

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

The screenshot displays the MFP interface with a patch named 'hello_world.mfp'. The patch is titled 'Print a familiar message to stdout'. It consists of a message box containing the text 'hello, world!' connected by a line to a 'print' object. The interface includes a 'Layers' panel on the left showing the patch structure, a 'Log' panel at the bottom showing system messages, and an 'Edit' button on the right.

hello_world.mfp
Print a familiar message to stdout

Flag-shaped elements are *messages*, which are constant data in Python syntax (with a few exceptions). Click to send the message from the outlet at the lower left.

Lines between elements are *connections* which can carry either Python messages or audio data (but not both on one connection).

Plain boxes are *processors* which perform some operation on data entering via inlets in the top of the box. `print` prints the message to standard output.

Log **Console**

```
[ 0.877 dsp] DSP process logging to GUI
[ 0.901 main] ALSA sequencer started
[ 0.901 main] MIDI started (ALSA Sequencer)
[ 0.926 main] OSC server started (UDP/5555)
[ 0.927 main] Collecting information about installed plugins...
[ 1.291 main] Found 167 LADSPA plugins in 135 files
[ 1.464 main] initfile: Looking for /home/grib/.mfp/mfprc.py
[ 1.465 main] initfile: Cannot find file /home/grib/.mfp/mfprc.py, skipping
[ 1.466 main] Opening patch file doc/hello_world.mfp
[ 1.467 main] Patch.register_file: registering type 'hello_world' from file 'doc/hello_world.mfp'
```

Hello, World: Message boxes and expressions

Layers Objects Keybindings

Name Scope

patch

Layer 0 __patch__

[message]: Store literal Python data as a message to emit when clicked/triggered

Name: message_004 **ID:** 5

Messages in: 1, **Messages out:** 1

Times triggered: 1, **Errors:** 0

Value: "hello, world"

OSC handlers:

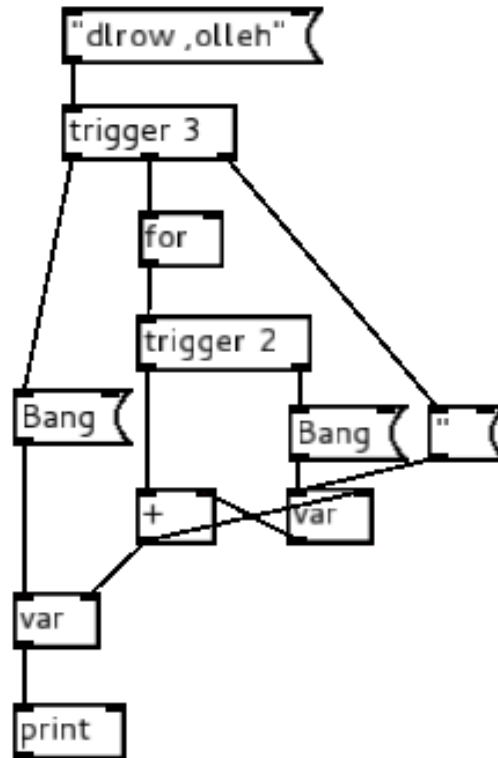
```
/mfp/patch/message_004/0 ['s', 'b', 'f']  
/mfp/patch/message_004/1 ['s', 'b', 'f']
```

Edit

Log Console

[94.738 print] hello, world
[98.744 print] hello, world
[100.164 print] hello, world
[100.681 print] hello, world
[101.132 print] hello, world
[101.495 print] hello, world
[102.467 print] hello, world
[102.885 print] hello, world
[103.267 print] hello, world
[107.975 print] hello, world

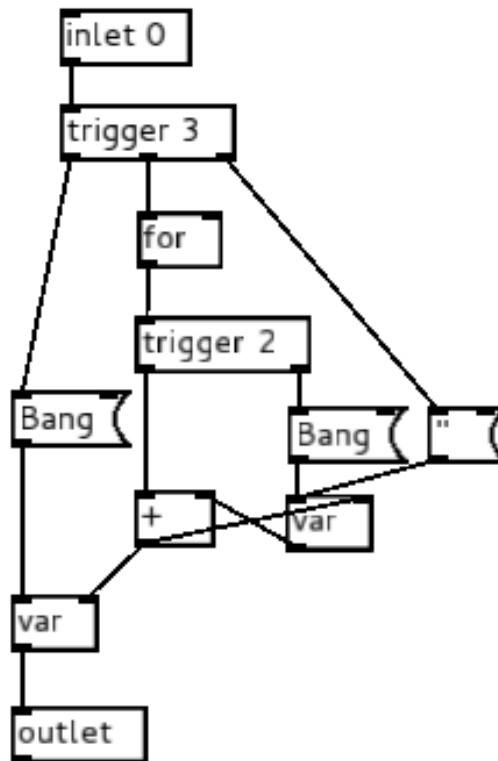
Hello, World: String reversal



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)

The screenshot shows a Pure Data patch window with the 'Layers' tab selected. The patch is named 'patch' and is located in the 'Scope' of 'Layer 0'. The patch itself is a 'string-reverse' patch, which is a sub-patch of 'Layer 0'. The patch is currently active, as indicated by the 'Layer 0' label in the console output.

The patch diagram shows a message box containing the string "dlrow,olleh", which is connected to a 'string-reverse' object, which is then connected to a 'print' object.

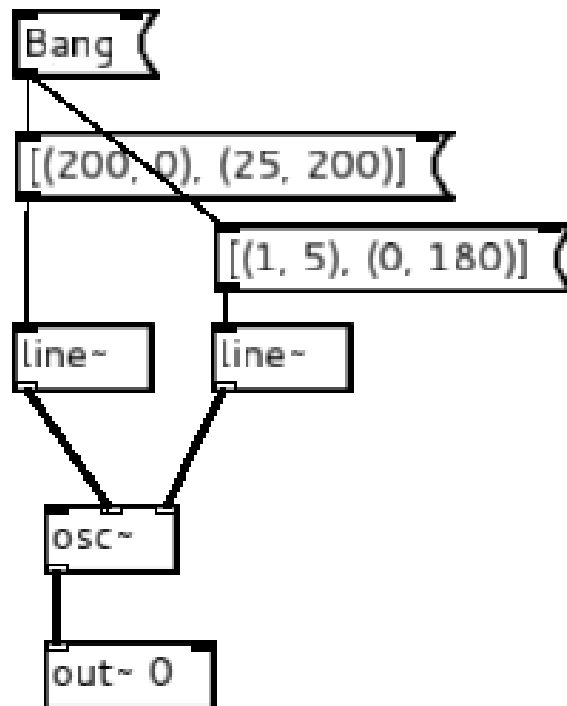
The console output shows the following messages:

```
[ 0.899 main] MIDI started (ALSA Sequencer)
[ 0.937 main] ALSA sequencer started
[ 0.954 main] OSC server started (UDP/5555)
[ 0.955 main] Collecting information about installed plugins...
[ 1.305 main] Found 167 LADSPA plugins in 135 files
[ 1.474 main] initfile: Looking for /home/grib/.mfp/mfprc.py
[ 1.475 main] initfile: Cannot find file /home/grib/.mfp/mfprc.py, skipping
[ 1.476 main] Opening patch file string-reverse.mfp
[ 1.477 main] Patch.register_file: registering type 'string-reverse' from file 'string-reverse.mfp'
[118.413 print] hello, world
```

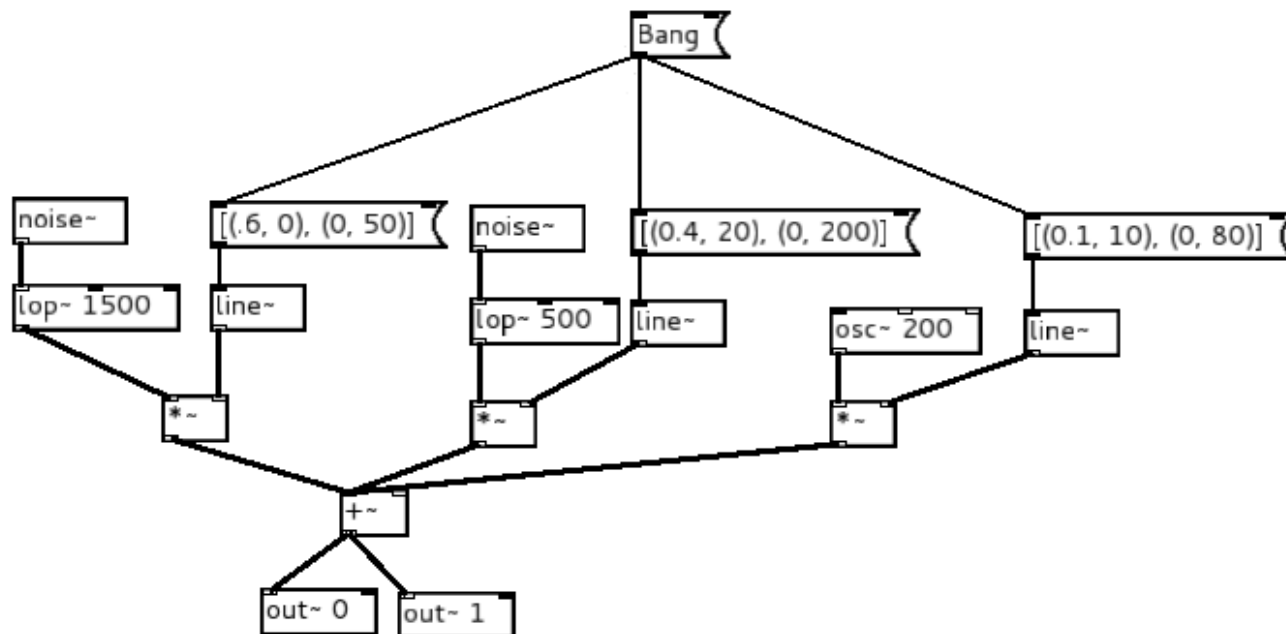
Example 2: Generating audio

- 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)

The screenshot displays a Pure Data patch window titled "ekit". The patch is organized into two main sections. The top section contains two inlet objects, "inlet 0" and "inlet 1", which are connected to "kick_trig" and "snare_trig" objects respectively. These objects are then connected to a central "audio" object. The bottom section contains two outlet objects, "outlet~ 0" and "outlet~ 1", which are connected to the "audio" object. The left sidebar shows a list of objects: "kick __patch__", "snare __patch__", and "IO __patch__". The bottom console shows a log of messages including ALSA sequencer and MIDI startup, OSC server startup, and plugin collection.

Layers Objects Keybindings

Name	Scope
ekit	
kick	__patch__
snare	__patch__
IO	__patch__

inlet 0 inlet 1

kick_trig snare_trig

audio

outlet~ 0 outlet~ 1

Edit

Log Console

```
[ 0.563 main] ALSA sequencer started
[ 0.563 main] MIDI started (ALSA Sequencer)
[ 0.572 main] OSC server started (UDP/5555)
[ 0.573 main] Collecting information about installed plugins...
[ 0.691 main] Found 167 LADSPA plugins in 135 files
[ 0.751 main] initfile: Looking for /home/grib/.mfp/mfprc.py
[ 0.752 main] initfile: Cannot find file /home/grib/.mfp/mfprc.py, skipping
[ 0.752 main] Opening patch ekit.mfp
[ 0.753 main] Found file /home/grib/devel/mfp/doc/lac2013/slides/ekit.mfp
[ 0.754 main] Patch.register_file: registering type 'ekit' from file '/home/grib/devel/mfp/doc/lac2013/slides/ekit.mfp'
```

Generating audio: Drum kit patch (using)

The screenshot displays a Pure Data patch window with the following components:

- Layers Panel:** Shows a hierarchy where 'Layer 0 __patch__' is selected, containing an 'ekit' sub-layer. The 'ekit' layer lists three objects: 'kick __patch__', 'snare __patch__', and 'IO __patch__'.
- Patch Diagram:** A visual representation of the patch. It features two 'Bang' objects at the top, labeled 'kick' and 'snare'. Arrows from these 'Bang' objects point to a central 'ekit' object. From the 'ekit' object, two arrows point downwards to 'out~ 0' and 'out~ 1' objects.
- Log Console:** A text area at the bottom showing the startup sequence of the patch. The log includes messages about ALSA sequencer and MIDI starting, OSC server starting, and the successful loading of the 'ekit.mfp' patch file.

```
graph TD
    kickBang[Bang] --> ekit[ekit]
    snareBang[Bang] --> ekit
    ekit --> out0[out~ 0]
    ekit --> out1[out~ 1]
```

```
[ 0.563 main] ALSA sequencer started
[ 0.563 main] MIDI started (ALSA Sequencer)
[ 0.572 main] OSC server started (UDP/5555)
[ 0.573 main] Collecting information about installed plugins...
[ 0.691 main] Found 167 LADSPA plugins in 135 files
[ 0.751 main] initfile: Looking for /home/grib/.mfp/mfprc.py
[ 0.752 main] initfile: Cannot find file /home/grib/.mfp/mfprc.py, skipping
[ 0.752 main] Opening patch ekit.mfp
[ 0.753 main] Found file /home/grib/devel/mfp/doc/lac2013/slides/ekit.mfp
[ 0.754 main] Patch.register_file: registering type 'ekit' from file '/home/grib/devel/mfp/doc/lac2013/slides/ekit.mfp'
```

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...