ShowControl: The Project

ShowControl is driven by a project database, which contains all the information needed to support a performance of a live stage play. In this document I will describe that database.

The database stores keyword-value pairs in a hierarchical structure. We are currently using XML to describe the data, but the format is under discussion. The database may be divided into several files, linked from a top-level file called the project file.

Project File

The project file, like all ShowControl files, has a top-level container called show_control. It contains exactly one container, called project. That contains the following data items and containers:

- Version, currently 1.0.
- Title, the name of the production and a short description
- equipment, a container for equipment information; see below
- cues, a container for the cues
- characters, a container for the characters in the production and the actors who will play them
- crew, a container for the crew for this production, with their duties
- script, a container for the script

Note that there can be more than one container of each type. The syntax of a container allows it to include a reference to a file as well as its contents. Multiple containers of the same type are merged, and the contents of all of the files referenced are merged when the project file is read.

We haven't yet defined the contents for the cues, characters, crew and script containers. Nevertheless, here is an example of a project file, using XML syntax:

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

<!-- The outer structure name is "show_control" to identify
     XML files belonging to the show_control project.  If the user
     tries to read an XML file intended for some other application,
     we can give a meaningful error message.  -->
<show_control>

  <!-- This is the project file.  Other XML files used by the
```

```
      show_control application are equipment, cues, characters, crew,
      script, sounds and sound_sequence. By putting the purpose of the
      file here, we can give a meaningful error message if the user
      tries to read the wrong file.  -->
<project>

  <!-- XML format lets us add more values without impacting old readers.
     However, if we need to make an incompatible change to an old
     value, having the version number lets us give a good error message
     if an old program tries to read a newer, incompatible, file.
     Every reader must check the version number and reject versions
     it does not know how to read.  The first part of the number is
     incremented on an incompatible change, the second part on a compatible
     change: one that adds a value that can be ignored by old readers,
     for example.  When reading, a program will check the first part of
     the number and adapt its reading to the indicated format.
     When writing, a program will use the oldest format capable of
     expressing the information correctly.  The first version of the
     show_control programs will reject version numbers in which the value
     before the decimal point is not equal to 1.  -->
  <version>1.0</version>

  <!-- a short description of the project -->
  <title>The Perils of Pauline, Amato Theater, April 15-20, 2015</title>

  <!-- The light board, sound mixer, etc, used at the Amato Theater.
     This information is independent of the production.  This file
     includes the IP address and port number used to communicate
     from the sequencer to the sound effects player, the light
     board controller and the microphone handler.  -->
  <equipment href="Amato_equipment.xml"/>

  <!-- Extra equipment brought in for this production.  The information
     in this file is used the same way as the standard equipment file,
     but is kept separate so the standard equipment file does not
     have to be edited for each production.  This file includes references
     to the sound effects and sequence used by the sound effects player.
  -->
  <equipment href="Pauline_equipment.xml"/>

  <!-- The way a cue is executed is specific to the script and the
     capabilities of the theater.  Every cue has a name.  The script
     specifies when during the performance each named cue is to be
     performed.  These files describe, for each named cue, what command
     or commands to send to the light board, sound effects player and/or
     microphone handler when the script calls for the cue to be executed.
     Cues which will be used in many shows are in Amato_cues.xml, whereas
     cues that are specific to The Perils of Pauline are in
     Pauline_cues.xml.  -->
  <cues href="Amato_cues.xml"/>
  <cues href="Pauline_cues.xml"/>

  <!-- These are the characters in this production, with the names of their
     actors. With a short list it is more convenient to place
```

```
       the information directly in the project file, though with
       larger productions it can be placed in its own file.  -->
    <characters>
      <character id="Pauline">Perl White</character>
      <character id="Harry Marvin">Crane Wilbur</character>
      <character id="Raymond Owen">Paul Panzer</character>
    </characters>

    <!-- The crew for this production, with their duties. -->
    <crew href="Amato_crew_for_Pauline.xml"/>

    <!-- The script, which is kept in a separate file because it is
       independent of production or even theater. -->
    <script lang="cueml" href="The_Perils_of_Pauline.xml"/>

  </project>
</show_control>
```

Equipment

The equipment container, which may be spread across several files, describes the sound and lighting hardware and software that will be used in the production. It contains the following items and containers:

- version, currently 1.0
- program, a container describing an item of software that will be used
- sound mixer, a container describing a sound mixer used for sound effects and microphones
- lighting controller, a container describing the lighting controller

There are three kinds of programs: sound effects, lighting and microphones. They all contain the following information:

- ID, either sound_effects, lighting or microphones
- name, the name of the program
- IP address, the IPv4 or IPv6 IP address that the program will listen on for commands
- port, the port number that the program will listen on

For the sound effects program, there are three additional items:

- Speaker count, the number of independent speakers available for sound effects. If not specified, defaults to the number of different speakers mentioned in the various sounds. Specify this as

the number of channels from the sound effects machine to the mixer, to avoid having to re-wire the mixer for every show.

- sounds, a container for the sounds that will be played
- sound sequence, a container for the sequence items that describe how and when to play each sound

We haven't yet defined the items for the lighting or microphones programs, or the contents for the sound mixer and lighting controller containers. Nevertheless, here is an example of an equipment container, as a separate file referenced from the project file.

```
<?xml version="1.0" encoding="utf-8"?>
<show_control>
  <!-- This file describes the standard equipment used at the Amato center.
  -->
  <equipment>
    <version>1.0</version>

    <!-- the various programs are identified by name, so they can be run
       automatically when the sequencer starts, and by type, so the
       sequencer knows which cues to send to which program.  The sequencer
       sends OSC or MIDI Show Control messages to the specified IP address
       and port.  The program reads this file to know which port to listen
       on.  When the main sequencer starts the program, it provides the
       name of the project file as the only argument.  The program then
       reads that file to learn everything it needs, including the
       information in this file.  -->
    <program id="sound_effects">
      <name>sound_effects_player</name>
      <type>sound</type>
      <IP_address>localhost6</IP_address>
      <port>5005</port>
      <sounds href="Amato_sounds.xml"/>
      <sound_sequence href="Amato_sound_sequence.xml"/>
      <speaker_count>8</speaker_count>
    </program>

    <program id="lighting">
      <name>light_board_controller</name>
      <type>lights</type>
      <IP_address>localhost6</IP_address>
      <port>1501</port>
      <board>Behringer Eurolight LC2412</board>
    </program>

    <program id="microphones">
      <name>microphone_controller</name>
      <type>microphone</type>
      <IP_address>localhost6</IP_address>
      <port>1502</port>
      <board>Behringer X32</board>
```

```
</program>

<!-- Here we describe the sound mixers used at the Amato Center.
   These definitions are read by the microphone_controller software
   to learn how to operate the mixers.  The microphone_controller
   program is given the name of the board, so it knows which board
   definition to use.  -->

<sound_mixer name="Beheringer X32" mfr="Behringer" model="X32">
  <protocol>osc</protocol>
  <mutestyle>illuminated</mutestyle>
  <port type="input" cnt="32" name="CH">
  <fader cmd="/ch/##/mix/fader" cmdtyp="level" range="0.0,1.0,1024" val="0"/>
  <mute cmd="/ch/##/mix/on" cmdtyp="enum" val="0"/>
  <scribble cmd="/ch/##/config/name" cmdtyp="string" text=""/>
  </port>
  <port type="auxin" cnt="6" name="AUX">
  </port>
  <port type="output" cnt="16" name="OUT">
  <mute cmd="/outputs/main/##/mix/on" cmdtyp="enum" val="0"/>
  <fader cmd="/ch/##/mix/fader" cmdtyp="level" range="0.0,1.0,1024" val="0"/>
  </port>
</sound_mixer>

<sound_mixer name="Yamaha 01V" mfr="Yamaha" model="01V">
  <protocol>midi</protocol>
  <mutestyle>illuminated</mutestyle>
  <port type="input" cnt="16" name="CH">
  <fader cmd="B#,01,XX" cmdtyp="level" range="0,127" val="0"/>
  <mute cmd="B#.1C,XX" cmdtyp="enum" val="0"/>
  <scribble cmd="" cmdtyp="string" text=""/>
  </port>
  <port type="auxmaster" cnt="4" name="AUX">
  <mute cmd="B#,2D,XX" cmdtyp="enum" val="0"/>
  <fader cmd="B#,11,XX" cmdtyp="level" range="0,127" val="0"/>
  </port>
  <port type="output" cnt="2" name="OUT">
  <mute cmd="B#,37,XX" cmdtyp="enum" val="0"/>
  <fader cmd="B#,1B,XX" cmdtyp="level" range="0,127" val="0"/>
  </port>
</sound_mixer>

<sound_mixer mfr="Default" model="">
  <protocol>midi</protocol>
  <port type="input" cnt="16" name="CH">
  <fader cmd="B#,01,XX" cmdtyp="level" range="0,127" val="0"/>
  <mute cmd="B#.1C,XX" cmdtyp="enum" val="0"/>
  <scribble cmd="" cmdtyp="string" text=""/>
  </port>
  <port type="output" cnt="2" name="OUT">
  <mute cmd="B#,37,XX" cmdtyp="enum" val="0"/>
  <fader cmd="B#,1B,XX" cmdtyp="level" range="0,127" val="0"/>
  </port>
</sound_mixer>
```

```
    <!-- Lighting boards go here. -->
    <lighting_controller>
      <!-- ... -->
    </lighting_controller>

  </equipment>
</show_control>
```

Sounds

A sounds container contains a version number and can contain several sound containers. Each sound container contains the following items:

- name, the name of the sound
- wav_file_name, the name of the WAV file that contains the sound. The WAV file holds sound data in PCM format. It can have any number of channels, anywhere from 8 to 64 bits per sample, and a sample rate of 6,000 samples per second or greater. A common format is two channels, 16 bits per sample, and 48,000 samples per second.
- Attack duration time, the time in seconds for the volume to increase from 0 to the attack level. Default is 0.0.
- Attack level, the peak hit by the attack. Default is 1.0.
- Decay duration time, the time in seconds for the volume to decay from the attack level to the sustain level. Default is 0.0.
- Sustain level, the volume reached by the end of the decay. Default is 1.0.
- Release start time, the time at which the sound starts the release stage of its envelope. Default is 0.0, which means that the release will not start unless commanded to start externally.
- Release duration time, the time in seconds for the volume to decrease to 0.0 after the release stage of the envelope starts. Default is 0.0. This can be specified as infinite, which means that the volume does not decrease during the release stage of the envelope.
- Loop from time, the time at which the sound goes back to the loop to time and repeats. Default is 0.0, which means no looping.
- Loop to time, the time the sound goes back to for looping.
- Loop limit, the number of times to loop. Default is 0, which means no limit.
- Start time, the time within the WAV file to start the sound. Default is 0.0, which means start at the beginning.

- Designer volume level, the amount of attenuate the sound. Default is 1.0, which means play at full volume.
- Designer pan, the amount of sound to send to the left and right channels. Default is 0, which means send a mono sound to both channels equally and send a stereo sound to both channels as specified in the WAV file. This is not applicable to WAV files with more than two channels.
- MIDI program number, MIDI note number and OSC name, used to activate a sound from an external sequencer
- Function key, used to activate a sound from a keyboard

The sound container can also contain information on how the sound channels are routed to speakers. Within the container <channels> is a <channel> container for each channel. A <channel> container has one item, the <number>, which is the 0-based number of the channel within the WAV file. Also in <channel> is a container <speakers> which contains one or more <speaker> containers. Each <speaker> container has two items: the <name> and <volume_level>. You can place a sound between two speakers by sending it to both speakers with suitable volume levels. The following are the valid speaker names:

Name	Description
number	Used for position-less channels, e.g. for a theater with 1,024 speakers. The number is the speaker number, which reflects how speakers are wired to the mixing board. The names below map to speaker numbers, as indicated.
NONE	Sound is not sent to any speakers
ALL	sound is sent to all speakers
FRONT_LEFT (0)	Front left
FRONT_RIGHT (1)	Front right
FRONT_CENTER (2)	Front center
LFE1 (3)	Low-frequency effects 1 (first sub woofer)
REAR_LEFT (4)	Rear left
REAR_RIGHT (5)	Rear right
FRONT_LEFT_OF_CENTER (6)	Front left of center
FRONT_RIGHT_OF_CENTER (7)	Front right of center

Name	Description
REAR_CENTER (8)	Rear center
LFE2 (9)	Low-frequency effects 2 (second sub woofer)
SIDE_LEFT (10)	Side left
SIDE_RIGHT (11)	Side right
TOP_FRONT_LEFT (12)	Top front left
TOP_FRONT_RIGHT (13)	Top front right
TOP_FRONT_CENTER (14)	Top front center
TOP_CENTER (15)	Top center
TOP_REAR_LEFT (16)	Top rear left
TOP_REAR_RIGHT (17)	Top rear right
TOP_SIDE_LEFT (18)	Top side right
TOP_SIDE_RIGHT (19)	Top rear right
TOP_REAR_CENTER (20)	Top rear center
BOTTOM_FRONT_CENTER (21)	Bottom front center
BOTTOM_FRONT_LEFT (22)	Bottom front left
BOTTOM_FRONT_RIGHT (23)	Bottom front right
WIDE_LEFT (24)	Wide left (between front left and side left)
WIDE_RIGHT (25)	Wide right (between front right and side right)
SURROUND_LEFT (26)	Surround left (between rear left and side left)
SURROUND_RIGHT (27)	Surround right (between rear right and side right)

If the speakers are not specified for a sound, the default assignment depends on the number of channels in the sound, as follows. This is also the speakers assumed to be present if the sound designer specifies a speaker count for the project, except that a speaker count of 1 implies the front center speaker.

Channels	Default Assignment of Sound Channels to Speakers
1	0: all speakers

Channels	Default Assignment of Sound Channels to Speakers
2	0: front_left, 1: front_right
3 (2.1)	0: front_left, 1: front_right, 2: LFE1
4 (4.0)	0: front_left, 1: front_right, 2: rear_left, 3: rear_right
5	0: front_left, 1: front_right, 2: rear_left, 3: rear_right, 4: front_center
6 (5.1)	0: front_left, 1: front_right, 2: rear_left, 3: rear_right, 4: front_center, 5: LFE1
7 (6.1)	0: front_left, 1: front_right, 2: read_left, 3: rear_right, 4: front_center, 5: LFE1, 6: rear_center
8 (7.1)	0: front_left, 1: front_right, 2: rear_left, 3: rear_right, 4: front_center, 5: LFE1, 6: side_left, 7: side_right.
More than 8	Channels are mapped 1-to-1 to speaker numbers.

Here is an example of a sounds container in a separate file, referenced from an equipment container.

```
<?xml version="1.0" encoding="utf-8"?>
<show_control>
<!-- these sounds are common to many shows. -->
  <sounds>
    <version>1.0</version>

    <!-- a test tone, used to verify the sound system -->
    <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>0.000000000</decay_duration_time>
      <sustain_level>1.00</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 -->
      <loop_from_time>0.000000000</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>1</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>
    <channels>
    <!-- Channel 0 to front_left speaker, 1 to front_right.  -->
          <channel>
                <number>0</number>
                <speakers>
                      <speaker>
                            <name>front_left</name>
                            <volume_level>1.0</volume_level>
                      </speaker>
                </speakers>
          </channel>
          <channel>
                <number>1</number>
                <speakers>
                      <speaker>
                            <name>front_right</name>
                            <volume_level>1.0</volume_level>
                      </speaker>
                </speakers>
          </channel>

    </channels>

</sound>

<!-- another test tone -->
<sound>
    <!-- The internal sequencer references sounds by name. -->
    <name>880Hz</name>
    <!-- the WAV file contains the waveform for the sound -->
    <wav_file_name>880Hz.wav</wav_file_name>
    <!-- volume ramps up from 0 to attack level in attack duration time -->
    <attack_duration_time>0.000000000</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>0.000000000</decay_duration_time>
```

```
    <sustain_level>1.00</sustain_level>
    <!-- release starts on external signal or when reaching the
       release start time -->
    <release_start_time>0.000000000</release_start_time>
    <!-- upon release, volume ramps down to 0 in release duration time -->
    <release_duration_time>0.000000000</release_duration_time>
    <!-- sound loops between loop times -->
    <loop_from_time>0.000000000</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>2</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>

  <!-- The ring and ringout sounds are used for a telephone. -->
  <sound>
    <name>ring</name>
    <wav_file_name>telephone_ring.wav</wav_file_name>
    <attack_duration_time>0.000000000</attack_duration_time>
    <attack_level>1.00</attack_level>
    <decay_duration_time>0.000000000</decay_duration_time>
    <sustain_level>1.00</sustain_level>
    <release_start_time>1.995000000</release_start_time>
    <release_duration_time>0.010000000</release_duration_time>
    <loop_from_time>0.000000000</loop_from_time>
    <loop_to_time>0.000000000</loop_to_time>
    <loop_limit>0</loop_limit>
    <start_time>0.000000000</start_time>
    <designer_volume_level>1.00</designer_volume_level>
    <designer_pan>0.00</designer_pan>
    <MIDI_program_number></MIDI_program_number>
    <MIDI_note_number></MIDI_note_number>
    <OSC_name></OSC_name>
    <function_key></function_key>
  </sound>

  <sound>
    <name>ringout</name>
```

```
      <wav_file_name>telephone_ringout.wav</wav_file_name>
      <attack_duration_time>0.010000000</attack_duration_time>
      <attack_level>1.00</attack_level>
      <decay_duration_time>0.000000000</decay_duration_time>
      <sustain_level>1.00</sustain_level>
      <release_start_time>0.000000000</release_start_time>
      <release_duration_time>∞</release_duration_time>
      <loop_from_time>0.000000000</loop_from_time>
      <loop_to_time>0.000000000</loop_to_time>
      <loop_limit>0</loop_limit>
      <start_time>0.000000000</start_time>
      <designer_volume_level>1.00</designer_volume_level>
      <designer_pan>0.00</designer_pan>
      <MIDI_program_number></MIDI_program_number>
      <MIDI_note_number></MIDI_note_number>
      <OSC_name></OSC_name>
      <function_key></function_key>
    </sound>

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

Sound sequence

The sound sequence container contains sequence items. Each sequence item describes a step in the handling of the sounds during a production. This amounts to a simple programming language. It does not have conditionals but it does have parallel flows of control. Each sequence item is a container with a name item and a type item. Which additional items it has depends on its type:

Operator Wait

The Operator Wait sequence item causes the sequencer to wait until the operator presses his Play button. It has these additional items

- text to display, shown to the sound effects operator when the sequencer is waiting for him to press his Play button
- next play, the name of the sequence item to execute when the operator presses his Play button

Start Sound

The Start Sound sequence item causes a sound to be played. It has these additional items

- sound name, the name of the sound to start playing

- program number, bank number and cluster number, specify where to display the sound on the operator's panel
- tag, a reference in case we wish to stop playing the sound in a later sequence item
- text to display, shown to the operator in the cluster
- importance, the progress of the most important sound playing is shown to the operator
- Q number, used for MIDI Show Control
- use external velocity, used for MIDI
- next starts, the sequence item to execute when the sound starts to play
- next release started, the sequence item to execute when the sound enters the release stage of its amplitude envelope
- next completion, the sequence item to execute when the sound finishes playing but has not been told to stop
- next termination, the sequence item to execute when the sound finishes playing after having been told to stop

Wait

The Wait sequence item causes the sequencer to wait for a specified time. It has these additional items

- time to wait, in seconds
- text to display, shown to the operator if this is the next wait to expire, and there is no Operator Wait in progress
- next completion, the sequence item to execute when the wait is over

Stop

The Stop sequence item is used to stop a playing sound. It has these additional items

- tag, the tag of the sounds to stop
- next, the name of the next sequence item to execute

If several sounds have the same tag, all are stopped.

Offer Sound

The Offer Sound sequence item is used to present a sound to the operator, so he can play it on demand. It has these additional items

- The program number, bank number and cluster number on which to offer this sound
- The name of the sequence item to execute when the operator presses the Start button on that cluster
- The tag for this sound offering, used by Cease Offering Sound
- the MIDI program number and note number, used by an external sequencer sending MIDI messages,
- the OSC name, used by an external sequencer sending Open Sound Control messages
- the Q number, used by an external sequencer sending MIDI Show Control
- the macro number, used by an external sequencer sending MIDI Show Control
- the function key, used from a computer keyboard to start this sound
- the text to display to the operator in the cluster
- the name of the next sequence item to execute

Cease Offering Sound

The Cease Offering Sound sequence item is used to stop offering a sound. It has these additional items

- tag, the tag on the Offer Sound sequence item to be stopped
- next, the sequence item to execute next

Here is an example of a sound sequences container, in a separate file referenced from an equipment container.

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

  <!-- These sound sequence items are common to many shows.  -->

  <sound_sequence>
    <version>1.0</version>

    <sequence_item>
      <name>telephone-ring</name>
      <type>operator_wait</type>
      <text_to_display>press play to make the telephone ring</text_to_display>
```

```
      <next_play>telephone-ring-2</next_play>
    </sequence_item>

    <sequence_item>
      <name>telephone-ring-2</name>
      <type>start_sound</type>
      <sound_name>ring</sound_name>
      <cluster_number>0</cluster_number>
      <tag>telephone-ring</tag>
      <text_to_display>ding-a-ling</text_to_display>
      <importance>2</importance>
      <next_starts>telephone-ring-3</next_starts>
      <next_release_started>telephone-ring-5</next_release_started>
      <next_termination>telephone-ring-7</next_termination>
    </sequence_item>

    <sequence_item>
      <name>telephone-ring-3</name>
      <type>operator_wait</type>
      <text_to_display>press play to stop the telephone from
ringing</text_to_display>
      <next_play>telephone-ring-4</next_play>
      </sequence_item>

    <sequence_item>
      <name>telephone-ring-4</name>
      <type>stop</type>
      <tag>telephone-ring</tag>
    </sequence_item>

    <sequence_item>
      <name>telephone-ring-5</name>
      <type>start_sound</type>
      <sound_name>ringout</sound_name>
      <cluster_number>0</cluster_number>
      <tag>telephone-ring</tag>
      <text_to_display>ringout between rings</text_to_display>
      <importance>2</importance>
      <next_completion>telephone-ring-6</next_completion>
      <next_termination>telephone-ring-8</next_termination>
    </sequence_item>

    <sequence_item>
      <name>telephone-ring-6</name>
      <type>start_sound</type>
      <sound_name>ring</sound_name>
      <cluster_number>0</cluster_number>
      <tag>telephone-ring</tag>
      <text_to_display>ding-a-ling</text_to_display>
      <importance>2</importance>
      <next_release_started>telephone-ring-5</next_release_started>
      <next_termination>telephone-ring-7</next_termination>
    </sequence_item>
```

```
    <sequence_item>
      <name>telephone-ring-7</name>
      <type>start_sound</type>
      <sound_name>ringout</sound_name>
      <cluster_number>0</cluster_number>
      <tag>telephone-ring</tag>
      <text_to_display>ringout after last ring</text_to_display>
      <next_completion>telephone-ring-8</next_completion>
      <importance>2</importance>
    </sequence_item>

    <sequence_item>
      <name>telephone-ring-8</name>
      <type>wait</type>
      <time_to_wait>1000000000</time_to_wait>
      <text_to_display>wait for ringing to finish</text_to_display>
      <next_completion>telephone-ring</next_completion>
    </sequence_item>


  </sound_sequence>
</show_control>
```

There are some additional XML files in the ShowControl kit which can function as additional examples.