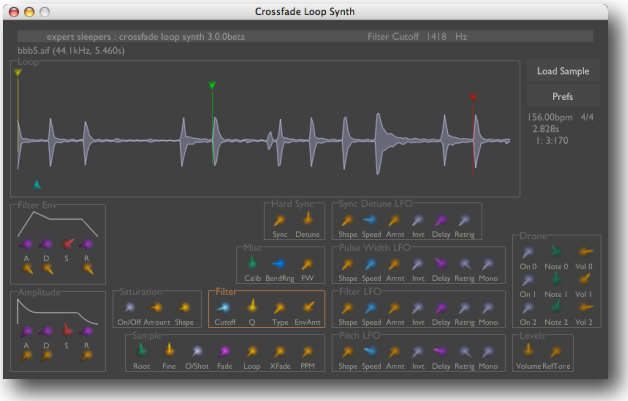
Expert Sleepers
Crossfade Loop Synth
v3.2.0
User Manual

Contents

Introduction

Crossfade Loop Synth



Crossfade Loop Synth is at heart a simple sample playback synth plug-in.

It plays samples in the classic sampler manner - by simply altering the playback speed according to the desired note. No fancy time-stretching or pitch shifting is applied.

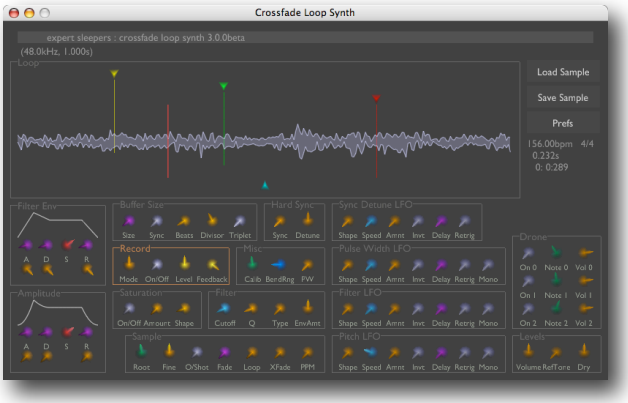
You can control how the sample is looped as it is played. An adjustable amount of crossfade can be applied to the loop endpoints to smooth the transitions. This makes it extremely easy to produce glitch-free pad sounds.

The loops can also be played backwards and alternately forwards and backwards (pendulum-style).

Further sound processing can be applied in the form of overdrive/saturation, filtering, envelopes (for amplitude and filter), pulse-width modulation, oscillator sync and LFOs (one each for filter, pitch, PWM and sync).

The synth is 32 note polyphonic, and all parameters can be controlled by MIDI controllers.

Crossfade Loop Synth Effect



Crossfade Loop Synth Effect is the same as Crossfade Loop Synth except you can record live audio to use as the sample, rather than just loading it from disk.

You can record continuously, even while playing back. This allows you to use the plug-in as an extremely unusual and creative delay effect. For example you can have the delays played back at double the pitch/speed you played them. Or at double the pitch and backwards. Or as a chord. Or repeat the first beat of every bar... the possibilities are legion.

You can save the recorded sample to disk as an AIFF (Mac OS X) or WAV (Windows) file for later use.

Installation

Mac OS X, Audio Unit (AU)

One file (`ExSlXFadeLooper.component`) contains all variants of the plug-in.

Simply copy the file to the folder:

`Library/Audio/Plug-Ins/Components`

Mac OS X, VST

There is one file per plug-in configuration, named `ExSlXFadeLooper.vst` and `ExSlXFadeLooperEffect.vst`.

Simply copy the files to the folder:

`Library/Audio/Plug-Ins/VST`

Windows (VST)

There is one file per plug-in configuration, named `xfadelooper.dll` and `xfadelooper_fx.dll`.

Simply copy the files to your VST plug-ins folder.

System Requirements

Mac OS X

Crossfade Loop Synth v3.2.0 and above require at least Mac OS X version 10.5.8 and are usable on Intel Macs only. Older versions of Crossfade Loop Synth also work on PowerPC Macs and OS X versions back to 10.2.8.

The Audio Unit version will work in any Audio Unit host.

The VST version requires a “VST 2.4” compatible host.[1]

Windows

Crossfade Loop Synth is supported on Windows XP SP2 and Windows 7. It may work with other versions of Windows but this is by no means guaranteed.

The plug-in requires a “VST 2.4” compatible host.

---

[1] VST is a trademark of Steinberg Media Technologies GmbH.

Registration

The downloadable version of Crossfade Loop Synth stops working after 15 minutes every time you use it. To stop this happening, you need to buy a registration.

You can buy a registration key online using a credit card or PayPal from the Expert Sleepers Licence Manager application. See here for more information. Note that you need at least version 1.0.4 (Mac OS X) or 1.0.11 (Windows) of the Licence Manager.

The e-commerce side of things is handled by eSellerate. If you have any security concerns, have a look at their website which is pretty informative.

Your registration key allows you to install Crossfade Loop Synth on up to 3 different computers (useful if for example you have a desktop computer in the studio and a laptop for live use).

You need an internet connection to activate the software, though not necessarily on the computer on which you want to use it.

Using Crossfade Loop Synth

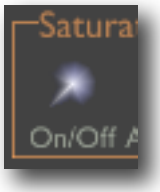
Using the controls

Knobs

Basic use of the knobs is to click on them and drag the mouse up and down. However you can obtain different results by holding keys as follows:

- Shift : Values change more slowly as you move the mouse.
- Command[1] (Mac OS X)/Alt (Windows) : The knob assumes its default position.

The exception to the above are knobs which are actually 'on/off' buttons. These are coloured light grey in the interface. Simply clicking on such a knob toggles the state between on and off.



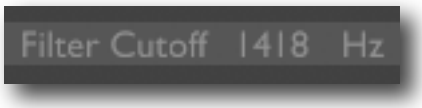
Sliders

Sliders behave similarly.

- Shift : Values change more slowly as you move the mouse.
- Command : The slider assumes its default position.

Name/value display

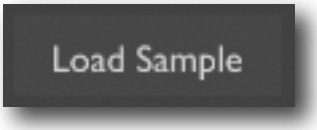
As you move the mouse around the interface, the name and current value of the control currently under the mouse is displayed in the top right of the window. This area also provides tooltips for buttons.



Loading and manipulating samples

Loading samples

The 'Load Sample' button lets you choose an audio file on disk to load as the sample to play. Files should be mono or stereo, 16 or 24 bit, uncompressed audio.



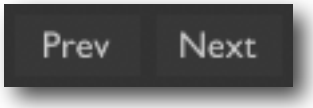
Mac OS X: Most popular formats (e.g. AIFF, WAV, SD2) should work.
Windows: Only WAV format is supported.

---

[1] The 'Command' key is also known as the 'Apple' key - the one next to the spacebar.

Stepping through a folder of samples

The two buttons below 'Load Sample', labelled 'Prev' and 'Next', let you quickly audition a number of samples, without having to repeatedly navigate through a file open dialogue.



Starting at the currently loaded sample, the 'Next' button loads the sample in the same folder that follows alphabetically. Similarly, the 'Prev' button loads the sample that precedes the current one.

Controlling sample playback

The selection of knobs labelled 'Sample' control some basic aspects of sample playback.

The 'Root Note' control lets you set the MIDI note number at which the sample will play back at its original pitch (this is essentially the same as a coarse tune control). A 'Fine Tune' control is also provided. A reference tone can be used to help tune the sample correctly - see below.

The 'No Loop On/Off' control lets you disable looping i.e. the sample stops playback once it hits the loop end position. The 'No Loop Fade' control lets you set a short fade-out for this, rather than an abrupt stop. NB the sample will continue to loop during this fade-out, so the settings of the loop controls will have an effect on the sound.

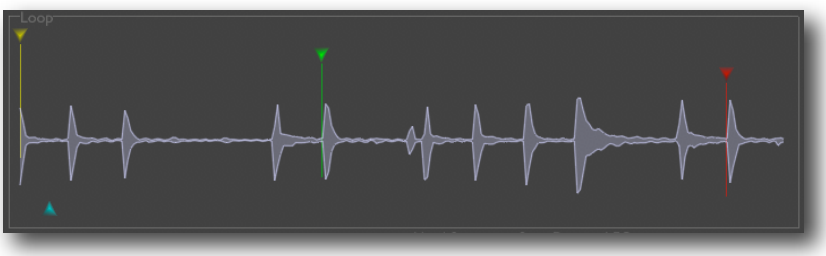
The 'Loop Mode' control chooses between the 3 basic loop modes: Forwards (loop plays from beginning to end), Reverse (loop plays from end to beginning) and Alternate (loop switches between Forward and Reverse whenever it hits a loop endpoint).

'Crossfade Type' selects between 'Equal Gain' and 'Equal Power' crossfade curves. The former is more appropriate when the ends of the loop are similar material, and phase-coherent (for example, if you're looping the sustain section of a sampled instrument). The 'Equal Power' mode is useful when looping less coherent material, and can help to prevent the apparent drop in volume during the crossfade that you can get with the 'Equal Gain' curve.

The final control in this section is Play Position Memory - see below.

Setting the loop points and crossfade

The remaining loop controls are the 4 coloured sliders that appear in the sample preview window.



'Start Offset' (yellow) sets the point at which playback will

start when the note is first triggered.

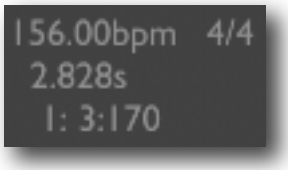
'Loop Start' (green) sets the beginning of the looped section.

'Loop End' (red) sets the beginning of the looped section.

NB when the Loop Mode is Reverse, Loop End is effectively the loop start and Loop Start is effectively the loop end. Also in this case the Start Offset should be set after Loop Start.

'Crossfade' (blue) sets the amount of crossfade or overlap applied at the beginning and end of the loop. Higher values will tend to smooth the loop transitions.

Loop calibration



Various information is displayed to the right of the sample waveform window to help in setting up loop points.

The first line gives the song's current tempo and time signature, as set by the host application.

The second line shows the current loop time, in seconds. NB this is not necessarily the time between the start and end markers - when crossfading is taken into account, the actual time after which the sample repeats is different.

The third line also shows the loop time, but in musical notation (bars, beats and 480ths of a beats).

To display the loop time, we have to assume a certain pitch for sample playback. This pitch is set with the 'Calibration Note' control in the 'Misc' group.

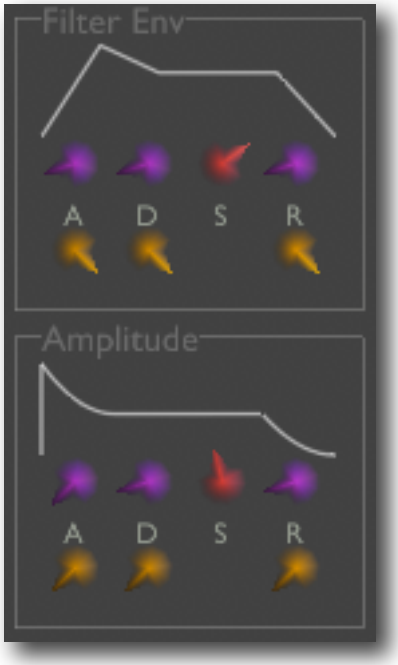
Reference Tone

The 'RefTone' knob fades up a pure tone at the reference pitch for the note being played. This can be useful when tuning the sample, but can also be used creatively to introduce another element into the synth's sound.

The sample needs to be tuned against this reference tone for the pulse width control to work properly (see below).

Envelopes

Controls



The amplitude and filter envelopes are both of the standard ADSR type, but with extra control over the shape of the curves.

The upper 4 knobs control the attack time, decay time, sustain level and release time.

The lower 3 knobs control the shape of the attack, decay and release curves.

Infinite Release

The filter envelope release setting has a special value (at full clockwise rotation of the control) labelled 'Inf'. When this setting is made, the envelope stops at the sustain level, and does not decay to zero when the note is released.

Saturation

The saturation section lets you apply subtle or extreme distortion/overdrive effects to the sample (before being fed to the filter section).

The 'Saturation Amount' control applies boost to the signal as it passes through the saturation effect, thus increasing its tendency to saturate.

The 'Saturation Shape' control affects the nature of the waveshaping applied to the signal. At a value of 0.0, the effect is simple hard clipping. At 1.0, the effect is soft clipping. Beyond 1.0 the effect is increasingly more extreme waveshaping.

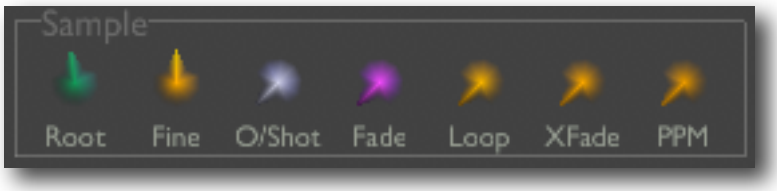
Filter

The filter 'type' control lets you smoothly choose between no filtering, lowpass filtering, bandpass filtering and highpass filtering.

The filter cutoff and resonance ('Q') controls function exactly as you would expect.

The envelope amount control sets how much the filter envelope modulates the filter cutoff.

Play Position Memory



The last control in the 'Sample' group, PPM provides variations on the idea that the sample should start playing from the position at which it last stopped playing. For example, if you had a sample of someone counting from one to ten, it might be useful to play a note which played 'one, two, three...', release the note, and then play the same note again for 'four, five, six...'.

The options are:

- Off.
- Global - playback restarts at the last position played, regardless of the notes involved.
- Per Note - each MIDI note (i.e. pitch) has its separate memory of the position it last stopped playing.
- Same Note - playback position is remembered if the same note is played again, otherwise playback starts from the normal sample start point.
- Different Note - playback position is remembered if different notes follow each other; if the same note is played twice, the playback position goes back to the start.

Pulse Width

The Pulse Width control lets you apply an effect similar to the pulse width modulation familiar from classic analogue synths. If the sample has been correctly tuned to the reference tone, adjusting the PW control will alter the tonal character of the sound in a manner similar to classic PWM.

Pulse width has its own LFO - see below.

Hard Sync

'Hard Sync' borrows another concept from classic analogue synths - that of oscillator sync.

In traditional synthesis, the frequency of one oscillator is forced onto the waveform of another. In Crossfade Loop Synth, this is done by adjusting the loop length so that the loop repeats at the desired audio frequency. The result is more like synthesis than sample playback - but synthesis with a waveform from the sample buffer. In effect, a kind of wavetable synthesis. By moving the loop point through your sample, you can get some nice tonal variation into your sound.

The ‘Sync’ control turns the effect on. It sets the length of the sample that is used for synthesis, in terms of a number of cycles at the reference frequency. Setting this to 1 will produce a most recognisable single tone - higher settings will produce more complex and possibly atonal waveforms.

When Hard Sync is turned on, a small section of the sample at the loop start point is all that will be used. However, at the start of a note the sample will still play from the start offset until it reaches the loop start. Therefore, if you just want the tone of the sync’d sound, set the start offset after the loop point so playback begins immediately at the loop. Or, use this creatively, to play an interesting section of the sample before hitting the loop - resulting in something similar to the old ‘S+S’ keyboards of the 80s.

Note that the crossfade control still has an effect. High crossfade values will tend to ‘soften’ the sound you get from Hard Sync.

The ‘Detune’ control lets you modulate the pitch of the sample playback, while keeping the sync’d pitch constant. This gives the characteristic ‘tearing’ sound.

The detune amount has its own LFO - see below.

LFOs

There are four LFOs which are independent and are each assigned a fixed function. They all have the same controls:

- ‘Shape’ - selects one of four LFO waveforms: sine, triangle, saw and square.
- ‘Speed’ controls the frequency of the LFO.
- ‘Amount’ sets the amount by which the LFO modulates its affected value.
- ‘Invert’ flips the LFO waveform - e.g. the saw waveform switches from a rising ramp to a falling ramp.
- ‘Delay’ sets the period over which the delay’s effect fades up after a note is pressed. This can give e.g. a nice vibrato effect that fades up during a note, as a violin is normally played.
- ‘Retrigger’ - if on, the LFO waveform resets when a note is pressed. If off, the LFO waveform runs continuously. NB retriggering is also affected by the LFO’s ‘Mono’ control.
- ‘Mono’ - if on, all notes playing use the same LFO. If off, each note has its own LFO.

Pitch LFO

Modulates the note pitch by up to 100 cents (one semitone).

Filter LFO

Modulates the filter cutoff frequency by up to 3 octaves.

Pulse Width LFO

Modulates the pulse width control (see above).

Sync Detune LFO

Modulates the hard sync detune frequency (see above) by up to an octave.

Drones

You can turn on up to three notes to play continuously (like the drones on the bagpipes). This is mainly useful for the effect plug-in (below) but has its uses for the synth version too.

Usage is very simple - for each of the 3 drones you have an on/off switch, a note control and volume control.

Pitch Wheel Range

Found in the 'Misc' section - simply sets the range of the MIDI pitch bend control, from 0 to 12 semitones.

Crossfade Loop Synth Effect

Compared to Crossfade Loop Synth

The effect version of Crossfade Loop Synth can do everything the synth version can - it simply adds more features.

The primary difference is that you can record audio into the sample buffer. You can still load samples from disk too.

You can play notes from the effect over MIDI, assuming your host application supports this (not all hosts allow passing notes to effects, only to synths).

Though they also feature in the synth version, the drones (see above) really come into their own in the effect version. For a simple delay effect, turn on one of the drones tuned to the root note, and enable recording with feedback (see below).

Recording

Record controls

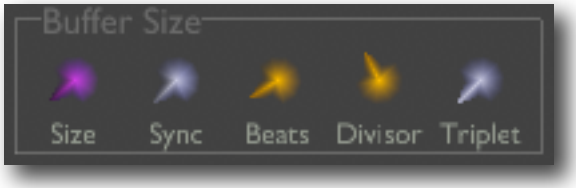
'Mode' chooses the recording mode:

- Continuous - recording is always on.
- Gated - recording is turned on and off with the 'On/Off' control.
- Gated With Reset - as Gated, but the record position is reset to the start of the buffer when recording starts.
- Gated By Notes - recording is active whenever any notes are playing (including drones).
- Gated By Notes With Reset - as Gated By Notes, but the record position is reset to the start of the buffer when recording starts.
- Triggered - recording starts when the 'On/Off' control is turned on, and stops at the end of the buffer. (This is the mode you'd use to treat Crossfade Loop Synth as a traditional sampler.)

'Level' sets the recording level. Raise this if the level going into the plug-in is too low.

'Feedback' controls the amount of feedback applied. At 0%, new recording completely replaces the audio already in the buffer. At 100%, new recording is layered on top of the old audio. At other settings, the old audio is reduced in volume on each pass.

Buffer size controls



These controls together set the size of the recording buffer.

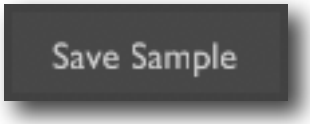
The first control 'Size' simply sets the buffer size in seconds. The range of this control is a preferences setting - see below.

The 'Sync' control has four settings - 'Off', 'Off (Adaptive)', 'On' and 'On (Adaptive)'.

In the 'Off' modes, the buffer size is set via the 'Size' control. In the 'On' modes, the remaining controls set the buffer size in terms of the host's tempo. 'Divisor' chooses a beat length to use as the basis for the size (e.g. a setting of 1/4 chooses quarter notes) and the 'Beats' control sets how many of those beats to use for the buffer length. 'Triplet' modifies the chosen beat length to its triplet version.

The '(Adaptive)' modes take into account the loop start, end and crossfade settings, so that the net length of the loop, rather than the actual buffer size, matches the size chosen with the size or beats controls.

Saving the sample to disk



The 'Save Sample' button lets you save the audio currently in the buffer to disk.

Currently there are no options. The saved file is always a 24 bit stereo AIFF (Mac OS X) or WAV (Windows) file. The audio is normalised during save.

Using the Effect as a Delay

Crossfade Loop Synth Effect can create some interesting variations on delay effects.

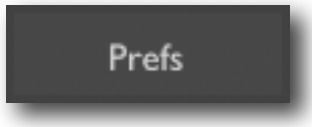
To get a basic delay, try the following settings:

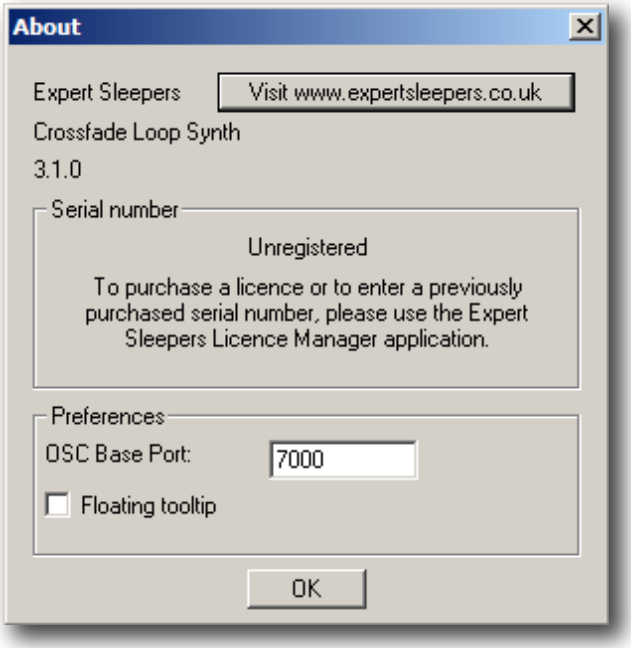
- Choose a buffer size that is rhythmically related to the tempo of your song (e.g. by using the 'Sync' settings above).
- Set the loop start and end points to the start and end of the buffer, and the crossfade to zero.
- Set the record mode to Gated By Notes With Reset.
- Activate the middle drone (which by default plays at MIDI note 60, the same as the default sample root note).

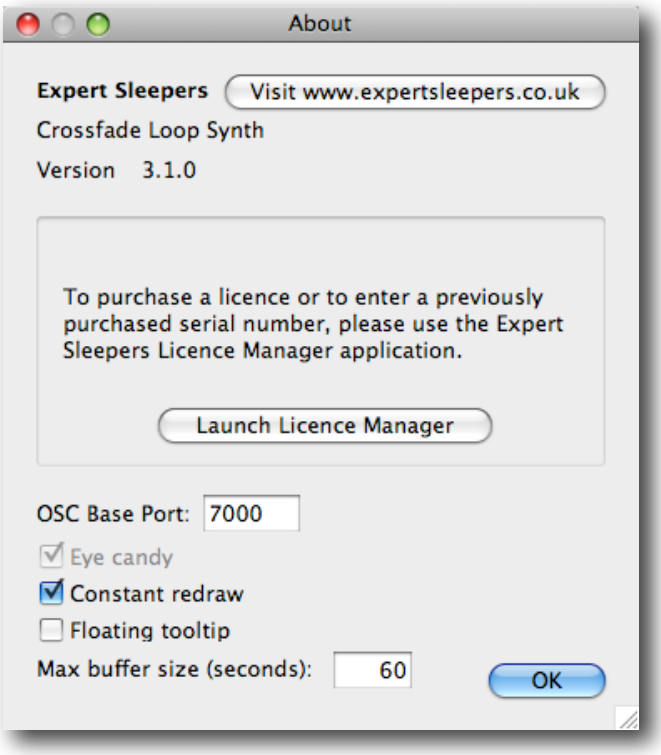
From there, try varying the loop mode, the loop start and end points etc. Try adding the other two drones to get audio an octave below and above the pitch you recorded.

Preferences

Pressing the 'Prefs' button brings up a dialog where various preferences are set. These settings are shared by all instances of Crossfade Loop Synth, and are not stored with presets.







The top section shows the product version.

The central section will show your serial number once you've bought a registration.

OSC Base Port

Sets the base port number for OSC. See the section on OSC, below.

Constant redraw

Is on by default. If turned off, the GUI is only redrawn when a control changes. Use this if you're concerned that the GUI is wasting your CPU resources. Note that the display of playback and record positions is useless if constant redraw is disabled.

Floating tooltip

Causes the parameter name and value display (usually in the top right of the GUI) to be displayed above the mouse pointer. Use this if you find it inconvenient to look away from the mouse to see the parameter value you're altering.

Max buffer size

This setting only affects the Effect version of the plug-in. It sets the range of the buffer size control. Setting this to higher values will consume a larger amount of your computer's memory.

MIDI control

All of Crossfade Loop Synth's parameters can be controlled via MIDI CC's (Continuous Controllers) according to the table below.

CC 64 (sustain pedal) is also supported.

2 Root Note
3 Fine Tune
4 Start Offset
5 Loop Start
7 Volume
8 Loop End
9 Crossfade
11 Attack
12 Decay
13 Release
14 Attack Shape
15 Decay Shape
16 Release Shape
17 Sustain
18 Filter Cutoff
19 Filter Q
20 Filter Type
21 Filter Env Amount
22 Filter Env Attack
23 Filter Env Decay
24 Filter Env Release
25 Filter Env Attack Shape
26 Filter Env Decay Shape
27 Filter Env Release Shape
28 Filter Env Sustain
29 No Loop Fade
30 No Loop On/Off
31 Loop Mode
32 Saturation On/Off
33 Saturation Amount
34 Saturation Shape
35 Crossfade Type
36 Pitch Wheel Range
37 Play Position Memory
39 Record Mode
40 Record On/Off
41 Record Level
42 Dry Level
43 Feedback
44 Drone 0 On/Off
45 Drone 0 Note
46 Drone 0 Volume
47 Drone 1 On/Off
48 Drone 1 Note
49 Drone 1 Volume
50 Drone 2 On/Off
51 Drone 2 Note
52 Drone 2 Volume
53 Filter LFO Shape
54 Filter LFO Speed
55 Filter LFO Amount
56 Filter LFO Invert
57 Filter LFO Delay
58 Filter LFO Retrigger
59 Filter LFO Mono
60 Pitch LFO Shape
61 Pitch LFO Speed
62 Pitch LFO Amount
63 Pitch LFO Invert
65 Pitch LFO Delay
66 Pitch LFO Retrigger
67 Pitch LFO Mono
68 Calibration Note
69 Buffer Size
70 Use Beats
71 Beats
72 Beat Divisor
73 Triplet
74 Pulse Width
75 Pulse Width LFO Shape
76 Pulse Width LFO Speed
77 Pulse Width LFO Amount
78 Pulse Width LFO Invert
79 Pulse Width LFO Delay
80 Pulse Width LFO Retrigger
81 Pulse Width LFO Mono
82 Reference Tone Level
83 Hard Sync Cycles
84 Hard Sync Detune
85 Sync Detune LFO Shape
86 Sync Detune LFO Speed
87 Sync Detune LFO Amount
88 Sync Detune LFO Invert
89 Sync Detune LFO Delay
90 Sync Detune LFO Retrigger
91 OSC Port Offset

OSC Control

From version 3.1.0 Crossfade Loop Synth can be controlled via the Open Sound Control (OSC) protocol.

If you're new to OSC, start by visiting opensoundcontrol.org.

Two settings control what port the plug-in uses to listen on for OSC commands. One is the base OSC port, set in the preferences. The second is the OSC Port Offset control. If the port offset is set to something other than 'Off', then the two numbers are added together and the result used as the port number. E.g. if the base port is 6000 and the port offset is 1, then the plug-in will listen on port 6001.

Received OSC Commands

In the documentation below, OSC parameters are prefixed with a string to indicate their type, as follows:

- s - string
- i - integer
- f - float
- b - boolean

All the examples assume that the plug-in is listening at address 10.0.0.1:6001.

/ping s:returnUrl s:returnPath

Responds by sending a message back to the returnUrl and returnPath with the parameters

```
s:hosturl s:version
```

E.g.

```
/ping osc.udp://10.0.0.2:7000 "/foo"
```

replies to 10.0.0.2:7000 with

```
/foo osc.udp://10.0.0.1:6001 "Crossfade Loop Synth 3.1.0"
```

/set i:param f:value

Sets the value of parameter 'param' to 'value'.

/get i:param s:returnUrl s:returnPath

Responds by sending a message back to the returnUrl and returnPath with the parameters

```
i:param f:value
```

where 'value' is the value of parameter 'param'. E.g.

`/get 14 osc.udp://10.0.0.2:7000 "/foo"`
replies to 10.0.0.2:7000 with (assuming parameter 14 has the value 64.0)

`/foo 14 64.0`

/getAll s:returnUrl s:returnPath

Behaves exactly as if a /get message was received for every parameter.

/getNumParameters s:returnUrl s:returnPath

Responds by sending a message back to the returnUrl and returnPath with the parameters

`i:numParameters`
where 'numParameters' is the total number of parameters defined by the plug-in. E.g.

`/getNumParameters osc.udp://10.0.0.2:7000 "/foo"`
replies to 10.0.0.2:7000 with (assuming the plug-in has 84 parameters)

`/foo 84`
Note that there can be 'gaps' in the array of parameters - see isParameterUsed below.

/isParameterUsed i:param s:returnUrl s:returnPath

Responds by sending a message back to the returnUrl and returnPath with the parameters

`i:param b:isUsed`
where 'isUsed' is 'true' if parameter 'param' is used, and 'false' otherwise. Parameters that are not used should not be used for any other call e.g. the getInfo call below.

/getInfo i:param s:returnUrl s:returnPath

Responds by sending a message back to the returnUrl and returnPath with the parameters

`i:param f:minValue f:maxValue f:defaultValue s:name i:unit`
where 'minValue' and 'maxValue' are the minimum and maximum values that parameter 'param' can take, 'defaultValue' is the default value of the parameter, 'name' is the name of the parameter, and 'unit' is a value that indicates the unit of the parameter (e.g. Hz, db, seconds). The unit is one of the values defined by Apple's Audio Unit specification.

/registerUpdate i:param s:returnUrl s:returnPath

Requests that when the parameter 'param' changes, a message is sent back to the returnUrl and returnPath with the parameters

`i:param f:value`
where the returned parameters have the same meaning as for the /get command (above).

/unregisterUpdate i:param s:returnUrl s:returnPath

Cancels a request made via /registerUpdate (above).

/getAllRegistered

Behaves exactly as if every parameter registered for updates with /registerUpdate had changed. A message will be sent for every such parameter.

/exec s:func ...

Executes the Lua function 'func', which is assumed to be defined by the MIDI & OSC scripting system (see below). OSC parameters following 'func' are passed through to the Lua function, as can best be managed given the varying limitations of the two. Specifically, the following table describes the mapping from OSC types to Lua types:

OSC	Lua
bool	bool
float	number
double	number
int32	number
int64	number
string	string
nil	nil

/call s:func s:returnUrl s:returnPath ...

As /exec, but also responds to the returnUrl and returnPath with the results of the Lua function call. The following table describes the mapping from Lua return values to OSC types:

Lua	OSC
number	float
string	string

Lua types not in the above table are not handled.

/startNote i:note f:velocity

Starts playback of a note - equivalent of a MIDI note on message.

'note' is the MIDI note number; 'velocity' should be between 0.0 and 1.0. E.g.

```
/startNote 60 0.7
```

/stopNote i:note f:velocity

Stops playback of a note - equivalent of a MIDI note off message.

‘note’ is the MIDI note number; ‘velocity’ should be between 0.0 and 1.0. E.g.

```
/stopNote 60 0.0
```

Note that the plug-in currently does not respond in any way to note off velocity.

/loadSample s:filename

Loads the given sample file - exactly as if the user had pressed the ‘Load Sample’ button and chosen the file from the dialogue. E.g.

```
/loadSample “/Users/sleeper/audio/samples/moo.aiff”
```

MIDI & OSC Scripting

Preamble

From Crossfade Loop Synth v3.1.0 onwards it is possible to extend the plug-in's MIDI & OSC functionality via user-writeable scripts. Indeed, the standard MIDI functionality described above has been re-implemented using such a script, which you can use as reference for your customisations.

The language used for the MIDI scripts is Lua. You will find a complete description of the language, and some useful tutorials, at the Lua website: www.lua.org

All the standard language features of Lua are available in the GUI scripts, plus some extra functions (documented below) specific to the Expert Sleepers system.

Learn by example

The best way to learn about scripting the MIDI & OSC functionality is to look at the existing examples, particularly the default script that ships with the plug-in. Just open up the plug-in bundle and find the midi.lua file within. (Windows users should download the Mac OS X version of the plug-in and get the script from there, since in the Windows version the script is munged into the plug-in as a Windows resource.)

Most of the example snippets in the documentation below are taken directly from the default midi script.

You should be able to find more scripts on the Expert Sleepers website.

Share your scripts!

You are encouraged to share your scripts with other users. For example, you could post them on the Expert Sleepers forum (linked from the website). Alternatively, email them to us, and we'll make the best of the bunch downloadable directly from the Expert Sleepers site.

Script locations

Crossfade Loop Synth looks for MIDI & OSC scripts in standard locations. Scripts must have the filename extension ".lua".

Mac OS X

The plug-in looks for scripts in

```
Library/Application Support/Expert Sleepers/Crossfade Loop Synth/
Scripts
```

Windows

The plug-in looks for scripts in

```
C:\Documents and Settings\<username>\Application Data\Expert
Sleepers\Crossfade Loop Synth\Scripts
```

Overriding the default script

Normally any scripts that the plug-in finds are run in addition to (and after) the default script ('midi.lua') that comes with the plug-in itself.

However, if you name your own script 'midi.lua', then the default script is not run. This lets you completely replace the plug-in's default MIDI behaviour (as described previously), rather than simply extend it.

MIDI & OSC Script Functions

The scripts are simply loaded and executed. You do not need to define any particular functions for the system to call.

The following functions are available for you to call to define your script behaviour.

getParameterID(param)

Returns the parameter ID of the named parameter. Use with setParameter()/ getParameter() (see below). E.g.

```
paramID_Pitch = getParameterID( "Pitch" )
```

getParameter(param)

Returns the value of the plug-in parameter. 'param' can either be the parameter name or the parameter ID (as returned from getParameterID()). Using the ID is more efficient. Typically you would obtain the ID in the main script body (which is only executed once) and then use it in a handler function (which can be called many times). E.g.

```
pitch = getParameter( “Pitch” )
pitch = getParameter( paramID_Pitch )
```

setParameter(param, value)

Sets the value of the plug-in parameter. See the description of getParameter() for the meaning of ‘param’. E.g.

```
setParameter( “Pitch”, 12.0 )
setParameter( paramID_Pitch, 12.0 )
```

getParameterMinMax(param)

Returns the minimum and maximum values allowable for a plug-in parameter. See the description of getParameter() for the meaning of ‘param’. E.g.

```
local minv, maxv = getParameterMinMax( paramID_Pitch )
```

getParameterUnit(param)

Returns an integer value that indicates the unit of the parameter (e.g. Hz, db, seconds). The unit is one of the values defined by Apple’s Audio Unit specification.

getParameterName(param)

Returns the name of the parameter. (This is the same name that appears at the top right of the GUI when the mouse is over a parameter’s control.)

isParameterUsed(param)

Returns a boolean value indicating whether the given parameter number is used by the plug-in. You should not attempt to set or get the value of an unused parameter.

getNumParameters()

Returns the total number of parameters that the plug-in defines. More strictly speaking - returns one more than the largest parameter ID that the plug-in uses, since there may be unused parameter IDs.

setOthersParameter(id, param, value)

As setParameter(), but sets the parameter on another instance of the plug-in, not necessarily the one running the script. This allows you to control several instances of the plug-in from a single script.

The ‘id’ is matched against the OSC Port Offset of the plug-ins. Any plug-in that matches the id will have its parameter set.

Note that all the plug-ins must be loaded by the same host application. For controlling instances of the plug-in loaded by other hosts, or running on other computers, use the ‘sendOSC’ command (below).

E.g.

```
setOthersParameter( 2, paramID_Pitch, 12.0 )
```

getOthersParameter(id, param)

As getParameter(), but gets the parameter from another instance of the plug-in. See setOthersParameter() for a fuller explanation. E.g.

```
pitch = getOthersParameter( 2, paramID_Pitch )
```

sendOSC(address, path [, format] [, values])

Sends an OSC message. 'values' is an optional array of data items to be sent with the message. If 'values' is used, then 'format' is an optional string that indicates how the items in the values array should be interpreted. This is required because Lua treats all numbers as being of the same type, whereas OSC differentiates between integers and floating point values. The number of characters in 'format' should be the same as the number of values. Each character may be one of 'i' (integer), 'f' (float) or 's' (string).

E.g.

```
sendOSC( "osc.udp://localhost:7001", "/foo" )
sendOSC( "osc.udp://localhost:7001", "/foo", { 3, 5.2, "hello" } )
sendOSC( "osc.udp://localhost:7001", "/foo", "ifs", { 3, 5.2, "hello" } )
```

Note that the second example sends two floats and a string; the third sends an integer, a float and a string.

requestAllNoteOn(function)

Request that the given function be called in response to any MIDI note on event. E.g.

```
local function handleNoteOn( channel, noteNumber, velocity )
   -- do stuff
end
requestAllNoteOn( handleNoteOn )
```

requestAllNoteOff(function)

Request that the given function be called in response to any MIDI note off event. E.g.

```
local function handleNoteOff( channel, noteNumber, velocity )
   -- do stuff
end
requestAllNoteOff( handleNoteOff )
```

requestAllCC(function)

Request that the given function be called in response to any MIDI continuous controller (CC) event. E.g.

```
local function handleCC( channel, cc, value )
   -- do stuff
```

```
end
requestAllCC( handleCC )
```

requestAllNRPN(function)

Request that the given function be called in response to any MIDI non-registered parameter number (NRPN) event. E.g.

```
local function handleNRPN( channel, nrpn, value )
   -- do stuff
end
requestAllNRPN( handleNRPN )
```

requestAllProgramChange(function)

Request that the given function be called in response to any MIDI program change event. E.g.

```
local function handlePC( channel, value )
   -- do stuff
end
requestAllProgramChange( handlePC )
```

requestAllPolyPressure(function)

Request that the given function be called in response to any MIDI poly pressure (polyphonic aftertouch) event. E.g.

```
local function handlePolyPressure( channel, key, value )
   -- do stuff
end
requestAllPolyPressure( handlePolyPressure )
```

requestNoteOn(note, function)

Request that the given function be called in response to a MIDI note on event matching the given note number. E.g.

```
local function handleNoteOn( channel, noteNumber, velocity )
   -- do stuff
end
requestNoteOn( 60, handleNoteOn )
```

requestNoteOff(note, function)

Request that the given function be called in response to a MIDI note off event matching the given note number. E.g.

```
local function handleNoteOff( channel, noteNumber, velocity )
   -- do stuff
end
requestNoteOff( 60, handleNoteOff )
```

requestCC(cc, function)

Request that the given function be called in response to the given MIDI continuous controller (CC) event. E.g.

```
local function handleCC( channel, cc, value )
   -- do stuff
end
requestCC( 20, handleCC )
```

requestNRPN(nrpn, function)

Request that the given function be called in response to the given MIDI non-registered parameter number (NRPN) event. E.g.

```
local function handleNRPN( channel, nrpn, value )
   -- do stuff
end
requestNRPN( 1000, handleNRPN )
```

requestProgramChange(pc, function)

Request that the given function be called in response to the given MIDI program change event. E.g.

```
local function handlePC( channel, value )
   -- do stuff
end
requestProgramChange( 2, handlePC )
```

requestPolyPressure(key, function)

Request that the given function be called in response to a MIDI poly pressure (polyphonic aftertouch) event on the given key. E.g.

```
local function handlePolyPressure( channel, key, value )
   -- do stuff
end
requestPolyPressure( 60, handlePolyPressure )
```

requestPitchWheel(function)

Request that the given function be called in response to a MIDI pitch wheel event. NB the value passed to the handler function is the raw 14 bit MIDI value, not e.g. a normalised ±1.0 value. E.g.

```
local function handlePitchWheel( channel, value )
   -- do stuff
end
requestPitchWheel( handlePitchWheel )
```

requestChannelPressure(function)

Request that the given function be called in response to a MIDI channel pressure (after-touch) event. E.g.

```
local function handleChannelPressure( channel, value )
   -- do stuff
end
requestChannelPressure( handleChannelPressure )
```

Pre-defined Global Values

The system defines some values before calling your script, which you can use to make the script's behaviour dependent on, for example, what kind of computer you're using. These values (which are pretty self-explanatory) are:

- isMac
- isWin
- isVST
- isAU
- majorVersion
- minorVersion
- dotVersion
- version

The plug-in's version number is of the form x.y.z (e.g. 2.1.4) where x is the major version number, y is the minor version number, and z is the dot version. The 'version' global variable contains a single value combining all three e.g. for version 2.1.4, 'version' is 20104. This is useful for making your scripts backwardly compatible - by testing for the version number and not trying to use features that were not present in a version of the plug-in older than the version you're testing for.

Debugging

You can use Lua's 'print' function to write out information to help you track what's going on (or what's not going on) in your script. Also any run-time errors, or errors in loading the script in the first place, are reported. In both cases, the output goes to:

Mac OS X

The system console.log. Use the standard Console utility (located in `Applications/Utilities`) to view it.

Windows

The system OutputDebugString API. Use an application like Sysinternal's DebugView to view it.

Version History

3.2.0 11/8/2011

- Mac OS X AU & VST versions now support 64 bit operation.
- Timing of note playback much improved in the synth versions of the plug-in (now sample accurate).
- Fixed a bug which would cause the plug-in to attempt to load an empty filename when loading a song in Plogue Bidule.

3.1.4 18/4/2011

- Fixed a rare crash that could occur when automating parameters in Cubase, using the 64 bit version of Cubase and the 32 bit plug-in bridge.

3.1.3 26/1/2011

- Fixed a bug in the plug-in's response to a MIDI All Notes Off message. In Plogue Bidule, this could cause a problem with the drones not reactivating when a plug-in was switched in and out of Bypass mode.

3.1.2 6/10/2009

- Fixed compatibility with Mac OS X 10.6 "Snow Leopard".

3.1.1 12/2/2009

- Fixed an issue in the Windows VST version that would prevent certain WAV files from loading successfully.

3.1.0 2/2/2009

- Added next/previous sample buttons
- Added OSC control.
- Added MIDI/OSC scripting.

3.0.5 20/5/2008

- First Windows version.

3.0.4 13/1/2008

- Fixed a crash when loading VST presets with the GUI hidden.

3.0.3 16/12/2007

- Fixed a possible crash when using very short buffer sizes in the Effect version.

3.0.2 30/9/2007

- (AU version) Now responds properly to AudioUnitReset(), so in e.g. Logic the delay buffer will be cleared when song playback is started etc.

3.0.1 18/3/2007

- Fixed crash bug when using Effect version with small (<64 samples) buffer sizes.
- Fixed failure in MOTU's AU Examiner that prevented use in Digital Performer.

3.0.0 15/3/2007

- First effect version.
- Added 'play position memory'.
- Added drones.
- Added pulse width modulation.
- Added 'hard sync'.
- Added LFOs.
- Added reference tone.
- Added calibration display.

2.1.5 4/1/2007

- VST version now built against VST SDK 2.4 and so is usable in current VST hosts.
- VST version now supports GUI resizing (as AU version previously did).

2.1.4 9/5/2006

- Fixed stability problems with filter.
- Added control of pitch wheel range.

2.1.2 22/4/2006

- First Universal Binary version.

2.1.1 27/1/2006

- First VST version.

2.1.1 2/2/2005

- fixed crash when using alternate and reverse loop modes
- fixed crash which could occur when GUI repeatedly opened and closed

2.1.0 22/12/2004

- added "equal power" crossfade option
- fixed issues with detuned playback of large samples
- cpu efficiency improvements

2.0.0 12/12/2004

- all-new GUI
- new 'reverse' and 'alternate' loop modes
- amplitude and filter envelopes now ADSR with adjustable shape
- added saturation section
- increased range of filter cutoff
- volume control now has a different response curve
- NB 2.0.0 is not compatible with 1.x presets

1.3 20/3/2004

- added MIDI CC control of most parameters
- resolved some possible (rare) crash & glitch issues
- added handling of MIDI 'All Notes Off' and 'All Sound Off' messages
- added support for 24 bit audio files
- added confirm dialogue when attempting to load a very large sample
- added support for MIDI pitch bend messages
- if two presets use the same sample, changing between them no longer reloads the sample or cuts off any sounding notes (though note that any glitching caused by sudden parameter changes will still occur)
- fixed possible audio cut-out when loop endpoints moved while notes sounding

1.2 12/2/2004

- added 'one shot' playback mode
- fixed possible crash when changing sample while notes are playing
- added volume control
- better audio file support, including support for wav and sd2 files

1.1 2/10/2003

- fixed component registry bug
- fixed crash when loop end set before loop start
- added fine tune control
- added filter sweep control

1.0 4/2/2003

- added Aqua GUI

0.2b 29/12/2002

- sample filename & path now stored with preset
- added proper handling of user pressing 'cancel' in file choosing dialogue
- improved voice allocation algorithm when repeatedly playing the same note
- added state-variable filter with its own envelope
- added a readme

0.1b 11/12/2002

- First beta release.

Release Notes

v3.0.x and v3.1.x have a known issue in that PWM and Hard Sync don't work well together. The intention is to re-implement PWM and release this as a free upgrade at some point in the future.

Contact

The Expert Sleepers website is here:

http://www.expert-sleepers.co.uk/

Or you can email

info@expertsleepers.co.uk

Or you can use the forum, which is here:
http://www.kvraudio.com/forum/viewforum.php?f=85