PixelController

PixelController - a matrix control project by Michael Vogt , (c) 2010-2014. The main goal of this application is to create an easy to use matrix controller software which creates stunning visuals!

Primary Website: http://www.pixelinvaders.ch

My Blog: http://www.neophob.com

Facebook: https://www.facebook.com/PixelInvaders

You can **download** PixelController on Google Code: http://code.google.com/p/pixelcontroller/downloads/

You can download the latest **SNAPSHOT** (=unstable) release of PixelController on the Jenkins Server

Demo

Check out PixelController Rough Cut #2. Featuring two PixelInvaders panels, PixelInvaders 3D RGB Panels and PixelInvaders panels controlled by a tablet (OSC) to see PixelController in action with two PixelInvaders panels. This should give you a quick overview what PixelController is.

How to use PixelController

Prerequisite:

- Java Runtime, v1.6+

Basic usage

You can start PixelController with the integrated GUI by double click on the file `PixelController.jar` .

By default PixelController use **a dummy** output device (= no configured LED Matrix). To change that open the `data/config.properties` configuration file and make the necessary changes, lines starting with # are ignored. The most important parts are:

```
output.resolution.x=8
output.resolution.y=8
```

which defines the resolution of your matrix. Next you need to define one or multiple Output devices, for example id you use two PixelInvaders panels (while the output for the second panel is rotated by 180 degrees) use:

```
pixelinvaders.layout.row1=NO_ROTATE,ROTATE_180
#pixelinvaders.layout.row2=NO_ROTATE,NO_ROTATE
```

Take a look at the config file, there are a lot of hints how to configure PixelController. You can start PixelController without touching the configuration file, by default the null output device is enabled.

You can put your own **image files** to the `data/pics` directory or delete existing images. Make sure your image fits your output resolution.

Main idea

The main idea of PixelController is, that you can create an nice looking Visual on your matrix by selecting the right "elements". The "elements" of a Visual are

- two **Generators** (create the content)
- two **Effects** (modify the content)
- a **Mixer** (mix the content)
- a **Colorset** (define the look of the content).

I try to visualize it:

```
[GENERATOR A] ---> [EFFECT A] ---> [MIXER] <--- [EFFECT B] <--- [GENERATOR B]
                                      |
                                 [Colorset]
                                      |
                                  [VISUAL]
```

A Visual can be assigned to one or more Output LED Matrices.

Exception: There are two exceptions, if you use the Capture generator or use the OSC Generator (that sends 24bpp data) PixelController switch to the **Pass-though mode**. This means no Colorset, Effect and Mixer can be used and the original input data is send to the panels.

Per default PixelController creates one Visual more than the number of connected Output devices. This allows you to play with a Visual that is not assigned to an Output (non-visible), that can be displayed later. All Visuals can be stored (and of course loaded) to a preset file.

Advanced usage

You can run the PixelController daemon without frontend, for example on a Raspberry Pi. Then you start the PixelController frontend on your computer, which will connect to the PixelController daemon.

It's important to know that the **configuration** of PixelController is made where the **PixelController deamon** is running.

Details

Start the **PixelController daemon** by execute `pixConServer/PixelController.sh` on Linux/OSX, `pixConServer/PixelControllerRPi.sh` on Raspberry Pi and `pixConServer\PixelController.cmd` on Windows. The PixelController daemon generate the visuals and send them to the Output device. PixelController create a Bonjour/Zeroconf service you can discover or simply ping at `PixelController.local` . You can control PixelController server by

- using the **PixelController frontend**/remote client, start it with `pixConClient/PixelControllerClient.jar` . The PixelController frontend will detect the PixelController daemon automatically via Bonjour/Zeroconf.
- using the PixelController command line tool by execute `pixConClient/PixConCli.sh` on Linux/OSX or `pixConClient\PixConCli.cmd`
- any other OSC client, see chapter **OSC Clients to control PixelController**

If Bonjour/Zeroconf does not work on your network you can edit the PixelController frontend configuration file `./pixConClient/clientRemote.properties` and add the IP address of the PixelController daemon.

Supported Hardware

PixelController supports different (LED) matrix hardware devices/controller:

- PixelInvaders 3D Panels serial device **(*1)**
- PixelInvaders 3D Panels network device **(*1)**
- Seeedstudios Rainbowduino V2 (see Readme.rainbowduinoV2)
- Seeedstudios Rainbowduino V3 (Using this firmware: https://code.google.com/p/rainbowduino-v3-streaming-firmware)
- ArtNet Devices, multiple universe are supported,510 Channels (170 RGB Pixels) per universe
- MiniDmx Devices (like the SEDU board of http://www.led-studien.de)
- Generic UDP Devices (for example Raspberry Pi, check out the PixelPi Software)
- TPM2 Serial devices (see http://www.led-studien.de for more information)
- TPM2 Net devices (see http://www.led-studien.de for more information)
- E1.31 devices (see http://www.opendmx.net/index.php/E1.31)
- RPi (Raspberry Pi) SPI controlled WS2801 LED pixels, see http://www.pixelinvaders.ch

Check out the `integration/ArduinoFw` directory, all Arduino based firmware files are stored there.

(*1): I sell PixelInvaders 3d panels as a DIY kit, see http://shop.pixelinvaders.ch for more details (and if you want to support PixelController).

[Arduino] Which firmware should I use?

If you don't have a hardware controller (like ArtNet or E1.31) and would like to use an Arduino/Teensy microcontroller you can choose between different firmwares.

- If you bought a PixelInvaders DIY Kit, use the `integration/ArduinoFw/pixelinvaders/neoLedLPD6803Spi` firmware
- If you want to create a ONE panel matrix with an arbitrary resolution, use the `integration/ArduinoFw/tpm2serial` firmware
- If you want to create multiple 8x8 panels, use the `integration/ArduinoFw/pixelinvaders/neoLedWS2801Spi` firmware. This firmware is based on the fastspi2 library and can handle different led chips.

I recommend a Teensy 2.0 microcontroller, as some Arduino boards suffer from bad serial latency (especially the Arduino UNO r3). You need to install the Arduino IDE, see the "Getting started with Arduino" (http://arduino.cc/en/Guide/HomePage) Tutorial.

You need to know how to install an Arduino Library (http://arduino.cc/en/Guide/Libraries). For PixelInvaders Panels (LPD6803) install the `integration/ArduinoFw/libraries/timer1` and `integration/ArduinoFw/libraries/neophob_lpd6803spi` libraries, for other panels (WS2801, WS281x...) install the `integration/ArduinoFw/libraries/FastSPI_LED2` library.

How does it work?

PixelController generates the content for the LED matrix and sends the data out to the controller. The controller then handle the LED module update (which depends on the used LED modules). There are two options to send the data to the controller: * sends the data via USB to the Arduino/Teensy board aka. DIY LED controller. * sends the data via ethernet to a PixelInvaders/E1.31/ArtNet... device.

Here are some primitive schemes how the visual is send to the LED matrix:

```
[PixelController]---<USB>---[Teensy with PixelInvaders firmware]---<SPI>---[LED#1]---[LED#2]...

[PixelController]---<USB>---[Teensy with TPM2 firmware using fastspi2 lib]---<SPI>---[LED#1]---[LED#2]...

[PixelController]---<Ethernet>---[Artnet Controller]---<SPI>---[LED#1]---[LED#2]...

[PixelController]---<SPI on RPi>---[LED#1]---[LED#2]...
```

Advanced PixelController configuration

There are a lot of options in the `config.properties` file, here I describe some of them.

PixelController updates all Visuals depending on the **sound input**. If a beat is detected, the Visuals are updated faster. You can disable this behavior by setting this option:

```
#=========================
#enable pixelcontroller sound analyzer (disable it if you don't have a sound card)
#=========================
sound.analyze.enabled=true
```

There is a Generator called "**Screen Capture**" which is disabled by default. If you want to enable this generator, edit the following settings:

```
#x/y offset for screen capturing generator
#if you define screen.capture.window.size.x as 0, the screen capture generator will be disabled
screen.capture.offset=100
screen.capture.window.size.x=500
screen.capture.window.size.y=300
```

This enables the Screen Capture Generator which captures a region of 500 x 300 pixels. Potential use cases for this Generator are: YouTube videos, other movie players...

Or you can start PixelController in the **random mode**, where PixelController changes the Visuals randomly:

```
#=========================
#start in random mode?
#=========================
startup.in.randommode=false
```

If the PixelController is running in **random mode** you can define a lifetime of the random pattern. If the lifetime is over a new random pattern is generated. If the lifetime is configured as 0, the random pattern will change only if an audio input is used and a beat is detected.

```
#=========================
#if the random mode is enabled, create a random visual each n seconds
#if this value is 0, then this feature is disabled
#=========================
randommode.lifetime.in.s=5
```

Or you can save a preset and load that one per default if you start PixelController (per default, preset 0 will be loaded)

```
#=========================
#load a preset if PixelController starts, default is preset nr 0
#=========================
#startup.load.preset.nr=0
```

You can define the **size of the PixelController GUI**, for example the size of the simulated LED Matrix (which is per default 16 pixels):

```
#=========================
#GUI: the size of the software output matrix
#=========================
led.pixel.size=16
```

Define the **listening port** of PixelController. This port is used if you want to send data or image content via OSC to the PixelController daemon. This port is also used if you want to connect from the PixelController frontend.

```
#=========================
#network port config
#=========================
osc.listening.port=9876
```

Or define the window size, depending on this setting, the Visuals are displayed larger or smaller.

```
#=========================
#define the maximal window size (control window)
#=========================
gui.window.maximal.width=820
gui.window.maximal.height=600
```

You can define **your own Colorsets**, they are defined in the file `data/palette.properties`. A Colorset definition consists of a name and multiple RGB color values. Here is an example:

```
MiamiVice=0x1be3ff, 0xff82dc, 0xffffff
```

There are more options in the config file, take a look - each option is documented in the config file.

Frontends

There are different frontends for PixelController (besides the GUI frontend). It doesn't matter how you control PixelController - you have the same functions. See chapter **OSC Messages** to get an overview of all available commands.

- **PixConCli**: Command Line Interface for PixelController, works also remote. The CLI tool is called `PixConCli.cmd` on Windows and `PixConCli.sh` on Linux/OSX.
- **OSC**: The OSC interface of PixelController is listening (by default) on port 9876. Processing examples are included in the `integration/Processing` directory. You can send messages to control PixelController or you can send image content via OSC to PixelController (using the OSC Generator's).
- **TouchOSC**: This mobile application can be used to controler PixelController. see `integration/TouchOsc` for ready to use layouts.

PixConCli Examples

You can send OSC messages to PixelController to control the software. PixelController includes a simple CLI tool to control the software by console. Start PixelController, then open the console:

Randomize current Visual

```
    # ./PixConCli.sh -c RANDOMIZE
```

Select Image Generator as Generator A (0 is Passthru, 1 is Blinkenlights...) for current Visual:

```
    # ./PixConCli.sh -c CHANGE_GENERATOR_A 2
```

Load image gradient.jpg (the image must be preset in the `data/pics` directory, where PixelController is running):

```
    # ./PixConCli.sh -c IMAGE gradient.jpg
```

Processing Examples

I included some Processing example Sketches. Maybe you can use one or several of those examples for your need. Those file should help you integrate PixelController into your environment. You need the OscP5 Processing library and ControlP5 Processing library to run the examples.

- `OscSendImageData/OscSenderRandom` : Randomize Visual 1 four times per second
- `OscSendImageData/OscSenderSelectVisual` : Change Generator, Effect and Mixer of Visual 0
- `OscSendImageData/PixelControllerRemote` : PixelController remote application to load presets via GUI
- `OscSendControllMessages/colorAdjust` : Sketch to adjust RGB values for multiple panels
- `OscSendControllMessages/kinect_MSAFluidDemo` : An example Sketch to use a MS Kinect together with PixelController
- `OscSendControllMessages/particleexample` : Particlesystem that can be controlled with your mouse
- `OscSendControllMessages/PixlInvCamAndroid` : Android application, use the camera of your Android phone and send a live video stream to PixelController
- `OscSendControllMessages/sendImageKreise` : An example of OpenProcessing used in PixelController
- `OscSendControllMessages/sendImageKreise24bpp` : The same example as above, but use the color settings of the Sketch, use the PixelController **pass through** mode where no effect, mixer and colorsets can be used.
- `OscSendControllMessages/sendImageSecretLifeOfTuring` : Another great animation of OpenProcessing
- `AdapterApp` : A more advanced Pixelcontroller example I used for an installation. Select a random preset after a specified time.

PureData Examples

PureData is a visual language, included are some examples.

- `PureData/ledgui5-onePanel.pd` : The old PixelController GUI, use it to create a frontend for your case...
- `PureData/ledgui5.pd` : The old PixelController GUI, use it to create a frontend for your case...
- `PureData/Midi2OSC.pd` : MIDI to OSC bridge - control PixelController with a MIDI device

More hints

Run PixelController on a RPi

As the RPi isn't the beefiest CPU (and PixelController doesn't use the GPU) it's not really practical to run it with the graphical frontend. But you can run the daemon version of PixelController (see chapter **Advanced usage**). You need to run PixelController **as root user** if you want to access the USB serial port (or open the /var/lock directory for the running user) or if you use the RPi SPI interface.

You need a Java Runtime on your RPi - I recomment using the Oracle Java as it's performance is much better than OpenJDK. You can install it by executing `# sudo apt-get install oracle-java7-jdk`.

The RPi has no **audio input** onboard, you must connect an USB audio card/USB microphone or a Webcam with a microphone. Use `sudo alsamixer --card 1` to verify the input volume is set correct. Use `sudo alsactl store` to save your current settings.

Make sure your RPi is up to date - run `#sudo apt-get update && sudo apt-get upgrade` and `sudo rpi-update`. If you want to enable the **SPI device**, make sure it's not blacklisted, see `/etc/modprobe.d/raspi-blacklist.conf` and remove the `spi-bcm2708` entry. After a reboot the SPI device should be visible:

```
pi@raspberrypi ~ $ ls -al /dev/spi*
crw------- 1 root root 153, 0 Jan  1  1970 /dev/spidev0.0
crw------- 1 root root 153, 1 Jan  1  1970 /dev/spidev0.1
```

See `integration/RPi-Startscript` for an example init.d startscript. Make sure to edit the `application_dir` variable in the startscript (default value is `/home/pi/pixcon`) and copy it to `/etc/init.d/`.

Non-rectangular LED matrix

If you have a non-rectangular LED matrix you want to use with PixelController you can use the custom mapping feature called `output.mapping`. While it's position define the target offset, the nr define the source, example:

```
output.mapping=5,8,2,...
```

This means the first physical pixel gets the content of 5th pixel of the visual, the second physical pixel gets the content of the 8th visual pixel, the third physical pixel gets the content of the 2nd visual pixel and so on.

For example you wired up this Christmas tree (Matrix resolution 9x12):

```
-- -- -- -- XX -- -- -- --         -- -- -- -- 01 -- -- -- --         -- -- -- -- 04 -- -- -- --
-- -- -- XX XX XX -- -- --         -- -- -- 02 03 04 -- -- --         -- -- -- 12 13 14 -- -- -- >> (direction)
-- -- XX XX XX XX XX -- --         -- -- 09 08 07 06 05 -- --         -- -- 20 21 22 23 24 -- -- <<
-- -- -- XX XX XX -- -- --         -- -- -- 10 11 12 -- -- --         -- -- -- 30 31 32 -- -- -- >>
-- -- XX XX XX XX XX -- --         -- -- 17 16 15 14 13 -- --         -- -- 38 39 40 41 42 -- -- <<
-- XX XX XX XX XX XX XX --         -- 18 19 20 21 22 23 24 --         -- 46 47 48 49 50 51 52 -- >>
-- -- XX XX XX XX XX -- --         -- -- 29 28 27 26 25 -- --         -- -- XX XX XX XX .. -- -- <<
-- XX XX XX XX XX XX XX --         -- 30 31 32 33 34 35 36 --         -- XX XX XX XX XX XX XX --
XX XX XX XX XX XX XX XX XX         45 44 43 42 41 40 39 38 37         XX XX XX XX XX XX XX XX XX
-- -- -- XX XX XX -- -- --         -- -- -- 46 47 48 -- -- --         -- -- -- XX XX XX -- -- --
-- -- -- -- XX -- -- -- --         -- -- -- -- 49 -- -- -- --         -- -- -- -- XX -- -- -- --
-- -- -- -- XX -- -- -- --         -- -- -- -- 50 -- -- -- --         -- -- -- -- XX -- -- -- --
     1) led position                    2) wiring order                    3) output.mapping
```

To create a valid mapping use this config:

```
output.mapping=4, 12,13,14, 24,23,22,21,20, 30,31,32, 42,41,40,39,38, 46,47,48,49,50,51,52 ...
```

With this feature you can use all kinds of matrices, for example a circle matrix.

To make the mapping process easier you can use the **online mapping tool** at http://pixelinvaders.ch/pixelcontroller/. First use the keys `a` / `y` and `s` / `x` to change the matrix size, then click the corresponding pixel order. Selected pixels turns yellow. The debug output on the lower end print out the pixel order you can copy and past into the PixelController config file.

How to use PixelInvaders.net

PixelController allows you to network enable the PixelInvaders panels. See my Blog post for details. Here is the Quick Guide:

- Install ser2net on your RPi
- configure ser2net: `5333:raw:500:/dev/ttyACM0:115200 8DATABITS NONE 1STOPBIT`
- connect the Teensy board via USB to the RPi
- start the ser2net daemon
- configure the PixelInvaders.net IP address in the `config.properties` file
- have fun

Send Image data via OSC to PixelController

You can control PixelController remotely by sending OSC Messages. But you can also send image data to PixelController via OSC (`/OSC_GENERATOR1` and `/OSC_GENERATOR2`).

First you need to find out the resolution for your Output device. Start PixelController and switch to the INFO tab. Search for the `Internal Buffersize` setting, this is the internal resolution. Now you have two options

- send 8bpp (greyscale) image data to PixelController (Resolution X * Resolution Y * 1bpp). You can use Effects, Mixer and Colorsets.
- send 24bpp image data to PixelController (Resolution X * Resolution Y * 3bpp). PixelController activates the pass through mode and Effects, Mixer and Colorsets cannot be used.

See the Processing examples `OscSendControllMessages/sendImageKreise` and `OscSendControllMessages/sendImageKreise24bpp`.

OSC Clients to control PixelController

OSC is a generic protocol that can be used to control PixelController.

If you want to use the **console** to control PixelController you can use tools like `liblo-tools`. An example to freeze the update would be `/usr/bin/oscsend 192.168.1.1 9876 /FREEZE`.

The included PixelController PixConCli tool can also send OSC messages - but are slower to execute and require an installed JVM - which is not always feasible.

If you prefer to use your mobile/tablet to control PixelController you can use the TouchOSC app, see the `integration/TouchOsc/` directory for pre made layouts. I've also included examples for the free Control OSC example in the `integration/ControlOSC/` directory.

- any other OSC Client

Create Blinkenlights movie files

PixelController can play Blinkenlight movie files (BML). BML was created by the CCC and it's a simple XML based file format. Links that help you create Blinkenlights files:

- Blinkenlights Dev Tools
- BLIMP - Blinkenlights Interactive Movie Program
- Image to BML Converter
- 162 Blinkenlights Movie files
- bmconv, GIF to BML Converter **German**
- Processing library to create Blinkenlights movie files

A note about BLIMP, if you save a file, make sure to add the ".bml" file extension, else it may fail.

To create a bml file out of an animated gif file, download and install blib and blinkentools (see Blinkenlights Dev Tools link). Example how to convert an animated gif to bml: `./b2b -o _NAMEIT_ -t bml -b 4 INPUT.gif`

Make sure you create blinkenlights files using 4 or 8 bits, using 1 channel (3 channels are not supported!).

Beat Workmode

The **Beat Workmode** define how the *sound event detection* (beat/hat/snare detection) should influence the Generator speed (=speed of the Visual). Currently three modes are implemented:

- **Linear**: Sound doesn't matter, the Generator speed is just, ermm, linear.
- **Heavy**: If no sound event was detected, the Visual stands still.
- **Moderate**: A mix between Linear and Heavy. There is a minimal Generator speed that will increase if a sound event was detected. This mode was used for all pre v2.0.0 releases.

Colorsets

The Colorsets is responsible to colorize the Visual:

```
[MIXER OUTPUT // 8bpp] --> [COLORSET] --> [VISUAL OUTPUT // 24 pp]
```

Each pixel before the transformation has a value from 0..255 (8 bit or greyscale) that will mapped to a color in the Colorset. The Colorset is also responsible to create a smooth transition between two colors (fade). Let's use a simple example to illustrate this:

```
BlackWhite=0x000000,0xffffff
```

The mapping looks like this:

- Pixel value 0 is mapped to color 0x000000
- Pixel value 1 is mapped to color 0x020202
- ...
- Pixel value 127 is mapped to color 0xffffff
- ...
- Pixel value 255 is mapped to color 0x000000

The more colors a Colorset has, the faster it changes the color.

OSC Messages

Here are all commands PixelController knows.

```
    CHANGE_GENERATOR_A          # of parameters: 1     <INT> change first generator for current visual
    CHANGE_GENERATOR_B          # of parameters: 1     <INT> change first generator for current visual
    CHANGE_EFFECT_A             # of parameters: 1     <INT> change first effect for current visual
    CHANGE_EFFECT_B             # of parameters: 1     <INT> change second effect for current visual
    CHANGE_MIXER                # of parameters: 1     <INT> change mixer for current visual
    CURRENT_VISUAL              # of parameters: 1     <INT> select actual visual
    CURRENT_COLORSET            # of parameters: 1     <INT> select actual ColorSet

    CHANGE_OUTPUT_VISUAL        # of parameters: 1     <INT> change visual for current output
    CHANGE_OUTPUT_FADER         # of parameters: 1     <INT> change fader for current output
    CHANGE_ALL_OUTPUT_VISUAL    # of parameters: 1     <INT> change visual for all outputs
    CHANGE_ALL_OUTPUT_FADER     # of parameters: 1     <INT> change fader for all outputs
    CURRENT_OUTPUT              # of parameters: 1     <INT> select current output

    BLINKEN                     # of parameters: 1     <STRING> file to load for the blinkenlights generator
    IMAGE                       # of parameters: 1     <STRING> image to load for the simple image generator
    TEXTDEF                     # of parameters: 1     <INT> select texture deformation option, 1-11
    ZOOMOPT                     # of parameters: 1     <INT> select zoom options 1-4
    COLOR_SCROLL_OPT            # of parameters: 1     <INT> select color scroll fading direction, 1-14
    TEXTWR                      # of parameters: 1     <STRING> update text for textwriter generator
    TEXTWR_OPTION               # of parameters: 1     <INT> set mode textwriter (pingpong scroller, left scroller)
    CHANGE_BRIGHTNESS           # of parameters: 1     <INT> output brightness 0 .. 100
    GENERATOR_SPEED             # of parameters: 1     <INT> generator speed 0 .. 200 (default speed is 100)
    BEAT_WORKMODE               # of parameters: 1     <INT> change beat workmode 0-2
    OSC_GENERATOR1              # of parameters: 1     <BLOB> contains Xres*Yres*8bpp bytes or Xres*Yres*24bpp bytes raw imagedata
    OSC_GENERATOR2              # of parameters: 1     <BLOB> contains Xres*Yres*8bpp bytes or Xres*Yres*24bpp bytes raw imagedata

    CHANGE_THRESHOLD_VALUE      # of parameters: 1     <INT> select current threshold for the threshold effect, 0-255
    CHANGE_ROTOZOOM             # of parameters: 1     <INT> select angle for the rotozoom effect, -127-127

    CHANGE_PRESET               # of parameters: 1     <INT> select current preset id
    CHANGE_SHUFFLER_SELECT      # of parameters: 18    <INT>, parameter contains 15 nibbles to enable or disable the shuffler option (gets changed in the random
  mode), 0=OFF, 1=ON, example: 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1
    SAVE_PRESET                 # of parameters: 0     <NO PARAM> save current preset settings
    LOAD_PRESET                 # of parameters: 0     <NO PARAM> load current preset settings
    RANDOM                      # of parameters: 1     <ON|OFF> enable/disable random mode
    RANDOM_PRESET_MODE          # of parameters: 1     <ON|OFF> enable/disable random preset mode
    RANDOMIZE                   # of parameters: 0     <NO PARAM> one shot randomizer
    PRESET_RANDOM               # of parameters: 0     <NO PARAM> one shot randomizer, use a pre-stored preset
    JMX_STAT                    # of parameters: 0     <NO PARAM> show JMX runtime statistic, default port: 1337 (use the -p switch)
    SCREENSHOT                  # of parameters: 0     <NO PARAM> save screenhot
    FREEZE                      # of parameters: 0     <NO PARAM> toggle pause mode
    TOGGLE_INTERNAL_VISUAL      # of parameters: 0     <NO PARAM> show/hide internal visual to save CPU
```

OSC Parameter

If you want to select another Generator, Effect or Mixer via OSC Message, you need to specify it's ID. Here is the list of all **Generator** ID's:

```
    PASSTHRU(0)
    BLINKENLIGHTS(1)
    IMAGE(2)
    PLASMA(3)
    COLOR_SCROLL(4)
    FIRE(5)
    METABALLS(6)
    PIXELIMAGE(7)
    COLOR_FADE(8)
    TEXTWRITER(9)
    DROPS(10)
    CELL(11)
    PLASMA_ADVANCED(12)
    FFT(13)
    SCREEN_CAPTURE(14)
    OSC_GEN1(15)
    OSC_GEN2(16)
    VISUAL_ZERO(17)
    NOISE(18)
```

Here is the list of all **Effect** ID's:

```
    PASSTHRU(0)
    INVERTER(1)
    ROTOZOOM(2)
    BEAT_HORIZONTAL_SHIFT(3)
    BEAT_VERTICAL_SHIFT(4)
    VOLUMINIZE(5)
    THRESHOLD(6)
    TEXTURE_DEFORMATION(7)
    ZOOM(8)
    FLIP_X(9)
    FLIP_Y(10)
    STROBO(11)
    ROTATE90(12)
    POSTERIZE(13)
    DARKEN(14)
```

Here is the list of all **Mixer** ID's:

```
    PASSTHRU(0)
    ADDSAT(1)
    MULTIPLY(2)
    MIX(3)
    NEGATIVE_MULTIPLY(4)
    CHECKBOX(5)
    VOLUMINIZER(6)
    EITHER(7)
    SUBSAT(8)
    HALFHALF(9)
    HALFHALFVERTICAL(10)
    MINIMUM(11)
    MAXIMUM(12)
```

It doesn't work!

Try to understand **WHAT** does not work, which component? is it the frontend? PixelController itself? or no output?

Here are some common errors:

- Is Java installed on your system? Open a terminal Windows (cmd.exe on Windows, terminal on OSX) and enter `java -version`.
- Did you forgot to **edit the configuration file** `config.properties`. Take a look at the config examples files in the `data/config.examples` directory!
- Did you flash the **correct firmware** to your Arduino/Teensy?
- **PixelInvaders panels**: Make sure that the Panel shows an **animated rainbow pattern** when the panels are powered on (make sure that you also power the Arduino/Teensy board). If you don't see a animated rainbow, make sure the direction of the modules is correct and that the Arduino/Teensy, LED modules and PSU share common ground. Verify the Arduino IDE don't spit out errors when you upload the firmware to the teensy
- **PixelInvaders panels**: Multiple users reported that the PixelInvader firmware did not work on a new Arduino UNO r3 board. I think the reason for this is the big serial latency. However using a Arduino UNO r1 worked flawlessly. Technically this is not a big deal, as the timeout value cold be adjusted in the firmware. Use a Teensy 2 board for best results.
- Make sure you're using an up-to date Java Runtime (JRE), this usually helps if the JVM crashes.
- If you use an extra long USB Cable (more than 5 meter) you might discover strange issues, try to use a short cable especially if you're uploading a firmware to the Arduino/Teensy.
- The **OSC Generator** does not work: make sure you select the correct resolution for the OSC sender, take a look at the INFO tab, there you see the PixelController internal buffer size. Use this resolution in your OSC sender (or Processing sketch).

Howto build PixelController from source

Prerequisite:

- Maven
- JDK 1.6+

Then run

```
# mvn clean package
```

to build PixelController, **the distribution directory** is `pixelcontroller-distribution/target/assembly`.

Hint: if you're using eclipse and you see an error like this `java.lang.NoClassDefFoundError: Could not initialize class gnu.io.RXTXVersionjava.lang.NoClassDefFoundError: Could not initialize class gnu.io.RXTXVersion` make sure you add the lib/serial directory as "Native library location"

Add new hardware support

It should be pretty simple to add support for new hardware. All Output code should go into the com.neophob.sematrix.output package (`src/main/java/com/neophob/sematrix/output` directory). All you need to do in the Output class is, take an array of int's (one int is used to store the 24 bpp) and send this buffer to your output device (via serial port, ethernet, spi...). Maybe you need to reduce the color depth, flip each second scanline due hardware wiring, such helper methods should go into the `OutputHelper.java` class.

As a starting point, add your hardware in the `OutputDeviceEnum.java` class and have a look where the other entries are referenced.

New Release

Optional, license header check for all source files (http://code.mycila.com/license-maven-plugin/)

```
# mvn license:check -Dyear=2014 -Demail=michu@neophob.com -Dlicense.header=./../pixelcontroller-distribution/src/main/resources/header.txt
# mvn license:format -Dyear=2014 -Demail=michu@neophob.com -Dlicense.header=./../pixelcontroller-distribution/src/main/resources/header.txt
```

Use the Maven version plugin to update your POM's versions:

```
# mvn versions:set -DnewVersion=2.0.0
# mvn versions:commit
```

Rebuild:

```
# mvn clean deploy
```

Test application, make sure the `config.properties` file is correct.

Verify the `ToDo.md` file is updated.

Update `readme.pdf` - use `README.md` as source.

Update Changelog, add git status:

```
# git diff v1.5.0 develop --stat
```

Commit and push new version:

```
# git commit pom.xml -m "release v1.5.1"
# git push
```

Tag the release branch:

```
# git tag -a v1.5.1
# git push --tags
```

Merge into the master branch and push:

```
# git checkout master
# git merge develop
# git push
```

Checkout the master branch (already done)

Do a deployment build:

```
# mvn clean deploy
```

Release

Performance

You can start a quick PixelController performance test, where common functions (hashing, generate visuals, resize images) are measured. You can start it by execute `./pixConServer/PixelController.sh -perf`.

Performance test using 500,000 rounds (Using PixelController v2.1.0-RC1)

Environment	Adler32	XXHash	Visual	PixelResize	QualityResize
OSX, 16x16	`191ms (0ns)`	`118ms (0ns)`	`614ms (245ns)`	`2ms (0ns)`	`28ms (11ns)`
OSX, 32x32	`179ms (0ns)`	`262ms (0ns)`	`618ms (247ns)`	`7ms (2ns)`	`123ms (49ns)`
BBB, 16x16	`806ms (1ns)`	`1799ms (3ns)`	`7985ms (3194ns)`	`31ms (12ns)`	`397ms (158ns)`
BBB, 32x32	`2554ms (5ns)`	`6883ms (13ns)`	`7971ms (3188ns)`	`151ms (60ns)`	`1633ms (653ns)`
RPi, 16x16	`1915ms (3ns)`	`3267ms (6ns)`	`18355ms (7342ns)`	`117ms (46ns)`	`575ms (230ns)`
RPi, 32x32	`5604ms (11ns)`	`12616ms (25ns)`	`17892ms (7156ns)`	`584ms (233ns)`	`2629ms (1051ns)`

BBB: Beagle Bone Black, ARMv7 Processor rev 2 (v7l), 300 BogoMIPS, **JRE**: `1.7.0_51-b13`, Kernel: `Linux beaglebone 3.8.13 #1 SMP Thu Sep 12 10:27:06 CEST 2013 armv7l GNU/Linux`

RPi: Raspberry Pi, Model B, 512MB Ram, ARMv6-compatible processor rev 7 (v6l), **JRE**: `1.7.0_40-b43`, Kernel: `Linux raspberrypi 3.10.27+ #630 PREEMPT Fri Jan 17 19:44:36 GMT 2014 armv6l GNU/Linux`

OSX: Machine: MacBook Air, 2x1.8GHz i5, **JRE**: `1.7.0_21-b12`, Kernel: `Darwin xxx.local 13.0.0 Darwin Kernel Version 13.0.0: Thu Sep 19 22:22:27 PDT 2013; root:xnu-2422.1.72~6/RELEASE_X86_64 x86_64`

Credits

- **Michael Vogt**: Project Lead, Main Developer
- **Markus Lang**: Maven enhancements, Output enhancements, Performance enhancements, Rainbowduino V3 support
- **McGyver666**: Contributor
- **Rainer Ostendorf**: Artnet Output
- **Pesi**: miniDMX Output, Tester
- **Scott Wilson**: Arduino/Rainbowduino Howto
- **Noxx6**: Bugfixes
- **okyeron**: Stealth output device
- **Dr. Stahl**: Documentation, Tester