;
    if (xmlStrEqual
        (name_data, (const xmlChar *) "start_sound"))
    {
        item_type = start_sound;
    }
    if (xmlStrEqual (name_data, (const xmlChar *) "stop"))
    {
        item_type = stop;
    }
    if (xmlStrEqual (name_data, (const xmlChar *) "wait"))
    {
        item_type = wait;
    }
    if (xmlStrEqual
        (name_data, (const xmlChar *) "offer_sound"))
    {
        item_type = offer_sound;
    }
    if (xmlStrEqual
        (name_data, (const xmlChar *) "cease_offering_sound"))
    {
        item_type = cease_offering_sound;
    }
    if (xmlStrEqual
        (name_data, (const xmlChar *) "operator_wait"))

```

```

    {
        item_type = operator_wait;
    }
    if (xmlStrEqual
        (name_data, (const xmlChar *) "start_sequence"))
    {
        item_type = start_sequence;
    }

    sequence_item_data->type = item_type;
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "sound_name"))
{
    /* For the Start Sound sequence item, the name of the sound
    * to start. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->sound_name =
        g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "tag"))
{
    /* The tag in Start Sound and Offer Sound is used by Stop
    * and Cease Offering Sound to name the sound or offering
    * to stop. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->tag = g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual
    (name, (const xmlChar *) "use_external_velocity"))
{
    /* For the Start Sound sequence item, if this is set to 1

```

```

    * we use the velocity of an external Note On message to
    * scale the volume of the sound. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sequence_item_data->use_external_velocity = long_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "volume"))
{
    /* For the Start Sound sequence item, scale the sound
    * designer's volume by this amount. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);
        xmlFree (name_data);
        sequence_item_data->volume = double_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "pan"))
{
    /* For the Start Sound sequence item, adjust the sound
    * designer's pan by this amount. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        double_data =
            g_ascii_strtod ((gchar *) name_data, NULL);

```

```

        xmlFree (name_data);
        sequence_item_data->pan = double_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "program_number"))
{
    /* For the Start Sound and Offer Sound sequence items,
     * the program number of
     * the cluster in which we display the sound. The program
     * number of the clusters being shown is controlled by
     * the sound effects operator. Unless there are a large
     * number of clusters being used, let this value default
     * to zero. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sequence_item_data->program_number = long_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "bank_number"))
{
    /* For the Start Sound and Offer Sound sequence items,
     * the bank number of the cluster in which we display
     * the sound. The bank
     * number of the clusters being shown is controlled by
     * the sound effects operator. Unless there are a large
     * number of clusters being used, let this value default
     * to zero. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);

```

```

        xmlFree (name_data);
        sequence_item_data->bank_number = long_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "cluster_number"))
{
    /* For the Start Sound and Offer Sound sequence items,
     * the cluster number in which we display the sound.
     * If none is specified,
     * one will be chosen at run time. Use this to place
     * a sound in the same cluster as a previous, related,
     * sound. For example, you might devote a particular
     * cluster to ringing a telephone even though it doesn't
     * ring throughout the show. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sequence_item_data->cluster_number = long_data;
        sequence_item_data->cluster_number_specified = TRUE;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "next_completion"))
{
    /* In the Start Sound sequence item, the next sequence item
     * to execute, when and if this sound completes normally.
     * In the Wait sequence item, the sequence item to execute
     * when the wait has completed. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->next_completion =
        g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

```

```

if (xmlStrEqual (name, (const xmlChar *) "next_termination"))
{
    /* The next sequence item to execute, when and if this
     * sound terminates due to an external event, such as
     * a MIDI Note Off or the sound effects operator pressing
     * his Stop key. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->next_termination =
        g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "next_starts"))
{
    /* The next sequence item to execute when this sound has
     * started. This can be used to fork the sequencer.
     */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->next_starts =
        g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual
    (name, (const xmlChar *) "next_release_started"))
{
    /* The next sequence item to execute when this sound has
     * reached the release stage of its amplitude envelope.
     * This can be used to fork the sequencer.
     */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->next_release_started =
        g_strdup ((gchar *) name_data);
    xmlFree (name_data);
}

```

```

    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "importance"))
{
    /* The importance of this sound to the sound effects
     * operator. The most important sound being played
     * is displayed on the console. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sequence_item_data->importance = long_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "Q_number"))
{
    /* The Q number of this sound, for MIDI Show Control. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->Q_number =
        g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "text_to_display"))
{
    /* The text to display to the sound effects operator when
     * this sound is playing. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->text_to_display =
        g_strdup ((gchar *) name_data);
    xmlFree (name_data);
}

```

```

    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "next"))
{
    /* In other than the Start Sound sequence item, the next
     * sequence item to execute when this one is done. The
     * Start Sound sequence item has three specialized next
     * sequence items, and so does not use this general one.
     */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    sequence_item_data->next = g_strdup ((gchar *) name_data);
    xmlFree (name_data);
    name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "time_to_wait"))
{
    /* In the Wait sequence item, the length of time to wait,
     * in nanoseconds. */
    name_data =
        xmlNodeListGetString (sequence_file,
                               sequence_item_loc->xmlChildrenNode,
                               1);
    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sequence_item_data->time_to_wait = long_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "next_to_start"))
{
    /* In the Offer Sound sequence item, the sequence item
     * that is to be executed when the sound effects operator
     * presses the Start button on the specified cluster.
     * The sequence item can also be started remotely.
     * This sequence item, like Start Sound, can be used
     * to fork the sequencer. */
    name_data =

```



```

        xmlNodeListGetString (sequence_file,
                                sequence_item_loc->xmlChildrenNode,
                                1);
sequence_item_data->next_to_start =
    g_strdup ((gchar *) name_data);
xmlFree (name_data);
name_data = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "MIDI_program_number"))
{
    /* In the Offer Sound sequence item, the MIDI program number
     * of the MIDI Note On message that will trigger the
     * specified sequence item. */
    name_data =
        xmlNodeListGetString (sequence_file,
                                sequence_item_loc->xmlChildrenNode,
                                1);

    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sequence_item_data->MIDI_program_number = long_data;
        name_data = NULL;
    }
}

if (xmlStrEqual (name, (const xmlChar *) "MIDI_note_number"))
{
    /* In the Offer Sound sequence item, the MIDI note number
     * of the MIDI Note On message that will trigger the
     * specified sequence item. */
    name_data =
        xmlNodeListGetString (sequence_file,
                                sequence_item_loc->xmlChildrenNode,
                                1);

    if (name_data != NULL)
    {
        long_data =
            g_ascii_strtoll ((gchar *) name_data, NULL, 10);
        xmlFree (name_data);
        sequence_item_data->MIDI_note_number = long_data;
        sequence_item_data->MIDI_note_number_specified = TRUE;
        name_data = NULL;
    }
}

```

```

    }

    if (xmlStrEqual (name, (const xmlChar *) "OSC_name"))
    {
        /* In the Offer Sound sequence item, the Open Show Control
         * (OSC) name used to trigger the specified sequence item
         * remotely. */
        name_data =
            xmlNodeListGetString (sequence_file,
                                   sequence_item_loc->xmlChildrenNode,
                                   1);
        sequence_item_data->OSC_name =
            g_strdup ((gchar *) name_data);
        xmlFree (name_data);
        name_data = NULL;
    }

    if (xmlStrEqual (name, (const xmlChar *) "macro_number"))
    {
        /* In the Offer Sound sequence item, the macro number used
         * by the Fire command of MIDI Show Control to trigger
         * the specified sequence item remotely. */
        name_data =
            xmlNodeListGetString (sequence_file,
                                   sequence_item_loc->xmlChildrenNode,
                                   1);

        if (name_data != NULL)
        {
            long_data =
                g_ascii_strtoll ((gchar *) name_data, NULL, 10);
            xmlFree (name_data);
            sequence_item_data->macro_number = long_data;
            name_data = NULL;
        }
    }

    if (xmlStrEqual (name, (const xmlChar *) "function_key"))
    {
        /* In the Offer Sound and Operator Wait sequence items,
         * the function key used to trigger the specified sequence
         * item remotely. */
        name_data =
            xmlNodeListGetString (sequence_file,
                                   sequence_item_loc->xmlChildrenNode,
                                   1);
        sequence_item_data->function_key =

```

```

        g_strdup ((gchar *) name_data);
        xmlFree (name_data);
        name_data = NULL;
    }

    if (xmlStrEqual (name, (const xmlChar *) "next_play"))
    {
        /* In the Operator Wait sequence item, the sequence item
         * to execute when the operator presses the Play button. */
        name_data =
            xmlNodeListGetString (sequence_file,
                                   sequence_item_loc->xmlChildrenNode,
                                   1);
        sequence_item_data->next_play =
            g_strdup ((gchar *) name_data);
        xmlFree (name_data);
        name_data = NULL;
    }

    if (xmlStrEqual (name, (const xmlChar *) "omit_from_display"))
    {
        /* In the Operator Wait sequence item, do not display this
         * item to the operator. */
        name_data =
            xmlNodeListGetString (sequence_file,
                                   sequence_item_loc->xmlChildrenNode,
                                   1);
        if (xmlStrEqual (name_data, (const xmlChar *) "True"))
        {
            sequence_item_data->omit_from_display = TRUE;
        }
        xmlFree (name_data);
        name_data = NULL;
    }

    /* Ignore fields we don't recognize, so we can read future
     * XML files. */

    sequence_item_loc = sequence_item_loc->next;
}

/* Append this sequence item to the sequence. */
sequence_append_item (sequence_item_data, app);
}
sequence_loc = sequence_loc->next;
}

```

```

    return;
}

/* Dig through the sound_effects program section of an equipment file
 * to find the sound and sequence information. */
static void
parse_program_info (xmlDocPtr equipment_file, gchar * equipment_file_name,
                    xmlNodePtr program_loc, GApplication * app)
{
    xmlChar *key;
    const xmlChar *name;
    xmlChar *prop_name;
    gchar *file_name;
    gchar *file_dirname;
    gchar *absolute_file_name;
    gint64 port_number;
    xmlNodePtr sounds_loc, sequence_loc;
    xmlDocPtr sounds_file, sequence_file;
    const xmlChar *root_name;
    const xmlChar *sounds_name, *sequence_name;
    gboolean sounds_section_parsed, sequence_section_parsed;

    /* We start at the children of a "program" section which has the
     * name "sound_effects". */
    /* We are looking for sound and sequence sections. */

    file_name = NULL;
    file_dirname = NULL;
    absolute_file_name = NULL;
    prop_name = NULL;

    while (program_loc != NULL)
    {
        name = program_loc->name;
        if (xmlStrEqual (name, (const xmlChar *) "port"))
        {
            /* This is the "port" section within "program". */
            key =
                xmlNodeListGetString (equipment_file,
                                      program_loc->xmlChildrenNode, 1);
            port_number = g_ascii_strtoll ((gchar *) key, NULL, 10);
            /* Tell the network module the new port number. */
            network_set_port (port_number, app);
            xmlFree (key);
        }
    }
}

```

```

if (xmlStrEqual (name, (const xmlChar *) "sounds"))
{
    /* This is the "sounds" section within "program".
     * It will have a reference to a sounds XML file,
     * content or both. First process the referenced file. */
    xmlFree (prop_name);
    prop_name = xmlGetProp (program_loc, (const xmlChar *) "href");
    if (prop_name != NULL)
    {
        /* We have a file reference. */
        g_free (file_name);
        file_name = g_strdup ((gchar *) prop_name);
        xmlFree (prop_name);
        prop_name = NULL;
        /* If the file name does not have an absolute path,
         * prepend the path of the equipment or project file.
         * This allows equipment and project files to be
         * copied along with the files they reference. */
        if (g_path_is_absolute (file_name))
        {
            g_free (absolute_file_name);
            absolute_file_name = g_strdup (file_name);
        }
        else
        {
            g_free (file_dirname);
            file_dirname = g_path_get_dirname (equipment_file_name);
            absolute_file_name =
                g_build_filename (file_dirname, file_name, NULL);
            g_free (file_dirname);
            file_dirname = NULL;
        }
        g_free (file_name);
        file_name = NULL;

        /* Read the specified file as an XML file. */
        xmlLineNumbersDefault (1);
        xmlThrDefIndentTreeOutput (1);
        xmlKeepBlanksDefault (0);
        xmlThrDefTreeIndentString ("  ");
        sounds_file = xmlParseFile (absolute_file_name);
        if (sounds_file == NULL)
        {
            g_printerr ("Load of sound file %s failed.\n",
                absolute_file_name);
        }
    }
}

```

```

    g_free (absolute_file_name);
    absolute_file_name = NULL;
    return;
}

/* Make sure the sounds file is valid, then extract
 * data from it. */
sounds_loc = xmlDocGetRootElement (sounds_file);
if (sounds_loc == NULL)
{
    g_printerr ("Empty sound file: %s.\n", absolute_file_name);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
    xmlFree (sounds_file);
    return;
}
root_name = sounds_loc->name;
if (!xmlStrEqual (root_name, (const xmlChar *) "show_control"))
{
    g_printerr ("Not a show_control file: %s; is %s.\n",
                absolute_file_name, root_name);
    xmlFree (sounds_file);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
    return;
}

/* Within the top-level show_control structure
 * should be a sounds structure. If there isn't,
 * this isn't a sound file and must be rejected. */
sounds_loc = sounds_loc->xmlChildrenNode;
sounds_name = NULL;
sounds_section_parsed = FALSE;
while (sounds_loc != NULL)
{
    sounds_name = sounds_loc->name;
    if (xmlStrEqual (sounds_name, (const xmlChar *) "sounds"))
    {
        parse_sounds_info (sounds_file, absolute_file_name,
                           sounds_loc->xmlChildrenNode, app);
        sounds_section_parsed = TRUE;
    }
    sounds_loc = sounds_loc->next;
}
if (!sounds_section_parsed)
{
    g_printerr ("Not a sounds file: %s; is %s.\n",

```

```

        absolute_file_name, sounds_name);
    }
    xmlFree (sounds_file);
}
/* Now process the content of the sounds section. */
parse_sounds_info (equipment_file, equipment_file_name,
                  program_loc->xmlChildrenNode, app);
g_free (absolute_file_name);
absolute_file_name = NULL;
}

if (xmlStrEqual (name, (const xmlChar *) "sound_sequence"))
{
    /* This is the "sound_sequence" section within "program".
     * It will have a reference to a sound sequence XML file,
     * content or both. First process the referenced file. */
    xmlFree (prop_name);
    prop_name = xmlGetProp (program_loc, (const xmlChar *) "href");
    if (prop_name != NULL)
    {
        /* We have a file reference. */
        g_free (file_name);
        file_name = g_strdup ((gchar *) prop_name);
        xmlFree (prop_name);
        prop_name = NULL;

        /* If the file name does not have an absolute path,
         * prepend the path of the equipment or project file.
         * This allows equipment and project files to be
         * copied along with the files they reference. */
        if (g_path_is_absolute (file_name))
        {
            g_free (absolute_file_name);
            absolute_file_name = g_strdup (file_name);
        }
        else
        {
            g_free (file_dirname);
            file_dirname = g_path_get_dirname (equipment_file_name);
            absolute_file_name =
                g_build_filename (file_dirname, file_name, NULL);
            g_free (file_dirname);
            file_dirname = NULL;
        }
        g_free (file_name);
        file_name = NULL;
    }
}

```

```

/* Read the specified file as an XML file. */
xmlLineNumbersDefault (1);
xmlThrDefIndentTreeOutput (1);
xmlKeepBlanksDefault (0);
xmlThrDefTreeIndentString ("  ");
sequence_file = xmlParseFile (absolute_file_name);
if (sequence_file == NULL)
{
    g_printerr ("Load of sound sequence file %s failed.\n",
                absolute_file_name);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
    return;
}

/* Make sure the sound sequence file is valid, then extract
 * data from it. */
sequence_loc = xmlDocGetRootElement (sequence_file);
if (sequence_loc == NULL)
{
    g_printerr ("Empty sound sequence file: %s.\n",
                absolute_file_name);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
    xmlFree (sequence_file);
    return;
}
root_name = sequence_loc->name;
if (!xmlStrEqual (root_name, (const xmlChar *) "show_control"))
{
    g_printerr ("Not a show_control file: %s; is %s.\n",
                absolute_file_name, root_name);
    xmlFree (sequence_file);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
    return;
}

/* Within the top-level show_control structure
 * should be a sound sequence structure. If there isn't,
 * this isn't a sound sequence file and must be rejected. */
sequence_loc = sequence_loc->xmlChildrenNode;
sequence_name = NULL;
sequence_section_parsed = FALSE;
while (sequence_loc != NULL)
{

```



```

        sequence_name = sequence_loc->name;
        if (xmlStrEqual
            (sequence_name, (const xmlChar *) "sound_sequence"))
        {
            parse_sequence_info (sequence_file, absolute_file_name,
                                sequence_loc->xmlChildrenNode,
                                app);
            sequence_section_parsed = TRUE;
        }
        sequence_loc = sequence_loc->next;
    }
    if (!sequence_section_parsed)
    {
        g_printerr ("Not a sound sequence file: %s; is %s.\n",
                    absolute_file_name, sequence_name);
    }
    xmlFree (sequence_file);
}

/* Now process the content of the sound sequence section. */
parse_sequence_info (equipment_file, equipment_file_name,
                    program_loc->xmlChildrenNode, app);
g_free (absolute_file_name);
absolute_file_name = NULL;
}

program_loc = program_loc->next;
}

}

/* Dig through an equipment xml file, or the equipment section of a project
 * xml file, looking for the sound effect player's sounds and network port.
 * When we find the network port, tell the network module about it.
 * When we find sounds, parse the description. */
static void
parse_equipment_info (xmlDocPtr equipment_file, gchar * equipment_file_name,
                    xmlNodePtr equipment_loc, GApplication * app)
{
    xmlChar *key;
    const xmlChar *name;
    xmlNodePtr program_loc;
    xmlChar *program_id;

    /* We start at the children of an "equipment" section. */
    /* We are looking for version and program sections. */

    while (equipment_loc != NULL)

```

```

{
    name = equipment_loc->name;
    if (xmlStrEqual (name, (const xmlChar *) "version"))
    {
        /* This is the "version" section within "equipment". */
        key =
            xmlNodeListGetString (equipment_file,
                                   equipment_loc->xmlChildrenNode, 1);
        if ((!g_str_has_prefix ((gchar *) key, (gchar *) "1.")))
        {
            g_printerr ("Version number of equipment is %s, "
                        "should start with 1.\n", key);
            return;
        }
        xmlFree (key);
    }
    if (xmlStrEqual (name, (const xmlChar *) "program"))
    {
        /* This is a "program" section. We only care about the sound
        * effects program. */
        program_id = xmlGetProp (equipment_loc, (const xmlChar *) "id");
        if (xmlStrEqual (program_id, (const xmlChar *) "sound_effects"))
        {
            /* This is the section of the XML file that contains information
            * about the sound effects program. */
            program_loc = equipment_loc->xmlChildrenNode;
            parse_program_info (equipment_file, equipment_file_name,
                               program_loc, app);
        }
        xmlFree (program_id);
    }
    equipment_loc = equipment_loc->next;
}

return;
}

/* Dig through the project xml file looking for the equipment references.
* Parse each one, since the information we are looking for might be scattered
* among them. */
static void
parse_project_info (xmlDocPtr project_file, gchar * project_file_name,
                   xmlNodePtr current_loc, GApplication * app)
{
    xmlChar *key;
    const xmlChar *name;

```

```

xmlChar *prop_name;
gchar *file_name;
gchar *file_dirname;
gchar *absolute_file_name;
xmlNodePtr equipment_loc;
xmlDocPtr equipment_file;
gboolean found_equipment_section;
const xmlChar *root_name;
const xmlChar *equipment_name;
gboolean equipment_section_parsed;

/* We start at the children of the "project" section.
 * Important child sections for our purposes are "version" and "equipment".
 */
found_equipment_section = FALSE;
file_name = NULL;
file_dirname = NULL;
absolute_file_name = NULL;
prop_name = NULL;

while (current_loc != NULL)
{
    name = current_loc->name;
    if (xmlStrEqual (name, (const xmlChar *) "version"))
    {
        /* This is the "version" section within "project". We can only
         * interpret version 1 of the project section, so reject all other
         * versions. The value after the decimal point doesn't matter,
         * since 1.1, for example, will be a compatible extension of 1.0. */
        key =
            xmlNodeListGetString (project_file, current_loc->xmlChildrenNode,
                                  1);
        if ((!g_str_has_prefix ((gchar *) key, (gchar *) "1.")))
        {
            g_printerr ("Version number of project is %s, "
                        "should start with 1.\n", key);
            return;
        }
        xmlFree (key);
    }
    if (xmlStrEqual (name, (const xmlChar *) "equipment"))
    {
        /* This is an "equipment" section within "project".
         * It will have a reference to an equipment XML file,
         * content, or both. First process the referenced file. */
        found_equipment_section = TRUE;
    }
}

```

```

xmlFree (prop_name);
prop_name = xmlGetProp (current_loc, (const xmlChar *) "href");
if (prop_name != NULL)
{
    g_free (file_name);
    file_name = g_strdup ((gchar *) prop_name);
    xmlFree (prop_name);
    prop_name = NULL;

    /* If the file name specified does not have an absolute path,
     * prepend the path to the project file. This allows project
     * files to be copied along with the files they reference. */
    if (g_path_is_absolute (file_name))
    {
        g_free (absolute_file_name);
        absolute_file_name = g_strdup (file_name);
    }
    else
    {
        g_free (file_dirname);
        file_dirname = g_path_get_dirname (project_file_name);
        absolute_file_name =
            g_build_filename (file_dirname, file_name, NULL);
        g_free (file_dirname);
        file_dirname = NULL;
    }
    g_free (file_name);
    file_name = NULL;

    /* Read the specified file as an XML file. */
    xmlLineNumbersDefault (1);
    xmlThrDefIndentTreeOutput (1);
    xmlKeepBlanksDefault (0);
    xmlThrDefTreeIndentString ("  ");
    equipment_file = xmlParseFile (absolute_file_name);
    if (equipment_file == NULL)
    {
        g_printerr ("Load of equipment file %s failed.\n",
                    absolute_file_name);
        g_free (absolute_file_name);
        absolute_file_name = NULL;
        return;
    }

    /* Make sure the equipment file is valid, then extract data from
     * it. */

```

```

equipment_loc = xmlDocGetRootElement (equipment_file);
if (equipment_loc == NULL)
{
    g_printerr ("Empty equipment file: %s.\n",
                absolute_file_name);
    xmlFree (equipment_file);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
    return;
}
root_name = equipment_loc->name;
if (!xmlStrEqual (root_name, (const xmlChar *) "show_control"))
{
    g_printerr ("Not a show_control file: %s; is %s.\n",
                absolute_file_name, root_name);
    xmlFree (equipment_file);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
    return;
}

/* Within the top-level show_control structure should be an
 * equipment structure. If there isn't, this isn't an equipment
 * file and must be rejected. */
equipment_loc = equipment_loc->xmlChildrenNode;
equipment_name = NULL;
equipment_section_parsed = FALSE;
while (equipment_loc != NULL)
{
    equipment_name = equipment_loc->name;
    if (xmlStrEqual
        (equipment_name, (const xmlChar *) "equipment"))
    {
        parse_equipment_info (equipment_file,
                              absolute_file_name,
                              equipment_loc->xmlChildrenNode,
                              app);
        equipment_section_parsed = TRUE;
    }
    equipment_loc = equipment_loc->next;
}
if (!equipment_section_parsed)
{
    g_printerr ("Not an equipment file: %s; is %s.\n",
                absolute_file_name, equipment_name);
}

```

```

        xmlFree (equipment_file);
    }

    /* Now process the content of the equipment section. */
    parse_equipment_info (project_file, project_file_name,
                          current_loc->xmlChildrenNode, app);
    g_free (absolute_file_name);
    absolute_file_name = NULL;
}
current_loc = current_loc->next;
}
if (!found_equipment_section)
{
    g_printerr ("No equipment section in project file: %s.\n",
                absolute_file_name);
}
g_free (absolute_file_name);
absolute_file_name = NULL;

return;
}

/* Open a project file and read its contents. The file is assumed to be in
 * XML format. */
void
parse_xml_read_project_file (gchar * project_file_name, GApplication * app)
{
    xmlDocPtr project_file;
    xmlNodePtr current_loc;
    const xmlChar *name;
    gboolean project_section_parsed;

    /* Read the file as an XML file. */
    xmlLineNumbersDefault (1);
    xmlThrDefIndentTreeOutput (1);
    xmlKeepBlanksDefault (0);
    xmlThrDefTreeIndentString ("  ");
    project_file = xmlParseFile (project_file_name);
    if (project_file == NULL)
    {
        g_printerr ("Load of project file %s failed.\n", project_file_name);
        g_free (project_file_name);
        project_file_name = NULL;
        return;
    }
}

```

```

/* Remember the file name. */
sep_set_project_filename (project_file_name, app);

/* Remember the data from the XML file, in case we want to refer to it
 * later. */
sep_set_project_file (project_file, app);

/* Make sure the project file is valid, then extract data from it. */
current_loc = xmlDocGetRootElement (project_file);
if (current_loc == NULL)
{
    g_printerr ("Empty project file.\n");
    return;
}
name = current_loc->name;
if (!xmlStrEqual (name, (const xmlChar *) "show_control"))
{
    g_printerr ("Not a show_control file: %s.\n", current_loc->name);
    return;
}

/* Within the top-level show_control section should be a project
 * section. If there isn't, this isn't a project file and must
 * be rejected. If there is, process it. */
current_loc = current_loc->xmlChildrenNode;
name = NULL;
project_section_parsed = FALSE;
while (current_loc != NULL)
{
    name = current_loc->name;
    if (xmlStrEqual (name, (const xmlChar *) "project"))
    {
        parse_project_info (project_file, project_file_name,
                           current_loc->xmlChildrenNode, app);
        project_section_parsed = TRUE;
    }
    current_loc = current_loc->next;
}
if (!project_section_parsed)
{
    g_printerr ("Not a project file: %s.\n", name);
}

return;
}

```

```

/* Write the project information to an XML file. */
void
parse_xml_write_project_file (gchar * project_file_name, GApplication * app)
{
    xmlDocPtr project_file;
    xmlNodePtr current_loc;
    xmlNodePtr equipment_loc;
    xmlNodePtr project_loc;
    xmlNodePtr program_loc;
    const xmlChar *name = NULL;
    xmlChar *prop_name = NULL;
    gint port_number;
    gchar *port_number_text = NULL;
    gboolean port_number_found;
    gchar text_buffer[G_ASCII_DTOSTR_BUF_SIZE];

    /* Write the project data as an XML file. */
    project_file = sep_get_project_file (app);
    if (project_file == NULL)
    {
        /* We don't have a project file--create one. */
        xmlLineNumbersDefault (1);
        xmlThrDefIndentTreeOutput (1);
        xmlKeepBlanksDefault (0);
        xmlThrDefTreeIndentString ("  ");
        project_file =
            xmlParseDoc ((xmlChar *) "<?xml version=\"1.0\" "
                "encoding=\"utf-8\"?> <show_control> <project>"
                "<version>1.0</version>"
                "<equipment> <program id=\"sound_effects\">"
                "<port>1500</port> </program> </equipment>"
                "</project> </show_control>");
        sep_set_project_file (project_file, app);
    }

    /* The network port might have been changed using the preferences
       * dialogue. Make sure we write out the current value. */

    /* Find the network node, then set the value of the port node
       * within it. This only works if the node number is in the top-level
       * project file. */
    port_number_found = FALSE;
    current_loc = xmlDocGetRootElement (project_file);
    /* We know the root element is show_control, and there is only one,
       * so we don't have to check it or iterate over it. */
    current_loc = current_loc->xmlChildrenNode;

```



```

while (current_loc != NULL)
{
    name = current_loc->name;
    if (xmlStrEqual (name, (const xmlChar *) "project"))
    {
        project_loc = current_loc->xmlChildrenNode;
        while (project_loc != NULL)
        {
            name = project_loc->name;
            if (xmlStrEqual (name, (const xmlChar *) "equipment"))
            {
                equipment_loc = project_loc->xmlChildrenNode;
                while (equipment_loc != NULL)
                {
                    name = equipment_loc->name;
                    if (xmlStrEqual (name, (const xmlChar *) "program"))
                    {
                        prop_name =
                            xmlGetProp (equipment_loc,
                                         (const xmlChar *) "id");
                        if (xmlStrEqual
                            (prop_name, (const xmlChar *) "sound_effects"))
                        {
                            program_loc = equipment_loc->xmlChildrenNode;
                            while (program_loc != NULL)
                            {
                                name = program_loc->name;
                                if (xmlStrEqual
                                    (name, (const xmlChar *) "port"))
                                {
                                    /* This is the "port" node within
                                     * program sound_effects. */
                                    port_number = network_get_port (app);
                                    port_number_text =
                                        g_ascii_dtostr (text_buffer,
                                                         G_ASCII_DTOSTR_BUF_SIZE,
                                                         (1.0 * port_number));
                                    xmlNodeSetContent (program_loc,
                                                         (xmlChar *)
                                                         port_number_text);
                                    port_number_found = TRUE;
                                }
                            }
                            program_loc = program_loc->next;
                        }
                    }
                }
            }
        }
    }
}

```

```
        equipment_loc = equipment_loc->next;
    }
}
project_loc = project_loc->next;
}
}
current_loc = current_loc->next;
}

if (port_number_found)
{
    /* Write the file, with indentations to make it easier to edit
     * manually. */
    xmlSaveFormatFileEnc (project_file_name, project_file, "utf-8", 1);

    /* Remember the file name so we can use it as the default
     * next time. */
    sep_set_project_filename (project_file_name, app);
}
else
{
    g_printerr ("The project file is complex, and must be edited "
               "with an XML editor such as Emacs.\n");
}
g_free (prop_name);

return;
}
```

23 parse_xml_subroutines.h

```
/*
 * parse_xml_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gtk/gtk.h>
/* Subroutines defined in parse_xml_subroutines.c */

void
parse_xml_read_project_file (gchar *project_file_name, GApplication *app);

void
parse_xml_write_project_file (gchar *project_file_name, GApplication *app);
```

24 preferences.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated with glade 3.18.3 -->
<interface>
  <requires lib="gtk+" version="3.12"/>
  <object class="GtkDialog" id="dialog1">
    <property name="width_request">300</property>
    <property name="height_request">100</property>
    <property name="can_focus">False</property>
    <property name="border_width">5</property>
    <property name="title" translatable="yes">Preferences</property>
    <property name="default_width">300</property>
    <property name="default_height">100</property>
    <property name="destroy_with_parent">True</property>
    <property name="type_hint">dialog</property>
    <child internal-child="vbox">
      <object class="GtkBox" id="dialog-vbox1">
        <property name="can_focus">False</property>
        <property name="orientation">vertical</property>
        <property name="spacing">2</property>
        <child internal-child="action_area">
          <object class="GtkButtonBox" id="dialog-action_area1">
            <property name="can_focus">False</property>
            <property name="layout_style">end</property>
            <child>
              <object class="GtkButton" id="button1">
                <property name="label">gtk-clear</property>
                <property name="visible">True</property>
                <property name="can_focus">True</property>
                <property name="receives_default">True</property>
                <property name="use_stock">True</property>
              </object>
            <packing>
              <property name="expand">True</property>
              <property name="fill">True</property>
              <property name="position">0</property>
            </packing>
          </child>
          <child>
            <object class="GtkButton" id="button2">
              <property name="label">gtk-close</property>
              <property name="visible">True</property>
              <property name="can_focus">True</property>
              <property name="receives_default">True</property>
              <property name="use_stock">True</property>
            </object>
          </child>
        </child>
      </object>
    </child>
  </object>
</interface>

```

```

        <signal name="clicked" handler="menu_preferences_close_clicked"
→  object="dialog1" swapped="no"/>
    </object>
    <packing>
        <property name="expand">True</property>
        <property name="fill">True</property>
        <property name="position">1</property>
    </packing>
</child>
</object>
<packing>
    <property name="expand">False</property>
    <property name="fill">False</property>
    <property name="position">0</property>
</packing>
</child>
<child>
    <object class="GtkNotebook" id="notebook1">
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property name="scrollable">True</property>
        <property name="enable_popup">True</property>
        <child>
            <object class="GtkLabel" id="label4">
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="label" translatable="yes">no general properties
→  yet</property>
            </object>
        </child>
        <child type="tab">
            <object class="GtkLabel" id="label1">
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="label" translatable="yes">general</property>
            </object>
            <packing>
                <property name="tab_fill">False</property>
            </packing>
        </child>
    </child>
    <object class="GtkBox" id="box1">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <child>
            <object class="GtkLabel" id="label5">

```

```

        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes">port</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">0</property>
    </packing>
</child>
<child>
    <object class="GtkEntry" id="entry1">
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property name="width_chars">6</property>
        <property name="progress_pulse_step">0</property>
        <property name="input_purpose">digits</property>
        <signal name="activate" handler="menu_network_port_changed"
→   object="dialog1" swapped="no"/>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">1</property>
    </packing>
</child>
</object>
<packing>
    <property name="position">1</property>
</packing>
</child>
<child type="tab">
    <object class="GtkLabel" id="label2">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes">network</property>
    </object>
    <packing>
        <property name="position">1</property>
        <property name="tab_fill">False</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="label6">
        <property name="visible">True</property>
        <property name="can_focus">False</property>

```

```

        <property name="label" translatable="yes">future properties will
→ go here</property>
    </object>
    <packing>
        <property name="position">2</property>
    </packing>
</child>
<child type="tab">
    <object class="GtkLabel" id="label3">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes">future</property>
    </object>
    <packing>
        <property name="position">2</property>
        <property name="tab_fill">False</property>
    </packing>
</child>
</object>
<packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">1</property>
</packing>
</child>
</object>
</child>
</object>
</interface>

```

25 sequence__structure.h

```

/*
 * sequence_structure.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/* Only define the structure once per source file. */
#ifdef SEQUENCE_STRUCTURE_H
#define SEQUENCE_STRUCTURE_H

/* Define the structure which holds the definition of a sequence item.
 * This structure is shared between
 * parse_xml_subroutines and sequence_subroutines. */

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

/* There are several types of sequence item, as follows: */
enum sequence_item_type
{ unknown, start_sound, stop, wait, offer_sound, cease_offering_sound,
  operator_wait, start_sequence
};

/* The following structure is used for all sequence items. No item uses
 * all the fields. */
struct sequence_item_info
{
  gchar *name; /* name of the sequence item */
  enum sequence_item_type type; /* The type of sequence item */
  gchar *sound_name; /* The name of the sound to start */
  gchar *tag; /* The sound's or offering's tag, used to

```



```

    * stop the sound or offering. */
guint use_external_velocity; /* If 1, use the velocity of an external
    * Note On message to scale the volume. */
gdouble volume;             /* Scale the sound designer's velocity by
    * this much when starting the sound. */

gdouble pan;                /* Pan the sound by this much, in addition
    * to the specification in the sound,
    * when starting the sound. */

guint program_number;       /* The program number, bank number and */
guint bank_number;          /* cluster number are used to display */
guint cluster_number;       /* a sound for the operator to control. */
gboolean cluster_number_specified; /* If a cluster is not specified,
    * one as chosen at run time. */

gchar *next_completion;     /* Sequence item to execute on completion. */
gchar *next_termination;   /* sequence item to execute on termination. */
gchar *next_starts;        /* sequence item to execute when this sound
    * starts. */
gchar *next_release_started; /* sequence item to execute when this sound
    * starts the release stage of its envelope. */
guint importance;           /* importance of this sound,
    * for display purposes. */
gchar *Q_number;            /* The Q_number is used by MIDI Show
    * Control. */
gchar *text_to_display;     /* What to show the operator */
gchar *next;                /* The next sequence item to execute. */
guint64 time_to_wait;       /* Nanoseconds to wait in the Wait sequence
    * item. */
gchar *next_to_start;       /* The sequence item to execute when
    * the operator presses the cluster's Start
    * button. */
gchar *next_play;           /* The sequence item to execute when
    * the operator presses the Play button. */
gint MIDI_program_number;   /* The MIDI program number and */
gint MIDI_note_number;      /* note number which trigger this cluster */
gboolean MIDI_note_number_specified; /* They may be omitted. */
gchar *OSC_name;            /* The Open Show Control (OSC) name which
    * triggers this cluster */
guint macro_number;         /* Used by MIDI Show Control's Fire command
    * to trigger this cluster. */
gchar *function_key;        /* The function key which triggers this
    * cluster */
gboolean omit_from_display; /* Do not show this item to the operator. */
};

```

```
#endif /* ifndef SEQUENCE_STRUCTURE_H */  
/* End of file sequence_structure.h */
```

26 sequence__subroutines.c

```

/*
 * sequence_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdlib.h>
#include <gst/gst.h>
#include "sequence_subroutines.h"
#include "sequence_structure.h"
#include "sound_effects_player.h"
#include "display_subroutines.h"
#include "sound_structure.h"
#include "sound_subroutines.h"
#include "button_subroutines.h"
#include "timer_subroutines.h"

/* When debugging it can be useful to trace what is happening in the
 * internal sequencer. */
#define TRACE_SEQUENCER FALSE
#define TRACE_SEQUENCER_DISPLAY_MESSAGE FALSE
#define DO_OPERATOR_DISPLAY TRUE

/* the persistent data used by the internal sequencer */
struct sequence_info
{
    GList *item_list;           /* The sequence */
    gchar *next_item_name;     /* The name of the next sequence item
 * to be executed. */
    GList *running;            /* The list of Start Sound items
 * that are still attached to a cluster */

```

```

GList *offering;           /* The list of Offer Sound items
                           * that are still attached to a cluster */
struct remember_info *current_operator_wait; /* The Operator Wait sequence
                           * item that is currently
                           * displaying its text to the
                           * operator. */

GList *operator_waiting;   /* The Operator Wait sequence items that are
                           * waiting for their turn at the operator. */
GList *waiting;            /* The Wait sequence items that are still
                           * pending. */
gboolean message_displaying; /* TRUE if the sequencer is displaying a
                           * message to the operator. */
guint message_id;          /* The ID of the message being displayed by the
                           * sequencer. */
};

/* an entry on the running, offering or operator waiting lists */
struct remember_info
{
    guint cluster_number;
    struct sound_info *sound_effect;
    struct sequence_item_info *sequence_item;
    gboolean active;
    gboolean being_displayed;
    gboolean release_sent;
    gboolean release_seen;
    gboolean off_cluster;
};

/* Forward declarations, so I can call these subroutines before I define them.
*/

static struct sequence_item_info *find_item_by_name (gchar * item_name,
                                                    struct sequence_info
                                                    *sequence_data);

static void execute_items (struct sequence_info *sequence_data,
                          GApplication * app);

static void execute_item (struct sequence_item_info *the_item,
                          struct sequence_info *sequence_data,
                          GApplication * app);

static void execute_start_sound (struct sequence_item_info *the_item,
                                 struct sequence_info *sequence_data,
                                 GApplication * app);

```

```

static void execute_stop_sound (struct sequence_item_info *the_item,
                                struct sequence_info *sequence_data,
                                GApplication * app);

static void execute_wait (struct sequence_item_info *the_item,
                           struct sequence_info *sequence_data,
                           GApplication * app);

static void wait_completed (void *remember_data, GApplication * app);

static void execute_offer_sound (struct sequence_item_info *the_item,
                                  struct sequence_info *sequence_data,
                                  GApplication * app);

static void execute_cease_offering_sound (struct sequence_item_info *the_item,
                                           struct sequence_info *sequence_data,
                                           GApplication * app);

static void execute_operator_wait (struct sequence_item_info *the_item,
                                   struct sequence_info *sequence_data,
                                   GApplication * app);

static void update_operator_display (struct sequence_info *sequence_data,
                                    GApplication * app);
static void clock_tick (void *sequence_data, GApplication * app);
static void cancel_operator_display (struct remember_info *remember_data,
                                    struct sequence_info *sequence_data,
                                    GApplication * app);

/* Subroutines for handling sequence items. */

/* Initialize the internal sequencer. */
void *
sequence_init (GApplication * app)
{
    struct sequence_info *sequence_data;

    sequence_data = g_malloc (sizeof (struct sequence_info));
    sequence_data->item_list = NULL;
    sequence_data->offering = NULL;
    sequence_data->running = NULL;
    sequence_data->current_operator_wait = NULL;
    sequence_data->operator_waiting = NULL;
    sequence_data->waiting = NULL;
    sequence_data->message_displaying = FALSE;

```

```

    sequence_data->message_id = 0;
    return (sequence_data);
}

/* Append a sequence item to the sequence. */
void
sequence_append_item (struct sequence_item_info *item, GApplication * app)
{
    struct sequence_info *sequence_data;

    sequence_data = sep_get_sequence_data (app);
    sequence_data->item_list = g_list_append (sequence_data->item_list, item);
    return;
}

/* Start running the sequencer. */
void
sequence_start (GApplication * app)
{
    struct sequence_info *sequence_data;
    GList *item_list;
    struct sequence_item_info *item, *start_item;

    sequence_data = sep_get_sequence_data (app);

    /* Find the Sequence Start item in the sequence. */
    start_item = NULL;
    for (item_list = sequence_data->item_list; item_list != NULL;
        item_list = item_list->next)
    {
        item = item_list->data;
        if (item->type == start_sequence)
        {
            start_item = item;
            break;
        }
    }
    if (start_item == NULL)
    {
        display_show_message ("No Sequence Start item.", app);
        return;
    }

    /* We have a sequence which contains a Start sequence item. Proceed to
     * the specified next item. */
    if (start_item->next == NULL)

```

```

    {
        display_show_message ("Sequence Start has no next item.", app);
        return;
    }

    sequence_data->next_item_name = start_item->next;

    /* Run sequence items starting at next_item_name. */
    execute_items (sequence_data, app);

    return;
}

/* Execute the named next item, and continue execution until we must
 * wait for something or we run out of items to execute. */
static void
execute_items (struct sequence_info *sequence_data, GApplication * app)
{
    struct sequence_item_info *next_item;

    while (sequence_data->next_item_name != NULL)
    {
        next_item =
            find_item_by_name (sequence_data->next_item_name, sequence_data);

        if (next_item == NULL)
        {
            display_show_message ("Next item not found.", app);
            break;
        }

        sequence_data->next_item_name = NULL;
        execute_item (next_item, sequence_data, app);
    }

    /* There are no more items to execute. If nothing is waiting to
     * execute later, we are done. */
    if ((sequence_data->offering == NULL) && (sequence_data->running == NULL)
        && (sequence_data->current_operator_wait == NULL)
        && (sequence_data->operator_waiting == NULL)
        && (sequence_data->waiting == NULL))
    {
        /* Signal the application to quit. This first shuts down the
         * Gstreamer pipeline, then exits. */
        g_application_quit (app);
    }
}

```

```

    return;
}

/* Find a sequence item, given its name.  If the item is not found,
 * returns NULL.  */
struct sequence_item_info *
find_item_by_name (gchar * item_name, struct sequence_info *sequence_data)
{
    struct sequence_item_info *item, *found_item;
    GList *item_list;

    if (TRACE_SEQUENCER)
    {
        g_print ("Searching for item %s.\n", item_name);
    }

    found_item = NULL;
    for (item_list = sequence_data->item_list; item_list != NULL;
        item_list = item_list->next)
    {
        item = item_list->data;

        if (g_strcmp0 (item_name, item->name) == 0)
        {
            found_item = item;
            break;
        }
    }

    return (found_item);
}

/* Execute a sequence item.  */
void
execute_item (struct sequence_item_info *the_item,
              struct sequence_info *sequence_data, GApplication * app)
{
    switch (the_item->type)
    {
        case unknown:
            display_show_message ("Unknown sequence item", app);
            break;
    }
}

```



```

    case start_sound:
        execute_start_sound (the_item, sequence_data, app);
        break;

    case stop:
        execute_stop_sound (the_item, sequence_data, app);
        break;

    case wait:
        execute_wait (the_item, sequence_data, app);
        break;

    case offer_sound:
        execute_offer_sound (the_item, sequence_data, app);
        break;

    case cease_offering_sound:
        execute_cease_offering_sound (the_item, sequence_data, app);
        break;

    case operator_wait:
        execute_operator_wait (the_item, sequence_data, app);
        break;

    case start_sequence:
        display_show_message ("Start sequence", app);
        break;

}

return;
}

/* Execute a Start Sound sequence item. */
void
execute_start_sound (struct sequence_item_info *the_item,
                    struct sequence_info *sequence_data, GApplication * app)
{
    gint cluster_number;
    struct sound_info *sound_effect;
    struct sound_info *old_sound_effect;
    struct remember_info *remember_data;
    gboolean item_found;
    GList *item_list;

    if (TRACE_SEQUENCER)

```

```

{
    g_print ("Start Sound, cluster = %d, sound name = %s, next = %s, "
            " complete = %s, terminate = %s.\n", the_item->cluster_number,
            the_item->sound_name, the_item->next_starts,
            the_item->next_completion, the_item->next_termination);

    g_print ("item_list = %p, " "next_item_name = %p, " "running = %p, "
            "offering = %p, " "current_operator_wait = %p, "
            "operator_waiting = %p, " "waiting = %p, "
            "message_displaying = %d, " "message_id = %d.\n",
            sequence_data->item_list, sequence_data->next_item_name,
            sequence_data->running, sequence_data->offering,
            sequence_data->current_operator_wait,
            sequence_data->operator_waiting, sequence_data->waiting,
            sequence_data->message_displaying, sequence_data->message_id);
}

cluster_number = the_item->cluster_number;

/* See if there is already a sound on this cluster. */
item_found = FALSE;
for (item_list = sequence_data->running; item_list != NULL;
     item_list = item_list->next)
{
    remember_data = item_list->data;
    old_sound_effect = remember_data->sound_effect;
    if ((remember_data->cluster_number == cluster_number)
        && (remember_data->off_cluster == FALSE))
    {
        item_found = TRUE;
        break;
    }
}
if (item_found)
{
    if (!old_sound_effect->release_has_started)
    {
        display_show_message ("Cannot start a sound on a busy cluster.",
                              app);

        return;
    }
    /* There is a sound on this cluster, but it is releasing.
     * Remove it from the cluster in favor of this new sound. */
    button_reset_cluster (old_sound_effect, app);
    remember_data->off_cluster = TRUE;
}

```

```

/* Set the name of the cluster to the specified text. */
sound_cluster_set_name (the_item->text_to_display, cluster_number, app);

/* Associate the sound with the cluster. */
sound_effect =
    sound_bind_to_cluster (the_item->sound_name, cluster_number, app);

/* If the sound does not exist, don't try to start it. */
if (sound_effect != NULL)
{
    /* Start that sound. */
    sound_start_playing (sound_effect, app);

    /* Show the operator that a sound is playing on this cluster. */
    button_set_cluster_playing (sound_effect, app);

    /* Remember that the sound is running. */
    remember_data = g_malloc (sizeof (struct remember_info));
    remember_data->cluster_number = cluster_number;
    remember_data->sequence_item = the_item;
    remember_data->sound_effect = sound_effect;
    remember_data->active = TRUE;
    remember_data->being_displayed = FALSE;
    remember_data->release_seen = FALSE;
    remember_data->release_sent = FALSE;
    remember_data->off_cluster = FALSE;
    sequence_data->running =
        g_list_append (sequence_data->running, remember_data);
}

/* In case this is the most important text to be displayed to the operator,
 * update the operator display. */
update_operator_display (sequence_data, app);

/* Advance to the next sequence item. */
sequence_data->next_item_name = the_item->next_starts;

return;
}

/* Execute a Stop sequence item. */
void
execute_stop_sound (struct sequence_item_info *the_item,
                   struct sequence_info *sequence_data, GApplication * app)
{

```

```

struct remember_info *remember_data;
struct sequence_item_info *sequence_item;
gboolean item_found, still_searching;
GList *item_list;

if (TRACE_SEQUENCER)
{
    g_print ("stop sound, tag = %s, next = %s.\n", the_item->tag,
            the_item->next);
}

/* Stop all running sounds with the specified tag. */
still_searching = TRUE;

while (still_searching)
{
    /* Find all running sounds whose Start Sound sequence item has
     * the specified tag. */
    item_found = FALSE;
    for (item_list = sequence_data->running; item_list != NULL;
        item_list = item_list->next)
    {
        remember_data = item_list->data;
        sequence_item = remember_data->sequence_item;
        if ((g_strcmp0 (the_item->tag, sequence_item->tag) == 0)
            && remember_data->active && !remember_data->release_sent)
        {
            item_found = TRUE;
            break;
        }
    }
    if (item_found)
    {
        /* Stop the sound. When the sound terminates we will clean up. */
        remember_data->release_sent = TRUE;
        sound_stop_playing (remember_data->sound_effect, app);
    }
    else
    {
        still_searching = FALSE;
    }
}

/* Advance to the next sequence item. */
sequence_data->next_item_name = the_item->next;

```

```

    return;
}

/* Execute a Wait sequence item. */
void
execute_wait (struct sequence_item_info *the_item,
              struct sequence_info *sequence_data, GApplication * app)
{
    struct remember_info *remember_data;

    if (TRACE_SEQUENCER)
    {
        g_print ("Wait, name = %s, time = %" G_GUINT64_FORMAT ", "
                  " when complete = %s, operator text = %s, next = %s.\n",
                  the_item->name, the_item->time_to_wait,
                  the_item->next_completion, the_item->text_to_display,
                  the_item->next);
    }

    /* Record information about the wait, since we will need it when
     * the wait is over. */
    remember_data = g_malloc (sizeof (struct remember_info));
    remember_data->cluster_number = 0;
    remember_data->sound_effect = NULL;
    remember_data->sequence_item = the_item;

    if ((sequence_data->waiting == NULL)
        && (sequence_data->current_operator_wait == NULL))
    {
        /* There are no prior Wait or Operator Wait commands running. */
        /* TODO: display the Wait that will end soonest. */
        remember_data->active = TRUE;
        display_set_operator_text (the_item->text_to_display, app);
    }
    else
    {
        remember_data->active = FALSE;
    }
    remember_data->release_seen = FALSE;
    remember_data->release_sent = FALSE;
    remember_data->off_cluster = FALSE;

    /* Place this item on the wait list. */
    sequence_data->waiting =
        g_list_append (sequence_data->waiting, remember_data);
}

```

```

    /* Arrange to call wait_completed when the wait is over. */
    timer_create_entry (wait_completed, (the_item->time_to_wait / 1e9),
                      remember_data, app);

    /* Advance to the next sequence item. */
    sequence_data->next_item_name = the_item->next;

    return;
}

/* Call here from the timer when a wait is completed. */
static void
wait_completed (void *user_data, GApplication * app)
{
    struct remember_info *remember_data = user_data;
    struct sequence_info *sequence_data;
    struct sequence_item_info *current_sequence_item;
    GList *list_element, *next_list_element;

    sequence_data = sep_get_sequence_data (app);
    current_sequence_item = NULL;

    /* Find this item on the wait list so we can remove it. */
    list_element = sequence_data->waiting;
    while (list_element != NULL)
    {
        next_list_element = list_element->next;
        if (remember_data == list_element->data)
        {
            /* This is the item on the list. Remove it. */
            remember_data->active = FALSE;
            current_sequence_item = remember_data->sequence_item;
            g_free (remember_data);
            sequence_data->waiting =
                g_list_delete_link (sequence_data->waiting, list_element);
        }
        list_element = next_list_element;
    }

    /* If we didn't find the item on the list, do nothing. */
    if (current_sequence_item == NULL)
    {
        return;
    }
}

```

```

if (TRACE_SEQUENCER)
{
    g_print ("Wait completed, name = %s, time = %" G_GUINT64_FORMAT ", "
            " when complete = %s, operator text = %s, next = %s.\n",
            current_sequence_item->name,
            current_sequence_item->time_to_wait,
            current_sequence_item->next_completion,
            current_sequence_item->text_to_display,
            current_sequence_item->next);
    g_print ("sequence_data->waiting == %p.\n", sequence_data->waiting);
}
/* TODO: display the wait list item that will end soonest,
 * or the current operator wait if there is one. */

/* Tell the sequencer to proceed from the specified item. */
sequence_data->next_item_name = current_sequence_item->next_completion;
execute_items (sequence_data, app);

return;
}

/* Execute an Offer Sound sequence item. */
void
execute_offer_sound (struct sequence_item_info *the_item,
                    struct sequence_info *sequence_data, GApplication * app)
{
    struct remember_info *remember_data;

    if (TRACE_SEQUENCER)
    {
        g_print ("Offer sound, name = %s, cluster = %d, Q number = %s, "
                "next = %s,\n next_to_start = %s, offering = %p.\n",
                the_item->name, the_item->cluster_number, the_item->Q_number,
                the_item->next, the_item->next_to_start,
                sequence_data->offering);
    }

    /* Set the name of the cluster to the specified text. */
    sound_cluster_set_name (the_item->text_to_display, the_item->cluster_number,
                           app);

    /* Remember that the sound is being offered. */

    remember_data = g_malloc (sizeof (struct remember_info));
    remember_data->cluster_number = the_item->cluster_number;
    remember_data->sound_effect = NULL;

```

```

remember_data->sequence_item = the_item;
remember_data->active = TRUE;
remember_data->release_seen = FALSE;
remember_data->release_sent = FALSE;
remember_data->off_cluster = FALSE;

sequence_data->offering =
    g_list_append (sequence_data->offering, remember_data);

if (TRACE_SEQUENCER)
{
    g_print ("End of Offer sound, offering = %p.\n",
            sequence_data->offering);
}

/* Advance to the next sequence item. */
sequence_data->next_item_name = the_item->next;

return;
}

/* Execute a Cease Offering Sound sequence item. */
void
execute_cease_offering_sound (struct sequence_item_info *the_item,
                             struct sequence_info *sequence_data,
                             GApplication * app)
{
    gint cluster_number;
    struct remember_info *remember_data;
    struct sequence_item_info *sequence_item;
    GList *list_element, *next_list_element;

    if (TRACE_SEQUENCER)
    {
        g_print ("Cease offering sound, name = %s, tag = %s, " "next = %s.\n",
                the_item->name, the_item->tag, the_item->next);
    }

    /* Process every Offer Sound sequence item with the same tag. */
    list_element = sequence_data->offering;
    while (list_element != NULL)
    {
        next_list_element = list_element->next;
        remember_data = list_element->data;
        sequence_item = remember_data->sequence_item;
    }
}

```



```

    if ((g_strcmp0 (the_item->tag, sequence_item->tag) == 0)
        && (remember_data->active))
    {
        /* We have a match. */
        remember_data->active = FALSE;

        /* Remove the Offer Sound from the cluster. */
        sequence_data->offering =
            g_list_remove_link (sequence_data->offering, list_element);

        /* Remove the Offer Sound's text from the cluster. */
        cluster_number = remember_data->cluster_number;
        sound_cluster_set_name ((gchar *) "", cluster_number, app);

        g_list_free (list_element);
        g_free (remember_data);
    }
    list_element = next_list_element;
}

/* Advance to the next sequence item. */
sequence_data->next_item_name = the_item->next;

return;
}

/* Execute an Operator Wait sequence item. */
void
execute_operator_wait (struct sequence_item_info *the_item,
                      struct sequence_info *sequence_data,
                      GApplication * app)
{
    struct remember_info *remember_data;

    if (TRACE_SEQUENCER)
    {
        g_print ("Operator Wait, name = %s, next play = %s, "
                  "operator text = %s, next = %s.\n", the_item->name,
                  the_item->next_play, the_item->text_to_display,
                  the_item->next);
    }

    /* Record information about the operator wait, since we will need it when
       * the operator wait is over. */
    remember_data = g_malloc (sizeof (struct remember_info));
    remember_data->cluster_number = 0;

```

```

remember_data->sound_effect = NULL;
remember_data->sequence_item = the_item;
remember_data->release_seen = FALSE;
remember_data->release_sent = FALSE;
remember_data->off_cluster = FALSE;

if (sequence_data->current_operator_wait == NULL)
{
    /* There are no prior operator wait commands running. */
    remember_data->active = TRUE;
    sequence_data->current_operator_wait = remember_data;
    display_set_operator_text (the_item->text_to_display, app);
}
else
{
    /* There is an Operator Wait already running; place this one
     * on the list to be executed later. */
    remember_data->active = FALSE;
    sequence_data->operator_waiting =
        g_list_append (sequence_data->operator_waiting, remember_data);
}

/* Advance to the next sequence item. */
sequence_data->next_item_name = the_item->next;

return;
}

/* Update the operator display. Show the most important item, preferring
 * the current item in case of a tie. */
static void
update_operator_display (struct sequence_info *sequence_data,
                        GApplication * app)
{
    struct remember_info *remember_data;
    struct sequence_item_info *sequence_item;
    struct remember_info *most_important;
    struct remember_info *current_display;
    struct sound_info *sound_effect;
    gboolean found_item;
    guint most_importance;
    GList *item_list;
    gchar *elapsed_time, *remaining_time;
    gchar *display_text;

    /* For debugging, optionally don't update the operator display. */

```

```

if (!DO_OPERATOR_DISPLAY)
    return;

found_item = FALSE;
most_importance = 0;
most_important = NULL;
current_display = NULL;
for (item_list = sequence_data->running; item_list != NULL;
     item_list = item_list->next)
{
    remember_data = item_list->data;

    /* Note which item is currently being displayed. */
    if (remember_data->being_displayed)
    {
        current_display = remember_data;
    }

    /* Find the item that should be displayed. */
    sequence_item = remember_data->sequence_item;
    if ((remember_data->active) && (sequence_item->importance > 0))
    {
        if (found_item == FALSE)
        {
            most_important = remember_data;
            most_importance = sequence_item->importance;
            found_item = TRUE;
        }
        else
        {
            if (sequence_item->importance > most_importance)
            {
                most_important = remember_data;
                most_importance = sequence_item->importance;
            }
            else
            {
                if ((sequence_item->importance == most_importance)
                    && (remember_data->being_displayed == TRUE))
                {
                    most_important = remember_data;
                }
            }
        }
    }
}
}

```

```

if (found_item == TRUE)
{
    /* "most_important" is the item we should be displaying.
     * "current_display" is the item we are currently displaying, if any.
     * These may be the same item. */
    sequence_item = most_important->sequence_item;
    sound_effect = most_important->sound_effect;
    /* Prepend the elapsed time to the operator message, and append
     * the remaining time. */
    elapsed_time = sound_get_elapsed_time (sound_effect, app);
    remaining_time = sound_get_remaining_time (sound_effect, app);
    display_text =
        g_strdup_printf ("%s %s (%s)", elapsed_time,
                        sequence_item->text_to_display, remaining_time);
    /* If there is a message already being displayed by the sequencer,
     * remove it. */
    if (sequence_data->message_displaying)
    {
        display_remove_message (sequence_data->message_id, app);
    }

    /* Display the most important message. */
    sequence_data->message_id = display_show_message (display_text, app);
    sequence_data->message_displaying = TRUE;
    g_free (display_text);
    display_text = NULL;
    g_free (elapsed_time);
    elapsed_time = NULL;

    /* Mark the most important item as the one currently being displayed. */
    if (current_display != NULL)
    {
        current_display->being_displayed = FALSE;
    }
    most_important->being_displayed = TRUE;

    if (TRACE_SEQUENCER && TRACE_SEQUENCER_DISPLAY_MESSAGE)
    {
        g_print ("Display message %d.\n", sequence_data->message_id);
    }
    /* Keep updating the display every 0.1 second until there is nothing
     * to show. */
    timer_create_entry (clock_tick, 0.1, sequence_data, app);
}
}

```

```

/* Come here when the 0.1-second clock ticks to update the operator display. */
static void
clock_tick (void *user_data, GApplication * app)
{
    struct sequence_info *sequence_data = user_data;
    update_operator_display (sequence_data, app);
}

/* Cease showing some text to the operator. */
static void
cancel_operator_display (struct remember_info *remember_data,
                        struct sequence_info *sequence_data,
                        GApplication * app)
{
    if (sequence_data->message_displaying && remember_data->being_displayed)
    {
        if (TRACE_SEQUENCER)
        {
            g_print ("Cancel message %d.\n", sequence_data->message_id);
        }
        display_remove_message (sequence_data->message_id, app);
        remember_data->being_displayed = FALSE;
        sequence_data->message_id = 0;
        sequence_data->message_displaying = FALSE;
    }
}

/* Execute the Go command from an external sequencer issuing MIDI Show Control
 * commands. */
void
sequence_MIDI_show_control_go (gchar * Q_number, GApplication * app)
{
    struct sequence_info *sequence_data;
    struct remember_info *remember_data;
    struct sequence_item_info *sequence_item;
    gboolean found_item;
    GList *item_list;

    sequence_data = sep_get_sequence_data (app);

    if (TRACE_SEQUENCER)
    {
        g_print ("MIDI show control go, Q_number = %s.\n", Q_number);
    }

```

```

    }

    /* Find the cluster whose Offer Sound sequence item has the specified
    * Q_number. */
    found_item = FALSE;
    for (item_list = sequence_data->offering; item_list != NULL;
        item_list = item_list->next)
    {
        remember_data = item_list->data;
        sequence_item = remember_data->sequence_item;
        if ((g_strcmp0 (Q_number, sequence_item->Q_number) == 0)
            && (remember_data->active))
        {
            found_item = TRUE;
            break;
        }
    }

    if (!found_item)
    {
        display_show_message ("No matching Q_number.", app);
        return;
    }

    /* Run the sequencer. A subsequent Start Sound sequence item
    * which names this same cluster will take possession of the cluster
    * until it completes or is terminated. */
    sequence_data->next_item_name = sequence_item->next_to_start;
    execute_items (sequence_data, app);

    return;
}

/* Execute the Go_off command from an external sequencer issuing MIDI Show
* Control commands. */
void
sequence_MIDI_show_control_go_off (gchar * Q_number, GApplication * app)
{
    struct sequence_info *sequence_data;
    struct remember_info *remember_data;
    struct sequence_item_info *sequence_item;
    GList *item_list;

    sequence_data = sep_get_sequence_data (app);

    /* Stop the running sounds whose Start Sound sequence item has the specified

```

```

    * Q_number. If there is no Q number, stop all sounds. */
    for (item_list = sequence_data->running; item_list != NULL;
        item_list = item_list->next)
    {
        remember_data = item_list->data;
        sequence_item = remember_data->sequence_item;
        if (((Q_number == NULL) || (g_strcmp0 (Q_number, (gchar *) ""))
            || (g_strcmp0 (Q_number, sequence_item->Q_number) == 0))
            && remember_data->active && !remember_data->release_sent)
        {
            /* Stop the sound. When the sound terminates we will clean up. */
            remember_data->release_sent = TRUE;
            sound_stop_playing (remember_data->sound_effect, app);
        }
    }

    return;
}

/* Process the Start button on a cluster. */
void
sequence_cluster_start (guint cluster_number, GApplication * app)
{
    struct sequence_info *sequence_data;
    struct remember_info *remember_data;
    struct sequence_item_info *sequence_item;
    gboolean found_item;
    GList *item_list;

    if (TRACE_SEQUENCER)
    {
        g_print ("sequence_cluster_start: cluster = %d.\n", cluster_number);
    }

    sequence_data = sep_get_sequence_data (app);

    /* See if there is an Offer Sound sequence item outstanding which names
    * this cluster. */
    found_item = FALSE;
    for (item_list = sequence_data->offering; item_list != NULL;
        item_list = item_list->next)
    {
        remember_data = item_list->data;
        if (remember_data->cluster_number == cluster_number)
        {
            found_item = TRUE;

```

```

        break;
    }
}
if (!found_item)
{
    display_show_message ("No sound offering on this cluster.", app);
    return;
}

/* We have an Offer Sound sequence item on this cluster.
   * Run the sequencer starting at its specified sequence item. */
sequence_item = remember_data->sequence_item;
sequence_data->next_item_name = sequence_item->next_to_start;
execute_items (sequence_data, app);

return;
}

/* Process the Stop button on a cluster. */
void
sequence_cluster_stop (guint cluster_number, GApplication * app)
{
    struct sequence_info *sequence_data;
    struct remember_info *remember_data;
    gboolean item_found;
    GList *item_list;

    sequence_data = sep_get_sequence_data (app);

    /* See if there is a Start Sound sequence item outstanding which names
       * this cluster. */
    item_found = FALSE;
    for (item_list = sequence_data->running; item_list != NULL;
         item_list = item_list->next)
    {
        remember_data = item_list->data;
        if ((remember_data->cluster_number == cluster_number)
            && remember_data->active && !remember_data->release_sent)
        {
            item_found = TRUE;
            break;
        }
    }
    if (!item_found)
    {
        /* There isn't. Ignore the stop button. */
    }
}

```



```

        display_show_message ("No sound to stop.", app);
        return;
    }

    /* We have a Start Sound sequence item on this cluster.
     * Tell the sound to stop. When it has stopped, the termination
     * process will be invoked. */
    remember_data->release_sent = TRUE;
    sound_stop_playing (remember_data->sound_effect, app);

    return;
}

/* Process the Play button. */
void
sequence_button_play (GApplication * app)
{
    struct sequence_info *sequence_data;
    struct remember_info *remember_data;
    struct sequence_item_info *current_sequence_item;
    struct sequence_item_info *next_sequence_item;

    sequence_data = sep_get_sequence_data (app);
    remember_data = sequence_data->current_operator_wait;
    GList *list_element;

    /* If we are not waiting for the operator to press the key,
     * do nothing. */
    if (remember_data == NULL)
        return;

    current_sequence_item = remember_data->sequence_item;

    /* This Operator Wait sequence item is no longer waiting. */
    sequence_data->current_operator_wait = NULL;
    g_free (remember_data);
    remember_data = NULL;

    /* See if there is another one ready to wait. */
    list_element = g_list_first (sequence_data->operator_waiting);
    if (list_element != NULL)
    {
        /* There is, give it its chance to display for the opeaaator. */
        sequence_data->operator_waiting =
            g_list_remove_link (sequence_data->operator_waiting, list_element);
        remember_data = list_element->data;
    }
}

```

```

        remember_data->active = TRUE;
        next_sequence_item = remember_data->sequence_item;
        display_set_operator_text (next_sequence_item->text_to_display, app);
        sequence_data->current_operator_wait = remember_data;
        g_list_free (list_element);
    }
    else
    {
        display_clear_operator_text (app);
    }

    /* Run the sequencer starting from the Operator Wait's specified label. */
    sequence_data->next_item_name = current_sequence_item->next_play;
    execute_items (sequence_data, app);

    return;
}

/* Process the completion of a sound. */
void
sequence_sound_completion (struct sound_info *sound_effect,
                           gboolean terminated, GApplication * app)
{
    struct sequence_info *sequence_data;
    struct remember_info *remember_data;
    struct remember_info *offer_remember_data;
    struct sequence_item_info *start_sound_sequence_item;
    struct sequence_item_info *offer_sound_sequence_item;
    gboolean item_found;
    GList *item_list, *found_item;

    sequence_data = sep_get_sequence_data (app);

    if (TRACE_SEQUENCER)
    {
        g_print ("completion of sound %s on cluster %d.\n", sound_effect->name,
                 sound_effect->cluster_number);
    }

    /* See if there is a Start Sound sequence item outstanding which names
     * this sound. */
    item_found = FALSE;
    for (item_list = sequence_data->running; item_list != NULL;
         item_list = item_list->next)
    {
        remember_data = item_list->data;

```

```

    if ((remember_data->sound_effect == sound_effect)
        && remember_data->active)
    {
        found_item = item_list;
        item_found = TRUE;
        break;
    }
}

if (!item_found)
{
    /* There isn't. Ignore the completion. */
    display_show_message ("Completion but sound not running.", app);
    return;
}

/* We have a Start Sound sequence item for this sound. It has
 * completed. */
start_sound_sequence_item = remember_data->sequence_item;

/* If we are showing the status of this sound to the operator,
 * stop doing that. */
cancel_operator_display (remember_data, sequence_data, app);

/* If this sound is still showing on the cluster, set the start label
 * back to "Start". */
if (!remember_data->off_cluster)
{
    button_reset_cluster (sound_effect, app);
    remember_data->off_cluster = TRUE;
}

/* See if there is an Offer Sound sequence item outstanding which names
 * this cluster. */
item_found = FALSE;
for (item_list = sequence_data->offering; item_list != NULL;
     item_list = item_list->next)
{
    offer_remember_data = item_list->data;
    if ((offer_remember_data->cluster_number ==
        sound_effect->cluster_number) && (offer_remember_data->active))
    {
        offer_sound_sequence_item = offer_remember_data->sequence_item;
        item_found = TRUE;
        break;
    }
}
}

```

```

/* If there is, restore its text to the cluster. If there isn't, clear
 * the text field. */
    if (item_found)
    {
        if (TRACE_SEQUENCER)
        {
            g_print ("Offer sound found.\n");
        }
        sound_cluster_set_name (offer_sound_sequence_item->text_to_display,
                                remember_data->cluster_number, app);
    }
    else
    {
        sound_cluster_set_name ((gchar *) "", sound_effect->cluster_number,
                                app);
    }
}

/* Remove the sequence item from the running list. */
sequence_data->running =
    g_list_remove_link (sequence_data->running, found_item);
g_free (remember_data);
g_list_free (found_item);

/* If there is another sound running, show its status. */
update_operator_display (sequence_data, app);

/* Now that the Start Sound has completed, run the sequencer
 * from its completion or termination label. */
if (terminated)
{
    sequence_data->next_item_name =
        start_sound_sequence_item->next_termination;
}
else
{
    sequence_data->next_item_name =
        start_sound_sequence_item->next_completion;
}
execute_items (sequence_data, app);

return;
}

/* Process the start of the release stage of a sound. */

```

```

void
sequence_sound_release_started (struct sound_info *sound_effect,
                               GApplication * app)
{
    struct sequence_info *sequence_data;
    struct remember_info *remember_data;
    struct sequence_item_info *start_sound_sequence_item;
    gboolean item_found;
    GList *item_list;

    sequence_data = sep_get_sequence_data (app);

    if (TRACE_SEQUENCER)
    {
        g_print ("release started for sound %s on cluster %d.\n",
                 sound_effect->name, sound_effect->cluster_number);
    }
    sound_effect->release_has_started = TRUE;

    /* See if there is a Start Sound sequence item outstanding for this
     * sound effect. */
    item_found = FALSE;
    for (item_list = sequence_data->running; item_list != NULL;
         item_list = item_list->next)
    {
        remember_data = item_list->data;
        if ((remember_data->sound_effect == sound_effect)
            && (remember_data->active))
        {
            item_found = TRUE;
            break;
        }
    }

    if (!item_found)
    {
        /* There isn't. Ignore the release. */
        display_show_message ("Release started but sound not running.", app);
        return;
    }

    remember_data->release_seen = TRUE;

    /* We have a Start Sound sequence item for this sound. It has
     * started the release stage of its amplitude envelope. */
    start_sound_sequence_item = remember_data->sequence_item;

```

```
/* Show the operator that the sound is now releasing. */
button_set_cluster_releasing (sound_effect, app);

/* If there is another sound running, show its status. */
update_operator_display (sequence_data, app);

/* Run the sequencer from the release_started label unless we have
 * sent a release command to the sound, in which case we will run
 * from the termination label when the sound completes, instead. */

if (!remember_data->release_sent)
{
    sequence_data->next_item_name =
        start_sound_sequence_item->next_release_started;
    execute_items (sequence_data, app);
}

return;
}
```

27 sequence__subroutines.h

```

/*
 * sequence_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <gtk/gtk.h>
#include <gst/gst.h>
#include "sequence_structure.h"
#include "sound_structure.h"

/* Subroutines defined in sequence_subroutines.c */

/* Initialize the internal sequencer */
void *sequence_init (GApplication * app);

/* Append a sequence item to the sequence. */
void sequence_append_item (struct sequence_item_info *sequence_item_data,
                           GApplication * app);

/* Start the internal sequencer. */
void sequence_start (GApplication * app);

/* Execute the MIDI Show Control command Go. */
void sequence_MIDI_show_control_go (gchar * Q_number, GApplication * app);

/* Execute the MIDI Show Control command Go_off. */
void sequence_MIDI_show_control_go_off (gchar * Q_number, GApplication * app);

/* Start the sound offered on a cluster. */
void sequence_cluster_start (guint cluster_number, GApplication * app);

```

```
/* Stop the sound offered on a cluster. */
void sequence_cluster_stop (guint cluster_number, GApplication * app);

/* Operator pushed the Play button. */
void sequence_button_play (GApplication * app);

/* A sound has completed. */
void sequence_sound_completion (struct sound_info *sound_effect,
                                gboolean terminated, GApplication * app);

/* A sound has reached its release stage. */
void sequence_sound_release_started (struct sound_info *sound_effect,
                                     GApplication * app);

/* End of file sequence_subroutines.h */
```


28 signal_subroutines.c

```

/*
 * signal_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <glib.h>
#include <glib-unix.h>
#include <gtk/gtk.h>
#include <gst/gst.h>
#include "signal_subroutines.h"
#include "sound_effects_player.h"
#include "gstreamer_subroutines.h"

/* When debugging it can be useful to trace what is happening in the
 * signal handler. */
#define TRACE_SIGNALS FALSE

/* the persistent data used by the signal handler */
/* none used at the moment, but we will probably need something
 * to implement sighup. */

struct signal_info
{
    gboolean not_used;
};

/* Forward declarations, so I can call these subroutines before I define them.
 */
static gboolean signal_term (gpointer user_data);
static gboolean signal_hup (gpointer user_data);

```

```

/* Initialize the signal handler. */
void *
signal_init (GApplication * app)
{
    struct signal_info *signal_data;

    /* Allocate the persistent data. */
    signal_data = g_malloc (sizeof (struct signal_info));

    /* Specify the routines to handle signals. */
    g_unix_signal_add (SIGTERM, signal_term, app);
    g_unix_signal_add (SIGHUP, signal_hup, app);

    return (signal_data);
}

/* Subroutine called when a term signal is received. */
static gboolean
signal_term (gpointer user_data)
{
    GApplication *app = user_data;

    if (TRACE_SIGNALS)
    {
        g_print ("signal term.\n");
    }

    /* Initiate the shutdown of the gstreamer pipeline. When it is complete
     * the application will terminate. */
    gstreamer_shutdown (app);

    return TRUE;
}

/* Subroutine called when a hup signal is received. */
static gboolean
signal_hup (gpointer user_data)
{
    if (TRACE_SIGNALS)
    {
        g_print ("signal hup.\n");
    }

    /* TODO: shutdown the gstreamer pipeline, re-read the current project,
     * and rebuild the gstreamer pipeline based on it. */

```

```
    return TRUE;
}

/* Shut down the signal handler. */
void
signal_finalize (GApplication * app)
{
    struct signal_info *signal_data;
    struct sigaction new_action;

    signal_data = sep_get_signal_data (app);

    /* Restore the signals. */
    new_action.sa_handler = SIG_DFL;
    sigemptyset (&new_action.sa_mask);
    new_action.sa_flags = 0;
    sigaction (SIGTERM, &new_action, NULL);
    sigaction (SIGHUP, &new_action, NULL);

    g_free (signal_data);
    signal_data = NULL;
    return;
}
```

29 signal_subroutines.h

```
/*
 * signal_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

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

/* Subroutines defined in signal_subroutines.c */

/* Initialize the signal handler */
void *signal_init (GApplication * app);

/* Terminate the signal handler */
void signal_finalize (GApplication * app);

/* End of file signal_subroutines.h */
```

30 sound_effects_player.c

```

/*
 * sound_effects_player.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * Sound_effects_player is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Sound_effects_player is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program. If not, see <http://www.gnu.org/licenses/>.
 */
#include <glib/gi18n.h>
#include <libxml/xmlmemory.h>
#include "sound_effects_player.h"
#include "sound_structure.h"
#include "gstreamer_subroutines.h"
#include "menu_subroutines.h"
#include "network_subroutines.h"
#include "parse_xml_subroutines.h"
#include "parse_net_subroutines.h"
#include "sound_subroutines.h"
#include "sequence_subroutines.h"
#include "signal_subroutines.h"
#include "timer_subroutines.h"
#include "display_subroutines.h"

G_DEFINE_TYPE (Sound_Effects_Player, sound_effects_player,
               GTK_TYPE_APPLICATION);

/* ANJUTA: Macro SOUND_EFFECTS_PLAYER_APPLICATION gets Sound_Effects_Player
 * - DO NOT REMOVE */

/* The private data associated with the top-level window. */
struct _Sound_Effects_PlayerPrivate
{
    /* The Gstreamer pipeline. */
    GstPipeline *gstreamer_pipeline;

```

```
/* A flag that is set by the Gstreamer startup process when it is
 * complete. */
gboolean gstreamer_ready;

/* The top-level gtk window. */
GtkWindow *top_window;

/* A flag that is set to indicate that we have told GTK to start
 * showing the top-level window. */
gboolean windows_showing;

/* The common area, needed for updating the display asynchronously. */
GtkWidget *common_area;

/* The place where messages to the operator are shown. */
GtkLabel *operator_text;

/* The status bar and context ID, used for showing messages. */
GtkStatusbar *status_bar;
guint context_id;

/* The list of sounds we can make. Each item of the GList points
 * to a sound_info structure. */
GList *sound_list;

/* The persistent information for the internal sequencer. */
void *sequence_data;

/* The persistent information for the signal handler. */
void *signal_data;

/* The persistent information for the timer. */
void *timer_data;

/* The list of clusters that might contain sound effects. */
GList *clusters;

/* The persistent network information. */
void *network_data;

/* The persistent information for the network commands parser. */
void *parse_net_data;

/* The XML file that holds parameters for the program. */
xmlDocPtr project_file;
```

```

/* The name of that file, for use in Save and as the default file
   * name for Save As. */
gchar *project_filename;

/* The path to the user interface files. */
gchar *ui_path;

/* ANJUTA: Widgets declaration for sound_effects_player.ui - DO NOT REMOVE */
};

/* Create a new window loading a file. */
static void
sound_effects_player_new_window (GApplication * app, GFile * file)
{
    GtkWidget *top_window;
    GtkWidget *common_area;
    GtkLabel *operator_text;
    GtkStatusbar *status_bar;
    guint context_id;
    GtkBuilder *builder;
    GError *error = NULL;
    gint cluster_number;
    gchar *cluster_name;
    GtkWidget *cluster_widget;
    gchar *filename;
    gchar *local_filename;
    guint message_code;

    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    /* Initialize the signal handler. */
    priv->signal_data = signal_init (app);

    /* Initialize the timer. */
    priv->timer_data = timer_init (app);

    /* Remember the path to the user interface files. */
    priv->ui_path = g_strdup (PACKAGE_DATA_DIR "/ui/");

    /* Load the main user interface definition from its file. */
    builder = gtk_builder_new ();
    filename = g_strconcat (priv->ui_path, "sound_effects_player.ui", NULL);
    if (!gtk_builder_add_from_file (builder, filename, &error))
    {

```

```

        g_critical ("Couldn't load builder file %s: %s", filename,
                    error->message);
        g_error_free (error);
    }

    /* Auto-connect signal handlers. */
    gtk_builder_connect_signals (builder, app);

    /* Get the top-level window object from the user interface file. */
    top_window =
        GTK_WINDOW (gtk_builder_get_object (builder, "top_level_window"));
    priv->top_window = top_window;
    if (top_window == NULL)
    {
        g_critical ("Widget \"top_level_window\" is missing in file %s.",
                    filename);
    }

    /* Also get the common area, operator text and status bar. */
    common_area = GTK_WIDGET (gtk_builder_get_object (builder, "common_area"));
    priv->common_area = common_area;
    if (common_area == NULL)
    {
        g_critical ("Widget \"common_area\" is missing in file %s.", filename);
    }

    operator_text =
        GTK_LABEL (gtk_builder_get_object (builder, "operator_text"));
    priv->operator_text = operator_text;
    if (operator_text == NULL)
    {
        g_critical ("Operator text is missing in file %s.", filename);
    }

    status_bar = GTK_STATUSBAR (gtk_builder_get_object (builder, "status_bar"));
    priv->status_bar = status_bar;
    if (status_bar == NULL)
    {
        g_critical ("Status bar is missing in file %s.", filename);
    }

    /* Generate a context ID for the status bar. */
    context_id =
        gtk_statusbar_get_context_id (status_bar, (gchar *) "messages");
    priv->context_id = context_id;

```



```

/* We are done with the name of the user interface file. */
g_free (filename);

/* Remember where the clusters are. Each cluster has a name identifying it. */
priv->clusters = NULL;
for (cluster_number = 0; cluster_number < 16; ++cluster_number)
{
    cluster_name = g_strdup_printf ("cluster_%2.2d", cluster_number);
    cluster_widget =
        GTK_WIDGET (gtk_builder_get_object (builder, cluster_name));
    if (cluster_widget != NULL)
    {
        priv->clusters = g_list_prepend (priv->clusters, cluster_widget);
    }
    g_free (cluster_name);
}

/* ANJUTA: Widgets initialization for sound_effects_player.ui
* - DO NOT REMOVE */

g_object_unref (builder);

gtk_window_set_application (top_window, GTK_APPLICATION (app));

/* If the invocation of sound_effects_player included a parameter,
* that parameter is the name of the project file to load before
* starting the user interface. */
if (file != NULL)
{
    priv->project_filename = g_file_get_parse_name (file);
}
else
    priv->project_filename = NULL;

/* Set up the menu. */
filename = g_strconcat (priv->ui_path, "app-menu.ui", NULL);
menu_init (app, filename);
g_free (filename);

/* Set up the remainder of the private data. */
priv->gstreamer_pipeline = NULL;
priv->gstreamer_ready = FALSE;
priv->sound_list = NULL;

/* Initialize the internal sequencer. */
priv->sequence_data = sequence_init (app);

```

```

/* Initialize the network message parser. */
priv->parse_net_data = parse_net_init (app);

/* Listen for network messages. */
priv->network_data = network_init (app);

/* The display is initialized; time to show it. */
gtk_widget_show_all (GTK_WIDGET (top_window));
priv->windows_showing = TRUE;

/* If we have a parameter, it is the project XML file to read for our sounds.
   * If we don't, the user will read a project XML file using the menu. */
if (priv->project_filename != NULL)
{
    message_code = display_show_message ("Loading...", app);
    local_filename = g_strdup (priv->project_filename);
    parse_xml_read_project_file (local_filename, app);
    priv->gstreamer_pipeline = sound_init (app);
    display_remove_message (message_code, app);
    message_code = display_show_message ("Starting...", app);
}
else
{
    message_code = display_show_message ("No sounds.", app);
}

return;
}

/* GApplication implementation */
static void
sound_effects_player_activate (GApplication * application)
{
    sound_effects_player_new_window (application, NULL);
}

static void
sound_effects_player_open (GApplication * application, GFile ** files,
                           gint n_files, const gchar * hint)
{
    gint i;

    for (i = 0; i < n_files; i++)
        sound_effects_player_new_window (application, files[i]);
}

```

```

static void
sound_effects_player_init (Sound_Effects_Player * object)
{
    object->priv =
        G_TYPE_INSTANCE_GET_PRIVATE (object,
                                      SOUND_EFFECTS_PLAYER_TYPE_APPLICATION,
                                      Sound_Effects_PlayerPrivate);
}

static void
sound_effects_player_finalize (GObject * object)
{
    GList *sound_effect_list;
    GList *next_sound_effect;
    struct sound_info *sound_effect;
    Sound_Effects_Player *self = (Sound_Effects_Player *) object;

    /* Deallocate the gstreamer pipeline. */
    if (self->priv->gstreamer_pipeline != NULL)
    {
        g_object_unref (self->priv->gstreamer_pipeline);
        self->priv->gstreamer_pipeline = NULL;
    }

    /* Deallocate the list of sound effects. */
    sound_effect_list = self->priv->sound_list;

    while (sound_effect_list != NULL)
    {
        sound_effect = sound_effect_list->data;
        next_sound_effect = sound_effect_list->next;
        g_free (sound_effect->name);
        g_free (sound_effect->wav_file_name);
        g_free (sound_effect->wav_file_name_full);
        g_free (sound_effect->OSC_name);
        g_free (sound_effect->function_key);
        g_free (sound_effect);
        self->priv->sound_list =
            g_list_delete_link (self->priv->sound_list, sound_effect_list);
        sound_effect_list = next_sound_effect;
    }

    G_OBJECT_CLASS (sound_effects_player_parent_class)->finalize (object);
}

```

```

static void
sound_effects_player_class_init (Sound_Effects_PlayerClass * klass)
{
    G_APPLICATION_CLASS (klass)->activate = sound_effects_player_activate;
    G_APPLICATION_CLASS (klass)->open = sound_effects_player_open;

    g_type_class_add_private (klass, sizeof (Sound_Effects_PlayerPrivate));

    G_OBJECT_CLASS (klass)->finalize = sound_effects_player_finalize;
}

Sound_Effects_Player *
sound_effects_player_new (void)
{
    return g_object_new (sound_effects_player_get_type (), "application-id",
                        "org.gnome.show_control.sound_effects_player", "flags",
                        G_APPLICATION_HANDLES_OPEN, NULL);
}

/* Callbacks from other modules. The names of the callbacks are prefixed
 * with sep_ rather than sound_effects_player_ for readability. */

/* This is called when the gstreamer pipeline has completed initialization. */
void
sep_gstreamer_ready (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    priv->gstreamer_ready = TRUE;

    /* If we aren't yet showing the top-level window, show it now. */
    if (!priv->windows_showing)
    {
        gtk_widget_show_all (GTK_WIDGET (priv->top_window));
        priv->windows_showing = TRUE;
    }

    /* Tell the operator we are ready. */
    display_show_message ("Ready.", app);

    /* Start the internal sequencer. */
    sequence_start (app);
    return;
}

```

```

/* Create the gstreamer pipeline by reading an XML file. */
void
sep_create_pipeline (gchar * filename, GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    gchar *local_filename;

    local_filename = g_strdup (filename);
    parse_xml_read_project_file (local_filename, app);
    priv->gstreamer_pipeline = sound_init (app);

    return;
}

/* Find the gstreamer pipeline. */
GstPipeline *
sep_get_pipeline_from_app (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    GstPipeline *pipeline_element;

    pipeline_element = priv->gstreamer_pipeline;

    return (pipeline_element);
}

/* Find the application, given any widget in the application. */
GApplication *
sep_get_application_from_widget (GtkWidget * object)
{
    GtkWidget *toplevel_widget;
    GtkWindow *toplevel_window;
    GtkApplication *gtk_app;
    GApplication *app;

    /* Find the top-level window. */
    toplevel_widget = gtk_widget_get_toplevel (object);
    toplevel_window = GTK_WINDOW (toplevel_widget);
    /* The top level window knows where to find the application. */
    gtk_app = gtk_window_get_application (toplevel_window);
    app = (GApplication *) gtk_app;
    return (app);
}

```

```

/* Find the cluster which contains the given widget. */
GtkWidget *
sep_get_cluster_from_widget (GtkWidget * object)
{
    GtkWidget *this_object;
    const gchar *widget_name;
    GtkWidget *cluster_widget;

    /* Work up from the given widget until we find one whose name starts
       * with "cluster_". */

    cluster_widget = NULL;
    this_object = object;
    do
    {
        widget_name = gtk_widget_get_name (this_object);
        if (g_str_has_prefix (widget_name, "cluster_"))
        {
            cluster_widget = this_object;
            break;
        }
        this_object = gtk_widget_get_parent (this_object);
    }
    while (this_object != NULL);

    return (cluster_widget);
}

/* Find the sound effect information corresponding to a cluster,
   * given a widget inside that cluster. Return NULL if
   * the cluster is not running a sound effect. */
struct sound_info *
sep_get_sound_effect (GtkWidget * object)
{
    GtkWidget *this_object;
    const gchar *widget_name;
    GtkWidget *cluster_widget = NULL;
    GtkWidget *toplevel_widget;
    GtkWindow *toplevel_window;
    GtkApplication *app;
    Sound_Effects_Player *self;
    Sound_Effects_PlayerPrivate *priv;
    GList *sound_effect_list;
    struct sound_info *sound_effect = NULL;
    gboolean sound_effect_found;

```

```

/* Work up from the given widget until we find one whose name starts
   * with "cluster_". */

this_object = object;
do
{
    widget_name = gtk_widget_get_name (this_object);
    if (g_str_has_prefix (widget_name, "cluster_"))
    {
        cluster_widget = this_object;
        break;
    }
    this_object = gtk_widget_get_parent (this_object);
}
while (this_object != NULL);

/* Find the application's private data, where the sound effects
   * information is kept. First we find the top-level window,
   * which has the private data. */
toplevel_widget = gtk_widget_get_toplevel (object);
toplevel_window = GTK_WINDOW (toplevel_widget);
/* Work through the pointer structure to the private data. */
app = gtk_window_get_application (toplevel_window);
self = SOUND_EFFECTS_PLAYER_APPLICATION (app);
priv = self->priv;

/* Then we search through the sound effects for the one attached
   * to this cluster. */
sound_effect_list = priv->sound_list;
sound_effect_found = FALSE;
while (sound_effect_list != NULL)
{
    sound_effect = sound_effect_list->data;
    if (sound_effect->cluster_widget == cluster_widget)
    {
        sound_effect_found = TRUE;
        break;
    }
    sound_effect_list = sound_effect_list->next;
}
if (sound_effect_found)
    return (sound_effect);

return NULL;
}

```

```

/* Find a cluster, given its number. */
GtkWidget *
sep_get_cluster_from_number (guint cluster_number, GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    GtkWidget *cluster_widget;
    GList *cluster_list;
    const gchar *widget_name;
    gchar *cluster_name;

    /* Go through the list of clusters looking for one with the name
       * "cluster_" followed by the cluster number. */
    for (cluster_list = priv->clusters; cluster_list != NULL;
         cluster_list = cluster_list->next)
    {
        cluster_widget = cluster_list->data;
        widget_name = gtk_widget_get_name (cluster_widget);
        cluster_name = g_strdup_printf ("cluster_%2.2d", cluster_number);
        if (g_ascii_strcasecmp (widget_name, cluster_name) == 0)
        {
            g_free (cluster_name);
            return (cluster_widget);
        }
        g_free (cluster_name);
    }

    return NULL;
}

/* Given a cluster, find its cluster number. */
guint
sep_get_cluster_number (GtkWidget * cluster_widget)
{
    const gchar *widget_name;
    guint cluster_number;
    gint result;

    /* Extract the cluster number from its name. */
    widget_name = gtk_widget_get_name (cluster_widget);
    result = sscanf (widget_name, "cluster_%u", &cluster_number);
    if (result != 1)
    {
        g_print ("result = %d, cluster number = %d.\n", result, cluster_number);
    }
}

```



```

    return (cluster_number);
}

/* Find the area above the top of the clusters,
 * so it can be updated. The parameter passed is the application, which
 * was passed through gstreamer_setup and the gstreamer signaling system
 * as an opaque value. */
GtkWidget *
sep_get_common_area (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    GtkWidget *common_area;

    common_area = priv->common_area;

    return (common_area);
}

/* Find the network information. The parameter passed is the application, which
 * was passed through the various gio callbacks as an opaque value. */
void *
sep_get_network_data (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    void *network_data;

    network_data = priv->network_data;
    return (network_data);
}

/* Find the network commands parser information.
 * The parameter passed is the application. */
void *
sep_get_parse_net_data (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    void *parse_net_data;

    parse_net_data = priv->parse_net_data;
    return (parse_net_data);
}

/* Find the top-level window, to use as the transient parent for

```

```
 * dialogs. */
GtkWidget *
sep_get_top_window (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    GtkWidget *top_window;

    top_window = priv->top_window;
    return (top_window);
}

/* Find the operator text label widget, which is used to display
 * text from the sequencer to the operator. */
GtkLabel *
sep_get_operator_text (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    GtkLabel *operator_text;

    operator_text = priv->operator_text;
    return operator_text;
}

/* Find the status bar, which is used for messages. */
GtkStatusbar *
sep_get_status_bar (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    GtkStatusbar *status_bar;

    status_bar = priv->status_bar;
    return status_bar;
}

/* Find the context ID, which is also needed for messages. */
guint
sep_get_context_id (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    guint context_id;

    context_id = priv->context_id;
}
```

```

    return context_id;
}

/* Find the project file which contains the parameters. */
xmlDocPtr
sep_get_project_file (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    xmlDocPtr project_file;

    project_file = priv->project_file;
    return (project_file);
}

/* Remember the project file which contains the parameters. */
void
sep_set_project_file (xmlDocPtr project_file, GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    if (priv->project_file != NULL)
    {
        xmlFreeDoc (priv->project_file);
        priv->project_file = NULL;
    }
    priv->project_file = project_file;

    return;
}

/* Find the name of the project file. */
gchar *
sep_get_project_filename (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    gchar *project_filename;

    project_filename = priv->project_filename;
    return (project_filename);
}

/* Find the path to the user interface files. */
gchar *

```

```
sep_get_ui_path (GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
    gchar *ui_path;

    ui_path = priv->ui_path;
    return (ui_path);
}

/* Set the name of the project file. */
void
sep_set_project_filename (gchar * filename, GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    if (priv->project_filename != NULL)
    {
        g_free (priv->project_filename);
        priv->project_filename = NULL;
    }
    priv->project_filename = filename;

    return;
}

/* Find the list of sound effects. */
GList *
sep_get_sound_list (GApplication * app)
{
    GList *sound_list;

    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    sound_list = priv->sound_list;
    return (sound_list);
}

/* Update the list of sound effects. */
void
sep_set_sound_list (GList * sound_list, GApplication * app)
{
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;
```

```
    priv->sound_list = sound_list;
    return;
}

/* Find the persistent data for the internal sequencer. */
void *
sep_get_sequence_data (GApplication * app)
{
    void *sequence_data;
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    sequence_data = priv->sequence_data;
    return (sequence_data);
}

/* Find the persistent data for the signal handler. */
void *
sep_get_signal_data (GApplication * app)
{
    void *signal_data;
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    signal_data = priv->signal_data;
    return (signal_data);
}

/* Find the persistent data for the timer. */
void *
sep_get_timer_data (GApplication * app)
{
    void *timer_data;
    Sound_Effects_PlayerPrivate *priv =
        SOUND_EFFECTS_PLAYER_APPLICATION (app)->priv;

    timer_data = priv->timer_data;
    return (timer_data);
}
```

31 sound_effects_player.h

```

/*
 * sound_effects_player.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * Sound_effects_player is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Sound_effects_player is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#ifndef _SOUND_EFFECTS_PLAYER_H_
#define _SOUND_EFFECTS_PLAYER_H_

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

G_BEGIN_DECLS

#define SOUND_EFFECTS_PLAYER_TYPE_APPLICATION (sound_effects_player_get_type ())
#define SOUND_EFFECTS_PLAYER_APPLICATION(obj) (G_TYPE_CHECK_INSTANCE_CAST ((obj),
    ↪ SOUND_EFFECTS_PLAYER_TYPE_APPLICATION, Sound_Effects_Player))
#define SOUND_EFFECTS_PLAYER_APPLICATION_CLASS(klass) (G_TYPE_CHECK_CLASS_CAST
    ↪ ((klass), SOUND_EFFECTS_PLAYER_TYPE_APPLICATION, Sound_Effects_PlayerClass))
#define SOUND_EFFECTS_PLAYER_IS_APPLICATION(obj) (G_TYPE_CHECK_INSTANCE_TYPE
    ↪ ((obj), SOUND_EFFECTS_PLAYER_TYPE_APPLICATION))
#define SOUND_EFFECTS_PLAYER_IS_APPLICATION_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE
    ↪ ((klass), SOUND_EFFECTS_PLAYER_TYPE_APPLICATION))
#define SOUND_EFFECTS_PLAYER_APPLICATION_GET_CLASS(obj)
    ↪ (G_TYPE_INSTANCE_GET_CLASS ((obj), SOUND_EFFECTS_PLAYER_TYPE_APPLICATION,
    ↪ Sound_Effects_PlayerClass))
typedef struct _Sound_Effects_PlayerClass Sound_Effects_PlayerClass;
typedef struct _Sound_Effects_Player Sound_Effects_Player;
typedef struct _Sound_Effects_PlayerPrivate Sound_Effects_PlayerPrivate;

```

```

struct _Sound_Effects_PlayerClass
{
    GtkApplicationClass parent_class;
};

struct _Sound_Effects_Player
{
    GtkApplication parent_instance;

    Sound_Effects_PlayerPrivate *priv;
};

GType
sound_effects_player_get_type (void) G_GNUC_CONST; Sound_Effects_Player
↪ *sound_effects_player_new (void);

/* Callbacks */

/* The gstreamer pipeline has completed initialization; we can show
 * the top-level window now. */
void sep_gstreamer_ready (GApplication *app);

/* Create the gstreamer pipeline by reading an XML file. */
void sep_create_pipeline (gchar * filename, GApplication *app);

/* Find the gstreamer pipeline. */
GstPipeline *sep_get_pipeline_from_app (GApplication * app);

/* Given a widget, get the app. */
GApplication *sep_get_application_from_widget (GtkWidget * object);

/* Given a widget within a cluster, find the cluster. */
GtkWidget *sep_get_cluster_from_widget (GtkWidget *the_widget);

/* Given a widget in a cluster, get its sound_effect structure. */
struct sound_info *sep_get_sound_effect (GtkWidget * object);

/* Given a cluster number, get the cluster. */
GtkWidget *sep_get_cluster_from_number (guint cluster_number,
                                         GApplication * app);

/* Given a cluster, get its number. */
guint sep_get_cluster_number (GtkWidget *cluster_widget);

/* Find the common area above the clusters. */

```

```
GtkWidget *sep_get_common_area (GApplication * app);

/* Find the network information. */
void *sep_get_network_data (GApplication * app);

/* Find the network messages parser information. */
void *sep_get_parse_net_data (GApplication * app);

/* Find the top-level window. */
GtkWindow *sep_get_top_window (GApplication * app);

/* Find the operator text label widget. */
GtkLabel *sep_get_operator_text (GApplication *app);

/* Find the status bar. */
GtkStatusbar *sep_get_status_bar (GApplication * app);

/* Find the context ID. */
guint sep_get_context_id (GApplication * app);

/* Find the project file. */
xmlDocPtr sep_get_project_file (GApplication * app);

/* Remember the project file. */
void sep_set_project_file (xmlDocPtr project_file, GApplication * app);

/* Find the name of the project file. */
gchar *sep_get_project_filename (GApplication * app);

/* Remember the name of the project file. */
void sep_set_project_filename (gchar * project_filename, GApplication * app);

/* Find the path to the user interface files. */
gchar *sep_get_ui_path (GApplication * app);

/* Find the list of sound effects. */
GList *sep_get_sound_list (GApplication *app);

/* Set the list of sound effects. */
void sep_set_sound_list (GList *sound_list, GApplication *app);

/* Find the sequence information. */
void *sep_get_sequence_data (GApplication *app);

/* Find the signal handler information. */
void *sep_get_signal_data (GApplication *app);
```



```
/* Find the timer information. */  
void *sep_get_timer_data (GApplication *app);  
  
G_END_DECLS  
#endif /* _SOUND_EFFECTS_PLAYER_H_ */
```

32 sound_effects_player.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated with glade 3.19.0 -->
<interface>
  <requires lib="gtk+" version="3.14"/>
  <object class="GtkApplicationWindow" id="top_level_window">
    <property name="width_request">1024</property>
    <property name="can_focus">False</property>
    <property name="title" translatable="yes">Sound Effects Player</property>
    <property name="icon_name">applications-multimedia</property>
    <signal name="delete-event" handler="menu_delete_top_window" object="top_box"
    ↪ swapped="no"/>
  <child>
    <object class="GtkBox" id="top_box">
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="orientation">vertical</property>
      <child>
        <object class="GtkBox" id="common_area">
          <property name="name">common_area</property>
          <property name="visible">True</property>
          <property name="can_focus">False</property>
          <property name="orientation">vertical</property>
          <child>
            <object class="GtkBox" id="VU_meter">
              <property name="name">VU_meter</property>
              <property name="visible">True</property>
              <property name="can_focus">False</property>
              <property name="orientation">vertical</property>
              <child>
                <object class="GtkBox" id="channel_01">
                  <property name="name">01</property>
                  <property name="visible">True</property>
                  <property name="can_focus">False</property>
                  <child>
                    <object class="GtkLabel" id="1">
                      <property name="name">01</property>
                      <property name="visible">True</property>
                      <property name="can_focus">False</property>
                      <property name="label" translatable="yes"></property>
                      <property name="width_chars">1</property>
                      <property name="max_width_chars">1</property>
                    </object>
                  <packing>
                    <property name="expand">False</property>

```

```

        <property name="fill">True</property>
        <property name="position">0</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="2">
        <property name="name">02</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">1</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="3">
        <property name="name">03</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">2</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="4">
        <property name="name">04</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>

```

```

        <property name="position">3</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="5">
        <property name="name">05</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">4</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="6">
        <property name="name">06</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">5</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="7">
        <property name="name">07</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">6</property>

```

```

    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="8">
      <property name="name">08</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">7</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="9">
      <property name="name">09</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">8</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="51">
      <property name="name">10</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">9</property>
    </packing>
  </child>

```

```

</child>
<child>
  <object class="GtkLabel" id="52">
    <property name="name">11</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">10</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="53">
    <property name="name">12</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">11</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="54">
    <property name="name">13</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">12</property>
  </packing>
</child>

```

```

<child>
  <object class="GtkLabel" id="55">
    <property name="name">14</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">13</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="56">
    <property name="name">15</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">14</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="57">
    <property name="name">16</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">15</property>
  </packing>
</child>
</child>

```

```

<object class="GtkLabel" id="58">
  <property name="name">17</property>
  <property name="visible">True</property>
  <property name="can_focus">False</property>
  <property name="label" translatable="yes"> </property>
  <property name="width_chars">1</property>
  <property name="max_width_chars">1</property>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">16</property>
</packing>
</child>
<child>
  <object class="GtkLabel" id="59">
    <property name="name">18</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">17</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="60">
    <property name="name">19</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">18</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="61">

```



```

    <property name="name">20</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">19</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="62">
    <property name="name">21</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">20</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="63">
    <property name="name">22</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">21</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="64">
    <property name="name">23</property>

```

```

    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">22</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="65">
    <property name="name">24</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">23</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="66">
    <property name="name">25</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">24</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="67">
    <property name="name">26</property>
    <property name="visible">True</property>

```

```

    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">25</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="68">
    <property name="name">27</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">26</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="69">
    <property name="name">28</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">27</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="70">
    <property name="name">29</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>

```

```

        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">28</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="71">
        <property name="name">30</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">29</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="72">
        <property name="name">31</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">30</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="73">
        <property name="name">32</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>

```

```

        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">31</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="74">
        <property name="name">33</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">32</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="75">
        <property name="name">34</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">33</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="76">
        <property name="name">35</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>

```

```

        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">34</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="77">
        <property name="name">36</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">35</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="78">
        <property name="name">37</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">36</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="79">
        <property name="name">38</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>

```

```

</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">37</property>
</packing>
</child>
<child>
  <object class="GtkLabel" id="80">
    <property name="name">39</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">38</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="81">
    <property name="name">40</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">39</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="82">
    <property name="name">41</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>

```

```

    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">40</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="83">
      <property name="name">42</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">41</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="84">
      <property name="name">43</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">42</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="85">
      <property name="name">44</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>

```



```

        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">43</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="86">
        <property name="name">45</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">44</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="87">
        <property name="name">46</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">45</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="88">
        <property name="name">47</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>

```

```

        <property name="fill">True</property>
        <property name="position">46</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="89">
        <property name="name">48</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">47</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="90">
        <property name="name">49</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">48</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="91">
        <property name="name">50</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>

```

```

        <property name="position">49</property>
    </packing>
</child>
</object>
<packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">0</property>
</packing>
</child>
<child>
    <object class="GtkBox" id="channel_02">
        <property name="name">02</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <child>
            <object class="GtkLabel" id="92">
                <property name="name">01</property>
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="label" translatable="yes"> </property>
                <property name="width_chars">1</property>
                <property name="max_width_chars">1</property>
            </object>
            <packing>
                <property name="expand">False</property>
                <property name="fill">True</property>
                <property name="position">0</property>
            </packing>
        </child>
        <child>
            <object class="GtkLabel" id="93">
                <property name="name">02</property>
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="label" translatable="yes"> </property>
                <property name="width_chars">1</property>
                <property name="max_width_chars">1</property>
            </object>
            <packing>
                <property name="expand">False</property>
                <property name="fill">True</property>
                <property name="position">1</property>
            </packing>
        </child>
    </child>

```

```

<object class="GtkLabel" id="94">
  <property name="name">03</property>
  <property name="visible">True</property>
  <property name="can_focus">False</property>
  <property name="label" translatable="yes"> </property>
  <property name="width_chars">1</property>
  <property name="max_width_chars">1</property>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">2</property>
</packing>
</child>
<child>
  <object class="GtkLabel" id="95">
    <property name="name">04</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">3</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="96">
    <property name="name">05</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">4</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="97">

```

```

    <property name="name">06</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">5</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="98">
    <property name="name">07</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">6</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="99">
    <property name="name">08</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">7</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="100">
    <property name="name">09</property>

```

```

    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">8</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="101">
    <property name="name">10</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">9</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="102">
    <property name="name">11</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">10</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="103">
    <property name="name">12</property>
    <property name="visible">True</property>

```

```

    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">11</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="104">
    <property name="name">13</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">12</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="105">
    <property name="name">14</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">13</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="106">
    <property name="name">15</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>

```

```

        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">14</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="107">
        <property name="name">16</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">15</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="108">
        <property name="name">17</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">16</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="109">
        <property name="name">18</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>

```



```

        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">17</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="110">
        <property name="name">19</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">18</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="111">
        <property name="name">20</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">19</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="112">
        <property name="name">21</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>

```

```

        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">20</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="113">
        <property name="name">22</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">21</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="114">
        <property name="name">23</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">22</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="115">
        <property name="name">24</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>

```

```

</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">23</property>
</packing>
</child>
<child>
  <object class="GtkLabel" id="116">
    <property name="name">25</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">24</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="117">
    <property name="name">26</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">25</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="118">
    <property name="name">27</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>

```

```

    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">26</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="119">
      <property name="name">28</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">27</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="120">
      <property name="name">29</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">28</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="121">
      <property name="name">30</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>

```

```

        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">29</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="122">
        <property name="name">31</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">30</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="123">
        <property name="name">32</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">31</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="124">
        <property name="name">33</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>

```

```

        <property name="fill">True</property>
        <property name="position">32</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="125">
        <property name="name">34</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">33</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="126">
        <property name="name">35</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">34</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="127">
        <property name="name">36</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>

```

```

        <property name="position">35</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="128">
        <property name="name">37</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">36</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="129">
        <property name="name">38</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">37</property>
    </packing>
</child>
<child>
    <object class="GtkLabel" id="130">
        <property name="name">39</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes"> </property>
        <property name="width_chars">1</property>
        <property name="max_width_chars">1</property>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">38</property>
    </packing>
</child>

```

```

    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="131">
      <property name="name">40</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">39</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="132">
      <property name="name">41</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">40</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="133">
      <property name="name">42</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes"> </property>
      <property name="width_chars">1</property>
      <property name="max_width_chars">1</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">41</property>
    </packing>
  </child>

```



```

</child>
<child>
  <object class="GtkLabel" id="134">
    <property name="name">43</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">42</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="135">
    <property name="name">44</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">43</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="136">
    <property name="name">45</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">44</property>
  </packing>
</child>

```

```

<child>
  <object class="GtkLabel" id="137">
    <property name="name">46</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">45</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="138">
    <property name="name">47</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">46</property>
  </packing>
</child>
<child>
  <object class="GtkLabel" id="139">
    <property name="name">48</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">47</property>
  </packing>
</child>
<child>

```

```

<object class="GtkLabel" id="140">
  <property name="name">49</property>
  <property name="visible">True</property>
  <property name="can_focus">False</property>
  <property name="label" translatable="yes"> </property>
  <property name="width_chars">1</property>
  <property name="max_width_chars">1</property>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">48</property>
</packing>
</child>
<child>
  <object class="GtkLabel" id="141">
    <property name="name">50</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes"> </property>
    <property name="width_chars">1</property>
    <property name="max_width_chars">1</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">49</property>
  </packing>
</child>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">1</property>
</packing>
</child>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">0</property>
</packing>
</child>
<child>
  <object class="GtkBox" id="box1">
    <property name="visible">True</property>

```

```

    <property name="can_focus">False</property>
    <property name="spacing">3</property>
    <child>
      <object class="GtkToggleButton" id="Mute">
        <property name="label" translatable="yes">Mute</property>
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property name="receives_default">True</property>
        <property name="xalign">0.56999999284744263</property>
        <property name="yalign">0.47999998927116394</property>
        <signal name="toggled" handler="button_mute_toggled"
→ object="top_level_window" swapped="no"/>
      </object>
      <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">0</property>
      </packing>
    </child>
    <child>
      <object class="GtkButton" id="Pause">
        <property name="label">gtk-media-pause</property>
        <property name="name">Pause</property>
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property name="receives_default">True</property>
        <property name="use_stock">True</property>
        <signal name="clicked" handler="button_pause_clicked"
→ object="top_level_window" swapped="no"/>
      </object>
      <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">1</property>
      </packing>
    </child>
    <child>
      <object class="GtkButton" id="continue">
        <property name="label" translatable="yes">Continue</property>
        <property name="name">Continue</property>
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property name="receives_default">True</property>
        <property name="xalign">0.55000001192092896</property>
        <signal name="clicked" handler="button_continue_clicked"
→ object="top_level_window" swapped="no"/>
      </object>

```

```

    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">2</property>
    </packing>
  </child>
  <child>
    <object class="GtkButton" id="play">
      <property name="label">gtk-media-play</property>
      <property name="name">play</property>
      <property name="visible">True</property>
      <property name="can_focus">True</property>
      <property name="receives_default">True</property>
      <property name="use_stock">True</property>
      <signal name="clicked" handler="button_play_clicked"
→ object="top_level_window" swapped="no"/>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">3</property>
    </packing>
  </child>
  <child>
    <object class="GtkLabel" id="operator_text">
      <property name="name">operator_text</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="label" translatable="yes">Operator
→ Text</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">4</property>
    </packing>
  </child>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">1</property>
</packing>
</child>
</object>

```

```

    <packing>
      <property name="expand">False</property>
      <property name="fill">True</property>
      <property name="position">0</property>
    </packing>
  </child>
  <child>
    <object class="GtkScrolledWindow" id="scrolledwindow1">
      <property name="visible">True</property>
      <property name="can_focus">True</property>
      <property name="shadow_type">in</property>
      <property name="min_content_width">500</property>
      <property name="min_content_height">180</property>
    <child>
      <object class="GtkViewport" id="viewport1">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="hscroll_policy">natural</property>
        <property name="vscroll_policy">natural</property>
      <child>
        <object class="GtkBox" id="frame_box">
          <property name="visible">True</property>
          <property name="can_focus">False</property>
        <child>
          <object class="GtkFrame" id="frame1">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label_xalign">0</property>
            <property name="label_yalign">1</property>
          <child>
            <object class="GtkAlignment" id="alignment1">
              <property name="visible">True</property>
              <property name="can_focus">False</property>
            <child>
              <object class="GtkGrid" id="cluster_00">
                <property name="name">cluster_00</property>
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="margin_start">5</property>
                <property name="margin_end">5</property>
                <property name="margin_top">5</property>
                <property name="margin_bottom">5</property>
                <property name="border_width">5</property>
              <child>
                <object class="GtkButton" id="button1">

```

```

        <property name="label"
→   translatable="yes">Start</property>
        <property name="name">start_button</property>
        <property name="width_request">80</property>
        <property name="height_request">50</property>
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property
→   name="receives_default">True</property>
        <signal name="clicked"
→   handler="button_start_clicked" object="cluster_00" swapped="no"/>
        </object>
        <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">1</property>
        </packing>
    </child>
    <child>
        <object class="GtkButton" id="button2">
        <property name="label"
→   translatable="yes">Stop</property>
        <property name="name">stop_button</property>
        <property name="width_request">50</property>
        <property name="height_request">50</property>
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property
→   name="receives_default">True</property>
        <signal name="clicked"
→   handler="button_stop_clicked" object="cluster_00" swapped="no"/>
        </object>
        <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">2</property>
        </packing>
    </child>
    <child>
        <object class="GtkLabel" id="label1">
        <property name="name">title</property>
        <property name="height_request">30</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label"
→   translatable="yes">Name of the Sound</property>
        <property name="ellipsize">end</property>
        </object>

```

```

        <packing>
        <property name="left_attach">0</property>
        <property name="top_attach">0</property>
        <property name="width">2</property>
        </packing>
    </child>
    <child>
        <object class="GtkGrid" id="grid2">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property
→ name="column_homogeneous">True</property>
        <child>
            <object class="GtkLabel" id="label2">
            <property
→ name="name">pan_label</property>
            <property name="visible">True</property>
            <property
→ name="can_focus">False</property>
            <property name="label"
→ translatable="yes">pan</property>
            <property
→ name="width_chars">10</property>
            <property
→ name="max_width_chars">10</property>
            </object>
            <packing>
            <property name="left_attach">0</property>
            <property name="top_attach">0</property>
            </packing>
        </child>
        <child>
            <object class="GtkScaleButton"
→ id="scalebutton1">
            <property
→ name="name">pan_button</property>
            <property name="visible">True</property>
            <property
→ name="can_focus">True</property>
            <property
→ name="receives_default">True</property>
            <property
→ name="focus_on_click">False</property>
            <property name="value">50</property>
            <signal name="value-changed"
→ handler="button_pan_changed" object="cluster_00" swapped="no"/>

```



```

        <child internal-child="plus_button">
          <object class="GtkButton"
→ id="scalebutton-plus_button1">
            <property
→ name="can_focus">True</property>
            <property
→ name="receives_default">True</property>
            <property
→ name="halign">center</property>
            <property
→ name="valign">center</property>
            <property
→ name="relief">none</property>
          </object>
        </child>
        <child internal-child="minus_button">
          <object class="GtkButton"
→ id="scalebutton-minus_button1">
            <property
→ name="can_focus">True</property>
            <property
→ name="receives_default">True</property>
            <property
→ name="halign">center</property>
            <property
→ name="valign">center</property>
            <property
→ name="relief">none</property>
          </object>
        </child>
      </object>
      <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">0</property>
      </packing>
    </child>
  </object>
  <packing>
    <property name="left_attach">0</property>
    <property name="top_attach">1</property>
  </packing>
</child>
<child>
  <object class="GtkGrid" id="grid3">
    <property name="visible">True</property>
    <property name="can_focus">False</property>

```

```

        <property
→  name="column_homogeneous">True</property>
        <child>
            <object class="GtkLabel" id="label3">
                <property
→  name="name">volume_label</property>
                <property name="visible">True</property>
                <property
→  name="can_focus">False</property>
                <property name="label"
→  translatable="yes">volume</property>
                <property name="width_chars">8</property>
                <property
→  name="max_width_chars">8</property>
            </object>
            <packing>
                <property name="left_attach">0</property>
                <property name="top_attach">0</property>
            </packing>
        </child>
        <child>
            <object class="GtkVolumeButton"
→  id="volumebutton1">
                <property name="name">volume</property>
                <property name="visible">True</property>
                <property
→  name="can_focus">True</property>
                <property
→  name="receives_default">True</property>
                <property
→  name="focus_on_click">False</property>
                <property name="value">1</property>
                <property
→  name="icons">audio-volume-muted-symbolic
audio-volume-high-symbolic
audio-volume-low-symbolic
audio-volume-medium-symbolic</property>
                <signal name="value-changed"
→  handler="button_volume_changed" object="cluster_00" swapped="no"/>
                <child internal-child="plus_button">
                    <object class="GtkButton"
→  id="volumebutton-plus_button1">
                        <property
→  name="can_focus">True</property>
                        <property
→  name="receives_default">True</property>

```

```

    <property
→   name="halign">center</property>
    <property
→   name="valign">center</property>
    <property
→   name="relief">none</property>
    </object>
  </child>
  <child internal-child="minus_button">
    <object class="GtkButton"
→   id="volumebutton-minus_button1">
      <property
→   name="can_focus">True</property>
      <property
→   name="receives_default">True</property>
      <property
→   name="halign">center</property>
      <property
→   name="valign">center</property>
      <property
→   name="relief">none</property>
    </object>
  </child>
</object>
<packing>
  <property name="left_attach">1</property>
  <property name="top_attach">0</property>
</packing>
</child>
</object>
<packing>
  <property name="left_attach">0</property>
  <property name="top_attach">2</property>
</packing>
</child>
</object>
</child>
</object>
</child>
<child type="label">
  <object class="GtkLabel" id="label14">
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes">cluster
→ 0</property>
  </object>

```

```

        </child>
    </object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">0</property>
    </packing>
</child>
<child>
    <object class="GtkFrame" id="frame2">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label_xalign">0</property>
        <property name="label_yalign">1</property>
        <child>
            <object class="GtkAlignment" id="alignment2">
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <child>
                    <object class="GtkGrid" id="cluster_01">
                        <property name="name">cluster_01</property>
                        <property name="visible">True</property>
                        <property name="can_focus">False</property>
                        <property name="margin_top">5</property>
                        <property name="margin_bottom">5</property>
                        <property name="border_width">5</property>
                        <child>
                            <object class="GtkButton" id="button3">
                                <property name="label"
↪   translatable="yes">Start</property>
                                    <property name="name">start_button</property>
                                    <property name="width_request">80</property>
                                    <property name="height_request">50</property>
                                    <property name="visible">True</property>
                                    <property name="can_focus">True</property>
                                    <property
↪   name="receives_default">True</property>
                                    <signal name="clicked"
↪   handler="button_start_clicked" object="cluster_01" swapped="no"/>
                                </object>
                            <packing>
                                <property name="left_attach">1</property>
                                <property name="top_attach">1</property>
                            </packing>
                        </child>
                    </child>
                </child>
            </object>
        </child>
    </object>

```

```

        <object class="GtkButton" id="button4">
          <property name="label"
→   translatable="yes">Stop</property>
            <property name="name">stop_button</property>
            <property name="width_request">50</property>
            <property name="height_request">50</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property
→   name="receives_default">True</property>
            <signal name="clicked"
→   handler="button_stop_clicked" object="cluster_01" swapped="no"/>
          </object>
          <packing>
            <property name="left_attach">1</property>
            <property name="top_attach">2</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="label4">
            <property name="name">title</property>
            <property name="height_request">30</property>
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label"
→   translatable="yes">Name of the Sound</property>
            <property name="ellipsize">end</property>
          </object>
          <packing>
            <property name="left_attach">0</property>
            <property name="top_attach">0</property>
            <property name="width">2</property>
          </packing>
        </child>
        <child>
          <object class="GtkGrid" id="grid6">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property
→   name="column_homogeneous">True</property>
            <child>
              <object class="GtkLabel" id="label5">
                <property
→   name="name">pan_label</property>
                <property name="visible">True</property>

```

```

    ↪ name="can_focus">False</property>
    ↪ translatable="yes">pan</property>
    ↪ name="width_chars">10</property>
    ↪ name="max_width_chars">10</property>
    </object>
    <packing>
    <property name="left_attach">0</property>
    <property name="top_attach">0</property>
    </packing>
  </child>
  <child>
    <object class="GtkScaleButton"
    ↪ id="scalebutton2">
    <property
    ↪ name="name">pan_button</property>
    <property name="visible">True</property>
    <property
    ↪ name="can_focus">True</property>
    <property
    ↪ name="receives_default">True</property>
    <property
    ↪ name="focus_on_click">False</property>
    <property name="value">50</property>
    <signal name="value-changed"
    ↪ handler="button_pan_changed" object="cluster_01" swapped="no"/>
    <child internal-child="plus_button">
      <object class="GtkButton"
    ↪ id="scalebutton-plus_button">
      <property
    ↪ name="can_focus">True</property>
      <property
    ↪ name="receives_default">True</property>
      <property
    ↪ name="halign">center</property>
      <property
    ↪ name="valign">center</property>
      <property
    ↪ name="relief">none</property>
    </object>
    </child>
    <child internal-child="minus_button">

```

```

        <object class="GtkButton"
→   id="scalebutton-minus_button">
        <property
→   name="can_focus">True</property>
        <property
→   name="receives_default">True</property>
        <property
→   name="halign">center</property>
        <property
→   name="valign">center</property>
        <property
→   name="relief">none</property>
        </object>
        </child>
        </object>
        <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">0</property>
        </packing>
        </child>
        </object>
        <packing>
        <property name="left_attach">0</property>
        <property name="top_attach">1</property>
        </packing>
        </child>
        <child>
        <object class="GtkGrid" id="grid7">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property
→   name="column_homogeneous">True</property>
        <child>
        <object class="GtkLabel" id="label6">
        <property
→   name="name">volume_label</property>
        <property name="visible">True</property>
        <property
→   name="can_focus">False</property>
        <property name="label"
→   translatable="yes">volume</property>
        <property
→   name="max_width_chars">11</property>
        </object>
        <packing>
        <property name="left_attach">0</property>

```

```

        <property name="top_attach">0</property>
    </packing>
</child>
<child>
    <object class="GtkVolumeButton"
↳ id="volumebutton2">
        <property name="name">volume</property>
        <property name="visible">True</property>
        <property
↳ name="can_focus">True</property>
        <property
↳ name="receives_default">True</property>
        <property
↳ name="focus_on_click">False</property>
        <property name="value">1</property>
        <property
↳ name="icons">audio-volume-muted-symbolic
audio-volume-high-symbolic
audio-volume-low-symbolic
audio-volume-medium-symbolic</property>
        <signal name="value-changed"
↳ handler="button_volume_changed" object="cluster_01" swapped="no"/>
        <child internal-child="plus_button">
            <object class="GtkButton"
↳ id="volumebutton-plus_button">
                <property
↳ name="can_focus">True</property>
                <property
↳ name="receives_default">True</property>
                <property
↳ name="halign">center</property>
                <property
↳ name="valign">center</property>
                <property
↳ name="relief">none</property>
            </object>
        </child>
        <child internal-child="minus_button">
            <object class="GtkButton"
↳ id="volumebutton-minus_button">
                <property
↳ name="can_focus">True</property>
                <property
↳ name="receives_default">True</property>
                <property
↳ name="halign">center</property>

```



```

        <property
→   name="valign">center</property>
        <property
→   name="relief">none</property>
        </object>
    </child>
</object>
    <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">0</property>
    </packing>
</child>
</object>
    <packing>
        <property name="left_attach">0</property>
        <property name="top_attach">2</property>
    </packing>
</child>
</object>
</child>
</object>
</child>
<child type="label">
    <object class="GtkLabel" id="label15">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label" translatable="yes">cluster
→ 1</property>
    </object>
</child>
</object>
    <packing>
        <property name="expand">False</property>
        <property name="fill">True</property>
        <property name="position">1</property>
    </packing>
</child>
<child>
    <object class="GtkFrame" id="frame3">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label_xalign">0</property>
        <property name="label_yalign">1</property>
    <child>
        <object class="GtkAlignment" id="alignment3">
            <property name="visible">True</property>

```

```

<property name="can_focus">False</property>
<child>
  <object class="GtkGrid" id="cluster_02">
    <property name="name">cluster_02</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="margin_top">5</property>
    <property name="margin_bottom">5</property>
    <property name="border_width">5</property>
    <child>
      <object class="GtkButton" id="button5">
        <property name="label"
→   translatable="yes">Start</property>
          <property name="name">start_button</property>
          <property name="width_request">80</property>
          <property name="height_request">50</property>
          <property name="visible">True</property>
          <property name="can_focus">True</property>
          <property
→   name="receives_default">True</property>
            <signal name="clicked"
→   handler="button_start_clicked" object="cluster_02" swapped="no"/>
        </object>
        <packing>
          <property name="left_attach">1</property>
          <property name="top_attach">1</property>
        </packing>
      </child>
      <child>
        <object class="GtkButton" id="button6">
          <property name="label"
→   translatable="yes">Stop</property>
            <property name="name">stop_button</property>
            <property name="width_request">50</property>
            <property name="height_request">50</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property
→   name="receives_default">True</property>
              <signal name="clicked"
→   handler="button_stop_clicked" object="cluster_02" swapped="no"/>
          </object>
          <packing>
            <property name="left_attach">1</property>
            <property name="top_attach">2</property>
          </packing>
        </child>
      </child>
    </child>
  </object>

```

```

</child>
<child>
  <object class="GtkLabel" id="label7">
    <property name="name">title</property>
    <property name="height_request">30</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label"
→   translatable="yes">Name of the Sound</property>
      <property name="ellipsize">end</property>
    </object>
    <packing>
      <property name="left_attach">0</property>
      <property name="top_attach">0</property>
      <property name="width">2</property>
    </packing>
  </child>
  <child>
    <object class="GtkGrid" id="grid9">
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property
→   name="column_homogeneous">True</property>
      <child>
        <object class="GtkLabel" id="label8">
          <property
→   name="name">pan_label</property>
            <property name="visible">True</property>
            <property
→   name="can_focus">False</property>
            <property name="label"
→   translatable="yes">pan</property>
            <property
→   name="width_chars">10</property>
            <property
→   name="max_width_chars">10</property>
          </object>
          <packing>
            <property name="left_attach">0</property>
            <property name="top_attach">0</property>
          </packing>
        </child>
      <child>
        <object class="GtkScaleButton"
→   id="scalebutton3">

```

```

    ↪ name="name">pan_button</property>
    ↪ name="can_focus">True</property>
    ↪ name="receives_default">True</property>
    ↪ name="focus_on_click">False</property>
    ↪ handler="button_pan_changed" object="cluster_02" swapped="no"/>
    ↪ id="scalebutton-plus_button3">
    ↪ name="can_focus">True</property>
    ↪ name="receives_default">True</property>
    ↪ name="halign">center</property>
    ↪ name="valign">center</property>
    ↪ name="relief">none</property>
    ↪ id="scalebutton-minus_button3">
    ↪ name="can_focus">True</property>
    ↪ name="receives_default">True</property>
    ↪ name="halign">center</property>
    ↪ name="valign">center</property>
    ↪ name="relief">none</property>

```

```

    <property
    <property name="visible">True</property>
    <property
    <property
    <property
    <property
    <property name="value">50</property>
    <signal name="value-changed"
    <child internal-child="plus_button">
    <object class="GtkButton"

    <property
    <property
    <property
    <property
    <property
    <property
    </object>
    </child>
    <child internal-child="minus_button">
    <object class="GtkButton"

    <property
    <property
    <property
    <property
    <property
    <property
    <property
    </object>
    </child>
    </object>
    <packing>
    <property name="left_attach">1</property>
    <property name="top_attach">0</property>

```

```

        </packing>
    </child>
</object>
<packing>
    <property name="left_attach">0</property>
    <property name="top_attach">1</property>
</packing>
</child>
<child>
    <object class="GtkGrid" id="grid10">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property
→ name="column_homogeneous">True</property>
        <child>
            <object class="GtkLabel" id="label9">
                <property
→ name="name">volume_label</property>
                <property name="visible">True</property>
                <property
→ name="can_focus">False</property>
                <property name="label"
→ translatable="yes">volume</property>
            </object>
            <packing>
                <property name="left_attach">0</property>
                <property name="top_attach">0</property>
            </packing>
        </child>
        <child>
            <object class="GtkVolumeButton"
→ id="volumebutton3">
                <property name="name">volume</property>
                <property name="visible">True</property>
                <property
→ name="can_focus">True</property>
                <property
→ name="receives_default">True</property>
                <property
→ name="focus_on_click">False</property>
                <property name="value">1</property>
                <property
→ name="icons">audio-volume-muted-symbolic
audio-volume-high-symbolic
audio-volume-low-symbolic
audio-volume-medium-symbolic</property>

```

```

        <signal name="value-changed"
→ handler="button_volume_changed" object="cluster_02" swapped="no"/>
        <child internal-child="plus_button">
            <object class="GtkButton"
→ id="volumebutton-plus_button3">
                <property
→ name="can_focus">True</property>
                <property
→ name="receives_default">True</property>
                <property
→ name="halign">center</property>
                <property
→ name="valign">center</property>
                <property
→ name="relief">none</property>
            </object>
        </child>
        <child internal-child="minus_button">
            <object class="GtkButton"
→ id="volumebutton-minus_button3">
                <property
→ name="can_focus">True</property>
                <property
→ name="receives_default">True</property>
                <property
→ name="halign">center</property>
                <property
→ name="valign">center</property>
                <property
→ name="relief">none</property>
            </object>
        </child>
    </object>
    <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">0</property>
    </packing>
</child>
</object>
<packing>
    <property name="left_attach">0</property>
    <property name="top_attach">2</property>
</packing>
</child>
</object>
</child>

```

```

        </object>
    </child>
    <child type="label">
        <object class="GtkLabel" id="label16">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">cluster
→ 2</property>

        </object>
    </child>
</object>
<packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">2</property>
</packing>
</child>
<child>
    <object class="GtkFrame" id="frame4">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label_xalign">0</property>
        <property name="label_yalign">1</property>
    <child>
        <object class="GtkAlignment" id="alignment4">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
        <child>
            <object class="GtkGrid" id="cluster_03">
                <property name="name">cluster_03</property>
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="margin_start">5</property>
                <property name="margin_end">5</property>
                <property name="margin_top">5</property>
                <property name="margin_bottom">5</property>
                <property name="border_width">5</property>
            <child>
                <object class="GtkButton" id="button7">
                    <property name="label"
→ translatable="yes">Start</property>
                    <property name="name">start_button</property>
                    <property name="width_request">80</property>
                    <property name="height_request">50</property>
                    <property name="visible">True</property>
                    <property name="can_focus">True</property>

```

```

        <property
→  name="receives_default">True</property>
        <signal name="clicked"
→  handler="button_start_clicked" object="cluster_03" swapped="no"/>
        </object>
        <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">1</property>
        </packing>
    </child>
    <child>
        <object class="GtkButton" id="button8">
        <property name="label"
→  translatable="yes">Stop</property>
        <property name="name">stop_button</property>
        <property name="width_request">50</property>
        <property name="height_request">50</property>
        <property name="visible">True</property>
        <property name="can_focus">True</property>
        <property
→  name="receives_default">True</property>
        <signal name="clicked"
→  handler="button_stop_clicked" object="cluster_03" swapped="no"/>
        </object>
        <packing>
        <property name="left_attach">1</property>
        <property name="top_attach">2</property>
        </packing>
    </child>
    <child>
        <object class="GtkLabel" id="label10">
        <property name="name">title</property>
        <property name="height_request">30</property>
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property name="label"
→  translatable="yes">Name of the Sound</property>
        <property name="ellipsize">end</property>
        </object>
        <packing>
        <property name="left_attach">0</property>
        <property name="top_attach">0</property>
        <property name="width">2</property>
        </packing>
    </child>
</child>

```



```

<object class="GtkGrid" id="grid12">
  <property name="visible">True</property>
  <property name="can_focus">False</property>
  <property
→ name="column_homogeneous">True</property>
  <child>
    <object class="GtkLabel" id="label11">
      <property
→ name="name">pan_label</property>
      <property name="visible">True</property>
      <property
→ name="can_focus">False</property>
      <property name="label"
→ translatable="yes">pan</property>
      <property
→ name="width_chars">10</property>
      <property
→ name="max_width_chars">10</property>
    </object>
    <packing>
      <property name="left_attach">0</property>
      <property name="top_attach">0</property>
    </packing>
  </child>
  <child>
    <object class="GtkScaleButton"
→ id="scalebutton4">
      <property
→ name="name">pan_button</property>
      <property name="visible">True</property>
      <property
→ name="can_focus">True</property>
      <property
→ name="receives_default">True</property>
      <property
→ name="focus_on_click">False</property>
      <property name="value">50</property>
      <signal name="value-changed"
→ handler="button_pan_changed" object="cluster_03" swapped="no"/>
      <child internal-child="plus_button">
        <object class="GtkButton"
→ id="scalebutton-plus_button5">
          <property
→ name="can_focus">True</property>
          <property
→ name="receives_default">True</property>

```

```

        <property
→  name="halign">center</property>
        <property
→  name="valign">center</property>
        <property
→  name="relief">none</property>
        </object>
    </child>
    <child internal-child="minus_button">
        <object class="GtkButton"
→  id="scalebutton-minus_button5">
            <property
→  name="can_focus">True</property>
            <property
→  name="receives_default">True</property>
            <property
→  name="halign">center</property>
            <property
→  name="valign">center</property>
            <property
→  name="relief">none</property>
            </object>
        </child>
        </object>
        <packing>
            <property name="left_attach">1</property>
            <property name="top_attach">0</property>
        </packing>
    </child>
</object>
<packing>
    <property name="left_attach">0</property>
    <property name="top_attach">1</property>
</packing>
</child>
<child>
    <object class="GtkGrid" id="grid13">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <property
→  name="column_homogeneous">True</property>
        <child>
            <object class="GtkLabel" id="label12">
                <property
→  name="name">volume_label</property>
                <property name="visible">True</property>

```

```

        <property
→   name="can_focus">False</property>
        <property name="label"
→   translatable="yes">volume</property>
        <property
→   name="width_chars">10</property>
        <property
→   name="max_width_chars">10</property>
        </object>
        <packing>
        <property name="left_attach">0</property>
        <property name="top_attach">0</property>
        </packing>
    </child>
    <child>
        <object class="GtkVolumeButton"
→   id="volumebutton4">
            <property name="name">volume</property>
            <property name="visible">True</property>
            <property
→   name="can_focus">True</property>
            <property
→   name="receives_default">True</property>
            <property
→   name="focus_on_click">False</property>
            <property name="value">1</property>
            <property
→   name="icons">audio-volume-muted-symbolic
audio-volume-high-symbolic
audio-volume-low-symbolic
audio-volume-medium-symbolic</property>
            <signal name="value-changed"
→   handler="button_volume_changed" object="cluster_03" swapped="no"/>
            <child internal-child="plus_button">
                <object class="GtkButton"
→   id="volumebutton-plus_button5">
                    <property
→   name="can_focus">True</property>
                    <property
→   name="receives_default">True</property>
                    <property
→   name="halign">center</property>
                    <property
→   name="valign">center</property>
                    <property
→   name="relief">none</property>

```

```

        </object>
      </child>
      <child internal-child="minus_button">
        <object class="GtkButton"
→ id="volumebutton-minus_button5">
          <property
→ name="can_focus">True</property>
          <property
→ name="receives_default">True</property>
          <property
→ name="halign">center</property>
          <property
→ name="valign">center</property>
          <property
→ name="relief">none</property>
        </object>
      </child>
    </object>
    <packing>
      <property name="left_attach">1</property>
      <property name="top_attach">0</property>
    </packing>
  </child>
</object>
<packing>
  <property name="left_attach">0</property>
  <property name="top_attach">2</property>
</packing>
</child>
</object>
</child>
</object>
</child>
<child type="label">
  <object class="GtkLabel" id="label17">
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="label" translatable="yes">cluster
→ 3</property>
  </object>
</child>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">3</property>

```

```

        </packing>
      </child>
    </object>
  </child>
</object>
</child>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">1</property>
</packing>
</child>
<child>
  <object class="GtkStatusbar" id="status_bar">
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="margin_start">10</property>
    <property name="margin_end">10</property>
    <property name="margin_top">6</property>
    <property name="margin_bottom">6</property>
    <property name="orientation">vertical</property>
    <property name="spacing">2</property>
  </object>
  <packing>
    <property name="expand">False</property>
    <property name="fill">True</property>
    <property name="position">2</property>
  </packing>
</child>
</object>
</child>
</object>
</interface>

```

33 sound_structure.h

```

/*
 * sound_structure.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/* Only define the structure once per source file. */
#ifndef SOUND_STRUCTURE_H
#define SOUND_STRUCTURE_H

/* Define the structure which holds the definition of a sound.
 * This structure is shared between sound_effects_player,
 * parse_xml_subroutines, button_subroutines, sound_subroutines
 * and sequence_subroutines. */

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

struct sound_info
{
    gchar *name; /* name of the sound */
    gboolean disabled; /* disabled because file is missing */
    gchar *wav_file_name; /* name of the file holding the waveform */
    gchar *wav_file_name_full; /* absolute path to the file */
    guint64 attack_duration_time; /* attack time, in nanoseconds */
    gdouble attack_level; /* 1.0 means 100% of volume */
    guint64 decay_duration_time; /* decay time, in nanoseconds */
    gdouble sustain_level; /* 1.0 means 100% of volume */
    guint64 release_start_time; /* release start time, in nanoseconds */
    guint64 release_duration_time; /* release duration time,
                                   * in nanoseconds */

```

```

gboolean release_duration_infinite; /* TRUE if duration is
                                     infinite */
gint64 loop_from_time; /* loop from time, in nanoseconds */
gint64 loop_to_time; /* loop to time, in nanoseconds */
gint loop_limit; /* loop limit, a count */
guint64 max_duration_time; /* maximum time taken from WAV file,
                             * in nanoseconds. */
guint64 start_time; /* start time, in nanoseconds */
gfloat designer_volume_level; /* 1.0 means 100% of waveform's volume */
gfloat designer_pan; /* -1.0 is left, 0.0 is center, 1.0 is right */
gint MIDI_program_number; /* MIDI program number, if specified */
gboolean MIDI_program_number_specified; /* TRUE if not empty */
gint MIDI_note_number; /* MIDI note number, if specified */
gboolean MIDI_note_number_specified; /* TRUE if not empty */
gchar *OSC_name; /* name used by OSC to activate */
gboolean OSC_name_specified; /* TRUE if not empty */
gchar *function_key; /* name of function key */
gboolean function_key_specified; /* TRUE if not empty */
GtkWidget *cluster_widget; /* The cluster this sound is in. */
GstBin *sound_control; /* The Gstreamer bin for this sound effect */
gint cluster_number; /* The number of the cluster the sound is in */
gboolean running; /* The sound is playing. */
gboolean release_sent; /* A Release command was given. */
gboolean release_has_started; /* The sound has started its release stage. */
gboolean omit_panning; /* Do not let the operator pan this sound. */
};

#endif /* ifndef SOUND_STRUCTURE_H */
/* End of file sound_structure.h */

```

34 sound_subroutines.c

```

/*
 * sound_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <stdlib.h>
#include <gst/gst.h>
#include "sound_subroutines.h"
#include "sound_structure.h"
#include "sound_effects_player.h"
#include "gstreamer_subroutines.h"
#include "button_subroutines.h"
#include "display_subroutines.h"
#include "sequence_subroutines.h"

/* Subroutines for processing sounds. */

/* Initialize the sound system. We have already read an XML file
 * containing sound definitions and put the results in the sound list. */
GstPipeline *
sound_init (GApplication * app)
{
    GstPipeline *pipeline_element;
    GstBin *bin_element;
    GList *sound_list;
    gint sound_number, sound_count;
    GList *l;
    struct sound_info *sound_data;

    sound_list = sep_get_sound_list (app);

```



```

/* Count the non-disabled sounds. */
sound_count = 0;
for (l = sound_list; l != NULL; l = l->next)
{
    sound_data = l->data;
    if (!sound_data->disabled)
        sound_count = sound_count + 1;
}

/* If we have any sounds, create the gstreamer pipeline and place the
 * sound effects bins in it. */
if (sound_count == 0)
{
    return NULL;
}

pipeline_element = gstreamer_init (sound_count, app);
if (pipeline_element == NULL)
{
    /* We are unable to create the gstreamer pipeline. */
    return pipeline_element;
}

/* Create a gstreamer bin for each enabled sound effect and place it in
 * the gstreamer pipeline. */
sound_number = 0;
for (l = sound_list; l != NULL; l = l->next)
{
    sound_data = l->data;
    if (!sound_data->disabled)
    {
        bin_element =
            gstreamer_create_bin (sound_data, sound_number, pipeline_element,
                                app);

        if (bin_element == NULL)
        {
            /* We are unable to create the gstreamer bin. This might
             * be because an element is unavailable. */
            sound_data->disabled = TRUE;
            sound_count = sound_count - 1;
            continue;
        }

        sound_data->sound_control = bin_element;
    }
}

```

```

        sound_number = sound_number + 1;
    }
}

/* If we have any sound effects, complete the gstreamer pipeline. */
if (sound_count > 0)
{
    gstreamer_complete_pipeline (pipeline_element, app);
}
else
{
    /* Since we have no sound effects, we don't need a pipeline. */
    g_object_unref (pipeline_element);
    pipeline_element = NULL;
}

return pipeline_element;
}

/* Set the name displayed in a cluster. */
void
sound_cluster_set_name (gchar * sound_name, guint cluster_number,
                        GApplication * app)
{
    GList *children_list;
    const char *child_name;
    GtkLabel *title_label;
    GtkWidget *cluster;

    /* find the cluster */
    cluster = sep_get_cluster_from_number (cluster_number, app);

    if (cluster == NULL)
        return;

    /* Set the name in the cluster. */
    title_label = NULL;
    children_list = gtk_container_get_children (GTK_CONTAINER (cluster));
    while (children_list != NULL)
    {
        child_name = gtk_widget_get_name (children_list->data);
        if (g_strcmp0 (child_name, "title") == 0)
        {
            title_label = children_list->data;
            break;
        }
    }
}

```

```

        children_list = children_list->next;
    }
    g_list_free (children_list);

    if (title_label != NULL)
    {
        gtk_label_set_label (title_label, sound_name);
    }
}

/* Append a sound to the list of sounds. */
void
sound_append_sound (struct sound_info *sound_effect, GApplication * app)
{
    GList *sound_list;

    sound_list = sep_get_sound_list (app);
    sound_list = g_list_append (sound_list, sound_effect);
    sep_set_sound_list (sound_list, app);
    return;
}

/* Associate a sound with a specified cluster. */
struct sound_info *
sound_bind_to_cluster (gchar * sound_name, guint cluster_number,
                      GApplication * app)
{
    GList *sound_effect_list;
    GtkWidget *cluster_widget;
    struct sound_info *sound_effect;
    gboolean sound_effect_found;

    sound_effect_list = sep_get_sound_list (app);
    sound_effect_found = FALSE;
    while (sound_effect_list != NULL)
    {
        sound_effect = sound_effect_list->data;
        if ((g_strcmp0 (sound_name, sound_effect->name)) == 0)
        {
            sound_effect_found = TRUE;
            break;
        }
        sound_effect_list = sound_effect_list->next;
    }

    if (!sound_effect_found)

```

```

    return NULL;

    cluster_widget = sep_get_cluster_from_number (cluster_number, app);
    sound_effect->cluster_number = cluster_number;
    sound_effect->cluster_widget = cluster_widget;

    return sound_effect;
}

/* Disassociate a sound from its cluster. */
void
sound_unbind_from_cluster (struct sound_info *sound_effect,
                           GApplication * app)
{
    sound_effect->cluster_number = 0;
    sound_effect->cluster_widget = NULL;

    return;
}

/* Start playing a sound effect. */
void
sound_start_playing (struct sound_info *sound_data, GApplication * app)
{
    GstBin *bin_element;
    GstEvent *event;
    GstStructure *structure;

    bin_element = sound_data->sound_control;
    if (bin_element == NULL)
        return;

    /* If the sound has already been started, and is not yet releasing,
     * don't try to start it again. A sound is releasing if we have sent
     * a release message or if it has entered its release stage on its own. */
    if (sound_data->running && !sound_data->release_sent
        && !sound_data->release_has_started)
    {
        return;
    }

    /* Send a start message to the bin. It will be routed to the source, and
     * flow from there downstream through the looper and envelope.
     * The looper element will start sending its local buffer
     * and the envelope element will start to shape the volume. */

```

```

    sound_data->running = TRUE;
    sound_data->release_sent = FALSE;
    sound_data->release_has_started = FALSE;
    structure = gst_structure_new_empty ((gchar *) "start");
    event = gst_event_new_custom (GST_EVENT_CUSTOM_UPSTREAM, structure);
    gst_element_send_event (GST_ELEMENT (bin_element), event);

    return;
}

/* Stop playing a sound effect. */
void
sound_stop_playing (struct sound_info *sound_data, GApplication * app)
{
    GstBin *bin_element;
    GstEvent *event;
    GstStructure *structure;

    bin_element = sound_data->sound_control;

    /* Send a release message to the bin. The looper element will stop
     * looping, and the envelope element will start shutting down the sound.
     * If the sound has a non-zero release time we should get a call to
     * release_started shortly, unless the sound has already completed
     * and the message is still on its way down the pipeline. */
    structure = gst_structure_new_empty ((gchar *) "release");
    event = gst_event_new_custom (GST_EVENT_CUSTOM_UPSTREAM, structure);
    gst_element_send_event (GST_ELEMENT (bin_element), event);

    sound_data->release_sent = TRUE;

    return;
}

/* Get the elapsed time of a playing sound. */
gchar *
sound_get_elapsed_time (struct sound_info * sound_data, GApplication * app)
{
    GstElement *looper_element;
    gchar *string_value;

    looper_element = gstreamer_get_looper (sound_data->sound_control);
    g_object_get (looper_element, (gchar *) "elapsed-time", &string_value,
                  NULL);
    return string_value;
}

```

```

/* Get the remaining run time of a playing sound. */
gchar *
sound_get_remaining_time (struct sound_info * sound_data, GApplication * app)
{
    GstElement *looper_element;
    gchar *string_value;

    looper_element = gstreamer_get_looper (sound_data->sound_control);
    g_object_get (looper_element, (gchar *) "remaining-time", &string_value,
                  NULL);
    return string_value;
}

/* Receive a completed message, which indicates that a sound has finished. */
void
sound_completed (const gchar * sound_name, GApplication * app)
{
    GList *sound_effect_list;
    struct sound_info *sound_effect = NULL;
    gboolean sound_effect_found;

    /* Search through the sound effects for the one with this name. */
    sound_effect_list = sep_get_sound_list (app);
    sound_effect_found = FALSE;
    while (sound_effect_list != NULL)
    {
        sound_effect = sound_effect_list->data;
        if (g_strcmp0 (sound_effect->name, sound_name) == 0)
        {
            sound_effect_found = TRUE;
            break;
        }
        sound_effect_list = sound_effect_list->next;
    }

    /* There isn't one--ignore the completion. */
    if (!sound_effect_found)
        return;

    /* Flag that the sound is no longer playing. */
    sound_effect->running = FALSE;

    /* Let the internal sequencer distinguish a sound that has completed
     * normally from one that has been stopped. */
    sequence_sound_completion (sound_effect, sound_effect->release_sent, app);
}

```

```

    return;
}

/* Receive a release_started message, which indicates that a sound has entered
 * its release stage. */
void
sound_release_started (const gchar * sound_name, GApplication * app)
{
    GList *sound_effect_list;
    struct sound_info *sound_effect = NULL;
    gboolean sound_effect_found;

    /* Search through the sound effects for the one with this name. */
    sound_effect_list = sep_get_sound_list (app);
    sound_effect_found = FALSE;
    while (sound_effect_list != NULL)
    {
        sound_effect = sound_effect_list->data;
        if (g_strcmp0 (sound_effect->name, sound_name) == 0)
        {
            sound_effect_found = TRUE;
            break;
        }
        sound_effect_list = sound_effect_list->next;
    }

    /* If there isn't one, ignore the termination message. */
    if (!sound_effect_found)
        return;

    /* Remember that the sound is in its release stage. */
    sound_effect->release_has_started = TRUE;

    /* Let the internal sequencer handle it. */
    sequence_sound_release_started (sound_effect, app);

    return;
}

/* The Pause button was pushed. */
void
sound_button_pause (GApplication * app)
{
    GList *sound_list;
    GList *l;
    struct sound_info *sound_data;

```

```

GstBin *bin_element;
GstEvent *event;
GstStructure *structure;

sound_list = sep_get_sound_list (app);

/* Go through the non-disabled sounds, sending each a pause command. */
for (l = sound_list; l != NULL; l = l->next)
{
    sound_data = l->data;
    if (!sound_data->disabled)
    {
        bin_element = sound_data->sound_control;

        /* Send a pause message to the bin. The looper element will stop
         * advancing its pointer, sending silence instead, and the envelope
         * element will stop advancing through its timeline. */
        structure = gst_structure_new_empty ((gchar *) "pause");
        event = gst_event_new_custom (GST_EVENT_CUSTOM_UPSTREAM, structure);
        gst_element_send_event (GST_ELEMENT (bin_element), event);
    }
}

return;
}

/* The Continue button was pushed. */
void
sound_button_continue (GApplication * app)
{
    GList *sound_list;
    GList *l;
    struct sound_info *sound_data;
    GstBin *bin_element;
    GstEvent *event;
    GstStructure *structure;

    sound_list = sep_get_sound_list (app);

    /* Go through the non-disabled sounds, sending each a continue command. */
    for (l = sound_list; l != NULL; l = l->next)
    {
        sound_data = l->data;
        if (!sound_data->disabled)
        {
            bin_element = sound_data->sound_control;

```



```
    /* Send a continue message to the bin. The looper element will  
    * return to advancing its pointer, and the envelope element will  
    * return to advancing through its timeline. */  
    structure = gst_structure_new_empty ((gchar *) "continue");  
    event = gst_event_new_custom (GST_EVENT_CUSTOM_UPSTREAM, structure);  
    gst_element_send_event (GST_ELEMENT (bin_element), event);  
}  
  
}  
  
return;  
}
```

35 sound_subroutines.h

```

/*
 * sound_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

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

/* Subroutines defined in sound_subroutines.c */

/* Initialize the sounds. */
GstPipeline *sound_init (GApplication * app);

/* Set the name of a cluster. */
void sound_cluster_set_name (gchar * sound_name, guint cluster_number,
                             GApplication * app);

/* Append a sound to the list of sounds. */
void sound_append_sound (struct sound_info *sound_data, GApplication * app);

/* Associate a sound with a cluster. */
struct sound_info *sound_bind_to_cluster (gchar * sound_name,
                                           guint cluster_number,
                                           GApplication * app);

/* Disassociate a sound from its cluster. */
void sound_unbind_from_cluster (struct sound_info *sound_data,
                                GApplication * app);

```

```
/* Start playing a sound. */
void sound_start_playing (struct sound_info *sound_data, GApplication * app);

/* Stop playing a sound. */
void sound_stop_playing (struct sound_info *sound_data, GApplication * app);

/* Get the elapsed time of a playing sound. */
gchar *sound_get_elapsed_time (struct sound_info *sound_data,
                               GApplication * app);

/* Get the remaining time of a playing sound. */
gchar *sound_get_remaining_time (struct sound_info *sound_data,
                                 GApplication * app);

/* Note that a sound has completed. */
void sound_completed (const gchar * sound_name, GApplication * app);

/* Note that a sound has entered the release stage of its amplitude envelope.
   */
void sound_release_started (const gchar * sound_name, GApplication * app);

/* The Pause button has been pushed. */
void sound_button_pause (GApplication * app);

/* The Continue button has been pushed. */
void sound_button_continue (GApplication * app);

/* End of file sound_subroutines.h */
```

36 timer__subroutines.c

```

/*
 * timer_subroutines.c
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <glib.h>
#include <glib-unix.h>
#include <time.h>
#include <gtk/gtk.h>
#include <gst/gst.h>
#include "timer_subroutines.h"
#include "sound_effects_player.h"

/* When debugging it can be useful to trace what is happening in the
 * timer. */
#define TRACE_TIMER FALSE

/* the persistent data used by the timer */
struct timer_info
{
    gdouble last_trace_time;
    GList *timer_entry_list; /* The list of timer items */
    guint tick_source;
};

/* an entry on the timer list */
struct timer_entry_info
{
    gdouble expiration_time; /* When to call the subroutine */
    void (*subroutine) (void *, GApplication *); /* The subroutine to call */

```

```

    void *user_data;                /* The first parameter to pass to the
                                    * subroutine */
};

/* Forward declarations, so I can call these subroutines before I define them.
 */
static gboolean timer_tick (gpointer user_data);

/* Initialize the timer. */
void *
timer_init (GApplication * app)
{
    struct timer_info *timer_data;

    /* Allocate the persistent data. */
    timer_data = g_malloc (sizeof (struct timer_info));

    if (TRACE_TIMER)
    {
        timer_data->last_trace_time = g_get_monotonic_time () / 1e6;
    }

    /* The list of timer entries is empty. */
    timer_data->timer_entry_list = NULL;

    /* Specify where to go on each tick. */
    timer_data->tick_source = g_timeout_add (100, timer_tick, app);

    return (timer_data);
}

/* Shut down the timer. */
void
timer_finalize (GApplication * app)
{
    GList *timer_entry_list;
    struct timer_entry_info *timer_entry_data;
    struct timer_info *timer_data;

    timer_data = sep_get_timer_data (app);
    timer_entry_list = timer_data->timer_entry_list;

    /* Remove all the pending timers. */
    while (timer_entry_list != NULL)
    {
        GList *timer_entry_next = timer_entry_list->next;

```

```

    timer_entry_data = timer_entry_list->data;
    g_free (timer_entry_data);
    timer_data->timer_entry_list =
        g_list_delete_link (timer_data->timer_entry_list, timer_entry_list);
    timer_entry_list = timer_entry_next;
}

/* Cancel the ticking source. */
g_source_remove (timer_data->tick_source);

g_free (timer_data);
timer_data = NULL;
return;
}

/* Arrange to call back after a specified interval, or slightly later. */
void
timer_create_entry (void (*subroutine) (void *, GApplication *),
                   gdouble interval, gpointer user_data, GApplication * app)
{
    struct timer_info *timer_data;
    struct timer_entry_info *timer_entry_data;
    gdouble current_time;

    timer_data = sep_get_timer_data (app);
    if (TRACE_TIMER)
    {
        g_print ("create timer entry at %p for %4.1f seconds from now.\n",
                 subroutine, interval);
    }
    current_time = g_get_monotonic_time () / 1e6;

    /* Construct the timer entry. */
    timer_entry_data = g_malloc (sizeof (struct timer_entry_info));
    timer_entry_data->subroutine = subroutine;
    timer_entry_data->expiration_time = current_time + interval;
    timer_entry_data->user_data = user_data;

    /* Place it on the timer entry list. We will see it on the next tick. */
    timer_data->timer_entry_list =
        g_list_append (timer_data->timer_entry_list, timer_entry_data);
    return;
}

/* Call here every 0.1 seconds to dispatch timers. */
static gboolean

```

```

timer_tick (gpointer user_data)
{
    GApplication *app = user_data;
    gdouble current_time;
    GList *timer_entry_list;
    struct timer_entry_info *timer_entry_data;
    struct timer_info *timer_data;

    /* Get our persistent data. */
    timer_data = sep_get_timer_data (app);

    /* Calculate the current time in seconds since the last reboot. */
    current_time = g_get_monotonic_time () / 1e6;

    /* Don't print the trace message oftener than once a second. */
    if (TRACE_TIMER && ((current_time - timer_data->last_trace_time) >= 1.0))
    {
        g_print ("current time is %f seconds.\n", current_time);
        timer_data->last_trace_time = current_time;
    }

    /* Check the timer entry list for expired timer entries. For each one found,
     * execute the specified subroutine and remove the entry from the list. */
    timer_entry_list = timer_data->timer_entry_list;

    while (timer_entry_list != NULL)
    {
        GList *timer_entry_next = timer_entry_list->next;
        timer_entry_data = timer_entry_list->data;
        if (current_time >= timer_entry_data->expiration_time)
        {
            /* The timer has expired. Call the specified subroutine with
             * its user data and the app as parameters. */
            if (TRACE_TIMER)
            {
                g_print ("timer routine called at %p.\n",
                    timer_entry_data->subroutine);
            }
            (*timer_entry_data->subroutine) (timer_entry_data->user_data, app);

            /* We are done with this timer entry item. */
            g_free (timer_entry_data);
            timer_entry_list->data = NULL;
            timer_data->timer_entry_list =
                g_list_delete_link (timer_data->timer_entry_list,
                    timer_entry_list);
        }
        timer_entry_list = timer_entry_next;
    }
}

```

```
        }

        timer_entry_list = timer_entry_next;
    }

    return G_SOURCE_CONTINUE;
}
```


37 timer_subroutines.h

```

/*
 * timer_subroutines.h
 *
 * Copyright © 2016 by John Sauter <John_Sauter@systemeyescomputerstore.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

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

/* Subroutines defined in timer_subroutines.c */

/* Initialize the timer */
void *timer_init (GApplication * app);

/* Terminate the timer */
void timer_finalize (GApplication * app);

/* Add an entry to the timer list. */
void timer_create_entry (void (*subroutine) (void *, GApplication *),
                        gdouble interval, gpointer user_data,
                        GApplication * app);

/* End of file timer_subroutines.h */

```