Expert Sleepers
Oomingmak
v1.0.0
User Manual

Contents

Introduction 8

Installation 9

Mac OS X, Audio Unit (AU) 9
Mac OS X, VST 9
Windows (VST) 9

System Requirements 9

Mac OS X 9
Windows 9

Registration 10

Quickstart 11

Factory presets 11

Defaults 11
Down One Octave 11
Down Two Octaves 11
Sync To The Max 11
Auto-Wah 11
PWM-tastic 11
Ring Mod 12
Auto Bassline 12
Try Me On Drums 12
Sweep The Pitch 12
Madness 12

Using Oomingmak 13

Using the controls 13

Knobs 13
Value edit boxes 13
Name/value display 13

Overview 14

Input Filter 15

Highpass/Lowpass 15
Filt Dry/Filt Wet 15

Envelope 15
Level meters 15
Gain 15
Attack/Release 15
Gate 16

Pitch Tracking Display 16

Oscillator 16
Tri 16
Saw 16
Square 16
PWM 16

Pitch 17
Pitch 17
Sweep 17
Sync 17
Osc 17

Filter 18
Cutoff 18
Q 18
Type 18

Env Mod 18
Osc Env 18
Trigger 18
Pitch 19
Filter 19
PWM 19

Pitch LFO 19
Shape 19
Speed 19
Amount/Fine 19

Filter LFO 19
Shape/Speed 19
Amount 20

PWM LFO 20
Shape/Speed/Amount 20

Mix 20
Dry 20
Wet 20
Osc 20
Dry RM 20
Wet RM 20

Preferences 21
OSC Base Port 21
Eye candy 21
Constant redraw 21
Floating tooltip 22

MIDI control 23

OSC Control 24

Received OSC Commands 24
/ping s:returnUrl s:returnPath 24
/set i:param f:value 24
/get i:param s:returnUrl s:returnPath 24
/getAll s:returnUrl s:returnPath 25
/getNumParameters s:returnUrl s:returnPath 25
/isParameterUsed i:param s:returnUrl s:returnPath 25
/getInfo i:param s:returnUrl s:returnPath 25
/registerUpdate i:param s:returnUrl s:returnPath 25
/unregisterUpdate i:param s:returnUrl s:returnPath 25
/getAllRegistered 26
/exec s:func ... 26
/call s:func s:returnUrl s:returnPath ... 26

MIDI & OSC Scripting 27

Preamble 27

Learn by example 27

Share your scripts! 27

Script locations 28
Mac OS X 28
Windows 28

Overriding the default script 28

MIDI & OSC Script Functions 28

getParameterID(param) 28
getParameter(param) 28
setParameter(param, value) 29
getParameterMinMax(param) 29
getParameterUnit(param) 29
getParameterName(param) 29
isParameterUsed(param) 29
getNumParameters() 29
setOthersParameter(id, param, value) 29
getOthersParameter(id, param) 30
sendOSC(address, path [, format] [, values]) 30
requestAllNoteOn(function) 30
requestAllNoteOff(function) 30
requestAllCC(function) 30
requestAllNRPN(function) 31
requestAllProgramChange(function) 31
requestAllPolyPressure(function) 31
requestNoteOn(note, function) 31
requestNoteOff(note, function) 31
requestCC(cc, function) 32
requestNRPN(nrpn, function) 32
requestProgramChange(pc, function) 32
requestPolyPressure(key, function) 32
requestPitchWheel(function) 32
requestChannelPressure(function) 33

Pre-defined Global Values 33

Debugging 33

Mac OS X 33
Windows 33

Version History 34

Contact 35

Acknowledgements 36

Lua 36

oscpack 36

glew 37

FreeType **38**

FTGL **38**

libpng **38**

zlib **38**

Introduction

Short version:

Oomingmak turns your guitar into an analogue monosynth.

Long version:

Oomingmak is a pitch- and envelope-tracking (re)synthesis effect.

It tracks the pitch and level of the incoming audio and uses them to:

- synthesize an entirely original sound, using classic analogue-style waveforms and subtractive synthesis.
- apply pitch modulation to the incoming audio, while syncing to the original waveform (the classic 'oscillator sync' effect, except that the 'oscillator' here is your audio).

The oscillator can also be synced to the incoming audio, and the tracked envelope can be used to modulate the pitch, filter cut-off and oscillator pulse-width. LFOs are also provided for further modulation possibilities.

Two ring modulators are also provided, allowing two different combinations of the above sounds to be modulated against each other.

As with all pitch-tracking devices, Oomingmak tends to work best on monophonic sound sources. It was designed particularly with the guitar in mind, in which case it tends to work well on solo lines rather than chords. However, sending chords, or even non-pitched sounds (e.g. drums) through Oomingmak can also have very interesting results.

Installation

Mac OS X, Audio Unit (AU)

The plug-in file is called `ExSlOomingmak.component`.

Simply copy the file to the folder:

`Library/Audio/Plug-Ins/Components`

Mac OS X, VST

The plug-in file is called `ExSlOomingmak.vst`.

Simply copy the file to the folder:

`Library/Audio/Plug-Ins/VST`

Windows (VST)

The plug-in file is called `oomingmak.dll`.

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

System Requirements

Mac OS X

Oomingmak requires at least Mac OS X version 10.2.8. Version 10.4 or higher is recommended.

The plug-ins are Universal Binaries and so will work on PowerPC or Intel Macs.

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

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

Windows

Oomingmak has been developed and tested with Windows XP SP2. It may work with other versions of Windows (Vista included) 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 Oomingmak 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 1.0.13 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 Oomingmak 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.

Quickstart

For a quick overview of Oomingmak, load up the plug-in in your host application of choice and try out the factory presets, which are listed below.

In all cases you will probably need to set up your levels appropriately, as described below in the Envelope section, so that the envelope tracking works optimally.

Factory presets

Defaults

In this preset all parameters are at their default value, and audio is passed through the plug-in unaltered. This is a plain 'vanilla' preset from which to start creating your own sounds.

Down One Octave

Uses the Oscillator to create a tone an octave below that being tracked. The key settings to note here are the Pitch (-12) and Sync (2) controls in the Pitch section.

Down Two Octaves

As Down One Octave, but another octave lower. Note the Pitch (-24) and Sync (4). This preset uses a different combination of Oscillator waveforms to produce a different timbre.

Sync To The Max

Most settings are at default values, except the Pitch in the Env Mod section which is set to its maximum value of 48. Therefore what you hear is the ReSynth section performing its deepest oscillator sync effect according to the incoming audio envelope.

Auto-Wah

No pitch modulation in this preset - instead the Filter envelope tracking is used to create a traditional 'auto-wah' effect, i.e. a low pass filter whose cut-off frequency depends on the audio envelope.

PWM-tastic

This preset demonstrates the PWM abilities of the Oscillator. The output mix is set to only output the square wave waveform, and the envelope and PWM LFO modulate the oscillator pulse-width.

Ring Mod

A fairly traditional ring modulation effect. The oscillator's pitch is fixed, rather than tracking the incoming audio, and the 'Dry RM' output is selected. However the oscillator's pitch is modulated slightly by the envelope, making this rather more than a standard ring mod.

Auto Bassline

The oscillator's output is used, and fixed to a bass note. The envelope controls the oscillator's level, and the filter. The result is a simple bassline that takes its rhythm from the incoming audio. Try feeding a drum loop through this preset.

Try Me On Drums

The clue is in the title. Applies a variety of sound mangling that sounds pretty interesting on drums.

Sweep The Pitch

An extreme sounding ring mod type effect. Interesting sounds can be obtained by messing about with the 'Sweep' parameter.

Madness

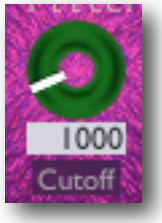
More extreme sound mangling.

Using Oomingmak

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.
- Option[2] (Mac OS X)/Control (Windows): The knob assumes integer values only.

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

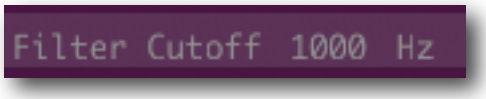
Value edit boxes

These boxes (below each knob) let you enter parameter values directly. Clicking on the value highlights it in green - you can then type the desired value using the keyboard. Press enter to finish and accept the new value.

While you're typing the value, the box goes red to indicate that the value you see has not yet been accepted.

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 tool-tips for buttons.

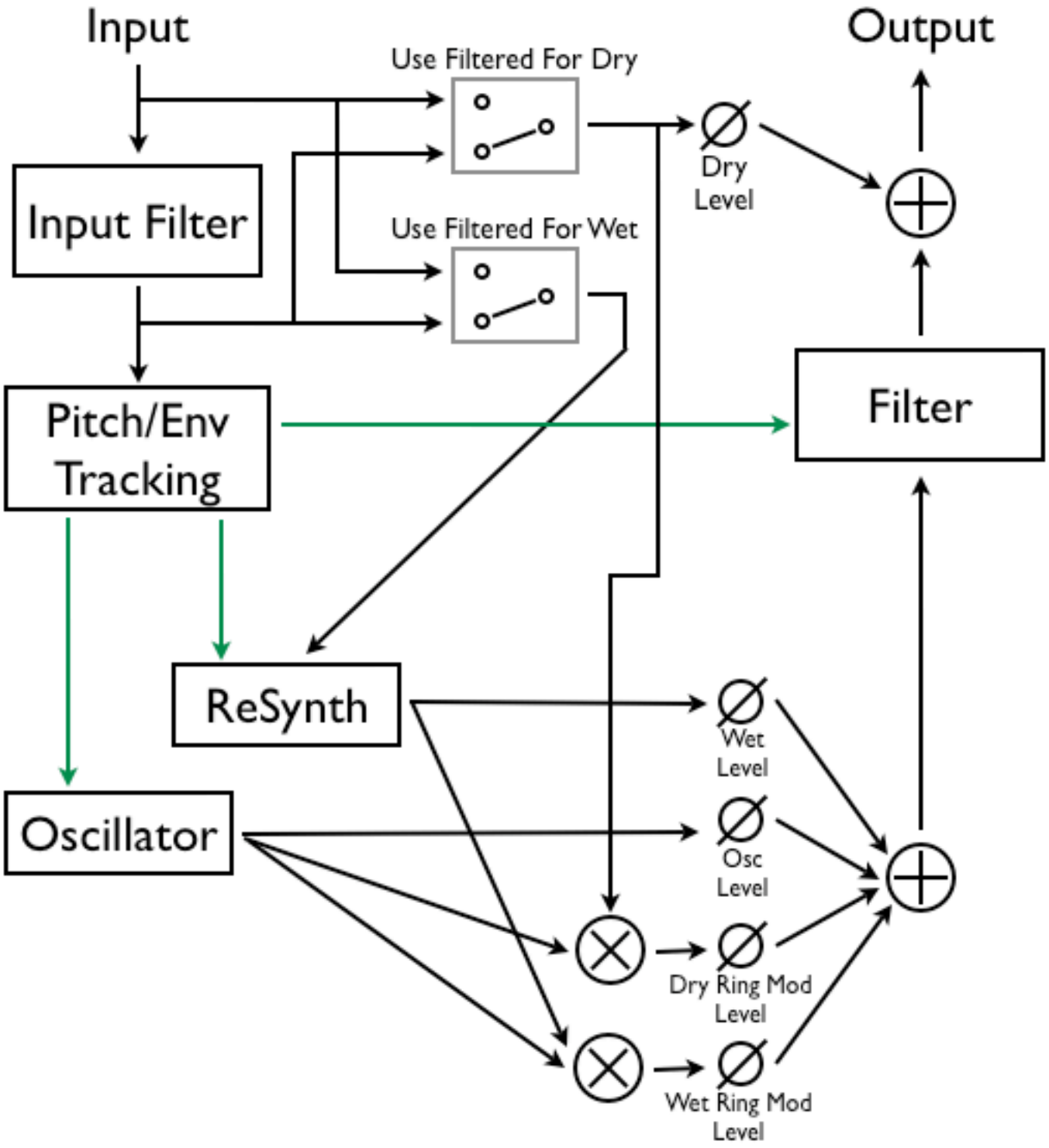


[1] The 'Command' key is also known as the 'Apple' key - the one next to the spacebar.

[2] The 'Option' (alt) key is the one between the Control (ctrl) key and the Command (cmd) key.

Overview

Here's a block diagram of the 'circuitry' of Oomingmak. It can help to visualise this when understanding the effect of some settings.



Input Filter

The input filter section filters the incoming audio before it reaches the pitch tracking section. By removing frequencies outside of the range of interest, the tracking can be made more reliable.

Highpass/Lowpass

The Highpass and Lowpass controls apply highpass (low frequencies removed) and lowpass (high frequencies removed) filtering, respectively, at the specified frequencies.

Filt Dry/Filt Wet

The 'Filt Dry' and 'Filt Wet' switches let you use the filtered signal elsewhere in the plug-in which by default use the original, unfiltered, signal. Refer to the overview diagram for details.

Envelope

The Envelope section controls Oomingmak's envelope (signal level) tracking.

Level meters

Two stereo level meters to the right of the Envelope controls show the detected and synthesised envelope levels.

The meters on the left show the detected level - the level of the incoming audio. Typically you want to aim to set the input level (or use the gain control, below) so that the meters just peak when you play your loudest sound.

The meters on the right show the synthesised level - the level that will be passed to the rest of the plug-in (see the overview).

Gain

The Gain control lets you boost the level of the signal going into the envelope detection, without actually changing the level of the audio elsewhere in the plug-in.

Attack/Release

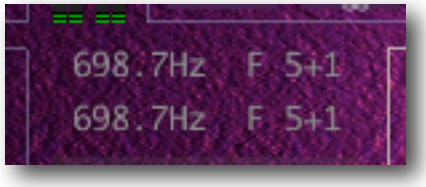
These controls let you set the attack and release times of the envelope that is actually used to control the rest of the plug-in. Higher attack settings will cause the envelope to rise

more slowly than that of the incoming audio; higher release settings will cause the envelope to fall more slowly than that of the incoming audio.

Gate

Once the envelope falls below the gate value, it is forced to exactly zero. (This is similar to the operation of a traditional noise gate.) This is useful if your incoming audio is quite noisy, and you don't want a very low level of synthesised sound leaking out when you're not actually playing.

Pitch Tracking Display



Just below the level meters is a readout of the currently tracked pitch, in Hz, and as note names and cents.

The numbers go red if Oomingmak fails to track the pitch of the incoming audio.

The display vanishes entirely once the envelope has fallen to zero.

Oscillator

These knobs control the output waveform of the oscillator section.

Tri

The level of the triangle wave output.

Saw

The level of the sawtooth wave output.

Square

The level of the square wave output.

PWM

The pulse-width of the square wave output.

Pitch

These knobs relate in various ways to the pitch of the Oscillator and ReSynth sections.

Pitch

Sets the pitch, in semitones, of the Oscillator and ReSynth outputs, relative to the tracked pitch as determined by the tracking section.

Sweep

Exactly the same as the 'Pitch' control except that the knob can take on any value, not just integer values. The two controls are simply added together to determine the total pitch value.

Sync

When set to a value other than 'Off', turns on oscillator sync for the Oscillator section. (The ReSynth section always uses sync.)

Oscillator sync is a traditional analogue synthesis technique, whereby the start of the waveform on one oscillator causes that of another to restart, regardless of the pitch of the second oscillator. In effect, the pitch of the first oscillator is imposed on the second, and the difference in their actual frequencies can give rise to very complex and interesting-sounding harmonics.

The value of the Sync control specifies the number of ReSynth oscillator cycles after which the Oscillator cycle should be reset. A value of 1 gives a basic sync effect (try modulating the pitch with the Sweep control or the envelope). A value of 2 allows you to sync to an octave below the tracked pitch; 4 tracks 2 octaves below, etc.

Osc

This control sets the pitch of the Oscillator section. If set to 'Track', then the Oscillator pitch tracks that of the incoming audio. If set to any other value, the Oscillator pitch is fixed at that value (though still modulated by the LFO and envelope).

A fixed pitch value is typically useful when using the Ring Modulator outputs, or when using Oomingmak to generate a fixed drone note to accompany your playing.

The values are calibrated in MIDI note numbers, allowing you to easily set musically meaningful values. Remember that holding Option/Control while moving the mouse lets you set integer values.

Filter

The Filter section applies a state-variable filter to the output of the other sections.

Cutoff

Controls the filter's cut-off frequency.

Q

Controls the filter's resonance.

Type

Allows you to smoothly change between the following filter types:

- Thru - no filtering
- Low - lowpass filter
- Band - bandpass filter
- High - highpass filter

Env Mod

This section controls how the envelope affects various other sections.

Osc Env

This knob controls the extent to which the tracked envelope controls the level of the Oscillator. At the default setting of 1.0, the Oscillator is fully controlled by the envelope. At settings below 1.0, the Oscillator does not entirely fade away as the envelope falls to zero. At a setting of 0.0, the Oscillator is always at full volume (use with care, as full volume is most likely a lot louder than the other audio in your system).

Trigger

Controls the level above which the envelope must rise before it has any affect on the controlled values.

This is typically useful to make it so the envelope modulation only affects the attack portion of your sound, while the (lower level) sustain section is not affected.

Pitch

Sets the amount by which the Oscillator and ReSynth sections' pitches are modulated by the envelope.

Filter

Sets the amount by which the Filter's cut-off frequency is modulated by the envelope.

PWM

Sets the amount by which the Oscillator's square wave's pulse-width is modulated by the envelope.

Pitch LFO

This section lets you apply a low frequency modulation to the Oscillator and ReSynth modules' pitch.

Shape

Lets you choose the LFO waveform from:

- Sine
- Tri(angle)
- SawUp
- Square
- SawD(ow)n

Speed

Sets the LFO frequency, in Hz.

Amount/Fine

Set the amount of pitch modulation applied. 'Amount' is in semitones; 'Fine' is in cents.

Filter LFO

This section lets you apply a low frequency modulation to the filter's cut-off frequency.

Shape/Speed

As for the Pitch LFO, above.

Amount

Sets the amount of modulation applied.

PWM LFO

This section lets you apply a low frequency modulation to the Oscillator's square wave's pulse-width.

Shape/Speed/Amount

As for the Filter LFO, above.

Mix

This section sets the output levels of the various other sections. Refer to the overview diagram if needed to clarify what these settings control exactly.

Dry

Sets the level of the unprocessed audio (or the audio filtered by the Input Filter if the Use Filtered For Dry option is set).

Wet

Sets the level of the ReSynth section.

Osc

Sets the level of the Oscillator.

Dry RM

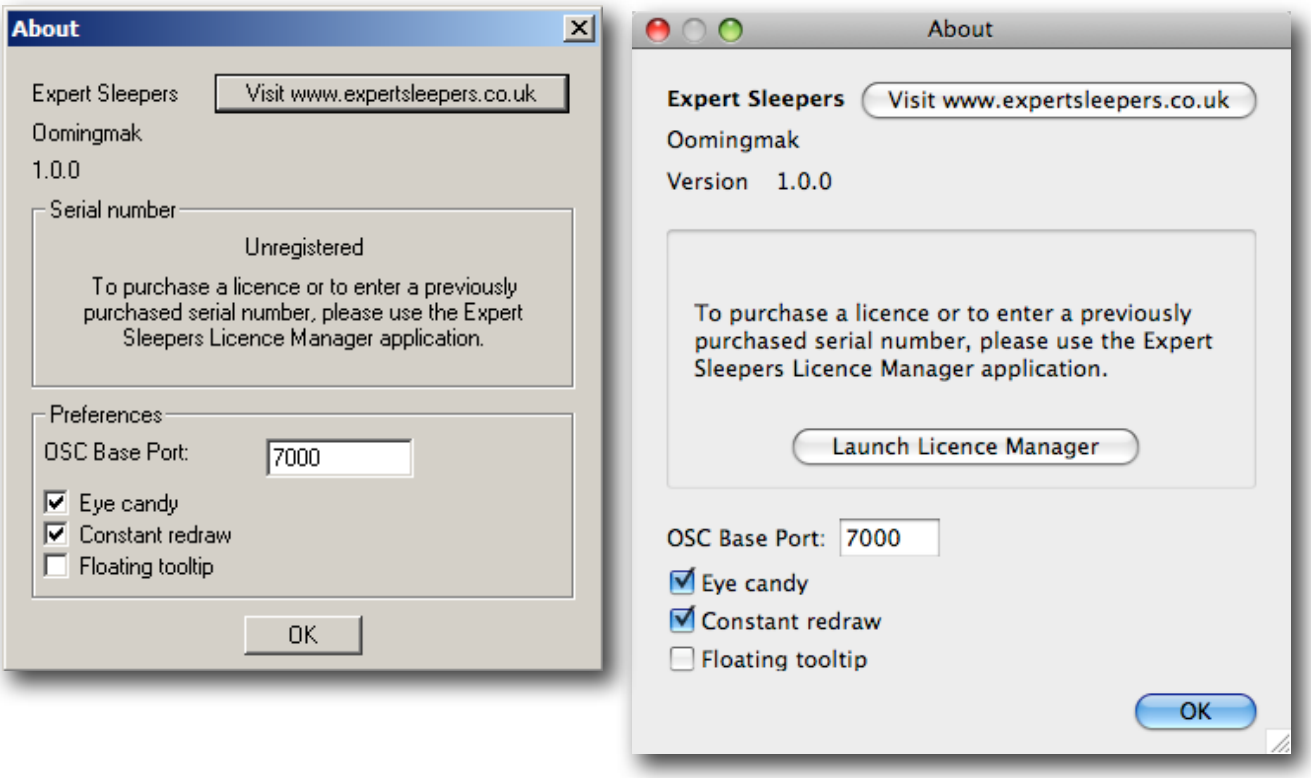
Sets the level of the ring modulator driven by the Dry signal and the Oscillator.

Wet RM

Sets the level of the ring modulator driven by the ReSynth signal and the Oscillator.

Preferences

Pressing the 'Prefs' button brings up a dialog where various preferences are set. These settings are shared by all instances of Oomingmak, 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.

Eye candy

Enables the pretty graphics. Turn off if you don't like them, or if your computer has compatibility issues with drawing such things.

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 tracked pitch and envelope 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.

MIDI control

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

0	Dry Level
2	Wet Level
3	Osc Level
4	Dry Ring Mod Level
5	Wet Ring Mod Level
8	Pitch Offset
9	Pitch Offset Sweepable
11	Lowpass Cutoff
12	Highpass Cutoff
13	Use Filtered For Dry
14	Use Filtered For Wet
15	Sync Cycles
16	Env Pre Gain
17	Env Attack
18	Env Release
19	Env Gate
20	Osc Env Follow
21	Trigger Level
22	Osc Pitch
23	Pitch Env Depth
24	Triangle
25	Saw
26	Square
27	Pulse Width
28	PWM Env Depth
29	Filter Cutoff
30	Filter Q
31	Filter Type
32	Filter Env Depth
33	Pitch LFO Shape
34	Pitch LFO Speed
35	Pitch LFO Amount
36	Pitch LFO Amount Fine
37	Filter LFO Shape
39	Filter LFO Speed
40	Filter LFO Amount
41	PWM LFO Shape
42	PWM LFO Speed
43	PWM LFO Amount
44	OSC Port Offset

OSC Control

Oomingmak 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 "Oomingmak 1.0.0"
```

/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.

MIDI & OSC Scripting

Preamble

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 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

The plug-in looks for MIDI & OSC scripts in standard locations. Scripts must have the file-name extension ".lua".

Mac OS X

The plug-in looks for scripts in

```
Library/Application Support/Expert Sleepers/Oomingmak/Scripts
```

Windows

The plug-in looks for scripts in

```
C:\Documents and Settings\<username>\Application Data\Expert
Sleepers\Oomingmak\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

1.0.0 26/3/2009

- First release.

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

Acknowledgements

The software described in this manual makes use of the following open source projects. The author is greatly indebted to them for their efforts and generosity.

Below are reproduced the various copyright notices and disclaimers that accompany these software projects, in accordance with their terms of use.

Lua

oscpack

oscpack -- Open Sound Control packet manipulation library
http://www.audiomulch.com/~rossb/code/oscpack

glew

PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS.

FreeType

The FreeType Project is copyright (C) 1996-2000 by David Turner, Robert Wilhelm, and Werner Lemberg.

THE FREETYPE PROJECT IS PROVIDED `AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE, OF THE FREETYPE PROJECT.

FTGL

Copyright (C) 2001-3 Henry Maddocks

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libpng

http://www.libpng.org/pub/png/libpng.html

zlib

http://www.zlib.net/