Introduction

The Sound Effects Player is a component of ShowControl, a project to automate the production of live stage plays. Sound Effects Player makes sounds at appropriate times in the performance.

There are two aspects to the use of the Sound Effects Player. The sound designer must create the sounds, and the sound operator must play them during the performance. The Sound Effects Player tries to make the latter job as simple as possible, at the cost of making the sound designer's job complex. This document is intended to support the sound designer by describing how to create sounds for the Sound Effects Player.

Installation

The Sound Effects Player is installed in the traditional GNU/Linux fashion: acquire the kit, place it in a subdirectory, install the prerequisite packages, then

```
./configure
make
sudo make install
```

The prerequisite packages for Fedora are gcc, intltool, gstreamer1-devel, gstreamer-plugins-base-devel, gtk3-devel and gtk-doc. Other distributions of GNU/Linux may have other names for these packages.

Sample Application

Because the Sound Effects Player is a Gstreamer 1.0 application that adds two Gstreamer elements, you will need to type the following before running Sound Effects Player;

```
export GST_PLUGIN_PATH=/usr/local/lib/gstreamer-1.0
```

There is a MAN page for Sound Effects Player which provides more detail, but briefly you must provide the Sound Effects Player with a file which describes how the stage play is to be automated. The distribution kit contains a sample subdirectory which plays a single sound with no operator interaction, then exits. You can verify that you have installed the Sound Effects Player successfully by running the sample application:

```
cd sample
./run_sample.sh
```

You should hear a 440 Hz sine wave that lasts 12 seconds and fades up and down during that time. We will examine each of the files in this subdirectory to see how they work together to make this sound. Here is the list of files and a brief description of their meaning:

File name	Description
440Hz.wav	3 seconds of a 440Hz sine wave, 8000 samples per second, 8 bits per

File name	Description
	sample
Makefile	Used by the build software
Makefile.am	Used by the build software
Makefile.in	Used by the build software
run_sample.sh	Program to run the sample application, marked executable
Sample_cues.xml	Dummy file which would contain the cues if this were a complete ShowControl project
Sample_equipment.xml	Pointer to the sounds and sound_sequence files; in a complete ShowControl project would also contain information about other equipment
Sample_project.xml	The top-level ShowControl project file
Sample_script.xml	Dummy file which would contain the script if this were a complete ShowControl project
Sample_sound_sequence.xml	The procedure for presenting sounds to the sound effects operator
Sample_sounds.xml	The sounds to be presented

The file run_sample.sh invokes sound_effects_player to play the sound specified by Sample_project.xml. Sample_project.xml consists mostly of comments explaining what each part does. You can read them to get more background on the ShowControl project, but here we are concerned just about the sound effects, so we only need to note that the equipment is described in the file Sample_equipment.xml. Sample_equipment.xml, in its sound effects section, refers to two other files: Sample_sounds.xml and Sample_sound_sequence.xml. These are the two files we will use to illustrate how to create sound effects for your live stage play. You should, of course, create files of your own for your play, but you can use the sample files to get started.

Sample_sounds.xml

The sounds XML file, in the sample application called Sample_sounds.xml, contains the information about each sound. Here is the Sample_sounds.xml file. The letters at the right margin are callouts referenced in the paragraphs below.

```
<?xml version="1.0" encoding="utf-8"?>
<show_control>
<!-- The sample production uses this sound. -->
  <sounds>
```

```
    <version>1.0</version>

    <!-- a sine wave at 440Hz, which is A in octave 6. -->
    <sound>
      <!-- The internal sequencer references sounds by name. -->
      <name>440Hz</name>
      <!-- the WAV file contains the waveform for the sound -->
      <wav_file_name>440Hz.wav</wav_file_name>
      <!-- volume ramps up from 0 to attack level in attack duration time -->
      <attack_duration_time>2.000000000</attack_duration_time>
      <attack_level>1.00</attack_level>
      <!-- after reaching attack level, volume ramps down to sustain level
         in decay time -->
      <decay_duration_time>1.000000000</decay_duration_time>
      <sustain_level>0.5</sustain_level>
      <!-- release starts on external signal or when reaching the
         release start time -->
      <release_start_time>10.000000000</release_start_time>
      <!-- upon release, volume ramps down to 0 in release duration time -->
      <release_duration_time>2.000000000</release_duration_time>
      <!-- sound loops between loop times; from=0 means no looping -->
      <loop_from_time>1.000000000</loop_from_time>
      <loop_to_time>0.000000000</loop_to_time>
      <!-- automatically stop looping after loop_limit loops -->
      <loop_limit>0</loop_limit>
      <!-- start at start_time offset in the wave file -->
      <start_time>0.000000000</start_time>
      <!-- attenuate the full-volume sound in the WAV file by
         designer_volume_level -->
      <designer_volume_level>1.00</designer_volume_level>
      <!-- pan mono sound or balance stereo sound by designer_pan -->
      <designer_pan>0.00</designer_pan>
      <!-- if no internal sequencer, MIDI program and note numbers to activate
         from an external sequencer  -->
      <MIDI_program_number>0</MIDI_program_number>
      <MIDI_note_number>69</MIDI_note_number>
      <!-- if no internal sequencer, OSC name to activate from an external
         sequencer  -->
      <OSC_name></OSC_name>
      <!-- if no internal sequencer, function key to activate by sound
         effects operator -->
      <function_key></function_key>
    </sound>

    <routing></routing>
  </sounds>
</show_control>
```

The sample application contains only one sound, which it calls 440Hz at A. Each sound refers to a WAV file which gets played when required. See B. In the simplest case the WAV file is just played once at full volume, but the Sound Effects Player can also repeat a portion of the sound, and can apply an amplitude envelope to it. In the Sample_sounds.xml file you see that the 440Hz sine wave spends two seconds (C), reaching full volume (D), one second (E) decaying to half volume (F), then, if it

hasn't been stopped by the sound effects operator, it stops itself after playing for 10 seconds (G). When it is stopped its volume decreases to 0 in two seconds (H). To see how the operator can stop a sound early, run run_sample.sh again but this time, instead of letting it finish by itself, click on the Stop button in cluster 0 and observe that the total run time is now less than 12 seconds.

The sample sound also saves memory and disk space by looping. If you examine the file 440Hz.WAV you will see that it is only three seconds long. The information in the <sound></sound> block specifies that the first second (I), (J) of the WAV file is to be repeated indefinitely until the sound is released (K). The sound is released when it has been playing for 10 seconds (G) or when the sound effects operator presses the Stop button.

If you examine the 440Hz.WAV file you will see that it has only 8 bits per sample, and only 8,000 samples per second. That is enough for a simple sine wave. The sound effects player uses the Gstreamer structure to accept sample rates from 8,000 to 96,000 samples per second, using formats S8, U8 (8 bits per sample), LE16 (16 bits per sample), S32, F32 (32 bits per sample) and F64 (64 bits per sample).

I use audacity to prepare WAV files for the Sound Effects Player. It provides all the flexibility I need, but is simple to use. Other tools are also available: I have used espeak and sox to prepare test files, for example.

Sample_sound_sequence.xml

The sound sequence XML file, in the sample application called Sample_sound_sequence.xml, presents the sounds to the sound effects operator. Some parts of a live stage play can be handled by sequential processing: simply making the next sound when the time comes to make it. Some parts, however, are best handled by presenting the sound effects operator with a selection of sounds which he triggers in response to activities on the stage. The sound effects player combines these methods in a very flexible way.

The Sample_sound_sequence.xml file simply plays the 440Hz sound without waiting for the operator. Here it is:

```
<?xml version="1.0" encoding="utf-8"?>
<show_control>

  <!-- The sequence items for the sample production.  -->

  <sound_sequence>
    <version>1.0</version>

    <sequence_item>
      <name>start_of_sequence</name>
      <type>start_sequence</type>
      <next>A6</next>
    </sequence_item>
```

L

M

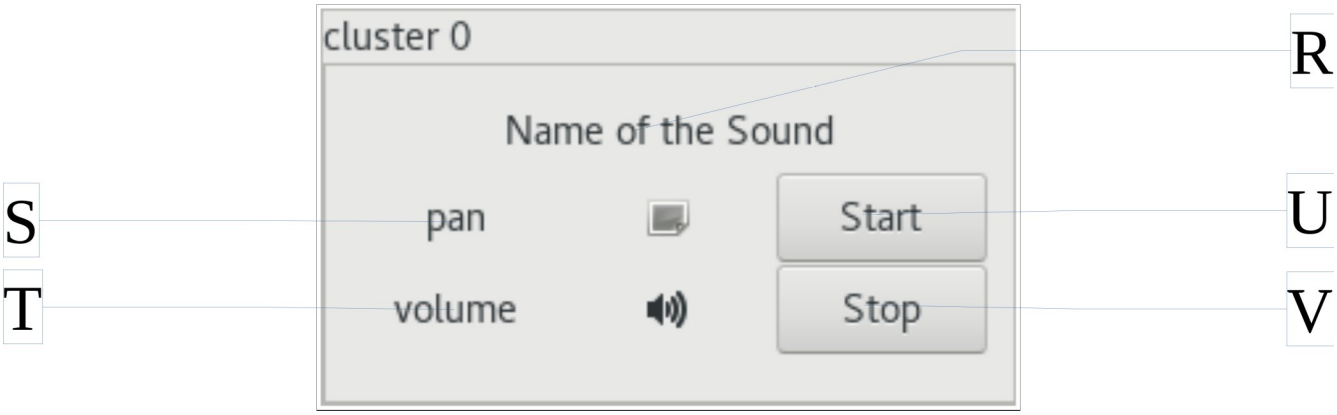
```
    <sequence_item>
      <name>A6</name>
      <type>start_sound</type>
      <sound_name>440Hz</sound_name>
      <MIDI_note_number>69</MIDI_note_number>
      <cluster_number>0</cluster_number>
      <tag>A6</tag>
      <text_to_display>A in octave 6</text_to_display>
      <importance>1</importance>
    </sequence_item>

  </sound_sequence>
</show_control>
```

N O P Q

A sound sequence file, in this sample program called Sample_sound_sequence.xml, contains sequence items. Each item performs a function when executed by the Sound Effects Player. Every sequence item has a name, a type and whatever other information that they need to carry out their function. The sample sequence consists of just two sequence items: a start_sequence and a start_sound.
Start_sequence (L) is where the Sound Effects Player starts. It then moves to the sequence item named in the <next></next> block (M), in this case A6. The sequence item named A6 (N) starts playing the sound named 440Hz (O), which we saw in Sample_sounds.xml (A). It specifies that the sound effects operator is to have control of the sound using cluster number 0 (P), and the text to display to the sound effects operator is “A in octave 6” (Q). This sequence item does not specify what the Sound Effects Player is to do when the sound finishes playing, so the Sound Effects Player exits.

Clusters

The sample application lets the sound effects operator control its sound using cluster 0. The Sound Effects Player displays 16 clusters, numbered 0 through 15. A cluster looks like this:



The text area across the top of the cluster (R) is controlled by the sequence item (Q). The four controls beneath are for panning the sound through the stereo field (S), adjusting its volume (T), starting it (U) and stopping it (V).

The global controls are above the clusters:

Above the buttons are horizontal bars that show how loud each channel is sounding (W). From left to right, the buttons silence the output (X), pause the sound (Y), continue from a pause (Z) and continue from an Operator Wait (AA). To the right of the Play button is a text area (AB) which is filled in by the Operator Wait. We will see the Operator Wait sequence item in the next section.

Example_01

To illustrate more of the capabilities of the Sound Effects Player we need a more complex example. Navigate to subdirectory example_01 and you will find file run_example_01.sh. Running this file executes the sound effects player, giving it Example_01_project.xml as its parameter. Type "./run_example_01.sh" to see what this project does.

Notice that it plays the 440Hz tone, then waits for you to press a key. The text to the right of the Play button (AB) is "Press a Start key to hear a number, Play to exit". Each of the clusters has some text in its display area (R) which invites you to press the Start button (U) and describes what you will hear when you do. Press some of the Start buttons to hear a synthesized voice saying numbers between one and sixteen. Notice that the text area over the cluster changes while the voice is speaking. When you have had enough, press the Play button in the global controls (AA) to cause the Sound Effects Player to exit.

The XML files in the example_01 subdirectory are structured very much like the files in the sample subdirectory, so we only need to be concerned with files example_01_sounds.xml and example_01_sound_sequences.xml. Example_01_sounds.xml contains, in addition to the 440Hz sound, sixteen sounds that look like this:

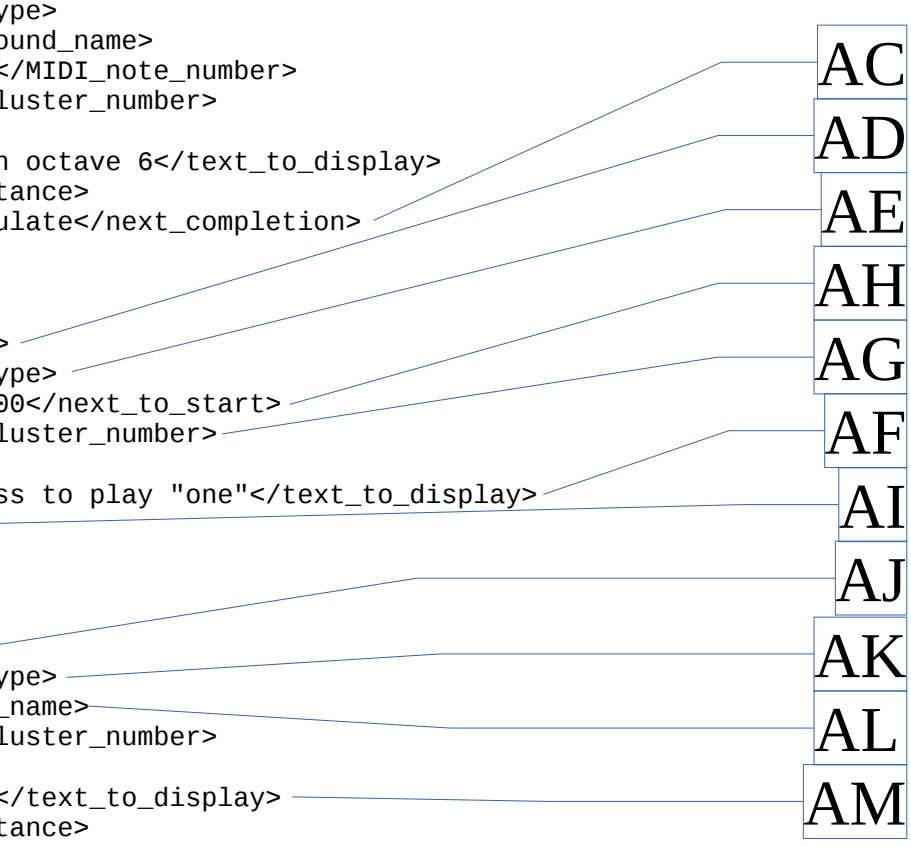
```
<sound>
      <!-- The internal sequencer references sounds by name. -->
      <name>1</name>
```

```
      <!-- the WAV file contains the waveform for the sound -->
      <wav_file_name>01.wav</wav_file_name>
    </sound>
```

Each block associates a sound name with a WAV file name, and lets all of the other sound parameters default. The default values play the sound once, at full volume.

Example_01_sound_sequence.xml starts like this:

```
<?xml version="1.0" encoding="utf-8"?>
<show_control>

  <!-- The sequence items for the example_01 production.  -->

  <sound_sequence>
    <version>1.0</version>

    <sequence_item>
      <name>start_of_sequence</name>
      <type>start_sequence</type>
      <next>A6</next>
    </sequence_item>

    <sequence_item>
      <name>A6</name>
      <type>start_sound</type>
      <sound_name>440Hz</sound_name>
      <MIDI_note_number>69</MIDI_note_number>
      <cluster_number>0</cluster_number>
      <tag>A6</tag>
      <text_to_display>A in octave 6</text_to_display>
      <importance>1</importance>
      <next_completion>populate</next_completion>
    </sequence_item>

    <sequence_item>
      <name>populate</name>
      <type>offer_sound</type>
      <next_to_start>play_00</next_to_start>
      <cluster_number>0</cluster_number>
      <tag>1</tag>
      <text_to_display>press to play "one"</text_to_display>
      <next>pop_01</next>
    </sequence_item>

    <sequence_item>
      <name>play_00</name>
      <type>start_sound</type>
      <sound_name>1</sound_name>
      <cluster_number>0</cluster_number>
      <tag>1</tag>
      <text_to_display>one</text_to_display>
      <importance>1</importance>
    </sequence_item>
```



The beginning is just like Sample_sound_sequence.xml (L through Q), but notice the new field at AC. The <next_completion>populate</next_completion> field causes the Sound Effects Player to execute the populate sequence item when the sound being played has completed.

The sequence item named populate (AD) is an offer_sound (AE). It puts some text (AF) on the text area (R) of cluster 0 (AG) and tells the Sound Effects Editor to execute sequence item play_00 (AH) when the sound effects operator pushes the Start button on cluster 0 (U). It then immediately goes on to execute sequence item pop_01 (AI) which does the same thing for cluster 1, and so on through cluster 15.

Pushing the Start button on cluster 0 causes the Sound Effects Player to execute sequence item play_00 (AJ). This is a start_sound sequence item (AK) which plays sound 1 (AL) and changes the text on cluster 0's text area (AM). Reading through the remainder of Example_01_sound_sequence.xml you will see that the same happens for clusters 1 through 15.

Further down in file Example_01_sound_sequence.xml we see this:

```
    <sequence_item>
      <name>pop_15</name>
      <type>offer_sound</type>
      <next_to_start>play_15</next_to_start>
      <cluster_number>15</cluster_number>
      <tag>16</tag>
      <text_to_display>press to play "sixteen"</text_to_display>
      <next>do_wait</next>
    </sequence_item>

    <sequence_item>
      <name>play_15</name>
      <type>start_sound</type>
      <sound_name>16</sound_name>
      <cluster_number>15</cluster_number>
      <tag>16</tag>
      <text_to_display>sixteen</text_to_display>
      <importance>1</importance>
    </sequence_item>

    <sequence_item>
      <name>do_wait</name>
      <type>operator_wait</type>
      <text_to_display>Press a Start key to hear a number, Play to
exit</text_to_display>
      <next_play>clean_01</next_play>
    </sequence_item>

    <sequence_item>
      <name>clean_01</name>
      <type>cease_offering_sound</type>
      <tag>1</tag>
```

AU AN AO AP AQ AR AS

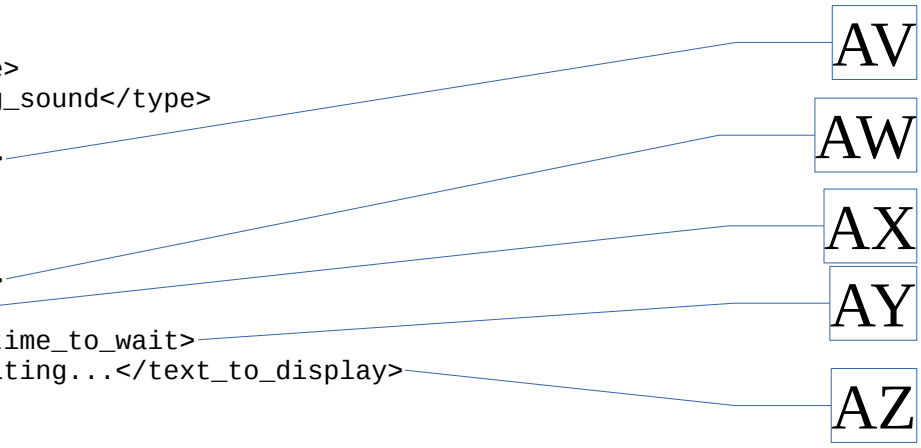
```
      <next>clean_02</next>
    </sequence_item>
```

The fifteeth offer_sound branches to do_wait (AN). Do_wait (AO) is an Operator_wait sequence item which displays “Press a Start key to hear a number, Play to exit” (AP). When the sound effects operator presses the Play button, the Sound Effects Editor executes the clean_01 sequence item (AQ). The clean_01 sequence item (AR) is a cease_offering_sound which specifies that the sound offered with a tag of 1 (AS) should be canceled. It then branches to clean_02 (AT) which does the same for tag 2, and so on to tag 16.

Each of the offer_sound sequence items contains a tag (AU), which is used by cease_offering_sound to terminate the offering.

At the bottom of file Example_01_sound_sequence.xml we find this:

```
    <sequence_item>
      <name>clean_16</name>
      <type>cease_offering_sound</type>
      <tag>16</tag>
      <next>exiting</next>
    </sequence_item>

    <sequence_item>
      <name>exiting</name>
      <type>wait</type>
      <time_to_wait>2.0</time_to_wait>
      <text_to_display>exiting...</text_to_display>
    </sequence_item>

  </sound_sequence>
</show_control>
```



The last of the cease_offering_sound sequence items branches to sequence item exiting (AV). The exiting sequence item (AW) is a wait (AX) which delays for 2 seconds (AY) while it displays “exiting...” to the sound effects operator (AZ). It has no <next></next> block, so when it is complete there are no sequence items in execution, and no sounds being offered on clusters. When that happens, the Sound Effects Editor exits.

Example_02

Example 2 is in subdirectory example_02. It has two sounds:

- car, the sound of a car engine starting, idling, then shutting down
- circular_saw, the sound of a circular saw starting, cutting, then stopping

The file Example_02_sound_sequence.xml simply offers the two sounds. The interesting file is Example_02_sounds.xml, which looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<show_control>
<!-- The example_02 production uses these sounds. -->
  <sounds>
    <version>1.0</version>

    <sound>
      <name>circular_saw</name>
      <wav_file_name>circular_saw.wav</wav_file_name>
      <!-- The saw runs until the sound effects operator
         presses the Stop key.  -->
      <loop_from_time>2.5</loop_from_time>
      <loop_to_time>2.45</loop_to_time>
      <loop_limit>0</loop_limit>
      <release_duration_time>∞</release_duration_time>
    </sound>

    <sound>
      <name>car</name>
      <wav_file_name>car.wav</wav_file_name>
      <!-- The car runs until the sound effects operator
         presses the Stop key.  -->
      <loop_from_time>11.01925</loop_from_time>
      <loop_to_time>10.00227</loop_to_time>
      <loop_limit>0</loop_limit>
      <release_duration_time>∞</release_duration_time>
    </sound>

    <routing></routing>
  </sounds>
</show_control>
```

Cars and saws are complex sounds: they don't just go silent when they stop, but have a shutdown sound. For a live stage play we want to make the sound until the script calls upon it to stop, which will be a variable length of time depending upon the actor. Thus we play the start of the sound, repeat the middle of the sound until time to stop, and finally play the end of the sound. This is done by setting the <loop_to_time></loop_to_time> and the <loop_from_time></loop_from_time> fields to the beginning and ending of the middle part of the sound. Also, we set <loop_limit></loop_limit> to 0, meaning loop until the Stop button is pushed, and <release_duration_time></release_duration_time> to infinity (∞) so that the envelope does not ramp the volume down after the stop button has been activated.

The tricky part of such a sound is finding good loop points. To avoid a click at the transition, choose points where the waveform crosses 0 and is headed in the same direction, either increasing or decreasing. I use Audacity for this, zooming in until I can see individual samples.

Another problem is the repetition. You don't want a 1-second sound to be repeated many times unless it is a very simple sound, like a hum. The audience will quickly become annoyed. To mask the repetition you can use a second sound which repeats at a different interval. Try playing the car and the saw together, to see how the repetition is less noticeable with both running.