Overview

Have you been wondering what displayio is and don’t know where to get started? Well, look no further. In this guide we show you how to get displayio up and running on our TFT FeatherWings. We currently offer a few different selections that all have drivers for CircuitPython.

What is displayio?

Displayio is a CircuitPython module that handles drawing to the display from within CircuitPython itself. It has been generally available for most displays since version 4.0 Beta 4. At this point there are displayio drivers available for most of our color displays. Getting started with displayio is very easy. You just need CircuitPython version 4.0 or higher loaded on your CircuitPython board and the appropriate display driver. Since the FeatherWings are the easiest TFT displays to connect we will cover those in this guide.
2.4" TFT FeatherWing

We'll start with the 2.4" TFT FeatherWing which has an ILI9341 display on it. If you would like more information on this display, be sure to check out our Adafruit 2.4" TFT FeatherWing guide (https://adafruit.it/vvE).

Parts

To use this display with displayio, you will only need two main parts. First, you will need the TFT FeatherWing itself.

And second, you will need a Feather such as the Feather M0 Express or the Feather M4 Express. We recommend the Feather M4 Express because it's much faster and works better for driving a display.

For this guide, we'll assume you have a Feather M4 Express. The steps should be about the same for the Feather M0 Express. To start, if you haven't already done so, follow the assembly instructions for the Feather M4 Express in our Feather M4 Express guide (https://adafruit.it/EEm). We'll start by looking at the back of the 2.4" TFT FeatherWing.
After that, it's just a matter of inserting the Feather M4 Express into the back of the TFT FeatherWing.

Required CircuitPython Libraries

To use this display with `displayio`, there is only one required library.

https://adafruit.it/EGe

https://adafruit.it/EGe
First, make sure you are running the latest version of Adafruit CircuitPython (https://adafruit.it/Amd) for your board.

Next, you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafruit.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafruit.it/ABU) for both express and non-express boards.

Remember for non-express boards, you'll need to manually install the necessary libraries from the bundle:

- adafruit_ili9341

Before continuing make sure your board's lib folder or root filesystem has the adafruit_ili9341 file copied over.

Code Example Additional Libraries

For the Code Example, you will need an additional library. We decided to make use of a library so the code didn't get overly complicated.

https://adafruit.it/FiA

Go ahead and install this in the same manner as the driver library by copying the adafruit_display_text folder over to the lib folder on your CircuitPython device.

CircuitPython Code Example
This test will initialize the display using displayio and draw a solid green background, a smaller purple rectangle, and some yellow text. All drawing is done using native displayio modules.

Pinouts are for the 2.4" TFT FeatherWing or Breakout with a Feather M4 or M0.

```python
import board
import terminalio
import displayio
from adafruit_display_text import label
import adafruit_ili9341

# Release any resources currently in use for the displays
displayio.release_displays()

spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10

display_bus = displayio.FourWire(
    spi, command=tft_dc, chip_select=tft_cs, reset=board.D6
)
display = adafruit_ili9341.ILI9341(display_bus, width=320, height=240)

# Make the display context
splash = displayio.Group(max_size=10)
display.show(splash)

# Draw a green background
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00  # Bright Green
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)
splash.append(bg_sprite)

# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(280, 200, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088  # Purple
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=20, y=20)
splash.append(inner_sprite)

# Draw a label
text_group = displayio.Group(max_size=10, scale=3, x=57, y=120)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
text_group.append(text_area)  # Subgroup for text scaling
displayio.release_displays()
while True:
    pass
```

Code Details
Let's take a look at the sections of code one by one. We start by importing the board so that we can initialize SPI, displayio, terminalio for the font, a label, and the adafruit_ili9341 driver.

```python
import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_ili9341
```

Next we release any previously used displays. This is important because if the Feather is reset, the display pins are not automatically released and this makes them available for use again.

```python
displayio.release_displays()
```

Next, we set the SPI object to the board's SPI with the easy shortcut function board.SPI(). By using this function, it finds the SPI module and initializes using the default SPI parameters. Next we set the Chip Select and Data/Command pins that will be used.

```python
spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10
```

In the next line, we set the display bus to FourWire which makes use of the SPI bus. The reset parameter is actually not needed for the FeatherWing, but was added to make it compatible with the breakout displays. You can either leave it or remove it if you need access to an additional GPIO pin.

```python
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=board.D6)
```

Finally, we initialize the driver with a width of 320 and a height of 240. If we stopped at this point and ran the code, we would have a terminal that we could type at and have the screen update.

```python
display = adafruit_ili9341.ILI9341(display_bus, width=320, height=240)
```
Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating the group.

```python
splash = displayio.Group(max_size=10)
display.show(splash)
```

Next we create a Bitmap which is like a canvas that we can draw on. In this case we are creating the Bitmap to be the same size as the screen, but only have one color. The Bitmaps can currently handle up to 256 different colors. We create a Palette with one color and set that color to 0x00FF00 which happens to be green. Colors are Hexadecimal values in the format of RGGGBB. Even though the Bitmaps can only handle 256 colors at a time, you get to define what those 256 different colors are.

```python
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green
```

With all those pieces in place, we create a TileGrid by passing the bitmap and palette and draw it at (0, 0) which represents the display’s upper left.

```python
bg_sprite = displayio.TileGrid(color_bitmap,
                               pixel_shader=color_palette,
                               x=0, y=0)
splash.append(bg_sprite)
```

This creates a solid green background which we will draw on top of.
Next we will create a smaller purple rectangle. The easiest way to do this is to create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case we will create a bitmap that is 20 pixels smaller on each side. The screen is 320x240, so we'll want to subtract 40 from each of those numbers.

We'll also want to place it at the position \((20, 20)\) so that it ends up centered.

```python
inner_bitmap = displayio.Bitmap(280, 200, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=20, y=20)
splash.append(inner_sprite)
```

Since we are adding this after the first rectangle, it's automatically drawn on top. Here's what it looks like now.
Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font and scale it up by a factor of three. To scale the label only, we will make use of a subgroup, which we will then add to the main group.

Labels are centered vertically, so we'll place it at 120 for the Y coordinate, and around 57 pixels make it appear to be centered horizontally, but if you want to change the text, change this to whatever looks good to you. Let's go with some yellow text, so we'll pass it a value of 0xFFFF00.

```python
text_group = displayio.Group(max_size=10, scale=3, x=57, y=120)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
text_group.append(text_area) # Subgroup for text scaling
splash.append(text_group)
```

Finally, we place an infinite loop at the end so that the graphics screen remains in place and isn't replaced by a terminal.

```python
while True:
    pass
```
Using Touch

We won't be covering how to use the touchscreen on the shield with CircuitPython in this guide, but the library required for enabling resistive touch is the Adafruit_CircuitPython_STMPE610 (https://adafruit.it/Fsz) library.

Where to go from here

Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafruit.it/EGh).
3.5" TFT FeatherWing

The steps to get the 3.5" TFT FeatherWing, which has an HX8357 display on it, are very similar to the 2.4" TFT FeatherWing. If you would like more information on this display, be sure to check out our Adafruit 3.5" TFT FeatherWing guide (https://adafruit.it/EEF).

Parts

To use this display with displayio, you will only need two main parts. First, you will need the TFT FeatherWing itself. One thing to note with this display is because of the increased resolution, updates will be a little slower for this display than for the other displays.

And second, you will need a Feather such as the Feather M0 Express or the Feather M4 Express. We recommend the Feather M4 Express because it's much faster and works better for driving a display.

For this guide, we'll assume you have a Feather M4 Express. The steps should be about the same for the Feather M0 Express. To start, if you haven't already done so, follow the assembly instructions for the Feather M4 Express in our Feather M4 Express guide (https://adafruit.it/EEm). We'll start by looking at the back of the 3.5" TFT FeatherWing.
After that, it's just a matter of inserting the Feather M4 Express into the back of the TFT FeatherWing.

Required CircuitPython Libraries

To use this display with `displayio`, there is only one required library.

https://adafruit.it/EGf
First, make sure you are running the latest version of Adafruit CircuitPython (https://adafruit.it/Amd) for your board.

Next, you'll need to install the necessary libraries to use the hardware—carefully follow the steps to find and install these libraries from Adafruit’s CircuitPython library bundle (https://adafruit.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafruit.it/ABU) for both express and non-express boards.

Remember for non-express boards, you'll need to manually install the necessary libraries from the bundle:

- adafruit_hx8357

Before continuing make sure your board's lib folder or root filesystem has the adafruit_hx8357 file copied over.

**Code Example Additional Libraries**

For the Code Example, you will need an additional library. We decided to make use of a library so the code didn't get overly complicated.

Go ahead and install this in the same manner as the driver library by copying the adafruit_display_text folder over to the lib folder on your CircuitPython device.

**CircuitPython Code Example**
This test will initialize the display using displayio and draw a solid green background, a smaller purple rectangle, and some yellow text.

```python
import board
import terminalio
import displayio
from adafruit_display_text import label
from adafruit_hx8357 import HX8357

# Release any resources currently in use for the displays
displayio.release_displays()

spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10

display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs)

display = HX8357(display_bus, width=480, height=320)

# Make the display context
splash = displayio.Group(max_size=10)
display.show(splash)

color_bitmap = displayio.Bitmap(480, 320, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00  # Bright Green

bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)
splash.append(bg_sprite)

# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(440, 280, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088  # Purple
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=20, y=20)
splash.append(inner_sprite)

# Draw a label

text_group = displayio.Group(max_size=10, scale=3, x=137, y=160)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
text_group.append(text_area)  # Subgroup for text scaling
splash.append(text_group)

while True:
    pass
```

Let's take a look at the sections of code one by one. We start by importing the board so that we can initialize SPI, displayio, terminalio for the font, a label, and the adafruit_hx8357 driver.
Next we release any previously used displays. This is important because if the Feather is reset, the display pins are not automatically released and this makes them available for use again.

```python
displayio.release_displays()
```

Next, we set the SPI object to the board's SPI with the easy shortcut function `board.SPI()`. By using this function, it finds the SPI module and initializes using the default SPI parameters. Next we set the Chip Select and Data/Command pins that will be used.

```python
spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10
```

In the next line, we set the display bus to FourWire which makes use of the SPI bus.

```python
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs)
```

Finally, we initialize the driver with a width of 480 and a height of 320. If we stopped at this point and ran the code, we would have a terminal that we could type at and have the screen update.

```python
display = HX8357(display_bus, width=480, height=320)
```
Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating the group.

```python
splash = displayio.Group(max_size=10)
display.show(splash)
```

After that we create a Bitmap which is like a canvas that we can draw on. In this case we are creating the Bitmap to be the same size as the screen, but only have one color. The Bitmaps can currently handle up to 256 different colors. We create a Palette with one color and set that color to 0x00FF00 which happens to be green. Colors are Hexadecimal values in the format of RRGGBB. Even though the Bitmaps can only handle 256 colors at a time, you get to define what those 256 different colors are.

```python
color_bitmap = displayio.Bitmap(480, 320, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green
```

With all those pieces in place, we create a TileGrid by passing the bitmap and palette and draw it at (0, 0) which represents the display’s upper left.

```python
bg_sprite = displayio.TileGrid(color_bitmap,
                              pixel_shader=color_palette,
                              x=0, y=0)
splash.append(bg_sprite)
```

This creates a solid green background which we will draw on top of.

![Green Background](https://learn.adafruit.com/using-circuitpython-displayio-with-a-tft-featherwing)

Next we will create a smaller purple rectangle. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case we will create a bitmap...
that is 20 pixels smaller on each side. The screen is 480x320, so we'll want to subtract 40 from each of those numbers.

We'll also want to place it at the position \((20, 20)\) so that it ends up centered.

```python
# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(440, 280, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
       pixel_shader=inner_palette,
       x=20, y=20)
splash.append(inner_sprite)
```

Since we are adding this after the first rectangle, it's automatically drawn on top. Here's what it looks like now.

Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font and scale it up by a factor of three. To scale the label only, we will make use of a subgroup, which we will then add to the main group.

Labels are centered vertically, so we'll place it at 160 for the Y coordinate, and around 137 pixels make it appear to be centered horizontally, but if you want to change the text, change this to whatever looks good to you. Let's go with some yellow text, so we'll pass it a value of \(0xFFFF00\).

```python
# Draw a label
text_group = displayio.Group(max_size=10, scale=3, x=137, y=160)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
text_group.append(text_area) # Subgroup for text scaling
splash.append(text_group)
```
Finally, we place an infinite loop at the end so that the graphics screen remains in place and isn't replaced by a terminal.

```python
while True:
    pass
```

Using Touch

We won't be covering how to use the touchscreen on the shield with CircuitPython in this guide, but the library required for enabling resistive touch is the Adafruit_CircuitPython_STMPE610 (https://adafruit.it/Fsz) library.

Where to go from here

Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafruit.it/EGh)
Mini Color TFT with Joystick FeatherWing

The Mini Color TFT with Joystick FeatherWing is a fun piece of hardware. In addition to a miniature TFT display, it also has a Joystick and two buttons. This could be used to make a miniature gaming system, MP3 Player and much more. If you would like more information on this display, be sure to check out our Adafruit Mini TFT with Joystick FeatherWing guide (https://adafruit.it/EGi). To use this display with displayio, you will need at least two main parts. First, you will need the TFT FeatherWing itself.

And second, you will need a Feather such as the Feather M0 Express or the Feather M4 Express. We recommend the Feather M4 Express because it’s much faster and works better for driving a display.

To connect the two, you have several options. One of the easiest ways is with the Doubler or Tripler which allows you to insert both items side-by-side. You can find out more about these including assembly information in our FeatherWing Proto, Doubler and Tripler guide (https://adafruit.it/EGj). If you do decide to go with the doubler, be aware that it comes with one set of female headers and one of the set of stacking header, so you may want to order an extra set of female headers with it or you could just trim off the extra pins with some flush cutters.
And finally, if you would like to go with a small profile, you could just go with either a set of female headers or stacking headers. In this guide we decided to just use a doubler with the extra female headers. Let's start by assembling the pieces.

Start with the doubler facing up.
Insert the Mini Color TFT with Joystick FeatherWing into the doubler.

Insert the Feather into the doubler. You're done! Wasn't that easy?

Required CircuitPython Libraries

Since this is no ordinary display and has some additional controls, we have written a special module for our FeatherWing Helper library that not only initializes the display for us, it also initializes the joystick and buttons. You will need the following libraries.

https://adafruit.it/EGk
https://adafruit.it/EGk
First, make sure you are running the latest version of Adafruit CircuitPython (https://adafruit.it/Amd) for your board.

Next, you'll need to install the necessary libraries to use the hardware—carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafruit.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafruit.it/ABU) for both express and non-express boards.

Remember for non-express boards, you'll need to manually install the necessary libraries from the bundle:

- adafruit_st7735r
- adafruit_featherwing
- adafruit_seesaw
- adafruit_bus_device

Before continuing make sure your board's lib folder or root filesystem has the adafruit_st7735r, adafruit_featherwing, adafruit_seesaw and adafruit_bus_device files and folders copied over.

CircuitPython Code Example

Let's take a look at the simpletest for the FeatherWing library.
This example display a CircuitPython console and 
print which button that is being pressed if any

```
import time
from adafruit_featherwing import minitft_featherwing

minitft = minitft_featherwing.MiniTFTFeatherWing()

while True:
    buttons = minitft.buttons

    if buttons.right:
        print("Button RIGHT!")

    if buttons.down:
        print("Button DOWN!")

    if buttons.left:
        print("Button LEFT!")

    if buttons.up:
        print("Button UP!")

    if buttons.select:
        print("Button SELECT!")

    if buttons.a:
        print("Button A!")

    if buttons.b:
        print("Button B!")

    time.sleep(0.001)
```

The first thing you might notice about this library is that compared to the other simpletests, it's quite small. Let's go 
through it a little at a time. First we start with the importing time so we can include a small delay at the end as well as the minitft_featherwing library itself.

````
import time
from adafruit_featherwing import minitft_featherwing
```

Next we initialize the library and leave all the defaults. The FeatherWing allows you to change the DC and CS pins as well as change the address in case you have multiple displays. If you have changed either of these, you can pass the overriding values as parameters when initializing the library.

````
minitft = minitft_featherwing.MiniTFTFeatherWing()
```

In the main loop we save the current state of all the buttons into a variable called buttons. From there we check if each button is \texttt{True} and print out a message.
while True:
    buttons = minitft.buttons
    if buttons.right:
        print("Button RIGHT!")
    if buttons.down:
        print("Button DOWN!")
    if buttons.left:
        print("Button LEFT!")
    if buttons.up:
        print("Button UP!")
    if buttons.select:
        print("Button SELECT!")
    if buttons.a:
        print("Button A!")
    if buttons.b:
        print("Button B!")
    time.sleep(.001)

Go ahead and run the program and try pushing some of the buttons. It displays what you pushed right on the display.

Where to go from here

Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafruit.it/EGh)