DecentSampler

Release 1.11

David Hilowitz

Apr 17, 2024

CONTENTS

DecentSampler is a free sampling plug-in that allows music composers to use multi-samples in the DecentSampler format. This document is a guide to creating samples in that format.

If you have no familiarity with the DecentSamples format, this video is a great place to start. After that, you'll want to start at the *File Format Overview* to learn a bit more about how to create sample libraries.

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 File Format Overview

At its core each DecentSampler sample library consists of two things: a folder containing a bunch of assets like audio files and pictures, and a single text file (called a **dspreset** file) which describes how the engine should use all of those files. This reference document is a guide to creating **dspreset** files.

dspreset files are just XML files. As such, each one begins with an XML declaration:

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

1.1.1 The top-level <DecentSampler> element (required)

At the top level of every **dspreset** file is a `<DecentSampler>` element. Every file **must** have one. Here is a list of attributes:

- **`minVersion`** (optional): This is the minimum version on which this preset is known to run. If a user is running an old version of DS, and a developer has specified a minVersion for their instrument, a dialog box will show up telling users that their version is outdated and that they should upgrade in order to get the full effect. They *can* than choose to ignore this warning or hit download. The dialog box does not show up for iOS users as most of them have auto-updates turned on.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<DecentSampler minVersion="1.0.0">
    <!-- More tags go here. :) -->
</DecentSampler>
```

Underneath the top-level `<DecentSampler>` element you can put any number of other elements, all of which are described in the sections that follow.

1.2 The <ui> element

The `<ui>` element is how you specify a user interface for your instrument. Each **dspreset** should have at most one `<ui>` element. There are several important attributes:

- **`coverArt`** (optional): A relative or absolutely path to a cover art image to use. After the first time this library is opened, this will get displayed on the "My Libraries" tab.
- **`bgImage`** (optional): A relative or absolutely path to a background image to use.
- **`bgColor`** (required): An eight digit hex value indicating the background color to be used for the background of the UI. This color will be drawn underneath any background image specified by `bgimage`.
- **`width`** (required): The width of your user interface. Recommended value: 812.
- **`height`** (required): The height of your user interface. Recommended value: 375.

Example:

```
<DecentSampler>
  <ui width="812" height="375">
    <tab name="main">
      <labeled-knob x="560" y="0" label="Tone" type="float" minValue="60" maxValue="22000
↪"
                    textColor="FF000000" value="22000.0" uid="y8AA4uuURh3">
        <binding type="effect" level="instrument" position="0" parameter="FX_FILTER_
↪FREQUENCY"/>
      </labeled-knob>
      <label x="360" y="0" width="50" height="30" text="Reverb"/>
      <control x="360" y="30" parameterName="Reverb" type="float" minValue="0" maxValue=
↪"1" textColor="FF000000" value="0.5">
      <!-- Your <binding /> elements should go here -->
      </control>
    </tab>
  </ui>
</DecentSampler>
```

1.2.1 The <tab> element

The `<tab>` element lives underneath the `<ui>` element. This architecture was chosen because we may, at some point, add support for multiple tabs. At present it is only possible to have a single tab within DecentSampler instruments. As such, every UI must have at most one `<tab>` element.

Attributes:

- **`name`** (optional): An optional name to be associated with this tab. This is currently not displayed anywhere.

1.2.2 The <button> element

The <`button`> element allows you to create a button within your UI. It lives underneath the <`tab`> element. There are two styles of buttons: text buttons and image buttons.

Attributes:

Attribute	Required	Description	Default
x	(required)	The x position of the menu	None
y	(required)	The y position of the menu	None
width	(required)	The width of the menu	None
heigh	(required)	The height of the menu	None
value	(optional)	The is the 0-based index of the button state that is currently selected. A value of 0 means that the first state is active.	0
style	(optional)	The type of button we want. There are two valid values: `text`, `image`.	`text`
mainIm	(required for `image` buttons)	For `image` buttons only. The path of the main image to display for this button. This can also be set at the state level so that it only applies to a specific state.	None
hoverI	(optional)	For `image` buttons only. The path of the main image to display when the user hovers their mouse over this button. This can also be set at the state level so that it only applies to a specific state.	None
clickI	(optional)	For `image` buttons only. The path of the main image to display when the user clicks down on this button. This can also be set at the state level so that it only applies to a specific state.	None
visibl	(optional)	This controls whether or not this button is visible. There are two valid values: `true`, `false`.	`true`

Example:

```
<button x="10" y="40"  width="120" height="30" style="image" value="0" mainImage=
↪"samples/ButtonMainImage.png" hoverImage="samples/ButtonHoverImage.png" clickImage=
↪"samples/ButtonSelectedImage.png">
    <!-- Your button states go here -->
</button>
```

The <state> element

In order for your button to work, it must contain at least one <`state`> elements.

Attributes:

Attribute	Required	Description
name	(required for `text` buttons)	The text to display on a text button when this state is active
mainIma	(required for `image` buttons)	For `image` buttons only. The path of the main image to display for this button when the button is in the current state.
hoverIm	(optional)	For `image` buttons only. The path of the image to display when the user hovers their mouse over this button when the button is in the current state.
clickIm	(optional)	For `image` buttons only. The path of the image to display when the user clicks down on this button when the button is in the current state.

In order to have your `<button>` elements actually do something useful, you need to attach bindings to them. Here's an example:

```
<button x="10" y="30"  width="70" height="50" style="image" value="0" >
    <state name="English" mainImage="samples/EFlag_MainImage.png" hoverImage="samples/
↪EFlag_HoverImage.png" clickImage="samples/EFlag_SelectedImage.png">
        <!-- Turn on this group -->
        <binding type="general" level="group" position="0" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true" />
        <!-- Turn off this group -->
        <binding type="general" level="group" position="1" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false" />
    </state>
    <state name="French" mainImage="Samples/FFlag_MainImage.png" hoverImage="Samples/
↪FFlag_HoverImage.png" clickImage="Samples/FFlag_SelectedImage.png">
        <!-- Turn off this group -->
        <binding type="general" level="group" position="0" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false" />
        <!-- Turn on this group -->
        <binding type="general" level="group" position="1" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true" />
    </state>
</button>
```

As you can see, this example uses a button to switch between two groups. You'll note the liberal use of the `fixed_value` translation mode above. This means that when any of these options are selected, a fixed predetermined value is used for the value of that binding.

The `<image>` element

The `<image>` element allows you to place a static image into your user interface. It lives underneath the `<tab>` element. Attributes:

- **`x`** (required): The `x` position of your image where (0,0) is the top-left corner
- **`y`** (required): The `y` position of your image where (0,0) is the top-left corner
- **`width`** (required): The width in pixels of the image component
- **`height`** (required): The height in pixels of the image component
- **`path`** (required): The relative path of the image file to show in this component
- `aspectRatioMode` (required): Whether or not the engine should preserve the aspect ratio of the image. Note: regardless of these settings, you still need to specify a width and height for your image element. Valid values: `preserve`, `stretch`. Default value is `preserve`.
- `visible` (optional): This controls whether or not this image is visible. There are two valid values: `true` (default), `false`.

The <label> element

The `<label>` element allows you to place a static block of text into yoru user interface. It lives underneath the `<tab>` element. Attributes:

- **`x`** (required): The `x` position of your control where (0,0) is the top-left corner
- **`y`** (required): The `y` position of your control where (0,0) is the top-left corner
- **`text`** (required): The actual text that should be displayed as part of the label.
- **`textColor`** (optional): An 8 digit hex value indicating the text color to be used for the label. See *Appendix A* for an explanation on these hex values.
- **`textSize`** (optional): A font size for the text label. Default: 12
- **`width`** (required): The width in pixels of the label.
- **`height`** (required): The height in pixels of the label.
- **`vAlign`** (optional): The vertical alignment of the text within the box described by the width and height attributes. Valid values: `top`,`bottom`, `center`. Default is `center`.
- **`hAlign`** (optional): The horizontal alignment of the text within the box described by the width and height attributes. Valid values: `left`,`right`, `center`. Default is `center`.
- **`visible`** (optional): This controls whether or not this text label is visible. There are two valid values: `true` (default), `false`.

A label's text can also be set dynamically using bindings using the `TEXT` binding parameter name.

The <labeled-knob> and <control> elements

The `<labeled-knob>` and `<control>` elements live underneath the `<tab>` element. These tags correspond to user controls (usually round radial dials) that can be used as part of a UI. These two element types are the same except that `<labeled-knob>` elements contain built-in labels, where as `<control>` elements do not. Every tab can have many `<control>` or `<labeled-knob>` elements underneath it.

For precise UI creation, it may be advisable to use a combination of `<control>` & `<label>` elements rather than `<labeled-knob>`.

Attributes:

- **`x`** (required): The `x` position of your control where (0,0) is the top-left corner
- **`y`** (required): The `y` position of your control where (0,0) is the top-left corner
- **`width`** (required): The width in pixels of the knob + label.
- **`height`** (required): The height in pixels of the knob + label.
- **`parameterName`** (required): In a situation where the sampler does not have enough room to display the full UI, a shrunken down version of the UI will be used. In such situations, this control will be labeled using the `parameterName`. It is good practice to always include a `parameterName`. If not `parameterName` is specified and a value `label` is specified, then that will be used instead.
- **`style`** (optional): The specific kind of control that is created. The following values are supported: `linear_bar`, `linear_bar_vertical`, `linear_horizontal`, `linear_vertical`, `rotary`, `rotary_horizontal_drag`, `rotary_horizontal_vertical_drag`, `rotary_vertical_drag`, `custom_skin_vertical_drag`, `custom_skin_horizontal_drag`. Default: `rotary_vertical_drag`.
- **`showLabel`** (optional): A true/false value dictating whether or not a built-in label should be displayed. Default: true for `<labeled-knob>` and false for `<control>` elements

- **label** (optional): If showLabel is true, the actual text that should be displayed above the control.
- **parameterName** (required): In a situation where the sampler does not have enough room to display the full UI, a shrunken down version of the UI will be used. In such situations, this control will be labeled using the parameterName. It is good practice to always include a parameterName.
- **minValue** (optional): The minimum value of your control. Default: 0
- **maxValue** (optional): The maximum value of your control. Default: 1
- **value** (optional): The initial value of your control. Default: 0
- **defaultValue** (optional): If a user double-clicks on the control, the control's value will be set to this default value. If no default value is specified, then nothing will happen on double-click.
- **valueType** (optional): There are three possible values for this float which yields numbers with two decimal points, integer which yields whole numbers, and musical_time which yields musical time increments in beats and measures. This last option is for use with the built-in delay effect only. Default: float
- **textColor** (optional): An 8 digit hex value indicating the text color to be used for the label. See *Appendix A* for an explanation on these hex values.
- **textSize** (optional): A font size for the text label. Default: 12
- **trackForegroundColor** (optional): An 8 digit hex value indicating the foreground color to use for the knob track. See *Appendix A* for an explanation on these hex values.
- **trackBackgroundColor** (optional): An 8 digit hex value indicating the background color to use for the knob track. See *Appendix A* for an explanation on these hex values.
- **visible** (optional): This controls whether or not this control is visible. There are two valid values: true (default), false.

It is also possible to use custom control graphics using the following attributes:

- **customSkinImage** (optional): This is path to an image to use for the control. This is expected to be a JPEG or PNG in KnobMan format. A huge gallery of compatible knobs can be found here (https://www.g200kg.com/en/webknobman/gallery.php).
- **customSkinNumFrames** (optional): The number of animation frames contained in the KnobMan image pointed to by customSkinImage.
- **customSkinImageOrientation** (optional): The orientation of the frames within the KnobMan image pointed to by customSkinImage. Valid values: horizontal, vertical. Default: vertical.
- **mouseDragSensitivity** (optional): An integer number describing how sensitive the control should be to mouse drags. The higher the number, the less sensitive the control will be to mouse movements.

If you are using custom knobs, it's important that you specify a style= value of either custom_skin_vertical_drag or custom_skin_horizontal_drag.

Example:

```
<DecentSampler>
  <ui>
    <tab>
      <labeled-knob x="560" y="0" label="Tone" type="float" minValue="60" maxValue="22000
↪" textColor="FF000000" value="22000.0">
      <!-- Your <binding /> elements should go here -->
      </labeled-knob>
      <label x="360" y="0" width="50" height="30" text="Reverb"/>
      <control x="360" y="30" parameterName="Reverb" type="float" minValue="0" maxValue=
```

(continues on next page)

(continued from previous page)

```
↪"1" textColor="FF000000" value="0.5" style="custom_skin_vertical_drag" customSkinImage=
↪"Samples/ENIGMA-nolight.png" customSkinNumFrames="31" customSkinImageOrientation=
↪"horizontal" mouseDragSensitivity="100">
      <!-- Your <binding /> elements should go here -->
      </control>
    </tab>
  </ui>
</DecentSampler>
```

To learn how to make knobs actually control parameters of your instrument, see "Appendix B: Bindings" section below.

1.2.3 The <menu> element

The <menu> element allows you to create a drop-down menu within your UI.

Attributes:

Attribute	Required	Description
x	(required)	The x position of the menu
y	(required)	The y position of the menu
width	(required)	The width of the menu
height	(required)	The height of the menu
value	(optional)	The is the 1-based index of the menu option that is currently selected. **NOTE: Index numbers for menu items start at 1.** A value of 0 means that no item is selected.
visible	(optional)	This controls whether or not this button is visible. There are two valid values: `true`, `false`. `true`

Example:

```
<menu x="10" y="40"  width="120" height="30" value="2">
    <!-- Your menu options go here -->
</menu>
```

The <option> element

In order for your drop-down menu to have options, it must contain <option> elements.

Attributes:

Attribute	Required	Description
name	(required)	The name of this element

That's right. The <option> element has only one attribute. In order to have your <option> elements actually do something useful, you need to attach bindings to them. Here's an example:

```
<menu x="10" y="40"  width="120" height="30" requireSelection="true" placeholderText=
↪"Choose..." value="2">
    <option name="Menu Option 1">
        <!-- Set the text of a label element -->
        <binding type="control" level="ui" position="2" parameter="TEXT" translation=
↪"fixed_value" translationValue="You chose the first option." />
        <!-- Turn on this group -->
        <binding type="general" level="group" position="0" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true" />
        <!-- Turn off this group -->
        <binding type="general" level="group" position="1" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false" />
    </option>
    <option name="Menu Option 2">
        <!-- Set the text of a label element -->
        <binding type="control" level="ui" position="2" parameter="TEXT" translation=
↪"fixed_value" translationValue="You chose the second option." />
        <!-- Turn off this group -->
        <binding type="general" level="group" position="0" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false" />
        <!-- Turn on this group -->
        <binding type="general" level="group" position="1" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true" />
    </option>
  </menu>
```

As you can see, this example uses a menu to switch between two groups. It also sets the text of a text label. You'll note the liberal use of the new `fixed_value` translation mode above. This means that when any of these options are selected, a fixed predetermined value is used for the value of that binding.

1.2.4 The <keyboard> element

The <`keyboard`> element lives underneath the <`ui`> element. This is where you specify settings relating to the on-screen keyboard. There should be only one <`keyboard`> element in your preset file. At this point, the only settings are color ranges which are specified using <`color`> sub-elements.

The <color> element

You can use <`color`> elements to change the color of portions of the on-screen keyboard. You can have as many <`color`> elements as you like. Only white keys are affected. It's worth noting that colors specified in the <`color`> elements are overlayed on top of the white keys using a 75% transparency, so choose your colors accordingly. This is done to preserve the readability of the key labels.

```
<DecentSampler>
    <ui>
        <!-- Other stuff here -->
        <keyboard>
            <color loNote="36" hiNote="50" color="FF2C365E" />
            <color loNote="51" hiNote="57" color="FF6D9DC5" />
            <color loNote="58" hiNote="67" color="FFCCF3F5" />
            <color loNote="68" hiNote="73" color="FFE8DA9B" />
```

(continues on next page)

(continued from previous page)

```
            <color loNote="74" hiNote="84" color="FFD19D61" />
        </keyboard>
    </ui>
    <!-- Other stuff here -->
</DecentSampler>
```

At-tribute		Description
loNote	(re-quired)	The bottom of the range for which this color should be displayed. Format: MIDI Note number.
hiNote	(re-quired)	The top of the range for which this color should be displayed. Format: MIDI Note number.
color	(re-quired)	A text representation of the color to be used for this key range. See *Appendix A* for an explanation on these hex values.

1.3 The <groups> element (required)

In this section, we'll find elements that pertain to samples and sample-mapping.

Every **dspreset** file should have one and only one `<groups>` element. This is where you specify the samples that make up your sample library. This element lives right underneath the top-level `<DecentSampler>` element. The basic structure is this:

```
<DecentSampler>
    <groups>
        <group>
            <sample /> <!-- This is where -->
            <sample /> <!-- the samples   -->
            <sample /> <!-- get defined   -->
        </group>
    </groups>
</DecentSampler>
```

At-tribute		Description
volum	(op-tiona	The volume of the instrument as a whole. This will be reflected in the UI in the top-right corner. Value can be in linear 0.0-1.0 or in decibels. If it's in decibels you must append dB after the value (example: "3dB"). Default: 1.0 (no volume change)
globa	(op-tiona	Global pitch adjustment for changing note pitch. In semitones. For example 1.0 would be a half-step up. Default: 0
glide	(op-tiona	The glide/portamento time in seconds. A value of 0.0 would mean no portamento. This value can also be set at the `<group>` and `<sample>` levels, although most people will want to set it globally at the `<groups>` level. Default: 0.0
glide	(op-tiona	Controls the glide/portamento behavior. Possible values are: `always` (glide is always performed), `legato` (glide is performed only when transitioning from one note to another), and `off`. This value can also be set at the `<group>` and `<sample>` levels, although most people will want to set it globally at the `<groups>` level. Default: `legato`

1.3.1 The <group> element

Samples live in groups. There can be many group elements under the `<groups>` element. It can be useful to sort your samples into groups in order to apply similar settings to them or to control them with a knob. The order of groups in a file matters insofar as bindings will often reference groups by using an index. The first group in a file is group 0, the second is group 1, etc.

Attribut	Description	
`enabl`	Whether or not this group is enabled. Possible values: true, false. Default: true	(optional)
`volum`	The volume of the group. Value can be in linear 0.0-1.0 or in decibels. If it's in decibels you must append dB after the value (example: "3dB"). Default: 1.0	(optional)
`ampVe`	The degree to which the velocity of the incoming notes affects the volume of the samples in this group. 0 = not at all. 1 = volume is completely determined by incoming velocity. When the value is 1, a velocity of 127 (max velocity) yields a gain 1.0 (full volume), a velocity of 63 (half velocity) yields a gain of 0.5 (half volume), etc.	(optional)
`grou`	Group-level pitch adjustment for changing note pitch. In semitones. For example 1.0 would be a half-step up and -1 would a half-step down. Default: 0	(optional)

The <sample> element

Underneath the `<group>` elements are `<sample>` elements. Each sample corresponds to a playable "zone" of your instrument. Attributes:

At- tribut	Description
`path` (re- quire	The relative path of the sample file to play for this zone.
`root` (re- quire	The MIDI note number (from 1 to 127) of the note.
`loNo` (op- tiona	The MIDI note number (from 1 to 127) of the lowest note for which the zone should be triggered. Default: 0.
`hiNo` (op- tiona	The MIDI note number (from 1 to 127) of the highest note for which the zone should be triggered. Default: 127.
`loVe` (op- tiona	The lowest velocity for which this zone should be triggered. Default: 0
`hiVe` (op- tiona	The highest velocity for which this zone should be triggered. Default: 127
`loCC` `hiCC` (op- tiona	Using these parameter, you can use MIDI continuous controllers to filter whether or not a note should be played. This lets you, for example, have one set of samples that get played when the piano sustain pedal is down and another set that get played when it is up. Each time a MIDI CC value comes for a specific CC#, the engine stores that value. When a "note on" signal is received, the engine makes a decision (based on the last received value and the range defined by these attributes) about whether or not this sample should be played. If you use `loCCN`, you must also use a corresponding `hiCCN` for the same MIDI CC number so that you are defining a range of values. Example: `loCC64="90"` and `hiCC64="127"` would mean that a "note on" message will only trigger this sample if the last received value for CC64 (Sustain Pedal) is between 90 and 127. This can also be set at the <`group`> level. Default:-1 (off)
`star` (op- tiona	The frame/sample position of the start of the sample audio. This is useful if the sample starts midway through the audio file. Default: 0
`end` (op- tiona	The frame/sample position of the end of the sample audio. The is useful is the zone ends before the end of the audio file. Default: the file's length in samples minus 1.
`tuni` (op- tiona	A fine-tuning number (in semitones) for changing the note pitch. e.g 1.0 would be a half-step up. Default: 0
`volu` (op- tiona	The volume of the sample. Value can be in linear 0.0-1.0 or in decibels. If it's in decibels you must append dB after the value (example: "3dB"). Default: 1.0
`pan` (op- tiona	A number of -100 to 100. -100 in panned all the way to the left, 100 is panned all the way to the right. This can also be set at the <`group`> or <`groups`> levels. Default: 0
`pitc` (op- tiona	A number from 0.0 to 1.0. 0 means that the pitch will stay the same regardless of what note is played. 1 means that the pitch will increase by one semitone when the note increases by one semitone (i.e. normal key pitch tracking). This can also be set at the <`group`> level. Default: 1
`trig` (op- tiona	Valid values: `attack` means a sample is played when the *note on* message is received. `release` means the sample is played when the *note off* message is received (aka a release trigger). `first` means that the sample will only be played if no other notes are playing. `legato` means that the sample will only be played if *some* other notes are already playing. This can also be set at the <`group`> level. Default: `attack`.
`tags` (op- tiona	A command-separated list of tags. Example: tags="rt,mic1". These are useful when controlling volumes using tags. See Appendix D.
`onLo` `onHi` (op- tiona	If you want a sample to be triggered when a MIDI CC controller message comes in, for example for piano pedal down and pedal up samples, you use these attributes to specify the range of values that should trigger the sample. If you use `onLoCCN`, you must also use a corresponding `onHiCCN` for the same MIDI CC number. Example: `onLoCC64="90"` and `onHiCC64="127"` would mean that values of CC64 (Sustain Pedal) between 90 and 127 will trigger the given sample. This can also be set at the <`group`> level. Default:-1 (off)
`play` (op- tiona	By default, the choice of playback engine is left up to the discretion of the user–they can set this in the Preferences screen. However, a sample creator can override this setting by setting the playbackMode for a specific sample, a group, or the entire instrument. Possible values are `memory`, `disk_streaming`, and `auto` (default).

Looping

At- tribute		Description
loopS	(op- tiona	The frame/sample position of the start of the sample's loop. If this is not specified, but the sample is a wave file with embedded loop markers, those will be used instead. Default: 0
loopE	(op- tiona	The frame/sample position of the end of the sample's loop. If this is not specified, but the sample is a wave file with embedded loop markers, those will be used instead. Default: the file's length in samples minus 1.
loopC	(op- tiona	When loop crossfades are used, instead of simply looping at a specific end point, a portion of the audio from before the loop point is faded in just as the audio from the end of the loop is faded out. In this way, smooth audio loops can be achieved on samples that weren't specifically prepared as looping. This parameter is used for specifying the length of the crossade region in frames/samples. This can also be set at the <`group`> level. Default: 0 (crossfades off).
loopC	(op- tiona	This parameter is used to specify the curve used for crossfading when loop crossfades are turned on. This can also be set at the <`group`> level. Value values: `linear`, `equal_power`. Default: `equal_power`.
loopE	(op- tiona	A boolean value indicating whether or not the loop should be used. Valid values: true, false

Amplitude Envelope

Each sample has its own ADSR amplitude envelope.

At- tribute		Description
attack	(op- tional)	The attack time in seconds of the amplitude envelope of this zone. This can also be set at the <`group`> or <`groups`> levels.
decay	(op- tional)	The decay time in seconds of the amplitude envelope of this zone. This can also be set at the <`group`> or <`groups`> levels.
sustai	(op- tional)	The sustain level (0.0 - 1.0) of the amplitude envelope of this zone. This can also be set at the <`group`> or <`groups`> levels.
releas	(op- tional)	The release time in seconds of the amplitude envelope of this zone. This can also be set at the <`group`> or <`groups`> levels.

The curve shapes of the attack, decay, and release zones can be changed as well. All three of the of the following parameters use the same system: a value from -100 to 100 that determines the shape of the curve. **-100 is a logarithmic curve, 0 is a linear curve, and 100 is an exponential curve.**

At- tribute		Description	Default Value
attack	(op- tional	A value from -100 to 100 that determines the shape of the attack curve. This can also be set at the <`group`> or <`groups`> levels.	-100 (log- arith- mic)
decayC	(op- tional	A value from -100 to 100 that determines the shape of the decay curve. The decay time in seconds of the amplitude envelope of this zone. This can also be set at the <`group`> or <`groups`> levels.	100 (expo- nential)
releas	(op- tional	A value from -100 to 100 that determines the shape of the release curves. The release time in seconds of the amplitude envelope of this zone. This can also be set at the <`group`> or <`groups`> levels.	100 (expo- nential)

Round Robins

Round robins allow different samples to be played each time a zone is triggered. This is especially useful with sounds that have short attacks (such as drums), and is a great way to keep your sample libraries from sounding fake. In order for round robins to work, you must specify both a `seqMode` and a `seqPosition` for all samples. If you have several different sets of round robins with different lengths, you'll want to set the `seqLength` value as well. There are several round-robin modes:

- `round_robin`: This causes samples to be triggered sequentially according to their `seqPosition` values.
- `random`: This causes random samples to be chosen from within the group of samples. If there are more than two round robins, then the algorithm makes sure not to hit the same one twice in a row.
- `true_random`: This causes random samples to be chosen from within the group of samples.
- `always`: This just turns round robins off.

Attribute		Description
seqMod	(optional	Valid values are `random`, `true_random`, `round_robin`, and `always`. A value indicating the desired round robin behavior for this sample or group of samples. This can also be set at the <`group`> and <`groups`> levels. Default: `always`
seqLen	(optional	The length of the round robin queue. This can also be set at the <`group`> or <`groups`> levels. If this is left out, then the engine will try to auto-detect the length of the roudn robin sequence. Default: 0
seqPos	(optional	A number indicating this zone's position in the round robin queue. This can also be set at the <`group`> level. Default: 1

Voice-Muting / Legato

Looping

Attribut		Description
sile	(optiona	A command-separated list of tags. Example: tags="rt,mic1". If a sample containing one of these tags gets triggered, then this sample will be stopped. This is useful when setting up drums as it will allow you mute one hi-hat when another hi-hat plays. See Appendix D.
sile	(optiona	Controls how quickly voices get silenced. `fast` = immediately; `normal` = triggers the sample's release phase. This second option, when used in conjunction with the `release` attribute, allows you to specify a longer release time. Values: `fast`, `normal`. Default: `fast`
prev	(optiona	Only play this sample if the previously triggered note equals one of these notes. Format: a comma-separated list of MIDI note numbers (from 1 to 127) of the note.
lega	(optiona	This is similar to the `previousNote` attribute. This causes the engine to only play the sample if the previously triggered note is exactly this semitone distance from the previous note. For example, if the note for which this sample is being triggered is a C3 and the `legatoInterval` is set to `-2`, then the sample will only play if the previous note was a D3 because D3 minus 2 semitones equals C3. Format: This can be a positive or negative whole number.

1.4 The <effects> element

Adding global, instrument-wide effects is easy: just add an `<effects>` element right below your top-level `<DecentSampler>` element.

It is also possible to have effects that only get added to a specific group. To adding effects that only apply to a specific group, all you need to do is create an `<effects>` group that lives underneath the `<group>` element for the group you want to affect.

Group level effects are initialized every time a note is started and destroyed every time a note is stopped. If you play two notes simultaneously, two instances of this effect will be created and these will be independent of eachother. As a result, they use more CPU than global effects.

NOTE: Only certain effects will work as group-level effects: lowpass filter, hipass filter, bandpass filter, gain, and chorus. Delay and reverb cannot work properly as they will be deleted before their tail peters out.

1.4.1 The <effect> element

Within the `<effects>` element, you can have any number of `<effect>` sub-elements. These specify parameters for each individual effect that you would like to have in your global effects chain. There are currently only a handful effects available although more could definitely be added on request:

Low-pass, Band pass, and Hi-pass filter

A 2-pole resonant filter that can be either a lowpass, bandpass, or highpass filter

Example:

```xml
<DecentSampler>
  <effects>
    <effect type="lowpass" resonance="0.7" frequency="22000" />
  </effects>
</DecentSampler>
```

There is also a single pole version of the lowpass filter that can be accessed using the `lowpass_1pl` effect type. This version does not have a resonance parameter.

```xml
<DecentSampler>
  <effects>
    <effect type="lowpass_1pl" frequency="22000" />
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Valid Range	Default
`type`	Required	The type of filter	Must be either `lowpass` (legacy: `lowpass_4pl`), `lowpass_1pl`, `bandpass`, or `highpass`	
`resonanc`	Optional	The filter resonance (Q)	0 - 1.0, where 1.0 is big, 0 is small.	0.7
`frequenc`	Optional	The filter frequency	0 - 1.0, where 0 is not damped, 1.0 is fully damped.	0.3

Notch EQ Filter

A simple notch filter.

Example:

```
<DecentSampler>
  <effects>
    <effect type="notch"  q="0.7" frequency="22000" />
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Valid Range	Default
`type`	Required	The type of filter	Must be `notch`	
`frequency`	Required	The filter frequency	60 - 22000.0	10000
`Q`	Optional	Q is the ratio of center frequency to bandwidth	0.01 - 18.0	0.7

Peak EQ Filter

A peak filter centred around a given frequency, with a variable Q and gain.

Example:

```
<DecentSampler>
  <effects>
    <effect type="peak"  q="0.7" frequency="22000" gain="2.0" />
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Valid Range	Default
`type`	Required	The type of filter	Must be `peak`	
`frequenc`	Required	The filter frequency	60 - 22000.0	10000
`Q`	Optional	Q is the ratio of center frequency to bandwidth	0.01 - 18.0	0.7
`gain`	Required	Values greater than 1.0 will boost the high frequencies, values less than 1.0 will attenuate them.	0 - 1.0	1.0

Gain effect

Applies a volume boost or cut to the output signal.

Example:

```
<DecentSampler>
  <effects>
    <effect type="gain" level="-6" />
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Default	Valid Range
`type`	Required	Must be `gain`		`gain`
`level`	Required	The amount of gain to be applied expressed in decibels. In other words, gain of -6dB reduces sound by 50%	0	-99 - 24

Reverb effect

Example:

```
<DecentSampler>
  <effects>
    <effect type="reverb" roomSize="" damping="" wetLevel="" />
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Valid Range	Default
`type`	Required	Must be `reverb`	`reverb`	
`roomSize`	Optional	The reverb “room size”	0 - 1.0, where 1.0 is big, 0 is small.	0.7
`damping`	Optional	The reverb damping level	0 - 1.0, where 0 is not damped, 1.0 is fully damped.	0.3
`wetLevel`	Optional	The volume of reverb signal	0 - 1.0	0

Delay effect

A simple delay effect that can be controlled either in seconds or using musical time increments based on the host tempo. For a complete explanation of how to use tempo-syncing, see here.

Attributes:

At-tribut		Type	Valid Range	De-fault
`type`	Re-quire	Must be `delay`	`delay`	
`dela`	Op-tiona	Determines whether the delay will be synced to DAW tempo or not, as well as what format you will be using for the `delayTime` parameter. There are two possible values: 1) `seconds`, which is the default, means that `delayTime` will be specified in seconds and will not change when the DAW tempo changes; 2) `musical_time` means that delay time will be specified using an integer value generated by a control which is setup to use the `musical_time` `valueType` parameter. In order for this to work, you will need to be using the plug-in within a DAW that provides a tempo to the plug-in.	`seconds` `musical`	`seconds`
`dela`	Op-tiona	The delay time in seconds	0 - 20.0	0.7
`feed`	Op-tiona	The feedback level.	0 - 1.0, where 0 is no feed-back, 1.0 is max feed-back.	0.2
`ster`	Op-tiona	The parameter allows you to introduce delay variations between the left and right channels. Half of this amount is subtracted from the left channel's delay time and half of this amount is added to the right channel's delay time. For example, if the `delayTime` is 0.5 seconds and the `stereoOffset` is 0.02 s, then the actual left channel delay time will be 0.49s and the actual right channel delay time will be 0.51s so that the two channels are offset by 0.02 seconds.	-10 - 10	0
`wetL`	Op-tiona	The volume of the delay signal	0 - 1.0	0.5

Example of using the delay effect when specifying time in seconds:

```
<DecentSampler>
  <effects>
    <effect type="delay" delayTime="0.5" stereoOffset="0.01" feedback="0.2" wetLevel="0.5
↪" />
  </effects>
</DecentSampler>
```

Example of using the delay effect when specifying time in musical time:

```
<DecentSampler>
  <ui>
    <tab>
      <labeled-knob x="180" y="40" label="Delay Time" valueType="musical_time"
```

(continues on next page)

(continued from previous page)

```
                    minValue="0" maxValue="20" value="10" defaultValue="10">
        <binding type="effect" level="instrument" position="0" parameter="FX_DELAY_TIME"␣
↪/>
      </labeled-knob>
    </tab>
  </ui>
  <effects>
    <effect type="delay" delayTimeFormat="musical_time" delayTime="10" stereoOffset="0.01
↪"
            feedback="0.3" wetLevel="0.5" />
  </effects>
</DecentSampler>
```

Chorus effect

Example:

```
<DecentSampler>
  <effects>
    <effect type="chorus" mix="0.5" modDepth="0.2" modRate="0.2"/>
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Valid Range	Default
`type`	Required	Must be `chorus`	`chorus`	
`mix`	Optional	The wet/dry mix which controls how much of the chorus signal we hear	0 - 1.0, where 1.0 is just chorus, 0 is just dry signal.	0.5
`modDep`	Optional	The modulation depth of the effect	0 - 1.0, where 0 is no modulation, 1.0 is max modulation.	0.2
`modRat`	Optional	The modulation speed in Hz.	0 - 10.0	0.2

Phaser effect

Example:

```
<DecentSampler>
  <effects>
    <effect type="phaser" mix="0.5" modDepth="0.2" modRate="0.2" centerFrequency="400"␣
↪feedback="0.7" />
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Valid Range	Default
`type`	Required	Must be `phaser`	`phaser`	
`mix`	Optional	The wet/dry mix which controls how much of the chorus signal we hear	0 - 1.0, where 1.0 is just chorus, 0 is just dry signal.	0.5
`modDepth`	Optional	The modulation depth of the effect	0 - 1.0, where 0 is no modulation, 1.0 is max modulation.	0.2
`modRate`	Optional	The modulation speed in Hz.	0 - 10.0	0.2
`centerFreq`	Optional	The center frequency (in Hz) of the phaser all-pass filters modulation	0 - 1.0	400
`feedback`	Optional	Sets the feedback volume of the phaser.	-1 - 1.0	0.7

Convolution effect

This effect allows you to use a convolution reverb or amp simulation to your sample library. Depending on the length of the impulse response, the convolution effect can use substantial CPU, so you'll definitely want to do some testing both with and without the convolution effect turned on.

Example:

```
<DecentSampler>
  <effects>
    <effect type="convolution" mix="0.5" irFile="Samples/Hall 3.wav" />
  </effects>
</DecentSampler>
```

Attributes:

Attribute		Type	Valid Range	Default
`type`	Required	Must be `convolution`	`convolution`	
`mix`	Optional	The wet/dry mix controls how much of the convolution signal we hear	0 - 1.0, where 1.0 is just convolution, 0 is just dry signal.	0.5
`irFile`	Required	The path of the WAV or AIFF to use as an Impulse Response (IR) file	String	No default

Wave Folder effect

Introduced in version 1.7.2. This effect allows you to fold a waveform back on itself. This is very useful for generating additional harmonic content.

Attributes:

Attribute		Type	Valid Range	Default
type	Required	Must be wave_folder	wave_folder	
drive	Optional	The volume of the input signal	1 - 100, where 100 means the signal is amplified by a factor of 100 and 1 means no amplification is applied	1
thresh	Optional	The amplitude above which wave folding should take place	0 - 10.0	0.25

Because wave folding tends to sound better when applied on a per-voice basis, it usually makes sense to set up the wave folder at the group level (separate group effects get created for each keypress). Example:

```
<DecentSampler pluginVersion="1">
  <ui>
    <tab>
      <labeled-knob x="180" y="40" label="Drive" type="float" minValue="1" maxValue="100
↪" textColor="FF000000" value="1">
        <binding type="effect" level="group" groupIndex="0" effectIndex="0" parameter=
↪"FX_DRIVE" translation="linear" />
      </labeled-knob>
      <labeled-knob x="280" y="40" label="Threshold" type="float" minValue="0" maxValue=
↪"1" value="1" textColor="FF000000">
        <binding type="effect" level="group" groupIndex="0" effectIndex="0" parameter=
↪"FX_THRESHOLD" translation="linear" />
      </labeled-knob>
    </tab>
  </ui>
  <groups>
    <group>
        <!-- Samples go here. -->
    <effects>
        <effect type="wave_folder" drive="1" threshold="1" />
      </effects>
    </group>
  </groups>
```

Wave Shaper effect

Introduced in version 1.7.2. This effect allows you to distort an audio signal. This is very useful for generating additional harmonic content.

Attributes:

At- tribute		Type	Valid Range	De- fault
`type`	Re- quire	Must be `wave_shaper`	`wave_folder`	
`drive`	Op- tiona	The amount of distortion. This really just controls the volume of the input signal. The volume of the input signal	1 - 1000, where 1000 means the signal is amplified by a factor of 1000 and 1 means no amplification is applied	1
`drive`	Op- tiona	Introduces an extra gain boost to the drive	0 - 1.0	1
`outpu`	Op- tiona	The linear output level of the signal	0 - 1.0	0.1
`highQ`	Op- tiona	Whether or not oversampling is performed. Over- sampling sounds better, but it's CPU intensive. If you want to save CPU, set this to false.	true, false	true

Because wave shaping tends to sound better when applied on a per-voice basis, it usually makes sense to set up the wave shaper at the group level (separate group effects get created for each keypress). Example:

```
<DecentSampler pluginVersion="1">
  <ui>
    <tab>
      <labeled-knob x="180" y="40" label="Drive" type="float" minValue="1" maxValue="40"␣
↪textColor="FF000000" value="0.5473124980926514">
        <binding type="effect" level="group" groupIndex="0" effectIndex="0" parameter=
↪"FX_DRIVE" translation="linear"/>
      </labeled-knob>
      <labeled-knob x="280" y="40" label="Drive Boost" type="float" minValue="0"␣
↪maxValue="1" value="0.328312486410141" textColor="FF000000">
        <binding type="effect" level="group" groupIndex="0" effectIndex="0" parameter=
↪"FX_DRIVE_BOOST" translation="linear"/>
      </labeled-knob>
      <labeled-knob x="380" y="40" label="Output Lvl" type="float" minValue="0" maxValue=
↪"1" value="0.328312486410141" textColor="FF000000">
        <binding type="effect" level="group" groupIndex="0" effectIndex="0" parameter=
↪"FX_OUTPUT_LEVEL" translation="linear"/>
      </labeled-knob>
    </tab>
  </ui>
  <groups>
    <group>
        <!-- Samples go here. -->
      <effects>
        <effect type="wave_shaper" drive="0.5473124980926514" shape="0.328312486410141"␣
↪outputLevel="0.1"/>
      </effects>
    </group>
  </groups>
```

1.5 The <midi> element

MIDI mappings can be added to your instrument by adding a `<midi>` element right below your top-level `<DecentSampler>` element.

1.5.1 The <cc> element

Within the `<midi>` element, you can have any number of `<cc>` elements. These allow you to map changes in incoming continuous controller messages to specific parameters of your instrument. To use this functionality, you'll want to add a separate `<cc>` element for each CC number you would like to respond to. The `<cc>` element has a single required attribute `number=""` which specifies the number (from 0 to 127) of the continuous controller you would like to listen on. Beneath the `<cc>` element, you can have any number of bindings.

```
<midi>
  <cc number="11">
    <binding level="ui" type="control" position="0" parameter="VALUE" translation="linear
↪"
             translationOutputMin="0" translationOutputMax="1"/>
  </cc>
  <cc number="1">
    <binding level="ui" type="control" position="1" parameter="VALUE" translation="linear
↪"
             translationOutputMin="0" translationOutputMax="1"/>
  </cc>
</midi>
```

1.5.2 The <note> element

Within the `<midi>` element, you can have any number of `<note>` elements. These allow you to map specific notes to specific parameters of your instrument. To use this functionality, you'll want to add a separate `<note>` element for each MIDI note or range of notes you would like to respond to.

Here are the attributes of the `<note />` element:

- **note** (required): This attribute specifies the MIDI note number (from 0 to 127) you would like to listen on. You can also specify ranges of notes by using a dash. For example `note="24-35"` would be used to specify bindings for the range of notes 24 thorugh 35.
- **enabled** (optional): A true/false value that specifies whether this note listener is turned on.
- **swallowNotes** (optional): The bindings that live below this note listener are called before any notes are played. By default, swallowNotes is false, which means that the keypress will then be received by the sampler. If `swallowNotes` is true, the sampler will not receive the note. This is useful if you wish to prevent certain keys from triggers notes.

It is possible to enable and disable a note listener by targeting the `enabled` attribute.

Beneath the `<note>` element, you can have any number of bindings. Here is an example of how keyswitches might be set up:

```
<midi>
  <note note="11" enabled="true">
    <binding enabled="true" type="general" level="group" groupIndex="0" parameter=
↪"ENABLED" translation="fixed_value" translationValue="true" />
```

(continues on next page)

(continued from previous page)

```
    <binding enabled="true" type="general" level="group" groupIndex="1" parameter=
↪"ENABLED" translation="fixed_value" translationValue="false" />
  </note>
  <note note="12" enabled="true">
    <binding enabled="true" type="general" level="group" groupIndex="0" parameter=
↪"ENABLED" translation="fixed_value" translationValue="false" />
    <binding enabled="true" type="general" level="group" groupIndex="1" parameter=
↪"ENABLED" translation="fixed_value" translationValue="true" />
  </note>
</midi>
```

In the above keyswitch example, MIDI note 11 turns on group 0 and turns off group 1, whereas MIDI note 12 does the opposite. Note the use of the `fixed_value` translation type.

Bindings within the `<midi>` section

The bindings that the `<cc>` and `<note>` element listens on are the same as those used by the UI controls. See *Appendix B* for a complete description of these.

If you have a UI control mapped to the same internal parameter as a MIDI mapping, you'll want to have your MIDI mapping control the UI control instead of the parameter directly. The benefit of doing this is that, as the MIDI CC input is received, the UI control will be updated as well as the desired internal parameter.

The way to accomplish this is to make use of the `labeled_knob` or `control` binding types (`control` was introduced in version 1.1.7) as follows:

```
<binding level="ui" type="control" position="0" parameter="VALUE" translation="linear"␣
↪translationOutputMin="0" translationOutputMax="1"/>
```

You'll notice that the `control` type has a `level` value of `ui` and a `parameter` value of `VALUE`. Another thing to notice is the `position=""` parameter. This contains the 0-based index of the control to be modified. **NOTE: The indexes of the parameter list includes all UI controls, including `<label>` and `menu` controls, so you'll want to account for that when calculating your positions.**

An example of changing a menu option based on a MIDI note (keyswitch) would look like this:

```
<midi>
  <note note="11">
    <binding type="control" level="ui" position="1" parameter="VALUE" translation="fixed_
↪value" translationValue="1" />
  </note>
  <note note="12">
    <binding type="control" level="ui" position="1" parameter="VALUE" translation="fixed_
↪value" translationValue="2" />
  </note>
</midi>
```

1.6 The <noteSequences> element

As of 1.11.1, DecentSampler has a built-in note sequencer. It can be mapped to keys so that clusters of notes are played every time certain keys are pressed. It can also be mapped to UI controls such as buttons.

The <`noteSequences`> element is how you specify note sequences that can be used by this playback engine. There should be exactly one <`noteSequences`> element in each <`DecentSampler`> file.

The <`noteSequences`> element can contain one or more <`sequence`> elements:

1.6.1 The <sequence> element

The <`sequence`> element has the following attributes:

- `name` (optional): An optional descriptive name for the sequence. This is only used in the sample editor UI to help you identify the sequence.
- `length` (required): The length of the sequence in beats. This is a floating point number.
- `rate` (optional): The rate at which the sequence is played. This is a floating point number. The default is 1.0.

The <`sequence`> element can contain one or more <`note`> elements:

The <note> element

The <`note`> element has the following attributes:

- `position` (required): The position of the note in the sequence, in beats. This is a whole number.
- `velocity` (required): The velocity of the note. This is a floating point number between 0 and 1.
- `note` (required): The MIDI note number of the note.
- `length` (required): The length of the note in beats. This is a whole number.

This is an an example showing the strum from an Omnichord:

```
<noteSequences>
    <sequence name="Maj1Slow" length="768.0" rate="96">
        <note position="0" velocity="1" note="48" length="768"/>
        <note position="11" velocity="1" note="52" length="757"/>
        <note position="29" velocity="1" note="55" length="739"/>
        <note position="46" velocity="1" note="60" length="722"/>
        <note position="63" velocity="1" note="64" length="705"/>
        <note position="81" velocity="1" note="67" length="687"/>
        <note position="99" velocity="1" note="72" length="669"/>
        <note position="117" velocity="1" note="76" length="651"/>
        <note position="134" velocity="1" note="79" length="634"/>
        <note position="153" velocity="1" note="84" length="615"/>
        <note position="171" velocity="1" note="88" length="597"/>
        <note position="192" velocity="1" note="91" length="576"/>
    </sequence>
</noteSequences>
```

For a full discussion of how to use note sequences, see the tutorial on *how to use note sequences*.

1.7 The <modulators> element

Version 1.6.0 of Decent Sampler officially introduces the new `<modulators>` section into the `.dspreset` format. This section lives below the top-level `<DecentSampler>` element and it is where all modulators for the entire sample library live.

```
<DecentSampler>
    <modulators>
        <!-- Your modulators go here. -->
    </modulators>
</DecentSampler>
```

1.7.1 The <lfo> element

Underneath the `<modulators>` section, you can have any number of different LFOs, which are defined using an `<lfo>` element, for example:

```
<modulators>1
  <lfo shape="sine" frequency="2" modAmount="1.0"></lfo>
</modulators>
```

This element has the following attributes:

- **`shape`**: controls the oscillator shape. Possible values are `sine`, `square`, `saw`.
- **`frequency`**: The speed of the LFO in cycles per second. For example, a value of 10 would mean that the waveform repeats ten times per second.
- **`modAmount`**: This value between 0 and 1 controls how much the modulation affects the things it is targeting. In conventional terms, this is like the modulation depth. Default value: 1.0.
- **`scope`**: Whether or not this LFO exists for all notes or whether each keypress gets its own LFO. Possible values are `global` (default for LFOs) and `voice`. If `voice` is chosen, a new LFO is started each time a new note is pressed.

1.7.2 The <envelope> element

In addition to LFOs, you can also have additional ADSR envelopes. These can be useful for controlling group-level effects, such as low-pass filters. If this is what you wish to achieve, make sure you check out the section on group-level effects below.

To create an envelope, use an `<envelope>` element:

This element has the following attributes:

- **`attack`**: The length in seconds of the attack portion of the ADSR envelope
- **`decay`**: The length in seconds of the decay portion of the ADSR envelope
- **`sustain`**: The height of the sustain portion of the ADSR envelope. This is expressed as a value between 0 and 1.
- **`release`**: The length in seconds of the release portion of the ADSR envelope
- **`modAmount`**: This value between 0 and 1 controls how much the modulation affects the things it is targeting. In conventional terms, this is like the modulation depth. Default value: 1.0.

- **scope**: Whether or not this LFO exists for all notes or whether each keypress gets its own LFO. Possible values are `global` and `voice` (default for envelopes). If `voice` is chosen, a new LFO is started each time a new note is pressed.

1.7.3 How to use <binding>s in conjunction with modulators

In order to actually have your LFOs and envelopes do anything, you need to have bindings under them. If you are not familiar with the concept of bindings, you may want to read this section then return here. Bindings tell the engine which parameters the LFO should be affecting and how. Here is an example:

```
<modulators>
    <lfo shape="sine" frequency="2" modAmount="1.0">
        <!-- This binding modifies the frequency of a low-pass filter  -->
        <binding type="effect" level="instrument" effectIndex="0" parameter="FX_FILTER_
↪FREQUENCY" modBehavior="add" translation="linear" translationOutputMin="0"␣
↪translationOutputMax="2000.0"  />
    </lfo>
</modulators>
```

There are a few differences between bindings as they are used by knobs and the ones used by modulators. Specifically, when you move a UI control that has a binding attached, the engine actually goes out and changes the value of the parameter that is targeted by that binding. For example, if you have a knob that controls a lowpass filter's cutoff frequency, moving that knob will cause that actual frequency of that filter to change. In other words, the changes that the knob is making on the underlying sample library are *permanent*. The same is also true for bindings associated with MIDI continuous controllers.

Modulators, on the other hand, do not work this way. If a modulator (such as an LFO) changes its value, the engine looks at the bindings associated with that LFO and then makes a list of *temporary* changes to the underlying data. When it comes time to render out the effect, it consults both the *permanent* value and the *temporary* modulation values. As a result of this difference in the way bindings are handled, only some parameters are "modulatable." At time of press, the following parameters are modulatable:

- All gain effect parameters
- All delay effect parameters
- All phaser effect parameters
- All filter effect parameters
- All reverb effect parameters
- All chorus effect parameters
- Group Volume
- Global Volume
- Group Pan
- Global Pan
- Group Tuning
- Global Tuning

1.7.4 Modulator scope: global or voice-level

By default, all modulators will be created at the global level. This means that there will be exactly one modulator that is shared by all voices. In many situations, such as an LFO modulating a single low-pass filter which is shared by all of voices, this is often what we want.

But there are other situations where we don't want our modulator to be global. In such cases we

```
<modulators>
    <envelope attack="2" decay="0" sustain="1" release="0.5" modAmount="1.0">
        <binding type="effect" level="group" groupIndex="0" effectIndex="0" parameter=
↪"FX_FILTER_FREQUENCY" translation="linear" translationOutputMin="0"␣
↪translationOutputMax="4000.0" modBehavior="add" />
    </envelope>
</modulators>
```

Note that this voice-level modulator is now targeting a group level effect.

1.8 The <tags> element

The `<tags>` element lives right below your top-level `<DecentSampler>` element. It allows you to specify details about the tags you use throughout your instrument. It is however not actually necessary to include a `<tags>` element for every tag you use. You only need to create this if you want to specify additional details about your tags.

1.8.1 The <tag> element

Underneath the `<tags>` element, you can have any number of `<tag>` elements. These specify details for each individual tag that you use throughout your sample mapping. Attributes:

Attribute		Description
enabled	(optional)	A number for 0.0 to 1.0 that specifies the initial volume for a tag. Default: 1.0
volume	(optional)	A number for 0.0 to 1.0 that specifies the initial volume for a tag. Default: 1.0
polyphony	(optional)	A whole number that specifies the number of voices allowed for this tag.

1.9 Appendix A: The Color Format

Colors are represented throughout the **dspreset** files using an 8-digit ARGB color format. These are identical to web color hex codes except with an additional 2-digit hex number in front of them. The first two digits are a hexadecimal representation of alpha level with 00 being fully transparent, 80 being 50% transparent, and FF being fully opaque.

Examples:

- Black (solid): FF000000
- Black (90% transparency): E6000000
- Red (solid): FFFF0000
- Red (50% transparency): 80FF0000
- Blue (solid): FF0000FF

1.10 Appendix B: The <binding> element

Adding a binding to a UI control, MIDI handler, or modulator tells the DecentSampler engine that it should take input from a source and use it to change values in another part of the engine. An example of this would be a knob which controls the volume of a group, a CC controller that changes an effect parameter, or an LFO that modulates an effect parameter.

In order to set up a binding for a specific source, create a `<binding>` element within the source element.

In this example, a labeled knob is controlling the volume of the first group of samples (group 0):

```
<DecentSampler>
  <ui>
    <tab>
      <labeled-knob x="420" y="100" label="RT" type="float" minValue="0" maxValue="1"␣
↪value="0.3" textSize="20">
        <binding type="amp" level="group" position="0" parameter="AMP_VOLUME"␣
↪translation="linear" translationOutputMin="0" translationOutputMax="1.0"  />
      </labeled-knob>
    </tab>
  </ui>
</DecentSampler>
```

Here’s a full list of parameters for the `<binding>` element:

At-tribute	Description	
`type`	This tells the engine what type of parameter this is. Valid values are: "amp", "effect", "control".	Re-quired
`level`	Valid values are `ui`, `instrument`, `group`	Re-quired
`positio`	The specific 0-based index of the element to be modified by this binding. If you are targeting a group, for example, the first group would be 0, the second group would be 1, etc.	Re-quired
`control`	When a binding is targeting a control, this is the same thing as the `position` attribute. It is a specific 0-based index of the control to be modified by this binding. If you are targeting an group-level effect, this would specified the group under which the effect lives.	Op-tional
`groupIn`	When a binding is targeting a group, this is the same thing as the `position` attribute. It is a specific 0-based index of the group to be modified by this binding. If you are targeting an group-level effect, this would specified the group under which the effect lives.	Op-tional
`effectI`	When a binding is targeting an effect, this is the same thing as the `position` attribute. It is a specific 0-based index of the effect to be modified by this binding.	Op-tional
`modulat`	When a binding is targeting a modulator, this is the same thing as the `position` attribute. It is a specific 0-based index of the modulator to be modified by this binding.	Op-tional
`tags`	A comma-separated list of tags to be modified by this binding. This allows you to set values for multiple groups at once by targeting a tag that is assigned to the groups.	Op-tional
`enabled`	A value that turns the binding on and off. Valid values are: `true`, `false`.	Op-tional
`identifi`	A string identifying the specific parameter that you wish to change. If you are modulating based on tags, you would put the tag you are targeting here. See Appendix D for example.	Re-quired
`paramet`	A token describing the specific kind of parameter that you wish to change. A list of controller parameters are below.	Re-quired
`transla`	Valid values are `fixed_value`, `linear` and `table`. Explanation of both translation modes is in a separate section below. Default: `linear`	Op-tional
`transla`	This is the min value this binding should send to the target parameter. This is only looked at if translation is set to `linear`.	Op-tional
`transla`	This is the max value this binding should send to the target parameter. This is only looked at if translation is set to `linear`.	Op-tional
`transla`	Valid values are `true` and `false`. Default: `false`. This is only looked at if translation is set to `linear`.	Op-tional
`transla`	A list of input-output pairs that make up the translation table. The input and output are separated by commas. The groups of coordinates themselves are separated by semi-colons. Default: `0,0;1,1`. You must have at least two coordinates in your list. This is only looked at if translation is set to `table`.	Op-tional
`transla`	The value that should be passed along when `translation` is set to `fixed_value`.	Op-tional

1.10.1 Binding Parameters for Targeting Note Sequences

A special set of binding attributes exist for targeting note sequences:

Attribute	Description	Default	Required
seqIndex	A 0-based index of a sequence underneath the <`noteSequences`> section	None	Required
seqTrigge	What the binding should do with the sequence in question. Valid values are `on` (start playing the sequence), `off` (stop playing the sequence), `midi_key` (special value that will cause the binding to follow a specific MIDI key note binding)	`midi_key`	Optional
seqPlayer	An identifier used for tracking the state of a sequence. This value can be any sequence of numbers or letters.	None	Required when `seqTriggerBehavior` is `on` or `off`
seqTrackM	Whether or not the sequence should respect the velocity of the incoming MIDI note. This can only be used when the sequence is being triggered by a MIDI note binding. Value should be a floating point number from 0.0 to 1.		Optional
seqTransp	Transpose the notes in the sequence by an arbitrary number of half steps. Value should be a floating point number from -36 to 36.	Any sequence of numbers of letters	Optional
seqTransp	Transpose the notes in the sequence relative to the pitch of the incoming MIDI note. This can only be used when the sequence is being triggered by a MIDI note binding. Value should be a floating point number from 0 to 127.	Any sequence of numbers of letters	Optional
seqPlayba	The speed of playback. Value should be a floating point number from 0.001 to 10000.	1.0	Optional
seqLoopMo	Valid values are: `forward`, `reverse`, `random`, `random_no_repeat`, `no_loop`	`forward`	Optional

1.10.2 Controllable Parameters

This is a list of parameters that can be used in conjunction with the <`binding`> element above. NOTE: The table below scrolls to the right.

Description	`type`	`level`	`parameter`
Global Volume	`amp`	`instrument`	`AMP_VOLUME`
Global Tuning	`amp`	`instrument`	`GLOBAL_TUNI`
Global Pan	`amp`	`instrument`	`PAN`
Sample Start (see note 2 below)	`general`	`instrument` or `group`	`SAMPLE_STAR`
Sample End (see note 2 below)	`general`	`instrument` or `group`	`SAMPLE_END`
Loop Start (see note 2 below)	`general`	`instrument` or `group`	`LOOP_START`
Loop End (see note 2 below)	`general`	`instrument` or `group`	`LOOP_END`
Amplitude Velocity Tracking	`amp`	`instrument`	`AMP_VEL_TRA`
Global Amp Envelope Attack	`amp`	`instrument`	`ENV_ATTACK`
Global Amp Envelope Attack Curve Shape	`amp`	`instrument`	`ENV_ATTACK_`
Global Amp Envelope Decay	`amp`	`instrument`	`ENV_DECAY`
Global Amp Envelope Decay Curve Shape	`amp`	`instrument`	`ENV_DECAY_C`
Global Amp Envelope Sustain	`amp`	`instrument`	`ENV_SUSTAIN`

Description	type	level	parameter
Global Amp Envelope Release	amp	instrument	ENV_RELEASE
Global Amp Envelope Release Curve Shape	amp	instrument	ENV_RELEASE
Glide/Portamento Time	amp	instrument	GLIDE_TIME
Effect Enabled (all effects)	effect	instrument	ENABLED
Convolution Mix Level	effect	instrument	FX_MIX
Convolution IR File	effect	instrument	FX_IR_FILE
Filter Frequency (for several filters)	effect	instrument	FX_FILTER_F
Peak or Notch Filter Q	effect	instrument	FX_FILTER_Q
Peak or Notch Filter Gain	effect	instrument	FX_FILTER_G
Low-pass or High-pass Filter Resonance	effect	instrument	FX_FILTER_R
Reverb Wet Level	effect	instrument	FX_REVERB_W
Reverb Room Size	effect	instrument	FX_REVERB_R
Reverb Damping	effect	instrument	FX_REVERB_D
Chorus/Phaser/Convolution Mix Level	effect	instrument	FX_MIX
Chorus/Phaser Mod Depth	effect	instrument	FX_MOD_DEPT
Chorus/Phaser Mod Rate	effect	instrument	FX_MOD_RATE
Phaser Center Frequency	effect	instrument	FX_CENTER_F
Phaser/Delay Feedback	effect	instrument	FX_FEEDBACK
Delay Time	effect	instrument	FX_DELAY_TI
Delay Time Format	effect	instrument	FX_DELAY_TI
Delay Stereo Offset	effect	instrument	FX_STEREO_O
Delay Wet Level	effect	instrument	FX_WET_LEVE
Gain Level	effect	instrument	LEVEL
Wave Folder Drive Level	effect	instrument	FX_DRIVE
Wave Folder Threshold	effect	instrument	FX_THRESHOL
Wave Shaper Drive Level	effect	instrument	FX_DRIVE
Wave Shaper Output Level	effect	instrument	FX_OUTPUT_L
Wave Shaper Drive Boost	effect	instrument	FX_DRIVE_BO
Group Enabled / Disabled	amp	group	ENABLED
Group Volume	amp	group	AMP_VOLUME
Group Tuning	amp	group	GROUP_TUNIN
Pan	amp	group	PAN
Amplitude Velocity Tracking	amp	group	AMP_VEL_TRA
Group Amp Envelope Attack	amp	group	ENV_ATTACK
Group Amp Envelope Decay	amp	group	ENV_DECAY
Group Amp Envelope Sustain	amp	group	ENV_SUSTAIN
Group Amp Envelope Release	amp	group	ENV_RELEASE
Group Glide/Portamento Time	amp	group	GLIDE_TIME
Tag Enabled	amp	tag	TAG_ENABLED
Tag Volume	amp	tag	TAG_VOLUME
MIDI Note Mapping Enabled	note	midi	ENABLED
MIDI Note Binding Enabled	note_binding	midi	ENABLED
MIDI Note Binding Change `seqIndex`	note_binding	midi	SEQ_INDEX
MIDI Note Binding Change `seqLoopMode`	note_binding	midi	SEQ_LOOP_MO
MIDI Note Binding Change `seqTransposeWithRootNote`	note_binding	midi	SEQ_TRANSPO
MIDI Note Binding Change `seqPlaybackRate`	note_binding	midi	SEQ_PLAYBAC
MIDI Note Binding Change `seqTrackMidiInputVelocity`	note_binding	midi	SEQ_TRACK_M
MIDI Note Binding Change `seqTranspose`	note_binding	midi	SEQ_TRANSPO
MIDI Velocity Binding Enabled	velocity_binding	midi	ENABLED
UI Button State Binding Enabled	button_state_binding	ui	ENABLED

Description	type	level	parameter
UI Control Enabled	control	ui	ENABLED
UI Control Visibile	control	ui	VISIBLE
UI Control Value	control	ui	VALUE
UI Control Text	control	ui	TEXT
UI Control Minimum Value	control	ui	MIN_VALUE
UI Control Maximum Value	control	ui	MAX_VALUE
UI Control Value Type	control	ui	VALUE_TYPE
UI Image Control Path	control	ui	PATH
UI Keyboard Coloring Enable / Disable	keyboard_color	ui.	ENABLED
Modulator Amount (Depth)	modulator	instrument	MOD_AMOUNT
LFO Modulator Rate (or Frequency)	modulator	instrument	FREQUENCY
Envelope Modulator Attack	modulator	instrument	ENV_ATTACK
Envelope Modulator Attack Curve Shape	modulator	instrument	ENV_ATTACK_
Envelope Modulator Decay	modulator	instrument	ENV_DECAY
Envelope Modulator Decay Curve Shape	modulator	instrument	ENV_DECAY_C
Envelope Modulator Sustain	modulator	instrument	ENV_SUSTAIN
Envelope Modulator Release	modulator	instrument	ENV_RELEASE
Envelope Modulator Release Curve Shape	modulator	instrument	ENV_RELEASE
Sequence Rate	note_sequence	instrument	RATE
All Notes Off	general	instrument	ALL_NOTES_O

1. NOTE: The indexes of the parameter list also include UI controls that are not editable, such as `<label>` UI controls, so you'll want to account for that when calculating your positions.

   Here's a quick example:

   If your UI's `<tab>` section has the following elements under it: `<label>`, `<control>`,`<label>`,`<control>`. The `position` indexes of the four elements will be 0, 1, 2, 3. Therefore, the indexes of the two `<control>` elements will be 1 and 3, respectively.

2. NOTE: If your sample library manipulates `start`, `end`, `loopStart`, or `loopEnd` after a sample library's initial load, the sample playback engine must be in RAM/Memory mode (not disk streaming) or you will get very unpredictable results. In order to enforce this, sample creators should use the `playbackEngine` attribute.

1.10.3 Translation Modes

There are currently three binding translation modes: `linear`, `table`, `fixed_value`

Mode #1: `linear`

linear mode allows values that come in to be scaled up or down before they get passed along to the binding's target. If you set your translation mode to `linear` you should also `translationOutputMin` and `translationOutputMax`.

Example usage:

```
<binding level="ui" type="control" position="0" parameter="value" translation="linear"
              translationOutputMin="0" translationOutputMax="1"/>
```

Mode #2: `table`

table mode allows you to transform the binding's input in a more complex fashion before it gets passed along to the binding's target. If you set your translation mode to `table` you must define the `translationTable` parameter as well. This consists of a series of input-output pairs, separated by semi-colons.

Mode #3: `fixed_value`

fixed_value mode allows you to completely disregard the input of a binding and instead always use a supplied value. In order to use this translation mode, you must also specify a `translationValue`. This can be very useful when trying to have menu options enable and disable groups. An example usage:

```
<binding type="general" level="group" position="1" parameter="ENABLED" translation=
↪"fixed_value" translationValue="true" />
```

1.11 Appendix C: Boilerplate .dspreset File

```
<?xml version="1.0" encoding="UTF-8"?>
<DecentSampler minVersion="1.0.0">
  <ui width="812" height="375" layoutMode="relative"
      bgMode="top_left">
    <tab name="main">
      <labeled-knob x="445" y="75" width="90" textSize="16" textColor="AA000000"
                    trackForegroundColor="CC000000" trackBackgroundColor="66999999"
                    label="Attack" type="float" minValue="0.0" maxValue="4.0" value="0.01
↪" >
        <binding type="amp" level="instrument" position="0" parameter="ENV_ATTACK" />
      </labeled-knob>
      <labeled-knob x="515" y="75" width="90" textSize="16" textColor="AA000000"
                    trackForegroundColor="CC000000" trackBackgroundColor="66999999"
                    label="Release" type="float" minValue="0.0" maxValue="20.0" value="1
↪" >
        <binding type="amp" level="instrument" position="0" parameter="ENV_RELEASE" />
      </labeled-knob>
      <labeled-knob x="585" y="75" width="90" textSize="16" textColor="AA000000"
                    trackForegroundColor="CC000000" trackBackgroundColor="66999999"
                    label="Chorus" type="float" minValue="0.0" maxValue="1" value="0" >
        <binding type="effect" level="instrument" position="1" parameter="FX_MIX" />
      </labeled-knob>
      <labeled-knob x="655" y="75" width="90" textSize="16" textColor="FF000000"
                    trackForegroundColor="CC000000" trackBackgroundColor="66999999"
                    label="Tone" type="float" minValue="0" maxValue="1" value="1">
        <binding type="effect" level="instrument" position="0" parameter="FX_FILTER_
↪FREQUENCY"
                 translation="table"
                 translationTable="0,33;0.3,150;0.4,450;0.5,1100;0.7,4100;0.9,11000;1.
↪0001,22000"/>
      </labeled-knob>
      <labeled-knob x="725" y="75" width="90" textSize="16" textColor="AA000000"
                    trackForegroundColor="CC000000" trackBackgroundColor="66999999"
```

(continues on next page)

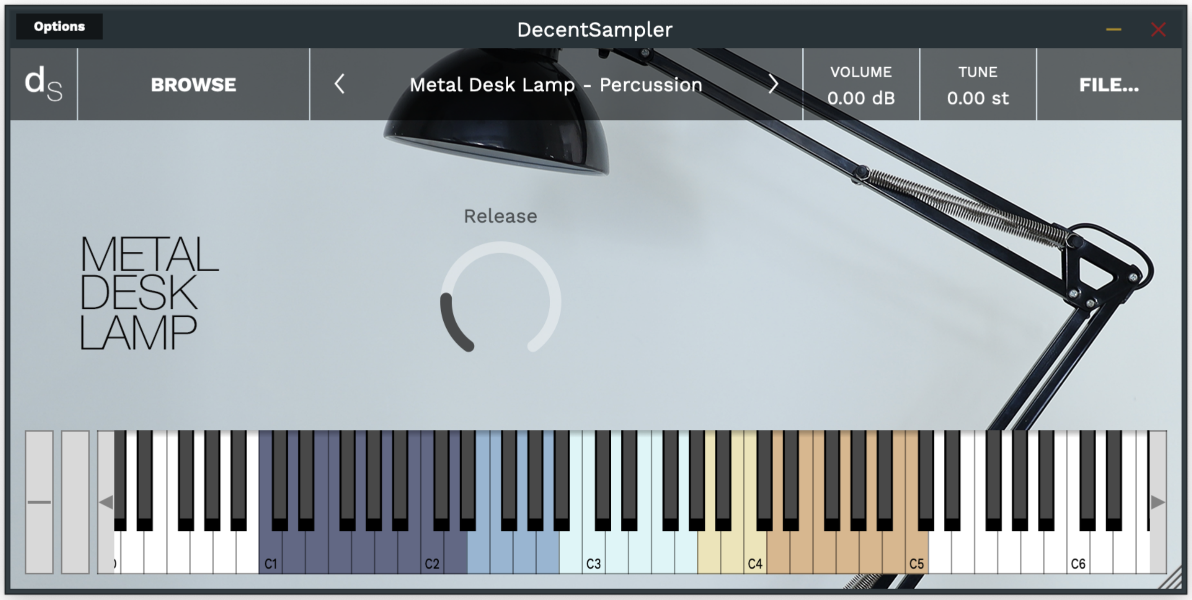
(continued from previous page)

```
                    label="Reverb" type="percent" minValue="0" maxValue="100"
                    textColor="FF000000" value="50">
        <binding type="effect" level="instrument" position="2"
                 parameter="FX_REVERB_WET_LEVEL" translation="linear"
                 translationOutputMin="0" translationOutputMax="1" />
      </labeled-knob>
    </tab>
  </ui>
  <groups attack="0.000" decay="25" sustain="1.0" release="0.430" volume="-3dB">
    <group>
      <sample loNote="21" hiNote="21" rootNote="21" path="DefaultPiano-21.aif"
              length="805888"/>
      <sample loNote="22" hiNote="33" rootNote="33" path="DefaultPiano-33.aif"
              length="807552"/>
      <sample loNote="34" hiNote="45" rootNote="45" path="DefaultPiano-45.aif"
              length="759168"/>
      <sample loNote="46" hiNote="57" rootNote="57" path="DefaultPiano-57.aif"
              length="756480"/>
      <sample loNote="58" hiNote="69" rootNote="69" path="DefaultPiano-69.aif"
              length="758656"/>
      <sample loNote="70" hiNote="77" rootNote="77" path="DefaultPiano-77.aif"
              length="595328"/>
      <sample loNote="78" hiNote="89" rootNote="89" path="DefaultPiano-89.aif"
              length="457600"/>
      <sample loNote="90" hiNote="96" rootNote="96" path="DefaultPiano-96.aif"
              length="469888"/>
      <sample loNote="94" hiNote="108" rootNote="108" path="DefaultPiano-108.aif"
              length="75264"/>
    </group>
  </groups>
  <effects>
    <effect type="lowpass" frequency="22000.0"/>
    <effect type="chorus"  mix="0.0" modDepth="0.2" modRate="0.2" />
    <effect type="reverb" wetLevel="0.5"/>
  </effects>
  <midi>
    <!-- This causes MIDI CC 1 to control the 4th knob (cutoff) -->
    <cc number="1">
      <binding level="ui" type="control" parameter="VALUE" position="3"
               translation="linear" translationOutputMin="0"
               translationOutputMax="1" />
    </cc>
  </midi>
</DecentSampler>
```

1.12 Useful Tutorials and Resources

1.12.1 UI Elements

How to Color the Keys of the On-Screen Keyboard



As of version 1.4.0, it is now possible to color the keys of the on-screen keyboard.

This can be useful for showing different ranges of notes that serve different purposes or highlighting notes used as keyswitches. In order to implement colored keys, make use of the new <`keyboard`> and <`color`> elements as follows:

```
<DecentSampler>
    <ui>
        <!-- Other stuffhere -->
        <keyboard>
            <color loNote="36" hiNote="50" color="FF2C365E" />
            <color loNote="51" hiNote="57" color="FF6D9DC5" />
            <color loNote="58" hiNote="67" color="FFCCF3F5" />
            <color loNote="68" hiNote="73" color="FFE8DA9B" />
            <color loNote="74" hiNote="84" color="FFD19D61" />
        </keyboard>
    </ui>
    <!-- Other stuff here -->
</DecentSampler>
```

By default, Decent Sampler highlights all of the notes that are mapped for a given sample. If you use the color keys feature, this default highlighting will be turned off and it will be up to you to color whatever keys you want.

Full documentation for the color element is *here*.

How to turn keyboard note coloring on and off using bindings

It also possible to programmically turn keyboard coloring on and off using bindings. Here's an example of how to do that:

```
<DecentSampler>
    <ui>
        <button name="Strum Enabled" x="464" y="82" width="14" height="14" style="image"␣
↪value="1" visible="false">
        <state name="Off" mainImage="Images/StrumsCheckboxUnchecked.png" hoverImage=
↪"Images/StrumsCheckboxUnchecked.png" clickImage="Images/StrumsCheckboxUnchecked.png">
            <binding level="ui" type="keyboard_color" colorIndex="0" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false"/>
            <binding level="ui" type="keyboard_color" colorIndex="1" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false"/>
            <binding level="ui" type="keyboard_color" colorIndex="2" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false"/>
            <binding level="ui" type="keyboard_color" colorIndex="3" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false"/>
            <binding level="ui" type="keyboard_color" colorIndex="4" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="false"/>
        </state>
        <state name="On" mainImage="Images/StrumsCheckboxChecked.png" hoverImage="Images/
↪StrumsCheckboxChecked.png" clickImage="Images/StrumsCheckboxChecked.png">
            <binding level="ui" type="keyboard_color" colorIndex="0" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true"/>
            <binding level="ui" type="keyboard_color" colorIndex="1" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true"/>
            <binding level="ui" type="keyboard_color" colorIndex="2" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true"/>
            <binding level="ui" type="keyboard_color" colorIndex="3" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true"/>
            <binding level="ui" type="keyboard_color" colorIndex="4" parameter="ENABLED"␣
↪translation="fixed_value" translationValue="true"/>
        </state>
      </button>
        <keyboard>
            <color loNote="36" hiNote="50" color="FF2C365E" />
            <color loNote="51" hiNote="57" color="FF6D9DC5" />
            <color loNote="58" hiNote="67" color="FFCCF3F5" />
            <color loNote="68" hiNote="73" color="FFE8DA9B" />
            <color loNote="74" hiNote="84" color="FFD19D61" />
        </keyboard>
    </ui>
    <!-- Other stuff here -->
</DecentSampler>
```

How to add Dropdown Menus

Dropdown Menus

In order to implemented dropdown menus, use the new `<menu>` and `<option>` elements. The `<menu>` element defines where the dropdown menu will show up in the ui, whereas the XML elements determine what menu options it has and what if anything those options actually do:

```
<menu x="10" y="40" width="120" height="30" value="2">
  <option name="Menu Option 1">
    <!-- Turn on this group -->
    <binding type="general" level="group" position="0" parameter="ENABLED"
    translation="fixed_value" translationValue="true" />
    <!-- Turn off this group -->
    <binding type="general" level="group" position="1" parameter="ENABLED"
    translation="fixed_value" translationValue="false" />
  </option>
  <option name="Menu Option 2">
    <!-- Turn off this group -->
    <binding type="general" level="group" position="0" parameter="ENABLED"
    translation="fixed_value" translationValue="false" />
    <!-- Turn on this group -->
    <binding type="general" level="group" position="1" parameter="ENABLED"
    translation="fixed_value" translationValue="true" />
  </option>
</menu>
```

In this example, a menu is being used to switch between two groups (the first menu option turns group 0 on and group 1 off; the section option turns group 0 off and group 1 on). Full documentation for the new `<menu>` and `<option>` elements is *here*.

The new `fixed_value` translation type

You'll note, in the example above, there's something new in the bindings: the four bindings elements have a `translation` parameter of type **fixed_value**. This is a new translation type. Up until now, binding translation has strictly been about taking an input parameter (such as a knob value or continuous controller amount) and translating it so that it is useful for some other purpose (it's our way of being able to do a little bit of math without having a full-blown scripting language). This new **fixed_value** binding is different. It ignores the input value completely and instead provides whatever is specified in the `translationValue` parameter. In this way, each menu option can have hardcoded values that it provides its bindings when it is selected.

1.12.2 Sample Mapping and Effects

How to Use Note Sequences within your Sample Libraries

As of version 1.11.1, **DecentSampler** now supports embedding note sequences with your sample libraries. This means that you can create patters or musical motifs that can be triggered either using MIDI or via the UI. This guide will show you how to create a sample library with note sequences and how to trigger them using MIDI.

Creating a sequence by hand

In this part of the guide, we will go over how to create a sequence by hand. This is useful if you want to create a sequence that is simple, and if you want the XML for your sequence to continue to be easily edited by hand. The first thing you need to do is add a `<noteSequences>` element to your `<DecentSampler>` file:

The `<noteSequences>` element

The `<noteSequences>` element is how you specify note sequences that can be used by this playback engine. There should be exactly one `<noteSequences>` element in each `<DecentSampler>` file.

The `<noteSequences>` element can contain one or more `<sequence>` elements:

The `<sequence>` element

The `<sequence>` element has the following attributes:

- `name` (optional): An optional descriptive name for the sequence. This is only used in the sample editor UI to help you identify the sequence.
- `length` (required): The length of the sequence in beats. This is a floating point number.
- `rate` (optional): The rate at which the sequence is played. This is a floating point number. The default is 1.0.

The `<sequence>` element can contain one or more `<note>` elements:

The `<note>` element

The `<note>` element has the following attributes:

- `position` (required): The position of the note in the sequence, in beats. This is a whole number.
- `velocity` (required): The velocity of the note. This is a floating point number between 0 and 1.
- `note` (required): The MIDI note number of the note.
- `length` (required): The length of the note in beats. This is a whole number.

An example sequence might looks like this:

```xml
<noteSequences>
    <sequence name="Maj1Slow" length="16" rate="1">
        <note position="0" velocity="1" note="48" length="16"/>
        <note position="1" velocity="1" note="52" length="15"/>
        <note position="2" velocity="1" note="55" length="14"/>
        <note position="3" velocity="1" note="60" length="13"/>
        <note position="4" velocity="1" note="64" length="12"/>
        <note position="5" velocity="1"  note="67" length="11"/>
        <note position="6" velocity="1"  note="72" length="10"/>
        <note position="7" velocity="1" note="76" length="9"/>
        <note position="8" velocity="1" note="79" length="8"/>
        <note position="9" velocity="1" note="84" length="7"/>
        <note position="10" velocity="1" note="88" length="6"/>
        <note position="11" velocity="1" note="91" length="5"/>
```

(continues on next page)

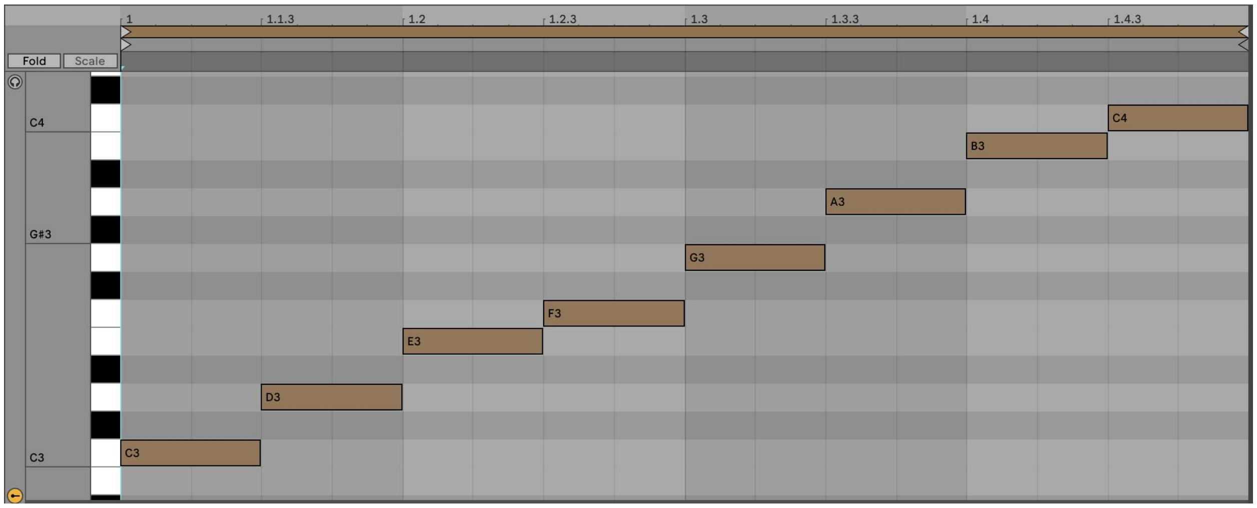
(continued from previous page)

```
    </sequence>
</noteSequences>
```

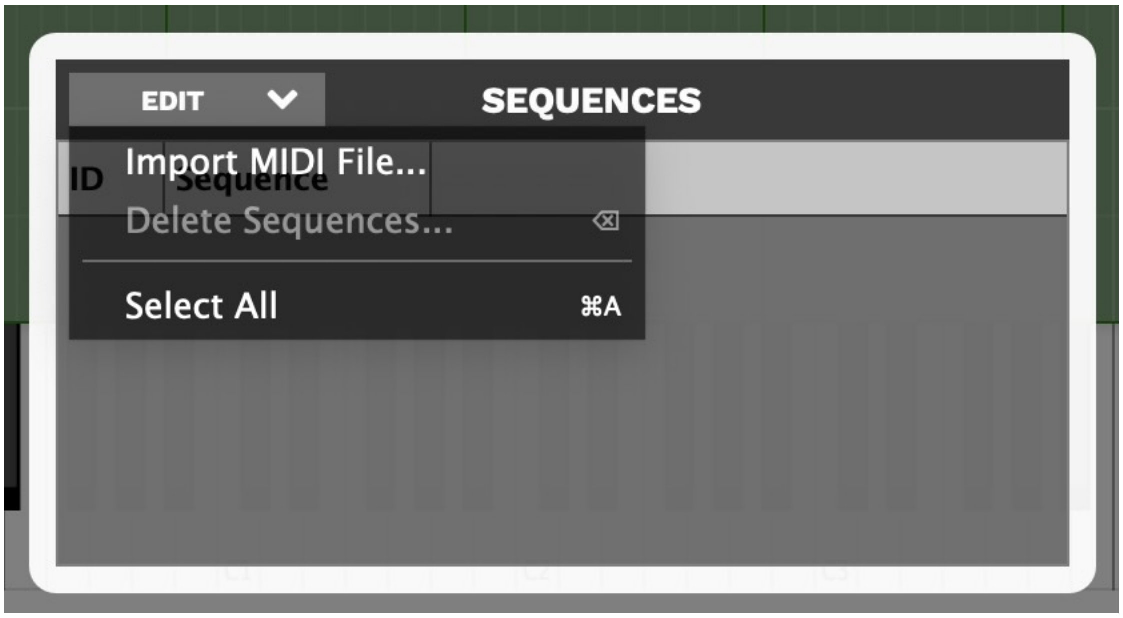
Creating a sequence using the sample editor

Another way to create sequences is via MIDI. While the Decent Sampler preset editor does not have a sequence editor built in, it does allow you to import MIDI files to create sequences. This means that you can use your favorite DAW to create sequences and then import them into Decent Sampler. This list of steps assumes that already have a preset created and that you are just looking to add sequences to it.

1. Create a MIDI sequence in your favorite DAW. Here is a sequence I created in Ableton Live:



2. Export the sequence as a MIDI file. The command for doing this will be different in every DAW. In Ableton Live, you can do this by right-clicking on the MIDI clip and selecting **Export MIDI Clip…**.
3. Load your preset in Decent Sampler
4. Open the preset editor by going to **File > Developer Tools > Preset Editor…**. You can also get there by hitting F12.
5. Once you're in the preset editor, choose the **File > Sequence Manager…** menu item. A little box will pop up. This is the sequence manager. It will be empty at first.
6. Click the **Edit > Import MIDI File** menu option, and select the MIDI file you created in step 2.



You will be presented with a dialog box that will ask you whether you want to quantize your sequence. Once you click OK, Decent Sampler will automatically convert the MIDI file into a sequence and add it to the sequence manager. You can then use the sequence in your preset.

7. Close the sequence manager window and save your preset.

Triggering sequences using MIDI

In order to set up your preset so that specific notes trigger specific sequences, you'll need to make use of the `<midi>` element in your `<DecentSampler>` file. Underneath the `<midi>` element, you can define any number of `<note>` elements. Each `<note>` element has the following attributes:

- `note` (required): The MIDI note number that will trigger the sequence. This can also be a range of notes, such as `24-35`.
- `enabled` (optional): A boolean value that determines whether the note handler is currently functional. The default is `true`.
- `swallowNotes` (optional): A boolean value that determines whether the note handler should swallow the incoming MIDI note or pass it on to the playback engine. The default is `false`, but most people will want to set this to `true`.

Within your `<note>` element, you can define any number of bindings. The bindings we will be using will look something like this:

```
<DecentSampler pluginVersion="1" minVersion="1.11.1">
  <midi>
    <!-- We define a listener for the notes 24 through 35... -->
    <note note="24-35" enabled="true" swallowNotes="true">
      <binding enabled="true" level="instrument" type="note_sequence" seqIndex="0"␣
↪seqLoopMode="forward" seqTriggerBehavior="midi_key" seqTransposeWithRootNote="12"␣
↪seqTrackMidiInputVelocity="1.0" seqPlaybackRate="1"/>
    </note>
  </midi>
  <noteSequences>
```

(continues on next page)

(continued from previous page)

```
    <sequence name="Maj1Slow" length="4" rate="1">
        <note position="0" velocity="1" note="48" length="1"/>
        <note position="1" velocity="1" note="52" length="1"/>
        <note position="2" velocity="1" note="55" length="1"/>
        <note position="3" velocity="1" note="60" length="1"/>
    </sequence>
  </noteSequences>
</DecentSampler>
```

Here is a description of all of the useful attributes for the `<binding>` element:

- `enabled` (optional): A boolean value that determines whether the binding is currently functional. The default is `true`.
- `level` (required): When triggering note sequences, this should always be `instrument`, as these all live at the `instrument` level.
- `type` (required): The type of binding. When triggering note sequences, this should always be `note_sequence`.
- `seqIndex` (required): The index of the sequence in the sequence manager that you want to trigger. This is a zero-based index.
- `seqLoopMode` (optional): The loop mode of the sequence. This can be `forward`, `reverse`, `random`, or `no_loop`.
- `seqTriggerBehavior` (optional): The trigger behavior of the sequence. this can be `midi_key`, `on`, or `off`. When you are triggering a sequence using a MIDI key, this should always be `midi_key`.
- `seqTransposeWithRootNote` (optional): The amount by which the sequence should be transposed when it is triggered, relative to the incoming note. For example, if you set this to 24, and the MIDI note that triggered it was 28, the sequence would be transposed up by 4 semitones, since 28 - 24 = 4.
- `seqTrackMidiInputVelocity` (optional): A floating point value between 0 and 1 that determines whether the sequence should track the velocity of the incoming MIDI note. The default is `1.0`.
- `seqPlaybackRate` (optional): A floating point value that determines the rate at which the sequence should be played. The default is `1.0`.

You can download the example file here and try it out for yourself. You will want to look in the `example-006-working-with-note-sequences/Example 6A - Triggering Sequences Using MIDI.dspreset` file.

Triggering note sequences using the UI

In order to trigger note sequences using the UI, you will need to make use of the `<ui>`, `<tab>`, and various UI controls elements in your `<DecentSampler>` file. This tutorial assumes you already have something like this set up. If you don't, you can refer to the *<ui> element section* for more information.

```
<DecentSampler pluginVersion="1" minVersion="1.11.1">
  <ui>
    <tab>
      <!-- This button has two states -->
      <button x="200" y="17" width="100" height="30" style="text" parameterName=
↪"Sequencer Enabled" value="0" defaultValue="0">
        <!-- In the Off state, we silence any sequence that might be playing that has␣
↪the identifier "sequence_button_1"  -->
```

(continues on next page)

(continued from previous page)

```
        <state name="Off">
          <binding level="instrument" type="note_sequence" seqIndex="0"␣
↪seqTriggerBehavior="off" seqPlayerIdentifier="sequence_button_1"/>
        </state>
        <!-- In the `On` state, we tell the engine to play the sequence "Maj1Slow"␣
↪(identifier as sequence with index 0).
        We will specify that it should be tracked internally using the identifier
↪"sequence_button_1". That fact that we are specify this idenfitier
        now will allow us to turn the sequence off later. -->
        <state name="On">
          <binding level="instrument" type="note_sequence" seqIndex="0" seqLoopMode=
↪"forward" seqTriggerBehavior="on" seqTrackMidiInputVelocity="1.0" seqPlaybackRate="1"␣
↪seqTranspose="12" seqPlayerIdentifier="sequence_button_1"/>
        </state>
      </button>
    </tab>
  </ui>
  <noteSequences>
    <sequence name="Maj1Slow" length="4" rate="1">
        <note position="0" velocity="1" note="48" length="1"/>
        <note position="1" velocity="1" note="52" length="1"/>
        <note position="2" velocity="1" note="55" length="1"/>
        <note position="3" velocity="1" note="60" length="1"/>
    </sequence>
  </noteSequences>
</DecentSampler>
```

In this example, we have a button that has two states: `Off` and `On`. When the button is in the `Off` state, we silence any sequence that might be playing that has the identifier `sequence_button_1`. When the button is in the `On` state, we tell the engine to play the sequence `Maj1Slow`. We will specify that it should be tracked internally using the identifier `sequence_button_1`. That fact that we are specifying this identifier now will allow us to turn the sequence off later.

You can download the example file here and try it out for yourself. You will want to look in the `example-006-working-with-note-sequences/Example 6B - Triggering Sequences Using UI.dspreset` file.

Conclusion

In this guide, we went over how to create note sequences by hand and how to import them from MIDI files. We also went over how to trigger sequences using MIDI and the UI. We hope this guide has been helpful to you.

How to control parameters using tags (Example: Mic-level Knobs)

As of version 1.0.2, the best way to implement mic-level knobs is using the new sample tagging feature. It is possible to assign tags to specific samples. In this way, you can specify which type of sound they are:

```
<sample volume="0.0dB" tags="note,mic1" />
<sample volume="0.0dB" tags="rt,mic1" />
<sample volume="0.0dB" tags="note,mic2" />
<sample volume="0.0dB" tags="rt,mic2" />
```

You can also assign tags at the group level. You can also mix and match, and the tags specified at the group level will be added to the list of tags already specified at the sample level:

```
<group tags="note">
  <sample volume="0.0dB" tags="mic1" />
  <sample volume="0.0dB" tags="mic2" />
</group>
<group tags="rt">
  <sample volume="0.0dB" tags="mic1" />
  <sample volume="0.0dB" tags="mic2" />
</group>
```

Then you can make controls with bindings that reference those tags:

```
<control x="246" y="115" parameterName="MIC 1" style="linear_bar_vertical" type="float"␣
↪minValue="0" maxValue="100" value="60" width="20" height="70" trackForegroundColor=
↪"FFFFFFFF" trackBackgroundColor="FF888888">
    <binding type="amp" level="tag" identifier="mic1" parameter="AMP_VOLUME" />
</control>

<control x="346" y="115" parameterName="MIC 2" style="linear_bar_vertical" type="float"␣
↪minValue="0" maxValue="100" value="60" width="20" height="70" trackForegroundColor=
↪"FFFFFFFF" trackBackgroundColor="FF888888">
    <binding type="amp" level="tag" identifier="mic2" parameter="AMP_VOLUME" />
</control>
```

How to do voice-muting for drums

Voice muting makes use of the tags functionality, these are text labels that you can use to identify samples or groups of samples. You start by adding tags to all of your groups (you can also add them to individual samples if you'd like). Next, you add a `silencedByTags` attribute to groups or sample elements that you want to be silenced by other samples. When a sample with a tag matching one of the tags in the `silencedByTags` is played, it will silence the current sample (or group).

Here's an example:

```
<DecentSampler>
    <groups>
        <group tags="hihat" silencedByTags="hihat" silencingMode="fast">
            <!-- Your samples go here. -->
        </group>
    </groups>
</DecentSampler>
```

Note the use of the `silencingMode` attribute as well (a value of "fast" means we immediately silence, whereas "normal" means we trigger the ADSR release phase).

How to implement true legato

Let's walk through how to build a basic true legato instrument. Legato instruments generally consist of either two or three groups:

1. First, there's the initial looping sustain sample that will get played when the note is first pressed down.
2. Then there is the legato transition sample that will get played when a second note gets pressed
3. (optional) Depending on the implementation, there may be a third looping sustain group that gets played after the legato transition sample plays. In such cases, this third group usually contains the same samples as were used in group 1.

Step 1: Voice muting

Before we get into legato, let's talk about voice muting. This is the behavior wherein one set of samples causes another set of samples to stop playing. This can be desirable in situations such as legato instruments where two samples should not be sounding at the same time.

In Decent Sampler, voice-muting makes use of tags. These are text labels that you can use to identify samples or groups of samples. You can add a `silencedByTags` attribute to groups or sample elements. This consists of a comma-separated list of tags that specify which samples should silence the current samples. When a sample with a tag matching one of the tags in the `silencedByTags` is played, it will silence the current sample (or group). Here's an example:

```
<DecentSampler>
    <groups>
        <group tags="sustain" silencedByTags="legato" silencingMode="normal" release="1.0
↪">
            <!-- Your sustain samples go here. -->
        </group>
        <group tags="legato" silencedByTags="legato" silencingMode="normal" release="1.0
↪">
            <!-- Your legato samples go here. -->
        </group>
    </groups>
</DecentSampler>
```

In the above scenario, if a legato sample is matched, any sample that might be playing from the "sustain" group will be stopped.

It's also worth mention the `silencingMode` attribute as well (a value of `fast` means we immediately silence that sample, whereas `normal` means we trigger the ADSR release phase).

Step 2: Legato

In order to specify which samples should be triggered first, we use the `trigger` attribute. This can that can be added to the <`group`> or individual <`sample`> tags. The default value is `attack`, but there are two useful new values:

- **`first`**: This value means that this sample will only trigger if no other notes are playing
- **`legato`**: This value means that this sample will only trigger if other notes are already playing

Here is an example of how this might look in cunjunction with our example from above:

```
<DecentSampler>
    <groups>
        <group trigger="first" tags="sustain" silencedByTags="legato" silencingMode=
↪"normal" release="1.0">
            <!-- Your sustain samples go here. -->
```

(continues on next page)

(continued from previous page)

```
        </group>
        <group trigger="legato" tags="legato" silencedByTags="legato" silencingMode=
↪"normal" release="1.0">
            <!-- Your legato samples go here. -->
        </group>
    </groups>
</DecentSampler>
```

Step 3: Specifying previous notes

When creating a legato instrument, it is often essential to limit which legato transition gets played based on which note we are transition from. This can be achieved using either the `previousNote` or the `legatoInterval` attributes. The `previousNote` attribute causes the engine to only play this sample if the previously triggered note matches the specified note. Example usage:

```
<sample path="Samples/LV_Legato_F2_G2.wav" rootNote="G2" loNote="G2" hiNote="G2"␣
↪previousNote="F2" start="43000" />
<sample path="Samples/LV_Legato_G2_A2.wav" rootNote="A2" loNote="A2" hiNote="A2"␣
↪previousNote="G2" start="43000" />
```

The `legatoInterval` attribute causes the engine to only play the sample if distance between the current note and the previously triggered note is exactly this semitone distance. For example, if the note for which this sample is being triggered is a C3 and the `legatoInterval` is set to `-2`, then the sample will only play if the previous note was a D3 because D3 minus two semitones equals C3.

Step 4: Polyphony

In legato instruments, it is sometimes useful to limit polyphony for a specific sample or set of samples. This is achieve using tags. At the top-level of your file, you can specify a element as follows:

```
<DecentSampler>
    <groups>
        <group tags="some-tag" >
            <!-- ... -->
        </group>
    </groups>
    <tags>
        <tag name="some-tag" polyphony="1" />
    </tags>
</DecentSampler>
```

Putting it all together

Here is a full example of a legato instrument:

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

<DecentSampler pluginVersion="1">
  <groups>
    <group tags="sustain" silencedByTags="legato" trigger="first"
           silencingMode="normal" release="0.3">
      <sample path="Samples/LV_Sustain_D3.wav" rootNote="D3" loNote="C3" hiNote="D3"/>
      <sample path="Samples/LV_Sustain_E3.wav" rootNote="E3" loNote="D#3" hiNote="E3"/>
      <!-- More samples go here -->
    </group>
```

(continues on next page)

(continued from previous page)

```xml
    <group tags="legato" silencedByTags="legato" trigger="legato"
           silencingMode="normal" attack="0.1" decay="0.2" sustain="0" release="1">
              previousNote="C#3"/>
      <sample path="Samples/LV_Legato_D3_E3.wav" rootNote="E3" loNote="E3"
              hiNote="E3" start="43000" previousNote="D3"/>
      <sample path="Samples/LV_Legato_E3_F3.wav" rootNote="F3" loNote="F3"
              hiNote="F3" start="43000" previousNote="E3"/>
      <!-- More samples go here -->
    </group>
    <group tags="legato-tails" silencedByTags="legato-tails" trigger="legato"
           attack="0.3" silencingMode="normal" release="0.3" start="5000">
      <sample path="Samples/LV_Sustain_D3.wav" rootNote="D3" loNote="C3" hiNote="D3"/>
      <sample path="Samples/LV_Sustain_E3.wav" rootNote="E3" loNote="D#3" hiNote="E3"/>
      <!-- More samples go here -->
    </group>
  </groups>
</DecentSampler>
```

How to add Keyswitches

As of DecentSampler 1.4.0, it is now possible to implement keyswitches. For a long time, it's been possible to trigger events when a MIDI continuous controller event is received: for example, using MIDI CCs we can change knob values or group volumes. Well, it is also possible to trigger events using MIDI notes as well. Here's what the setup for a MIDI note-based event mapping would look like:

```xml
<midi>
  <note note="11">
    <binding type="general" level="group" position="0" parameter="ENABLED"
             translation="fixed_value" translationValue="true" />
    <binding type="general" level="group" position="1" parameter="ENABLED"
             translation="fixed_value" translationValue="false" />
  </note>
  <note note="12">
    <binding type="general" level="group" position="0" parameter="ENABLED"
             translation="fixed_value" translationValue="false" />
    <binding type="general" level="group" position="1" parameter="ENABLED"
             translation="fixed_value" translationValue="true" />
  </note>
</midi>
```

In this example, MIDI note 11 turns on group 0 and turns off group 1, whereas MIDI note 12 does the opposite. Note the use of the `fixed_value` translation type.

More documentation on this can be found *here*.