MFP (Music For Programmers)

A graphical patching language

Bill Gribble

grib@billgribble.com

May 10, 2013

Presented at the 2013 Linux Audio Conference (LAC-2013)

Institute of Electronic Music and Acoustics

University of Music and Performing Arts, Graz, Austria

What is MFP?

Context

- Graphical dataflow patching system inspired by Max/MSP and Pure Data
- Patches are diagrams with "boxes" (builtins, plugins, or extensions) and "connections" (carrying audio signals or data messages)
- "The diagram is the program" -- patches are computer programs, the tool is a sort of IDE

Purpose

- Build tools for audio analysis and synthesis
- Explore some concepts about programming languages
- Experiment with user interface and information display

Metadata

About MFP

Description:	Environment for building graphical "patches" (programs) with special support for real-time audio data
Similar to:	Pure Data, Max/MSP
Applications:	Synthesis, MIDI/OSC control, performance, engineering, algorithmic music, analysis...
Team:	Solo developer
Timeline:	2010-present
Status:	Pre-alpha/experimental, active development
User base:	0-10 bold pioneers
OS:	Linux
License:	GPL
Languages:	Python with C extensions
Supports:	JACK, NSM, LADSPA, MIDI, OSC, GTK+, Clutter
Source:	https://www.github.com/bgribble/mfp
Bug tracking:	https://www.github.com/bgribble/mfp/issues

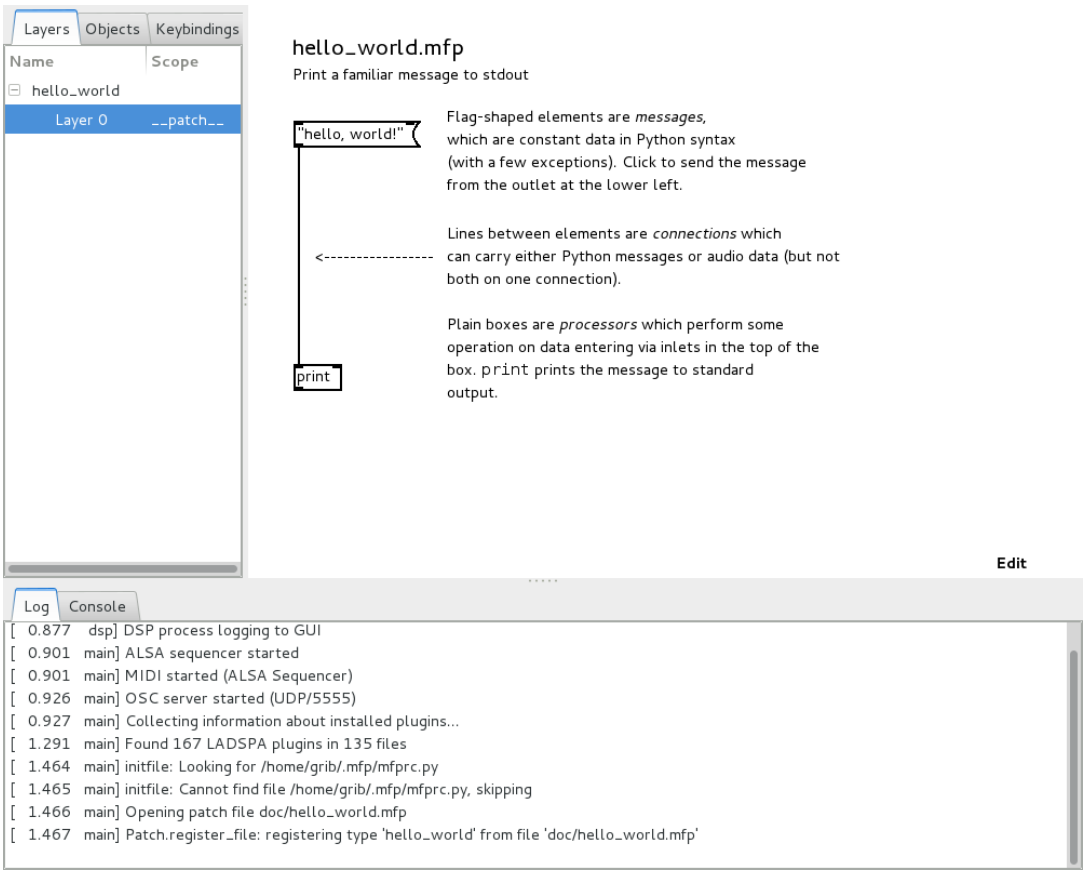
Structure of Talk

5 min:	Context and overview
25 min:	Examples and live-coding
10 min:	Q&A

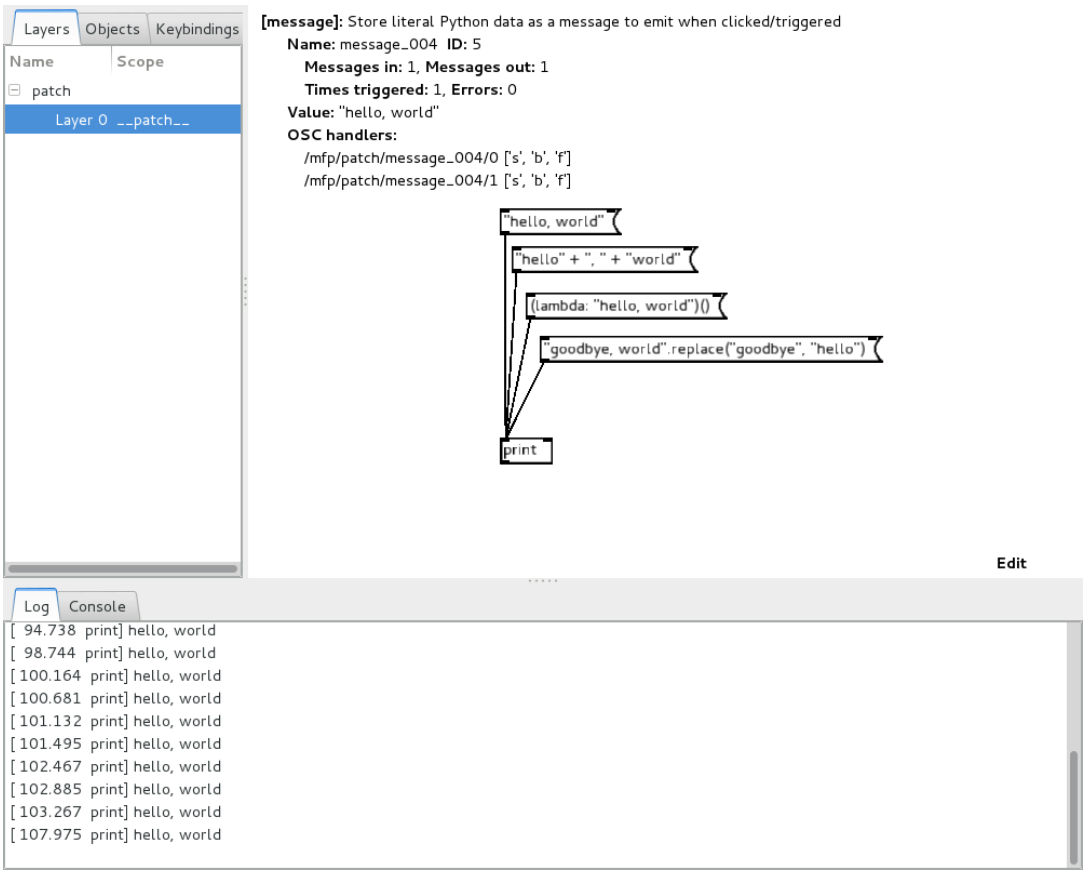
Example 1: Hello, World/Tour

- Work through Hello, World program and variants
- See basics of patch authoring and language features
- Have a tour of the MFP interface

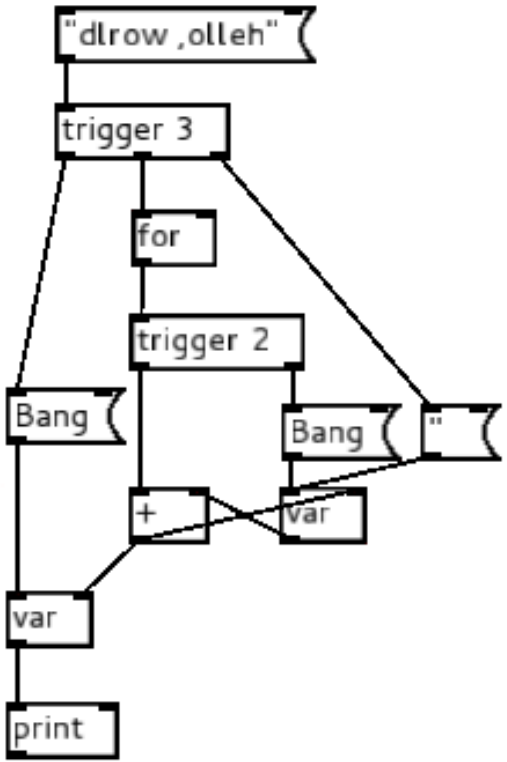
Hello, World: Basic program



Hello, World: Message boxes and expressions



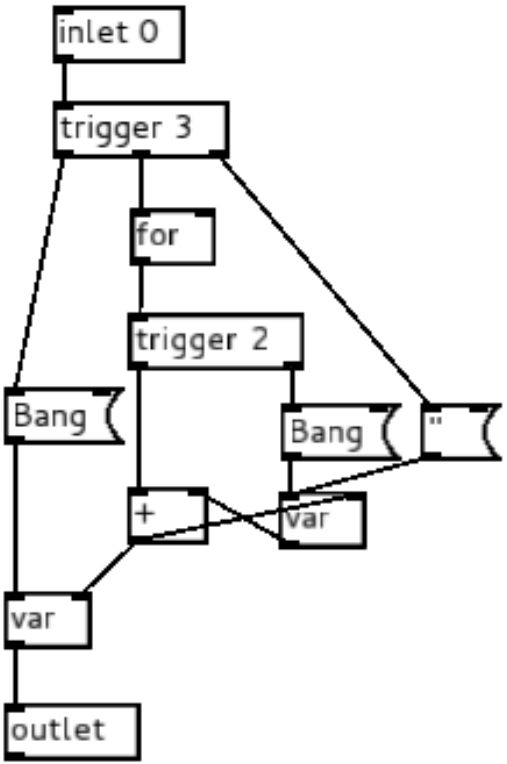
Hello, World: String reversal



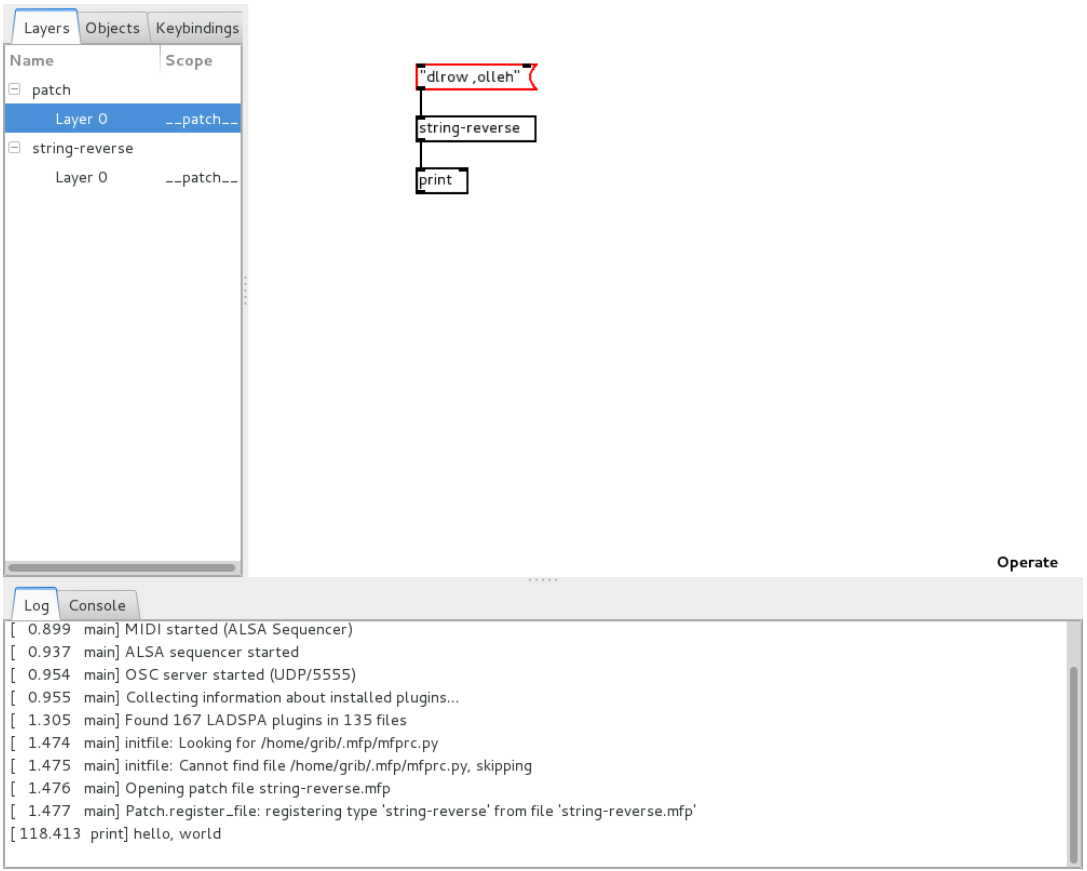
Hello, World: Basics of data flow

- "Hot" inlets trigger processing, other inputs are buffered
- Depth first (sequencing of steps)
- Right-to-left output order (sequencing of steps)
- Multiple connections on an outlet may be followed in any order

Hello, World: Saving and using a patch (1)



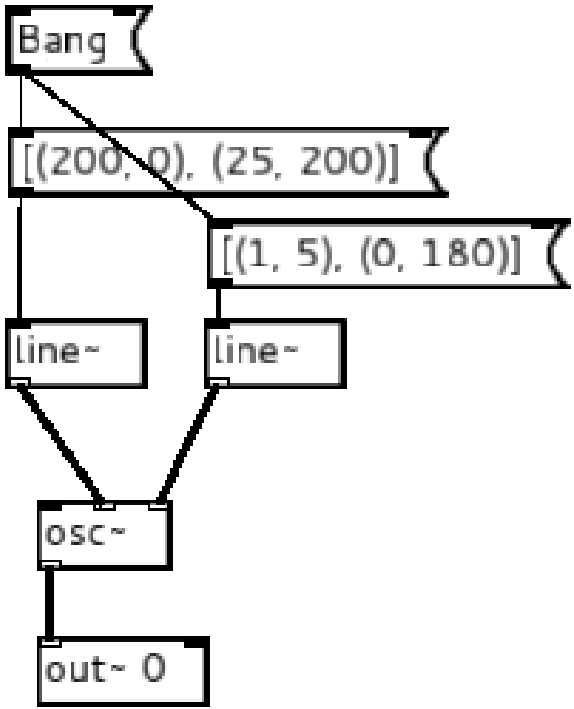
Hello, World: Saving and using a patch (2)



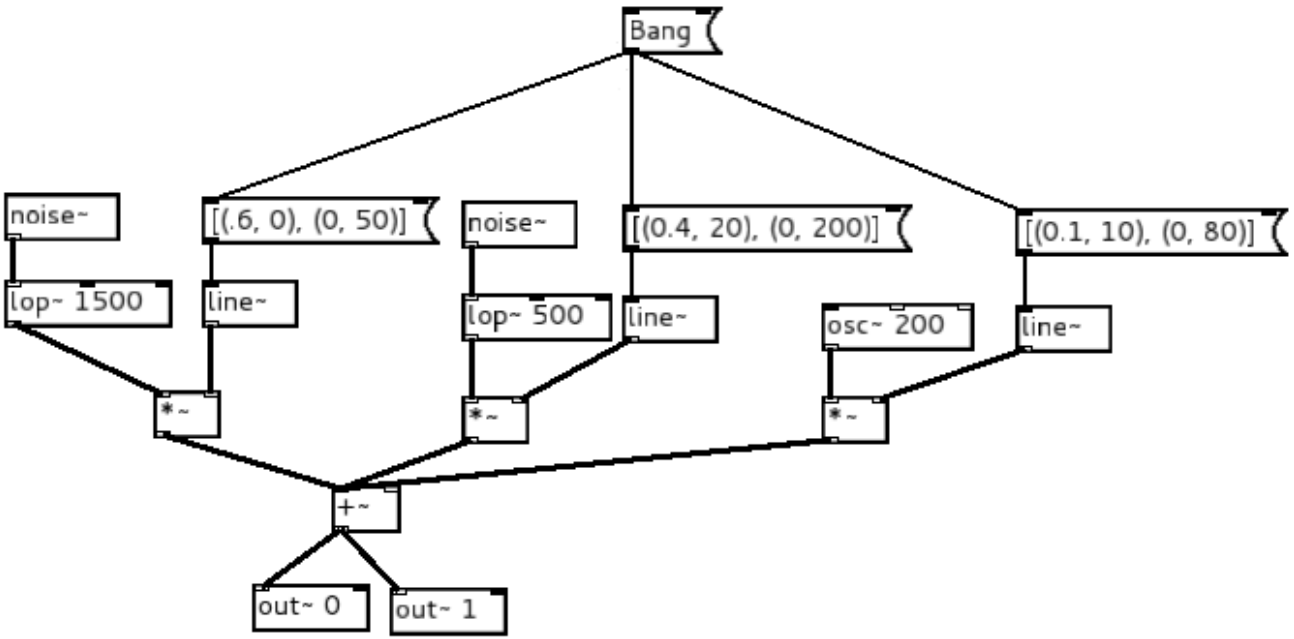
Example 2: Generating audio

- Show how signals and controls work together
- Create a simple kick and snare drum synth
- Use `[osc~]`, `[noise~]`, `[line~]`, `[lop~]`
- Connect it to external MIDI control

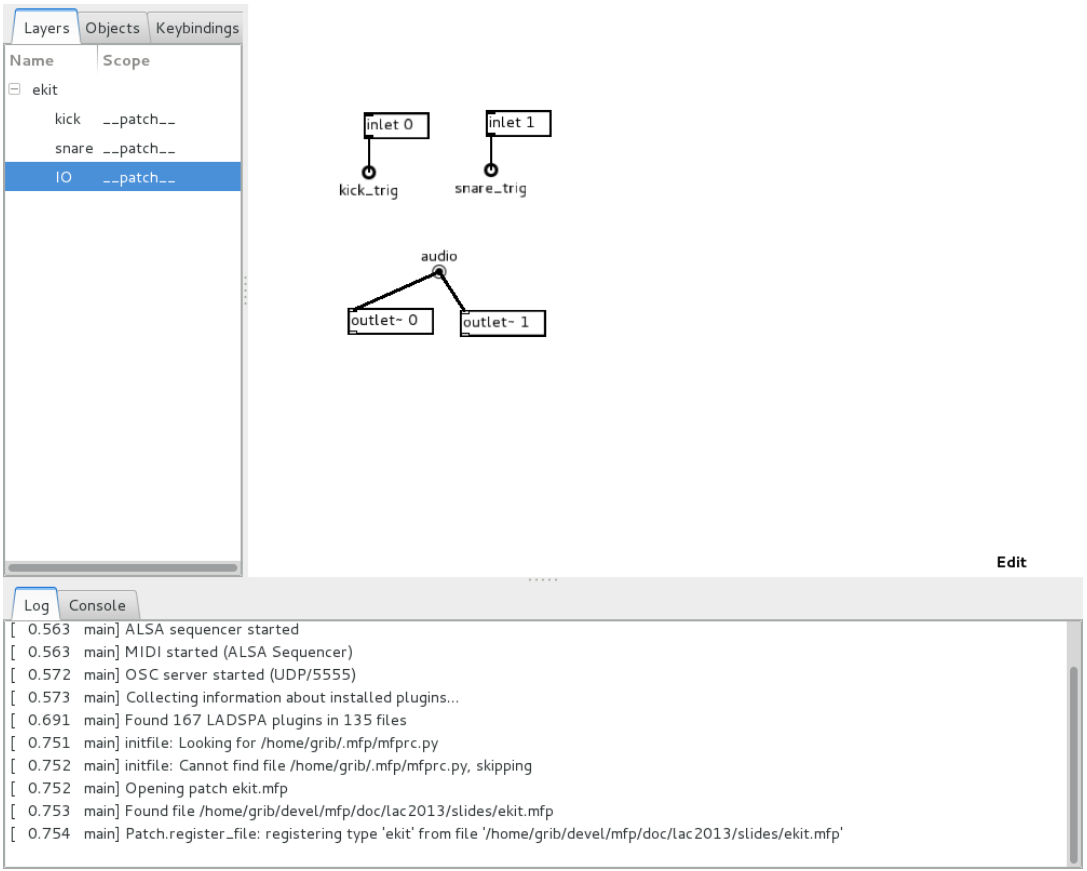
Generating audio: Simple kick drum



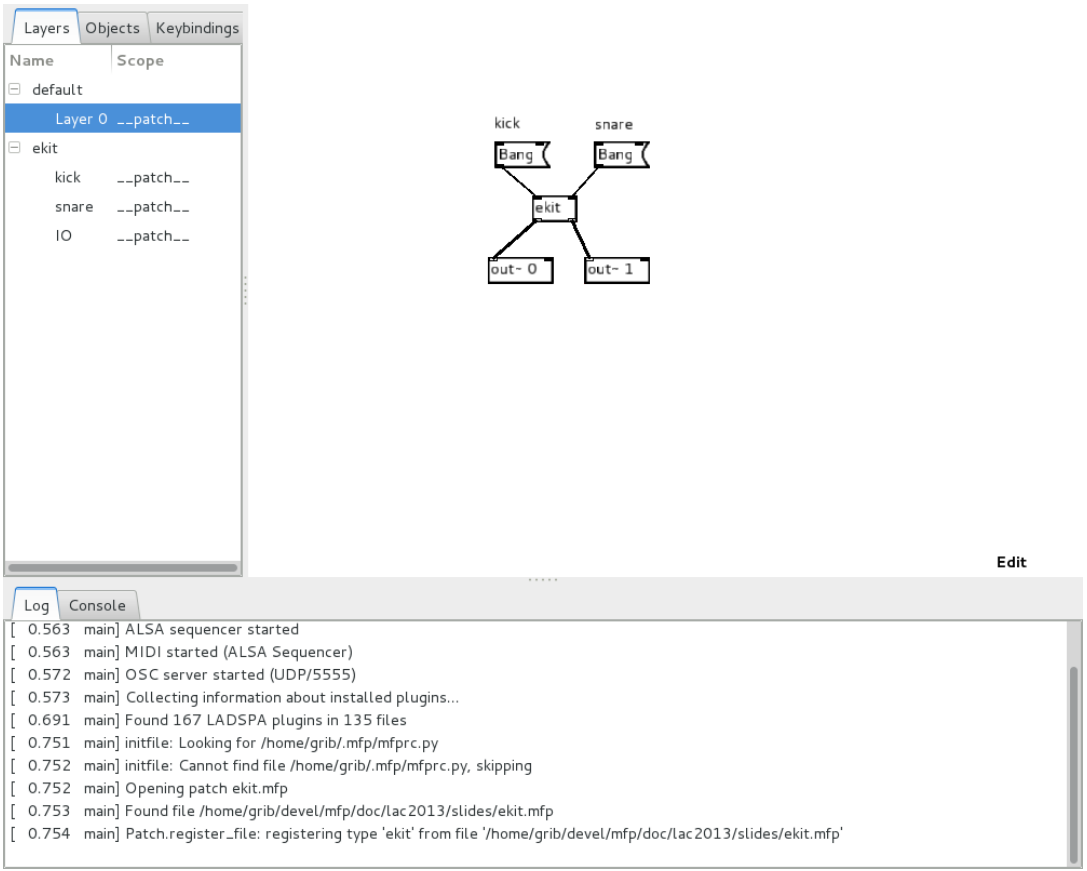
Generating audio: Simple snare drum



Generating audio: Drum kit patch (patch)



Generating audio: Drum kit patch (using)



Architecture

- Three processes connected with Python `multiprocessing`: GUI, engine, DSP
- DSP uses a C extension `mfpdsp` for all signal operations
- Only simple messages (float, array of float, string) transferred between engine and DSP

Future work

- Documentation and online help
- UI improvements: Undo/redo, click/drag to connect, file dialogs for load/save, menus
- Debugging tools: step execution, better error management
- Audio file handling: libsndfile for loading and saving samples and generated output
- Standards: JACK MIDI and transport, LV2 hosting, possible LV2 client mode, improved NSM support
- Bug fixes, test coverage, optimization...