F-Prop

Falcon Prop Controller

(Includes up to Build 2)

Last Updated: May, 2024

Written and Edited by:

Keith Westley

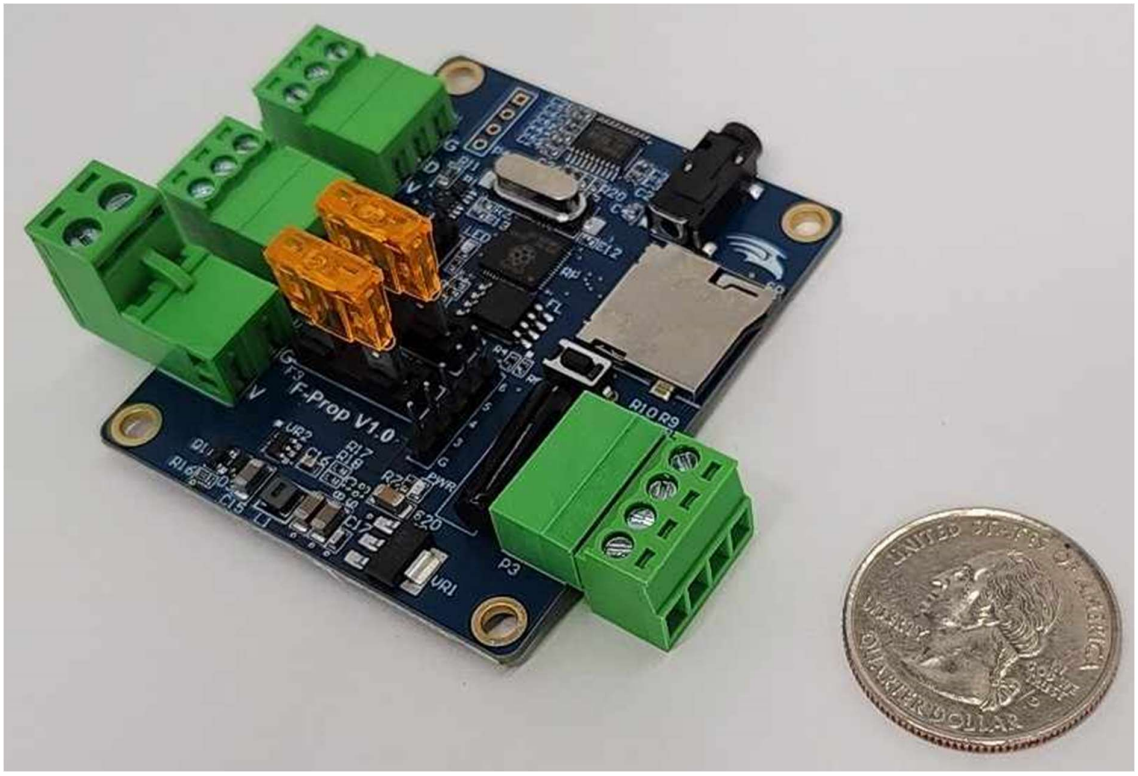


Table of Contents

Contents

Overview

The F-Prop is a stand alone controller for running a single (or small group of) props. It is not designed to be integrated with other props or controllers.

When we say prop we use it in the most general sense possible. Essentially it can be used to run any prop with up to 2048 pixels and audio including the ability to react to events going on in the environment around the prop.

There are many ways to configure and use this board but its most common usage would be to by default play pixel and/or audio data when in its default state. Then when an event is detected by one of its sensors and timers it would switch to an alternate pixel/audio display and once that was completed return to the default state.

That's said this is not even close to the full capabilities of the board. Other things it could do:

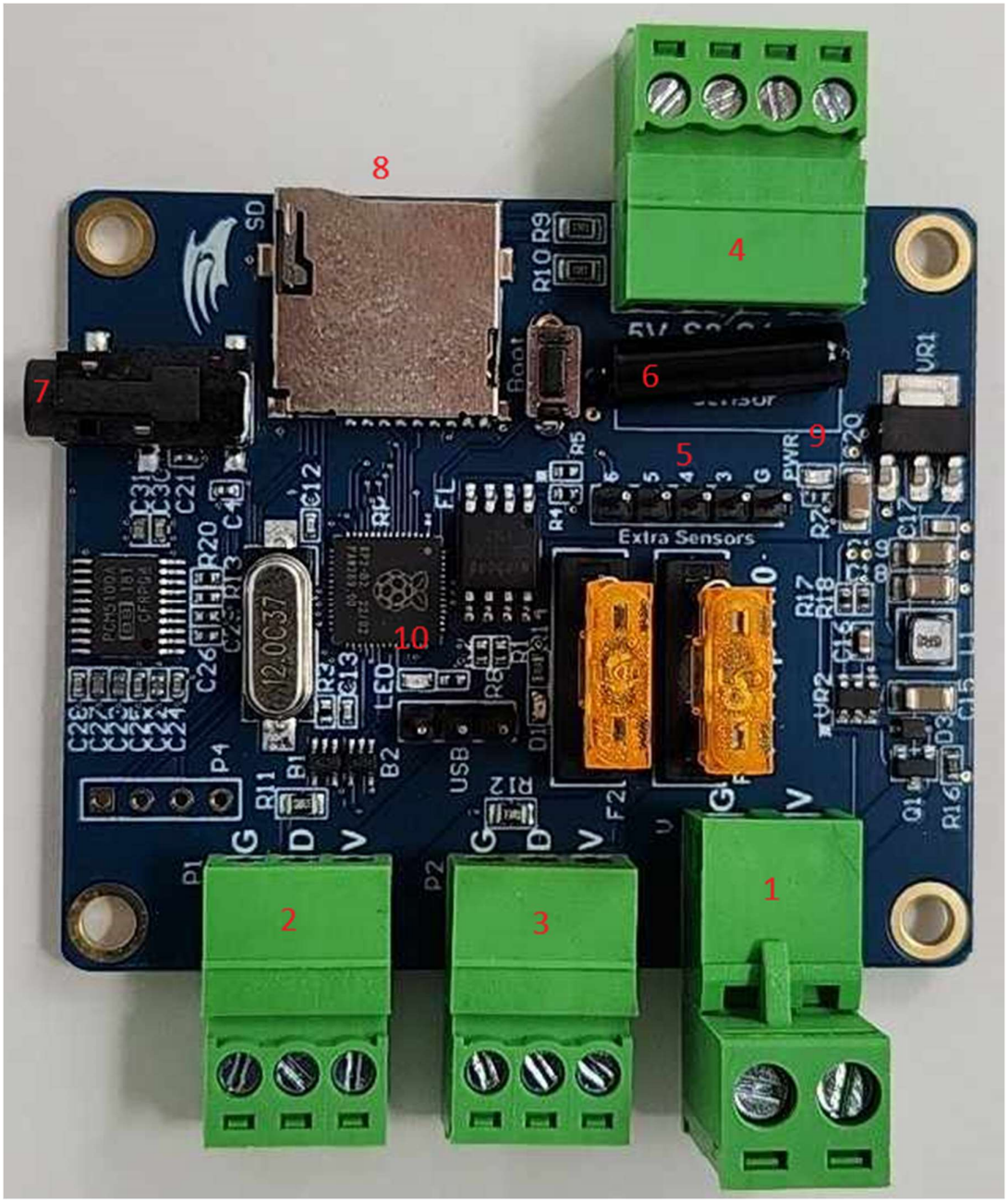
- Run kitchen cabinet lighting
- Run mood lighting
- Run architectural lighting
- Run a jukebox of lights and/or audio

Once configured (and particularly if you are able to fit all the data into the onboard flash) the prop essentially becomes completely self contained. You plug it in and it just works.

Feature Highlights

- 5V or 7-12V power
- 15MB of onboard flash memory to hold pixel data/audio data
- Micro SD card slot for configuring the board and for holding data if it cannot fit within the 15MB limit
- 2 fused pixel ports supporting 1024 3 channel pixels each (or 768 4 channel pixels)
- 1 audio output (3.5mm jack) supporting audio output.
- 6 input/output pins that can be used to
    - Detect closure of a normally open switch (or a high from a normally low output from a sensor)
    - Detect opening of a normally closed switch (or a low from a normally high output from a sensor)
    - Detect an object within range of an ultrasonic sensor
    - Detect movement of the board
- Ability to define time based events triggering

Physical Board



1. 5/12V Power connector
2. Pixel Port 1
3. Pixel Port 2
4. Sensor 1/2 along with 5V/GND for sensors
5. Sensors 3-6 along with GND for sensors
6. Vibration sensor
7. 3.5mm Audio jack
8. Micro SD Card reader
9. Power LED
10. Indicator LED

LED

Immediately behind and in the middle of the 2 pixel ports is an indicator LED (10) that provides basic information about what is going on with your prop controller. These are the possible things you could see it doing and what it means.

Fading on/off (sometimes quickly and sometimes quite slowly) – This occurs during the booting phase of the device (including any configuration actions the controller may be performing). Depending on the exact actions being performed this can be quite slow or quite fast. If your device is doing this just wait for it to finish.

Blinking on/off every second – This occurs when the device is running normally. It means the device has valid firmware and is running some valid configuration.

Blinking on/off rapidly 8 times followed by a brief pause then repeating – This occurs when the controller is in an error state. It may be because the device has no firmware or it may be because the configuration it was trying to load contained an error it could not recover from. If this occurs boot the device again with a SD card in the slot then look for fepbl.log and fep.log on the SD card. These files are written in that order and will contain information about any errors that may have occurred. You may want to delete these files and repeat the process to ensure they are both from the last booting as you may just get a fepbl.log if the error occurs in the bootloading (generally an error applying firmware or finding firmware). fep.log will contain errors during the configuration load or even while running (if you enabled that logging in your configuration).

Pixel Outputs

The onboard pixel outputs can provide both power and data to the pixels as long as the current draw remains under 5A. For 12V pixels at 100% brightness this is commonly 100-200 pixels. Beyond that you will need to route power to the pixels bypassing the onboard fuses. If you do decide to power the pixels directly rather than from the board then it is essential that the pixel data wire and the ground from the first pixel is connected to the pixel port. Failing to do so will likely result in pixel flickering.

Between the F-Prop and the first pixel should be no more than 6 feet. Beyond that again you may experience flickering or even the inability to control pixels. If you must run pixel data further than 6 feet we recommend the use of F-Amps roughly every 15 feet to boost the data signal.

Pixels supported are WS2811 or one of the many compatible chipsets including:

- 3 channel pixels : APA104, APA106, CS8812, GS8202, GS8206, GS8208, INK1002, INK1003, LPD1882, LX1203, P9883, SK6812 (RGB), SK6813, SK6822, SM16703, SM16711, SM16712, TM1803, TM1804, TM1809, TM1812, TM1914, UCS1903, UCS1904, UCS1909, UCS1912, UCS2903, UCS2909, UCS2912, UCS9812, WS2811, WS2812, WS2813, WS2815, WS2818
- 4 channel pixels: APA109, SK6812 (RGBW), SK6818, SM16704, UCS2904, WS2814

Pixel data can either come from some of the inbuilt patterns or from FSEQ files created by xLights or Vixen software. If you are using FSEQ files they must:

- Be either V1 format or V2 Uncompressed or V2 ZLIB compressed. ZSTD compressed is not supported.
- They should contain the data for pixel port 1 at the first channel of each frame
- They should contain the data for pixel port 2 immediately after the end of the pixel port 1 data
- Frame rates are not critical but due to the time taken to send out the data if you are running 40 frames per second (or 25ms frames) then you should limit the pixels on each port to ~700.
- If you are running > 700 pixels on one or both of the ports you should use 20 frames per second (or 50ms frames)
- Where you are playing FSEQ files from the SD Card they should be named 001.fseq – 999.fseq. Names are not important when you are playing them from the onboard memory.

Audio Output

The audio output jack (7) is a standard stereo 3.5mm jack. You can plug either a headphone or a powered speaker into this jack for audio playback.

Audio files need to be WAV files ideally at bitrates of 22kbps or lower. 44.1kbps may play but they are stretching the capacity of the processor. They can be mono or stereo files but they must use the signed 16 bit integer format.

This tool (https://pixelcontroller.com/tools/audioconverter.html) will allow you to easily convert files into a compatible format. It runs locally in your browser.

As WAV files can be quite large if you plan to load it into flash we recommend you try the lower bitrates (even as low as 8kbps) as they are significantly smaller files.

When playing audio files from the SD Card but not associated with a FSEQ file then they should be named 001.wav - 999.wav. If the audio files are associated with a FSEQ file then you should leave the base filename the same as it was when you created the FSEQ file in xLights or Vixen. Names are not important when you are loading them into the onboard flash but the slot you load them into is if they are associated with a FSEQ file. In those cases you must always load the associated audio in the slot immediately after the FSEQ file slot.

Sensor Inputs

Vibration Sensor

The vibration sensor (6) is built onto the board. It triggers when the board is jolted in some way.

Sensor 1/2

The 2 middle pins of the 4 pin connector (4) are sensor pins 1 and 2. These pins are the easiest to connect to and can be configured for multiple uses:

- Normally Open/Normally High – Sensor 1 and 2 can be independently configured to sense the closure of a switch or a sensor which normally outputs a high signal

outputting a low signal. When using a switch or other similar device one input should be connected to the sensor pin and the other to the ground pin.

- Normally Closed/Normally Low – Sensor 1 and 2 can be independently configured to sense the opening of a switch or a sensor which normally outputs a low signal outputting a high signal. When using a switch or other similar device one input should be connected to the sensor pin and the other to the ground pin.
- A HC-SR04 ultrasonic proximity sensor can be attached to the 4 pin connector and can be used to detect objects approaching within a range of 0-5 feet. Sensor 1 should be attached to the echo pin and sensor 2 to the trigger pin. Power for the sensor can be taken from the power pins on this connector.
- Output Pin – These sensor pins can also be configured as output pins outputting low or high signals or pulses when events occur. This can be used to control other devices. Care should be taken to limit the current draw from these pins. Generally the signal should be used to drive a FET which then can switch a higher current device.

Sensor 3-6

Sensors 3-6 are the header pins (5) between the vibration sensor and the fuses. These pins can be used just like sensor pins 1 and 2 with the exception that they cannot drive a HC-SR04 proximity sensor.

Timers

The controller also has inbuilt 16 timers which can also be used to trigger events.

Sensor Hardware

If you are not a current user of the sorts of sensors some of this terminology may be vague and confusing … so lets look at some sensors which could be attached.

These sensors can generally be brought cheaply from online market places like Amazon, EBay and AliExpress. Often including Arduino in the search can help you find sensors likely to meet your specific needs. If you want high quality sensors companies like Adafruit are a great source for well designed sensors. Sensors requiring power should be 5V tolerant as the board does not provide a 3.3V source.

HC-SR04 – Ultrasonic Proximity Detector

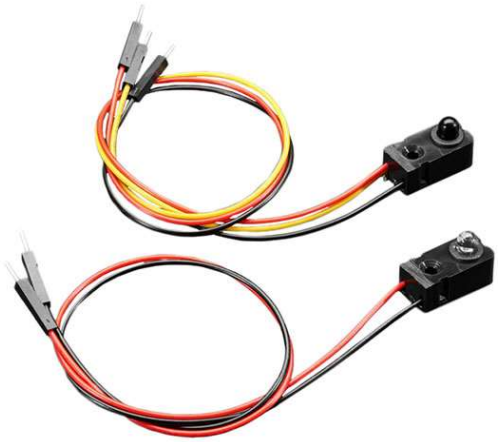
Door Reed Switch – Normally closed switch which triggers when separated

Floor mat – Normally open switch which triggers when stepped on

Sound sensor – Pulses signal high when sound is detected

Passive Infrared (PIR) sensor – Pulses signal high when movement is detected

Infrared beam sensor – Closes when the beam between the two elements is broken.

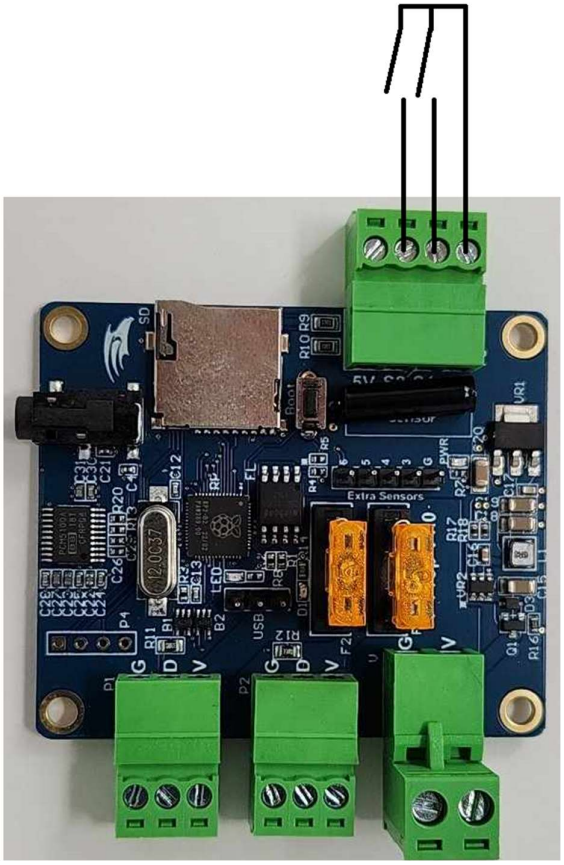


Momentary Push Buttons – close when pressed

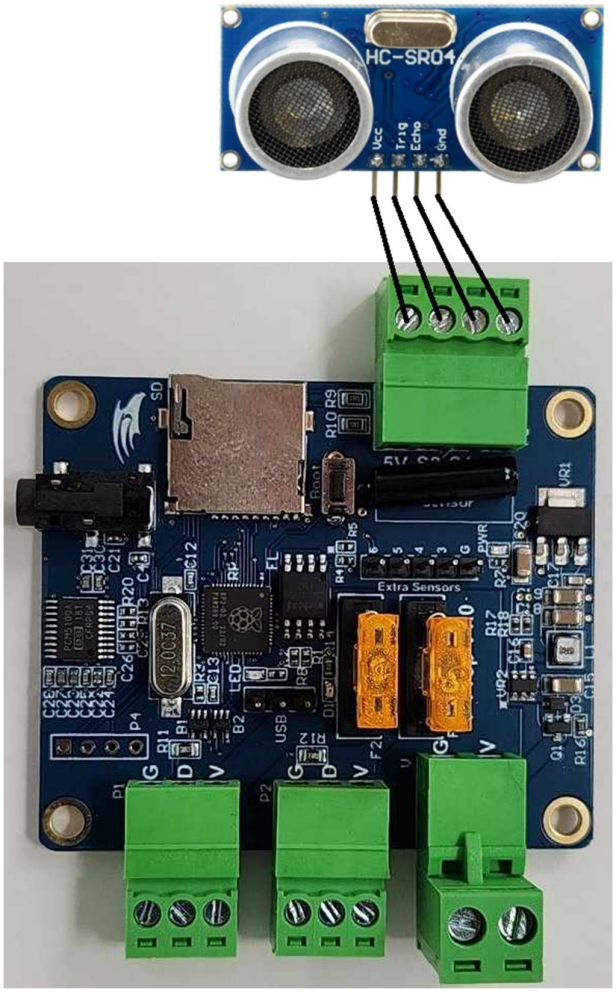
Light Sensors – go high when sufficient light is detected

Attaching sensors

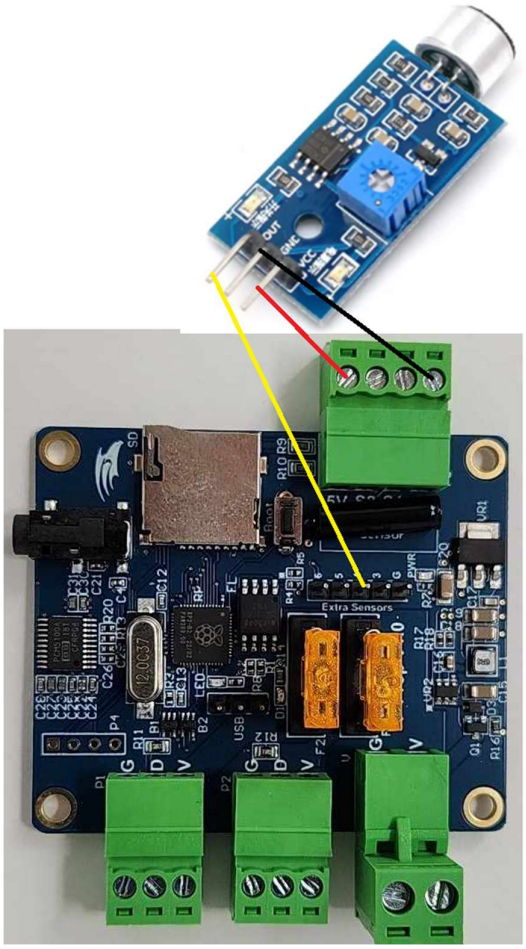
Sensors which are like switches should be connected to ground on the board and one of the 6 sensor pins. This picture shows how you might wire 2 switches to sensor 1 and 2.



Ultrasonic sensors need 2 data wires and power. They must be wired this way to sensors 1 and 2.



Powered sensors need to be 5V tolerant. To attach these follow this wiring diagram this time using sensor 4.



Configuring

Configuring the device is achieved through the creation of a fep.cfg file. This is a text file containing all the settings where you don't want the default setting value. Placing this file onto a SD card along with any FSEQ/WAV files the configuration may require and then booting the device will apply the configuration to the board. If the configuration loaded is not playing sequences/audio from the SD card then the SD card can be removed and the device will continue to run with the loaded configuration.

Configurations of course will only load if they are valid. If you try to load an invalid file the controller may not boot (the LED will flash rapidly 8 times, pause and repeat) or it may continue to run the old configuration or even apply a default configuration. To work out why it did not load you will need to look into the fep.log file on the SD card. This logs the configuration load process and will tell you if any parameters were missing, had invalid values or conflicted with other parameters.

Simple configurations will generally load quite quickly. In circumstances where you are loading many megabytes of sequence/audio data it can take quite some time to load ... even as long as a minute or two. As long as the LED continues to fade on/off it is happily doing its thing and should not be interrupted (although doing so will not hurt the device).

Using the online configuration tool

Available at https://pixelcontroller.com/store/tools/propcontroller/fep.html is a web tool which can help you generate a fep.cfg file rather than needing to create the file manually. There are a couple of things to note about this tool before you use it:

1. It targets a specific build of the F-Prop firmware. Use matching versions for best results. Mismatched versions are likely to work unless you happen to hit something that has changed.
2. While the page is on the internet it actually processes all the data locally in the browser. No FSEQ/Audio files are uploaded to the web.
3. While the tool allows you to save and then reload the configuration to a fep.json file when it loads the file all references to fseq/audio files will load onto the screen but the browser cannot actually access the files due to browser security. This means when you generate the fep.zip file the files that are referenced will be included in the fep.cfg file but omitted from the zip file.
4. The page does not validate everything that could be wrong in the configuration (at least not yet).

Key page features

Generate button – generates and 'downloads' a zip file which will contain the fep.cfg file along with any fseq/wav files. These should be extracted and placed onto the SD card. If you are not prompted by your browser this will by default go to your downloads folder.

Save button – 'downloads' the page to a json file that can be re-loaded later. If you are not prompted by your browser this will by default go to your downloads folder.

Load button – loads a previously saved page. Note that any Onboard or SD files will be loaded as references but will not generate into the zip file due to browser security controls.

Name – used purely in the names of the files saved/generated.

Pixel output – this section contains the settings for defining what pixels are connected to the board.

Audio output – this section contains audio settings

Onboard files – this section contains the files which are going to be loaded into the onboard flash memory. In total these must not exceed 15.5MB. Any MP3 files loaded here will be automatically converted to the WAV files that the board requires.

SD files – this section contains the files which are going to be loaded onto the SD card and played from there. Any MP3 files loaded here will be automatically converted to WAV files that the board requires.

Player – this section defines the default playing action when the board is started and not running an event.

Events – this section defines the events the board will respond to and what it will do when that event occurs.

Miscellaneous – this section has some miscellaneous settings.

See field tooltips for more information on field use.

fep.cfg Basics

The fep.cfg file is a simple text file.

When it is processed the following rules are observed:

- Blank lines are ignored.
- Any line whose first non blank character is a # are ignored. This allows you to write comments in the file explaining to your future self what you were doing.
- Upper/Lower case is irrelevant. The loader will totally ignore it. Use capitals, don't use capitals it is totally up to you. In the documentation we do capitalise the second word where multiple words are joined together for readability reasons but the process does not require you to do that.
- All actual settings are of the form.

```
setting=value
```

- Spaces are allowed to appear before and after the equal sign but will be ignored.
- Spaces at the beginning or end of the line will be ignored
- Spaces within the setting name or the value are not ignored and must be valid for the file .to load.
- Settings where you just want the value to be the same as the default for that setting do not need to be specified.
- Order of the settings within the file is not important
- Duplicate setting will generate warnings but the latest value will be the one applied.

Concepts

When you are creating your fep.cfg file the settings can be loosely grouped into things which are used to configure certain elements/modes of the device

Modes

One of the most critical things to understand in the player is the player modes. There are two types of modes that can be set:

playerMode – There is only one of these and it represents the state the player will be in when it is first booted and no sensors have been detected. It is also the mode it will return to after the event action is completed.

eventMode_X – There are up to 16 of these (ie X can be 1-16). These represents the actions the player will perform when an event occurs.

Modes can have parameters that further configure the mode. There can be up to 5 of these parameters (A-E). The meaning and number varies depending on the mode being used.

Event Settings

Event settings associate the sensors with the events and how that event will be processed. It allows events to be prioritised, suppressed and even controls how the event behaves.

Example settings include: eventAction_X, eventTrigger_X, eventPriority_X, eventSuppressLow_X, eventSuppressHigh_X, minEventGap, minTimerGap, equalPriorityWins

Sensor Settings

Sensors are used to trigger events. Depending on the sensor there are different settings you can set to fine tune how the sensor will behave such as its sensitivity or the minimum time that must elapse between the sensor triggering.

Example settings include: vibrationSensitivity, vibrationGap, sensorSensitivity_X, sensorGap_X, ultrasonicLevel, ultrasonicGap, timerInterval_X, timerGap_X

Pixel Settings

There are settings for the pixel ports including control over the number of pixels on each port, the number of channels each pixel expects, the colour order of the pixels and the brightness that should be applied to the pixel.

Example settings include: pixelType, colorOrder, stringPixels_1, stringPixels_2, brightness

Audio Settings

You can configure the audio volume.

Example settings include: volume

Content Settings

Where you are choosing to load FSEQ or WAV data into the 15MB of flash memory on the device you will specify the files to load and which slot they should be placed into.

Example settings include: file_1-file_32

Miscellaneous Settings

These settings are not as commonly used but can be useful. Settings which turn on additional logging, cause the configuration to just be played from the SD card and force the device back to its default configuration when it arrived from pixelcontroller.

Example settings include: logging, fromSD, defaultConfig

Simple Examples

These are some simple examples to give you a feel for the configuration you may want

Just play rainbow colours

On 100 pixels on pixel port 1 at full brightness

```
# This tells us to play rainbow colours

playerMode=Rainbow

# This tells us there are 100 pixels on the first port

stringPixels_1=100

# Everything else does not need to be specified as our pixels
have all the defaults of full brightness and RGB color order
```

Play colourwash but with two buttons which turn brightness up/down

Using 2 normally open, momentary action push buttons on sensor 1 and 2 to adjust brightness by 10%

```
# This tells it to display a color wash

playerMode=ColorWash


# This defines the decrease brightness button on sensor 1

eventMode_1=brightness

# Reduce brightness by 10%

eventParam_A_1=-10

# This says process the event immediately

eventAction_1=Immediate

# This says the sensor is normally open and thus will trigger
when closed

eventTrigger_1=sensorNO_1

# This requires at least 200ms between detecting the button
presses

sensorGap_1=200


eventMode_2=brightness

# Increase brightness by 10%

eventParam_A_2=10

eventAction_2=Immediate

eventTrigger_2=sensorNO_2

sensorGap_2=200


stringPixels_1=100
```

Play an animation sequence from flash

This loads a FSEQ into flash and then plays that FSEQ forever in a loop on two strings of pixels of length 100 and 45 respectively.

```
# This is the fseq which must be on the SD card when loading this
configuration so it can be copied into flash

file_1=mysequence.fseq

# This plays the fseq from flash

playerMode=PlayFSEQ

# This tells the player that the sequence being played is in slot
file_1

playerParam_A=1

# This specifies the number of pixels on each port

stringPixels_1=100

stringPixels_2=45
```

Play an animation from flash. Interrupt it with an audio + pixels when vibration triggered

This loads 2 FSEQs and an audio into flash and then plays the animation sequence until the vibration sensor triggers at which time the sequence and audio plays. Once done it returns to the animation. Pixel brightness is set to 50%.

```
# We need to load 3 files. As joke.fseq and joke.wav go together they must be loaded in sequential slots

file_1=animation.fseq

file_2=joke.fseq

file_3=joke.wav


# By default we play the animation sequence

playerMode=playFSEQ

playerParam_A=1


# When the vibration trigger files we play the joke sequence and the associated audio

eventMode_1=PlayFSEQAudio

# We just point to the sequence file. The audio is automatically assumed to be in the next slot

eventParam_A_1=2

eventTrigger_1=Vibration

eventAction_1=Immedate


# This specifies the number of pixels on each port

stringPixels_1=100

stringPixels_2=100

# As 128 is 50% of 255 … this is how we set brightness to 50%

brightness=128
```

Do nothing until sensor 1 fires and then play one of 10 random sequence + audio from SD card

Here we need to leave the SD card in the player. Sequences must be named 001.fseq through 010.fseq. Audio files should be as they were in xLights but the WAV file versions of them

```
# Turn all the pixels off normally

playerMode=Blank


eventMode_1=SDFSEQAudio

# The 0xFFFF magic value means play a random sequence

eventParam_A_1=0xFFFF

# this is the first sequence in the range of sequences to choose
from

eventParam_B_1=1

# this is the last sequence in the range of sequences to choose
from

eventParam_C_1=10

# We will play the sequence immediately

eventAction_1=Immedate

# use sensor 1 but it is normally closed so we trigger when the
circuit breaks

eventTrigger_1=sensorNC_1

# we wont refire for at least 200ms

sensorGap_1=200


# This specifies the number of pixels on each port

stringPixels_1=100

stringPixels_2=100
```

Play 5 animations in sequence from flash

This loads 5 FSEQs into flash and then plays them one after another forever.

```
# load the files - the name is not important
file_1=animation1.fseq
file_2=animation2.fseq
file_3=animation3.fseq
file_4=animation4.fseq
file_5=animation5.fseq

# play them all
playerMode=playFSEQ
playerParam_A=0xFF00

# This specifies the number of pixels on each port
stringPixels_1=100
stringPixels_2=100
```

Configuration Reference

file_x (X=1-32)

```
file_1=animation.fseq

file_2=song.wav

…

file_32=xyzzy.fseq
```

These settings load files into the 32 slots on the flash memory chip. The total size of all the files loaded must not exceed 15MB. The amount of free space will be displayed in the fep.log file when you load the configuration if you want to check how much space is left.

Where these are not specified no file is loaded into the slot. If you then refer to an unloaded slot elsewhere in the configuration that will be reported as an error and it will not load.

playerMode and eventMode_X (X=1-16)

```
playerMode=Blank

eventMode_1=White

…

eventMode_16=Rainbow
```

These setting describe the action that is taken either in its default state (playerMode) or when an event occurs (eventMode_x)

By default the mode is None which does nothing.

playerMode and eventMode_X can have between 0 and 5 parameters represented by playerParam_A-playerParam_E and eventParam_A_X-eventParam_E_X respectively. The meaning of these parameters varies depending on the mode value.

A few modes like brightness and volume are only valid in events.

This is the full list of available modes and their parameters:

None	Does nothing	This is the default mode
Blank	Turns off all configured pixels	
ColorWash or ColourWash	Washes through the colour palette with every pixel the same colour	When used on an eventMode_X eventParam_A_X specified the number of milliseconds to hold the colorwash for
Rainbow	Displays a rainbow of colors on the pixels	When used on an eventMode_X eventParam_A_X specified the number of milliseconds to hold the rainbow for
RGBW	Cycles between red, green, blue and white	When used on an eventMode_X eventParam_A_X specified the number of milliseconds to hold the rgbw for

Color or Colour	Turns all pixels to the specified color	playerParam_B/eventParam_B_X has the colour expressed as 0x00RRGGBB – so yellow would be 0x00FFFF00. When used on an eventMode_X eventParam_A_X specified the number of milliseconds to hold the color for
White	Turns all pixels to white	When used on an eventMode_X eventParam_A_X specified the number of milliseconds to hold the rgbw for
SetPinHigh	Sets the pin specified by parameter A to high and holds it in that state. This means that pin cannot be used as a sensor pin.	This cannot be used as a playerMode. eventParam_A_X specifies the sensor pin (1-6) to set high.
SetPinLow	Sets the pin specified by parameter A to low and holds it in that state. This means that pin cannot be used as a sensor pin.	This cannot be used as a playerMode. eventParam_A_X specifies the sensor pin (1-6) to set low.
PulsePinHigh	Sets the pin specified by parameter A to high and holds it in that state for the number of milliseconds specified by parameter B. This means that pin cannot be used as a sensor pin.	This cannot be used as a playerMode. eventParam_A_X specifies the sensor pin (1-6) to pulse high. eventParam_B_X specifies the number of milliseconds to hold the pin high for before it is returned to a low state.
PulsePinLow	Sets the pin specified by parameter A to low and holds it in that state for the number of milliseconds specified by parameter B. This means that pin cannot be used as a sensor pin.	This cannot be used as a playerMode. eventParam_A_X specifies the sensor pin (1-6) to pulse low. eventParam_B_X specifies the number of milliseconds to hold the pin low for before it is returned to a high state.
Reboot	Reboots the device	This cannot be used as a playerMode.
Volume	Adjusts the volume up/down depending on the value of parameter A which is the percentage amount to adjust by	This cannot be used as a playerMode. eventParam_A_X is between -100 and 100 representing adjust by -100% to adjust by +100%. This is the absolute adjustment so if currently 20% and you add 10% then the result is 30%.
Brightness	Adjusts the brightness up/down depending on the value of parameter A which is the percentage amount to adjust by	This cannot be used as a playerMode. eventParam_A_X is between -100 and 100 representing adjust by -100% to adjust by +100%. This is the absolute adjustment so if currently 20% and you add 10% then the result is 30%.
Index	Moves forward or backward between sequences/audio/effects.	This cannot be used as a playerMode. eventParam_A_X is 0 when we wish to move to the prior sequence/effect and 1 when we wish to move to the next sequence/effect.

	Parameter A determines the direction of movement.	This mode only makes sense if you are using a mode like fx or SDFSEQ, SDFSEQAudio or SDAudio.
playFSEQ	Plays a FSEQ from flash. Parameter A specifies which slot to play from or one of the special modes. Parameter B and C are used by those special modes to limit the slots used.	playerParam_A/eventParam_A_X – When 1-32 it specified the slot to play the FSEQ from. When set to 0xFFFF it tells the player to randomly choose a sequence to play. When set to 0xFF00 it tells the player to play the sequences one after another. playerParam_B/eventParam_B_X – when parameter A is 0xFFFF or 0xFF00 then this is the first slot to use when choosing a sequence to play. playerParam_C/eventParam_C_X – when parameter A is 0xFFFF or 0xFF00 then this is the last slot to use when choosing a sequence to play. When in 0xFFFF mode or 0xFF00 mode the Index mode can be used to manually move between sequences
playAudio	Plays a WAV from flash. Parameter A specifies which slot to play from or one of the special modes. Parameter B and C are used by those special modes to limit the slots used.	playerParam_A/eventParam_A_X – When 1-32 it specified the slot to play the WAV from. When set to 0xFFFF it tells the player to randomly choose a WAV to play. When set to 0xFF00 it tells the player to play the WAVs one after another. playerParam_B/eventParam_B_X – when parameter A is 0xFFFF or 0xFF00 then this is the first slot to use when choosing a WAV to play. playerParam_C/eventParam_C_X – when parameter A is 0xFFFF or 0xFF00 then this is the last slot to use when choosing a WAV to play. When in 0xFFFF mode or 0xFF00 mode the Index mode can be used to manually move between WAVs
playFSEQAudio	Plays a FSEQ/WAV from flash. Parameter A specifies which slot to play from or one of the special modes. Parameter B and C are used by those special modes to limit the slots used.	playerParam_A/eventParam_A_X – When 1-31 it specified the slot to play the FSEQ/WAV from. When set to 0xFFFF it tells the player to randomly choose a FSEQ/WAV to play. When set to 0xFF00 it tells the player to play the FSEQ/WAVs one after another. playerParam_B/eventParam_B_X – when parameter A is 0xFFFF or 0xFF00 then this is the first slot to use when choosing a FSEQ/WAV to play. This must point to a FSEQ file slot. playerParam_C/eventParam_C_X – when parameter A is 0xFFFF or 0xFF00 then this is the last slot to use when choosing a FSEQ/WAV to play. This must point to a FSEQ file slot. When in 0xFFFF mode or 0xFF00 mode the Index mode can be used to manually move between WAVs FSEQ files must have their associated WAV files loaded in the immediate next slot.
SDFSEQ	Plays a FSEQ from SD Card. Parameter A specifies which file to play or one of the special	playerParam_A/eventParam_A_X – When 1-999 it specifies the FSEQ to play. When set to 0xFFFF it tells the player to randomly choose a sequence to play.

	modes. Parameter B and C are used by those special modes to limit the files used.	When set to 0xFF00 it tells the player to play the sequences one after another. playerParam_B/eventParam_B_X – when parameter A is 0xFFFF or 0xFF00 then this is the first file to use when choosing a sequence to play. playerParam_C/eventParam_C_X – when parameter A is 0xFFFF or 0xFF00 then this is the last file to use when choosing a sequence to play. When in 0xFFFF mode or 0xFF00 mode the Index mode can be used to manually move between sequences Sequences on the SD card must be named 001.fseq-999.fseq
SDAudio	Plays a WAV from SD Card. Parameter A specifies which file to play or one of the special modes. Parameter B and C are used by those special modes to limit the files used.	playerParam_A/eventParam_A_X – When 1-999 it specifies the WAV to play. When set to 0xFFFF it tells the player to randomly choose a WAV to play. When set to 0xFF00 it tells the player to play the WAVs one after another. playerParam_B/eventParam_B_X – when parameter A is 0xFFFF or 0xFF00 then this is the first file to use when choosing a WAV to play. playerParam_C/eventParam_C_X – when parameter A is 0xFFFF or 0xFF00 then this is the last file to use when choosing a WAV to play. When in 0xFFFF mode or 0xFF00 mode the Index mode can be used to manually move between WAVs WAVs on the SD card must be named 001.wav-999.wav
SDFSEQAudio	Plays a FSEQ/Audio from SD Card. Parameter A specifies which file to play or one of the special modes. Parameter B and C are used by those special modes to limit the files used.	playerParam_A/eventParam_A_X – When 1-999 it specifies the FSEQ/Audio to play. When set to 0xFFFF it tells the player to randomly choose a sequence/audio to play. When set to 0xFF00 it tells the player to play the sequences/audio one after another. playerParam_B/eventParam_B_X – when parameter A is 0xFFFF or 0xFF00 then this is the first file to use when choosing a sequence/audio to play. playerParam_C/eventParam_C_X – when parameter A is 0xFFFF or 0xFF00 then this is the last file to use when choosing a sequence/audio to play. When in 0xFFFF mode or 0xFF00 mode the Index mode can be used to manually move between sequences/audio Sequences on the SD card must be named 001.fseq-999.fseq. The WAV files can be the same name as the audio used to create the sequence in xLights/Vixen but in WAV format. If that file is not there then the wav file with the number one more than the sequence will be chosen … so 005.fseq will play 006.wav with it.

FX	Plays an effect from the onboard effect library on the pixels.	When used on an eventMode_X eventParam_A_X specified the number of milliseconds to hold the effect for playerParam_B/eventParam_B_X – specifies the effect to play (1-55). When set to 0xFFFF it tells the player to randomly choose an effect to play. When set to 0xFF00 it tells the player to play the effects one after another. playerParam_C/eventParam_C_X – specifies either a palette to use or the first color in a custom 3 color palette. Not all effects pay attention to the palette. Possible values are: - 0x01000000 or 0x00000000 - RGB - 0x02000000 - Red Green White - 0x03000000 - Yellow Cyan Purple - 0x04000000 - Purple Orange - 0x05000000 - Greens - 0x06000000 - Reds - 0x07000000 - Blues - 0x08000000 - red Orange Yellow - 0x09000000 - Purple Blue - 0x0A000000 - Green Blue - 0x0B000000 - Red Green - 0x00RRGGBB - Where RR GG and BB are the hex colour components - 0x00000000 does not work here playerParam_D/eventParam_D_X – specifies the second color in a custom 3 color palette - 0x00RRGGBB - Where RR GG and BB are the hex colour components - 0x00000000 does not work here playerParam_E/eventParam_E_X – specifies the third color in a custom 3 color palette - 0x00RRGGBB - Where RR GG and BB are the hex colour components - 0x00000000 does not work here When in 0xFFFF mode or 0xFF00 mode the Index mode can be used to manually move between effects The available effects are: - 0 - Static - 1 - Blink - 2 - Breath - 3 - Color Wipe - 4 - Color Wipe Inverse - 5 - Color Wipe Reverse - 6 - Color Wipe Reverse Inverse - 7 - Color Wipe Random - 8 - Random Color - 9 - Single Dynamic - 10 - Multi Dynamic - 11 - Rainbow - 12 - Rainbow Cycle - 13 - Scan

		- 14 - Dual Scan - 15 - Fade - 16 - Theater Chase - 17 - Theater Chase Rainbow - 18 - Running Lights - 19 - Twinkle - 20 - Twinkle Random - 21 - Twinkle Fade - 22 - Twinkle Fade Random - 23 - Sparkle - 24 - Flash Sparkle - 25 - Hyper Sparkle - 26 - Strobe - 27 - Strobe Rainbow - 28 - Multi Strobe - 29 - Blink Rainbow - 30 - Chase White - 31 - Chase Color - 32 - Chase Random - 33 - Chase Rainbow - 34 - Chase Flash - 35 - Chase Flash Random - 36 - Chase Rainbow White - 37 - Chase Blackout - 38 - Chase Blackout Rainbow - 39 - Color Sweep Random - 40 - Running Color - 41 - Running Red Blue - 42 - Running Random - 43 - Larson Scanner - 44 - Comet - 45 - Fireworks - 46 - Fireworks Random - 47 - Merry Christmas - 48 - Fire Flicker - 49 - Fire Flicker soft - 50 - Fire Flicker intense - 51 - Circus Combustus - 52 - Halloween - 53 - Bicolor Chase - 54 - Tricolor Chase - 55 - TwinkleFOX

All parameters default to a value of 0 if not specified

eventAction_X (X=1-16)

```
eventAction_1=Immediate
```

This determines how the event is handled when it fires. Possible values include:

- None – Ignore the event (default)
- Immediate – process the event immediately

- Graceful - process the event when the current mode finishes. This does not always make sense. For example if your player mode was Rainbow and your event was graceful the event would never fire because rainbow never ends
- ImmediateToggle – rather than switching to the event mode and returning when done this will remain in the event mode until something else triggers a change. The switch occurs immediately
- GracefulToggle – process the event when the current mode finishes and then stay in that event mode until something else happens to get us out of this mode.

eventTrigger_X (X=1-16)

```
eventTrigger=Vibration
```

Specifies which sensor should trigger the event. Possible values include:

- None – monitor nothing – default
- Vibration – trigger based on the on board vibration sensor
- SensorNO_X – (X=1-6) Triggers when one of the sensor pins is shorted to ground by closing of the circuit. The number of times this needs to happen to trigger the event is determined by the sensitivity.
- SensorNC_X – (X-1-6) Triggers when one of the sensor pins which is shorted to ground is made open circuit. The number of times this needs to happen to trigger the event is determined by the sensitivity.
- SensorNOPoll_X – (X=1-6) Triggers when one of the sensor pins is shorted to ground by closing of the circuit and held in that state for the sensitivity duration.
- SensorNCPoll_X – (X-1-6) Triggers when one of the sensor pins which is shorted to ground is made open circuit and held in that state for the sensitivity duration.
- Timer_X – (X=1-16) Triggers when the timer goes off which happens randomly within a specified time interval.
- Ultrasonic – Triggers based on a HC-SR04 sensor connected to sensor pins 1 and 2 detects an object within a specified distance.

eventPriority_X (X=1-16)

```
eventPriority_1=100
```

Specifies the priority of this event compared to other events. When an event is running if another event of higher priority fires then it will be interrupted and the higher priority event will play. On completion it will NOT return to the interrupted event but back to the playerMode. Valid values are 0-255 with 0 being the default value.

Events of equal priority will not interrupt each other unless the equalPriorityWins setting has been set to 1.

eventSuppressLow_X (X=1-16)

```
eventSuppressLow_1=2
```

If eventSuppressLow is set to a value other than zero then this event will be suppressed if the nominated sensor pin is held low.

Valid values are 0-6. Default value is 0 which means never suppress.

eventSuppressHigh_X (X=1-16)

```
eventSuppressHigh_1=2
```

If eventSuppressHigh is set to a value other than zero then this event will be suppressed if the nominated sensor pin is held high.

Valid values are 0-6. Default value is 0 which means never suppress.

vibrationSensitivity

```
vibrationSensitivity=3
```

This setting determines how many times the vibration sensor need to be sensed before it is counted as an event. Zero is the default which is also the most sensitive.

vibrationGap

```
vibrationGap=500
```

This is the minimum number of milliseconds that must pass after a vibration event before another can be detected.

sensorSensitivity_X (X=1-6)

```
sensorSensitiivity_1=2
```

When using sensorNO or sensorNC this setting determines how many times the sensor need to be sensed before it is counted as an event. Zero is the default which is also the most sensitive.

When using sensorNOPoll or sensorNCPoll this setting determines how many milliseconds the sensor must be triggered for before the trigger will fire. Zero is the default which is also the most sensitive.

Valid values are 0-1000.

sensorGap_X (X=1-6)

```
sensorGap_2=500
```

This is the minimum number of milliseconds that must pass after a sensor event before another can be detected.

ultrasonicLevel

```
ultrasonicLevel=1024
```

This is the threshold value that is compared to the value returned from the HC-SR04 sensor. If the value is less than this number then the trigger fires.

Zero is the least sensitive (it pretty much will never fire and the default). Based on experimentation the maximum value I have ever seen is about 9500.

These values convert to distance using a scale of 10mm = 58, 1m – 5800. 147 = 1”, 1764 = 1’.

The maximum of 9500 ~- 163cm/64”

ultrasonicGap

```
ultrasonicGap=500
```

This is the minimum number of milliseconds that must pass after a ultrasonic event before another can be detected.

timerInterval_X (X=1-16)

```
timerInterval_1=30000
```

This is the span of time in milliseconds after the timerGap_X during which the timer will randomly go off. If this is zero which is the default then it will immediately trigger after the timerGap_X time interval.

timerGap_X (X=1-16)

```
timerGap_2=60000
```

This is the minimum gap in milliseconds between the timer firing. Default is 0.

minTimerGap

```
minTimerGap=5000
```

This is the minimum gap in milliseconds between any timers firing. This allows you to have multiple timers firing but ensure they never happen too close together.

minEventGap

```
minEventGap=1000
```

Like the minTimerGap this prevents any two events from occurring too closely together. Essentially the board ignores all event triggers for this number of milliseconds after any event fires.

equalPriorityWins

```
equalPriorityWins=1
```

When set to 1 two events with the same priority can interrupt each other. When set to zero which is the default they cannot.

channelsPerPixel

```
channelsPerPixel=4
```

This is used by internally generated patterns to correctly generate pixel data for RGBW style pixels. Valid values are 3 or 4 with the default being 3.

colorOrder (colourOrder)

```
colorOrder=BRG
```

This is used by internally generated patterns to correctly generate pixel data for pixels with different colour orders. Possible values include

- RGB or RGBW – Default
- RBG or RBGW
- GRB or GRBW
- GBR or GBRW
- BRG or BRGW
- BGR or BGRW

stringPixels_1 and stringPixels_2

```
stringPixels_1=100

stringPixels_2=50
```

Specifies the number of pixels on each of the pixel ports on the board.

Default value is zero. Valid values are 0-1024.

When reading data from FSEQ files string 1s data is assumed to be in the first channel of each frame. String 2 data is assumed to start immediately after the last channel of string 1. The fseq file is expected to have at least (stringPixels_1 + stringPixels_2) * channelsPerPixel channels in each frame

brightness

```
brightness=128
```

Sets the brightness of the pixels. Valid values are 0-255. Default value is 255. Percentage is calculated as (brightness * 100)/255 so 128 -> 50%.

volume

```
volume=128
```

Sets the volume of the audio output. Valid values are 0-255. Default value is 255. Percentage is calculated as (volume * 100)/255 so 128 -> 50%.

logging

```
logging=1
```

Logging is normally disabled on the player once it boots to ensure maximum performance. If however you are experiencing playing issues you may want to enable logging by setting this setting to 1. This will log any unexpected states or significant events. You should not leave it on long term but it is useful for troubleshooting. Default is zero which disables logging.

fromSD

```
fromSD=1
```

When this parameter is set the configuration is not loaded into flash (although any file_X entries will be). This allows you to quickly switch between configurations without permanently changing the player configuration just by swapping SD cards.

Default value is 0 which causes the config to be loaded into the players memory.

defaultConfig

```
defaultConfig=1
```

Please don't use this option unless asked to do so by support. This essentially wipes all configuration from the player and returns it to the state you received it from your vendor. Anything else in the file is totally ignored.

Firmware Upgrades

From time to time we may release firmware updates to address bugs or add features to your device. In general we would recommend you only apply such updates where the bugs reported as fixed are impacting you or you specifically require a feature released in a more recent build.

To determine which build you are running boot the device with a SD card in the slot. Once booted remove the card and place it in computer and access the fepbl.log file. The first line will contain text such as:

```
Falcon Event Player Bootloader 1.0 booting.
```

This tells you that it is running build 1.0 of the bootloader.

Now open fep.log. The first line of this file should contain text such as:

```
Falcon Event Player Firmware Build 1 booting.
```

This tells you that it is running Build 1 of the firmware.

The firmware update process will update the firmware only. Never the bootloader.

To update the firmware:

- Visit https://github.com/Pixelcontroller/f-prop_issues/tree/main to access the appropriate build of the firmware and download the FL3 file.
- Place the FL3 file onto a SD card and ensure no other FL3 files are in the root folder on that card. Also ensure there is no fep.cfg file on the card.
- Power down the device
- Insert the SD card into the device
- Power on the device
- The LED will start fading on/off - Wait for the LED to commence flashing
- To check the success of the process pop out the SD card and place it in your computer and open the fepbl.log file which should have text similar to this:

```
        Firmware file found on SD card.

        Firmware checks out as VALID.

        Loading firmware.

        Firmware checks out as VALID.

        Upgrading Firmware 0x10050000 : 141656 [1].

        Erasing firmware.
```

```
        Erasing 0x10050000-0x10073000 (0x00023000:35).
        ..
            Erased.
        Flashing firmware.
        Upgrading firmware FW
        .
        Flashing done.
        Firmware applied.
        Firmware loaded.
```

Advanced Topics

Resetting the device

In some scenarios you may have the need (typically based on a support instruction) to reset the device to a known state.

There are two reset modes the device supports:

Reset to defaults

This action will reset the device to the state it was in when you received the device from the vendor. Essentially it erases all configuration of the device including all data you may have loaded onto it.

To reset the device

- Turn off the power to the device
- Disconnect all pixels and sensors from the device
- Connect a wire between the data pin (middle pin) on Pixel Port 2 and the Sensor 1 Pin
- Apply power to the device
- Watch the LED go through its fade on/off until it blinks on and off every second
- At this point the device has completed its reset
- Remove the wire

Deep erase

This is a much more extreme erase after which you will need to reload the firmware FL3 file downloaded from our support page.

To reset the device

- Turn off the power to the device
- Disconnect all pixels and sensors from the device
- Connect a wire between the data pin (middle pin) on Pixel Port 1 and the Sensor 1 Pin
- Apply power to the device
- Watch the LED go through its fade on/off until it blinks rapidly 8 times, pauses then repeats. This is the board saying it has no firmware to run.

- Turn off the power to the device
- Insert a SD card with the FL3 file of the firmware you wish to load
- Apply power to the device.
- Watch the LED go through its fade on/off until it blinks on and off every second
- At this point the device has completed its reset
- Remove the wire

Snapshot Save

In some support scenarios we need to be able to exactly reproduce the state your board is in. This process allows us to do this. It should only be performed under instruction from us.

To save to a SD card a snapshot of the device in a file named fep_snapshot.bin

- Turn off power to the device
- Insert a SD card (this should not contain a FL3 or fep.cfg file
- Connect a wire between the data pin (middle pin) on Pixel Port 1 and the Sensor 2 Pin
- Apply power to the device
- Watch the LED go through its fade on/off until it blinks on and off every second
- At this point the device has completed its save of the snapshot
- Remove the SD card and send the fep_snapshot.bin file to us
- Remove the wire

Snapshot Load

In even rarer support scenarios we may send you a configuration to load onto the board. This will be sent to you as a fep_snapshot.bin file.

To load the fep_snapshot.bin file

- Turn off power to the device
- Copy the fep_snapshot.bin file into the root folder of a SD Card. Also ensure there is no FL3 files or fep.cfg file on the SD card.
- Insert the SD card
- Connect a wire between the data pin (middle pin) on Pixel Port 2 and the Sensor 2 Pin
- Apply power to the device
- Watch the LED go through its fade on/off until it blinks on and off every second
- At this point the device has completed its load of the snapshot
- Remove the wire