PPM-java

User guide

Introduction

ppm-java is a BBC Type II Peak Programme Meter indicating the level of a stereo audio signal. The meter has a short rise time (22 dB in 10 ms) and a long return time (24 dB in 2800 ms). Therefore it is more suitable to show the peak level of the incoming audio signal.

The audio level maps to the meter scale as:

Input level [dB]		Meter mark	
min	max	min	max
-130	-24	0	1
-24	-20	1	2
-20	-16	2	3
-16	-12	3	4
-12	-8	4	5
-8	-4	5	6
-4	0	6	7
0	...	7	7

System requirements

ppm-java requires

- Java (Tested with Java 1.8.)
- JackD audio server (Tested with version 1.9.10)

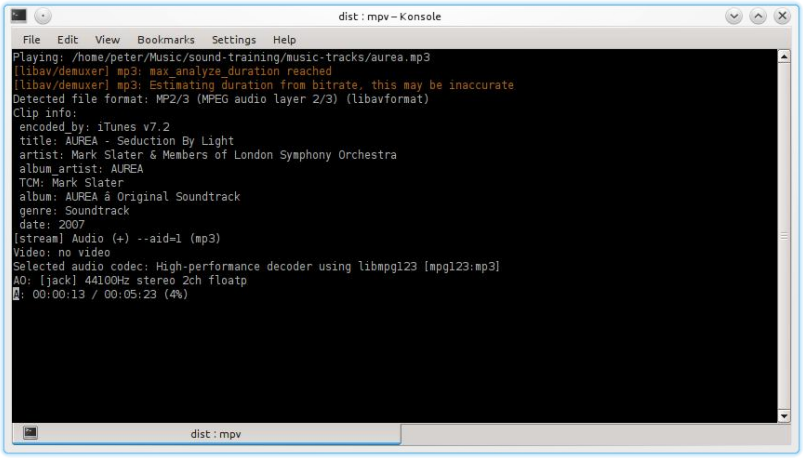
Test setup was a Linux machine (Kubuntu 14); program hasn't been tested on other platforms.

Running ppm-java

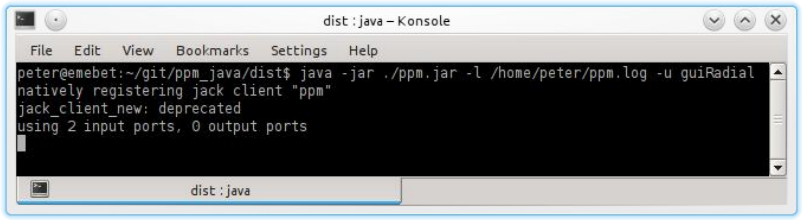
ppm-java requires a running instance of jackd.

Play any audio file with a media player that can use jackd. I used mpv:

```
mpv --ao=jack <file>
```

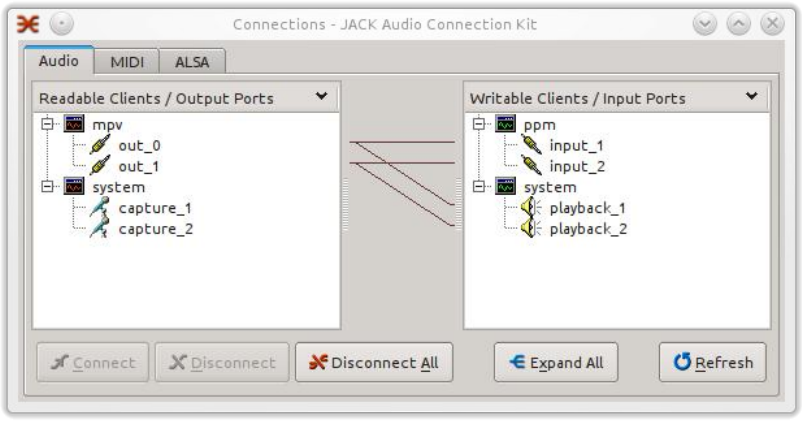


Start **ppm-java** with the **guiRadial** front end.



This will bring up the meter frontend showing the audio level from the media file played.

You need to be sure that the media player and the **ppm-java** meter are connected.



Front ends

ppm-java offers four different front ends (Commandline option **-u**). Two are using a graphical user interface, the others work inside the console from which you started **ppm-java**.

Most frontends feature peak indication on each channel. On a graphical frontend there's an LED, and on a commandline frontend the resp. indicator element changes color. The exception is the text streaming frontend; that one does no peak indication as it's up to the recipient to implement that.

Level	Indication	
Range	**Graphical**	**Commandline**
-130 dB .. -4 dB	No indication	No indication
-4 dB .. 0dB	LED yellow	Element orange
0dB ..	LED red	Element red

Graphical front ends

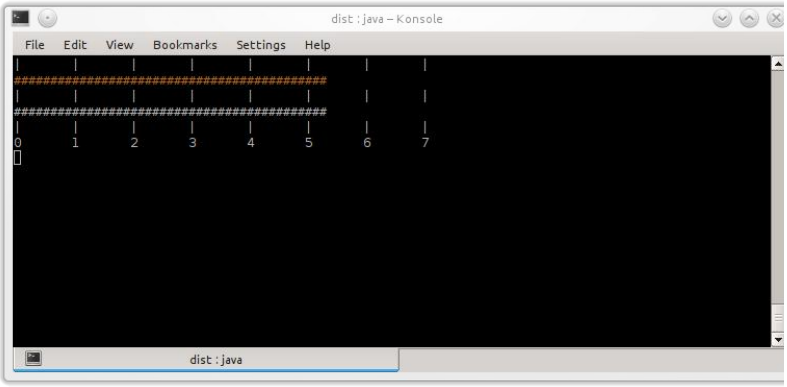
A PPM front end with a look resembling a physical PPM.

Note: Due to javax.swing internals on the test setup the meter needles didn't update smoothly. See section “Graphics issues” below for details.

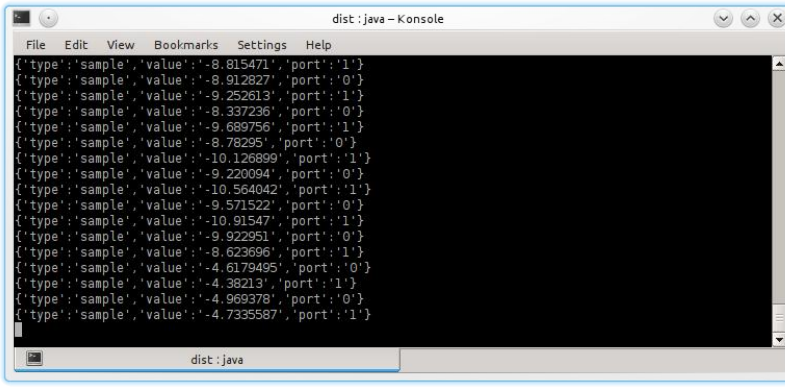
A horizontal LED bar graph display.

Commandline frontends

A horizontal bar graph.



Text stream. Useful to stream the data over a network (e.g. via a web socket). The data is streamed in JSON format. See section “Data streaming” for details.



Commandline options

```
usage: java -jar /path/to/ppm.jar
          [-h | --help]
          [-d | --debug]
          [-x | --dPPMProc]
          -l <path> | --logFile=<path>
          -u <ui_type> | --uiType=<ui_type>
          -w <n> | --consoleWidth=<n>
          [-x]
```

-u <ui_type> **--uiType=<ui_type>**	(required). The frontend to use; Can have the following values: **guiRadial**: Classic PPM look. Unfortunately, there are problems with the graphics. See section 'Issues'. **guiLinear**: Horizontal, linear bargraph. **consoleLinear**: Horizontal linear bargraph on the console. **consoleText**: Network data streaming. Will omit PPM ballistics if the option x is **not** used.
-w <n> **--consoleWidth=<n>**	(required, if **uiType** is **consoleLinear**). The width of the console based linear bargraph in characters (columns).
-l <path> **--logFile=<path>**	(required) The path to the log file.
-h | --help]	(optional) Prints the usage information onto the console and quits.
-d **--debug**	(optional) Shows the debug window. The program prints live runtime statistics onto the window. It's recommended to use this option when you use the PPM GUI frontend (See section

	“Issues”)
-x **--dPPMProc**	(optional) Use the (deprecated) monolithic PPM processor. This module does the entire PPM processing - from raw data to frontend (meter) value stream, incl. PPM ballistics.[1]

Data streaming

With the frontend set to consoleText the PPM streams all the peak values to stdout. In this mode the PPM does not implement any PPM integration, i.e. it just sends each peak value directly. This mode is useful if you want to employ the PPM program as capturing backend and send the data over a network connection. The usage scenario envisaged is a web application where the PPM runs on the server side and is invoked by the web server which then pipes the PPM data via a web socket to the client. The client has to implement the graphics and PPM ballistics itself. The streamed records have the following format:

- Info record

  ```
  {
      'type':'info'
      'value':<info_value>
  }
  ```

  Sends a piece of information to the client, so the client can adjust. Provides the following fields:

  - **type**: The type of record (i.e. info)
  - **value**: The associated value. Possible values:
    - **event_meter_start**: Notification - we are about to stream data.
    - **event_meter_stop**: Notification - we are about to stop streaming.

- Sample record

  ```
  {
      'type':'sample'
      'value':<float_value>
      'port':<int_value>
  }
  ```

  A peak sample. This is the actual data for the GUI meter. We only send peak values, no interpolated values (i.e. data contains no PPM ballistics); the client has to implement all the presentation specific behaviour. Provides the following fields:

  - **type**: The type of record (i.e. info)

1 Now replaced by a more scalable, modular framework.

- **`value`**: The peak value (float)
- port: The port (left channel or right channel) for which the data is destined. **`0`** (zero) means left channel, **`1`** means right channel.

- Null record

```
{
    'type':'null'
}
```

Sent if an internal, transient error occured on the backend side. Just here for completeness. In most cases, clients can ignore this.

Issues

- There's a problem when displaying the PPM frontend (-u guiRadial) in that the meter updates are rather jerky. As a workaround additionally specify the -d commandline option which brings up the debug window. Each time it's being updated it seems to force a meter redrawing as well, with the result that the meter updates smoothly. This issue is probably due to the way the underlying javax.swing framework operates.
- The meter is likely to miss true peaks (i.e. peaks in the waveform) as it only catches sample peaks which may fall between waveform peaks. This results in an under-read of approx. 3dB.[2]
- There is an imbalance in processed frame size. Approx. every second cycle there are many samples, and the next cycle uses those samples that have been missed in the previous one. During that cycle the meter still gets the peak, albeit from a smaller sample base. Visually, the meter doesn't show any anomaly; it does not seem to be an issue.[3]

2 In a future version it's planned to include an oversampling low pass filter which makes the reading more accurate.

3 In a future version we'll implement some sort of balancing system that ensures all frames get an approx. equal amount of data to work with (tech. detail: Each cycle we use a moving average of samples per cycle).