
Adafruit HT16K33 Library Documentation

Release 1.0

Radomir Dopieralski

Mar 10, 2020

Contents

1	Dependencies	3
2	Installing from PyPI	5
3	Usage Example	7
4	Contributing	9
5	Documentation	11
6	Table of Contents	13
6.1	Simple test	13
6.2	adafruit_ht16k33.ht16k33	23
6.3	Matrix Displays	24
6.4	Segment Displays	25
7	Indices and tables	27
	Python Module Index	29
	Index	31

This is a library for using the I²C-based LED matrices with the HT16K33 chip. It supports both 16x8 and 8x8 matrices, as well as 7- and 14-segment displays.

- **Notes**

1. This library is intended for Adafruit CircuitPython's API. For a library compatible with MicroPython machine API see this [library](#).
2. This library does not work with the Trellis 4x4 LED+Keypad board. For that product use: [CircuitPython Trellis Library](#)

CHAPTER 1

Dependencies

This driver depends on:

- [Adafruit CircuitPython](#)
- [Bus Device](#)

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the [Adafruit library and driver bundle](#).

CHAPTER 2

Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally [from PyPI](#). To install for current user:

```
pip3 install adafruit-circuitpython-ht16k33
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-ht16k33
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-ht16k33
```


CHAPTER 3

Usage Example

```
# Import all board pins and bus interface.
import board
import busio

# Import the HT16K33 LED matrix module.
from adafruit_ht16k33 import matrix

# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)

# Create the matrix class.
# This creates a 16x8 matrix:
matrix = matrix.Matrix16x8(i2c)
# Or this creates a 8x8 matrix:
#matrix = matrix.Matrix8x8(i2c)
# Or this creates a 8x8 bicolor matrix:
#matrix = matrix.Matrix8x8x2
# Finally you can optionally specify a custom I2C address of the HT16k33 like:
#matrix = matrix.Matrix16x8(i2c, address=0x70)

# Clear the matrix.
matrix.fill(0)

# Set a pixel in the origin 0,0 position.
matrix[0, 0] = 1
# Set a pixel in the middle 8, 4 position.
matrix[8, 4] = 1
# Set a pixel in the opposite 15, 7 position.
matrix[15, 7] = 1
matrix.show()

# Change the brightness
matrix.brightness = 8
```

(continues on next page)

(continued from previous page)

```
# Set the blink rate
matrix.blink_rate = 2
```

CHAPTER 4

Contributing

Contributions are welcome! Please read our [Code of Conduct](#) before contributing to help this project stay welcoming.

CHAPTER 5

Documentation

For information on building library documentation, please check out [this guide](#).

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/ht16k33_matrix_simpletest.py

```
1  # Basic example of clearing and drawing a pixel on a LED matrix display.
2  # This example and library is meant to work with Adafruit CircuitPython API.
3  # Author: Tony DiCola
4  # License: Public Domain
5
6  # Import all board pins.
7  import time
8  import board
9  import busio
10
11 # Import the HT16K33 LED matrix module.
12 from adafruit_ht16k33 import matrix
13
14
15 # Create the I2C interface.
16 i2c = busio.I2C(board.SCL, board.SDA)
17
18 # Create the matrix class.
19 # This creates a 16x8 matrix:
20 matrix = matrix.Matrix16x8(i2c)
21 # Or this creates a 16x8 matrix backpack:
22 #matrix = matrix.MatrixBackpack16x8(i2c)
23 # Or this creates a 8x8 matrix:
24 #matrix = matrix.Matrix8x8(i2c)
25 # Or this creates a 8x8 bicolor matrix:
26 #matrix = matrix.Matrix8x8x2(i2c)
27 # Finally you can optionally specify a custom I2C address of the HT16k33 like:
```

(continues on next page)

(continued from previous page)

```

28 #matrix = matrix.Matrix16x8(i2c, address=0x70)
29
30 # Clear the matrix.
31 matrix.fill(0)
32
33 # Set a pixel in the origin 0, 0 position.
34 matrix[0, 0] = 1
35 # Set a pixel in the middle 8, 4 position.
36 matrix[8, 4] = 1
37 # Set a pixel in the opposite 15, 7 position.
38 matrix[15, 7] = 1
39
40 time.sleep(2)
41
42 # Draw a Smiley Face
43 matrix.fill(0)
44
45 for row in range(2, 6):
46     matrix[row, 0] = 1
47     matrix[row, 7] = 1
48
49 for column in range(2, 6):
50     matrix[0, column] = 1
51     matrix[7, column] = 1
52
53 matrix[1, 1] = 1
54 matrix[1, 6] = 1
55 matrix[6, 1] = 1
56 matrix[6, 6] = 1
57 matrix[2, 5] = 1
58 matrix[5, 5] = 1
59 matrix[2, 3] = 1
60 matrix[5, 3] = 1
61 matrix[3, 2] = 1
62 matrix[4, 2] = 1
63
64 # Move the Smiley Face Around
65 while True:
66     for frame in range(0, 8):
67         matrix.shift_right(True)
68         time.sleep(0.05)
69     for frame in range(0, 8):
70         matrix.shift_down(True)
71         time.sleep(0.05)
72     for frame in range(0, 8):
73         matrix.shift_left(True)
74         time.sleep(0.05)
75     for frame in range(0, 8):
76         matrix.shift_up(True)
77         time.sleep(0.05)

```

Listing 2: examples/ht16k33_segments_simpletest.py

```

1 # Basic example of setting digits on a LED segment display.
2 # This example and library is meant to work with Adafruit CircuitPython API.
3 # Author: Tony DiCola

```

(continues on next page)

(continued from previous page)

```

4  # License: Public Domain
5
6  import time
7
8  # Import all board pins.
9  import board
10 import busio
11
12 # Import the HT16K33 LED segment module.
13 from adafruit_ht16k33 import segments
14
15 # Create the I2C interface.
16 i2c = busio.I2C(board.SCL, board.SDA)
17
18 # Create the LED segment class.
19 # This creates a 7 segment 4 character display:
20 display = segments.Seg7x4(i2c)
21 # Or this creates a 14 segment alphanumeric 4 character display:
22 #display = segments.Seg14x4(i2c)
23 # Or this creates a big 7 segment 4 character display
24 #display = segments.BigSeg7x4(i2c)
25 # Finally you can optionally specify a custom I2C address of the HT16k33 like:
26 #display = segments.Seg7x4(i2c, address=0x70)
27
28 # Clear the display.
29 display.fill(0)
30
31 # Can just print a number
32 display.print(42)
33 time.sleep(2)
34
35 # Or, can print a hexadecimal value
36 display.print_hex(0xFF23)
37 time.sleep(2)
38
39 # Or, print the time
40 display.print("12:30")
41 time.sleep(2)
42
43 display.colon = False
44
45 # Or, can set individual digits / characters
46 # Set the first character to '1':
47 display[0] = '1'
48 # Set the second character to '2':
49 display[1] = '2'
50 # Set the third character to 'A':
51 display[2] = 'A'
52 # Set the forth character to 'B':
53 display[3] = 'B'
54 time.sleep(2)
55
56 # Or, can even set the segments to make up characters
57 if isinstance(display, segments.Seg7x4):
58     # 7-segment raw digits
59     display.set_digit_raw(0, 0xFF)
60     display.set_digit_raw(1, 0b11111111)

```

(continues on next page)

(continued from previous page)

```

61     display.set_digit_raw(2, 0x79)
62     display.set_digit_raw(3, 0b01111001)
63 else:
64     # 14-segment raw digits
65     display.set_digit_raw(0, 0x2D3F)
66     display.set_digit_raw(1, 0b0010110100111111)
67     display.set_digit_raw(2, (0b00101101, 0b00111111))
68     display.set_digit_raw(3, [0x2D, 0x3F])
69 time.sleep(2)
70
71 #Show a looping marquee
72 display.marquee('Deadbeef 192.168.100.102... ', 0.2)

```

Listing 3: examples/ht16k33_bicolor24_simpletest.py

```

1  # Basic example of using the Bi-color 24 segment bargraph display.
2  # This example and library is meant to work with Adafruit CircuitPython API.
3  # Author: Carter Nelson
4  # License: Public Domain
5
6  import time
7
8  # Import board related modules
9  import board
10 import busio
11
12 # Import the Bicolor24 driver from the HT16K33 module
13 from adafruit_ht16k33.bargraph import Bicolor24
14
15 # Create the I2C interface
16 i2c = busio.I2C(board.SCL, board.SDA)
17
18 # Create the LED bargraph class.
19 bc24 = Bicolor24(i2c)
20
21 # Set individual segments of bargraph
22 bc24[0] = bc24.LED_RED
23 bc24[1] = bc24.LED_GREEN
24 bc24[2] = bc24.LED_YELLOW
25
26 time.sleep(2)
27
28 # Turn them all off
29 bc24.fill(bc24.LED_OFF)
30
31 # Turn them on in a loop
32 for i in range(24):
33     bc24[i] = bc24.LED_RED
34     time.sleep(0.1)
35     bc24[i] = bc24.LED_OFF
36
37 time.sleep(1)
38
39 # Fill the entire bargraph
40 bc24.fill(bc24.LED_GREEN)

```

Listing 4: examples/ht16k33_matrix_pillow_image.py

```

1  # Basic example of drawing an image
2  # This example and library is meant to work with Adafruit CircuitPython API.
3  #
4  # This example is for use on (Linux) computers that are using CPython with
5  # Adafruit Blinka to support CircuitPython libraries. CircuitPython does
6  # not support PIL/pillow (python imaging library)!
7  #
8  # Author: Melissa LeBlanc-Williams
9  # License: Public Domain
10
11 # Import all board pins.
12 import board
13 import busio
14 from PIL import Image
15
16 # Import the HT16K33 LED matrix module.
17 from adafruit_ht16k33 import matrix
18
19 # Create the I2C interface.
20 i2c = busio.I2C(board.SCL, board.SDA)
21
22 # Create the matrix class.
23 # This creates a 16x8 matrix:
24 mtrx = matrix.Matrix16x8(i2c)
25 # Or this creates a 16x8 matrix backpack:
26 #mtrx = matrix.MatrixBackpack16x8(i2c)
27 # Or this creates a 8x8 matrix:
28 #mtrx = matrix.Matrix8x8(i2c)
29 # Or this creates a 8x8 bicolor matrix:
30 #mtrx = matrix.Matrix8x8x2(i2c)
31 # Finally you can optionally specify a custom I2C address of the HT16k33 like:
32 #mtrx = matrix.Matrix16x8(i2c, address=0x70)
33
34 if isinstance(mtrx, matrix.Matrix8x8x2):
35     image = Image.open("squares-color.png")
36 elif isinstance(mtrx, matrix.Matrix16x8):
37     image = Image.open("squares-mono-16x8.png")
38 else:
39     image = Image.open("squares-mono-8x8.png")
40
41 # Clear the matrix
42 mtrx.fill(0)
43 mtrx.image(image)

```

Listing 5: examples/ht16k33_animation_demo.py

```

1  """
2      Test script for display animations on an HT16K33 with alphanumeric display
3
4      The display must be initialized with auto_write=False.
5  """
6
7  from time import sleep
8  import board
9  import busio

```

(continues on next page)

(continued from previous page)

```

10 from adafruit_ht16k33.segments import Seg14x4
11
12 #
13 #   Segment bits on the HT16K33 with alphanumeric display.
14 #
15 #   Add the values of the segments you need to create a bitmask
16 #
17
18 N = 16384
19 M = 8192
20 L = 4096
21 K = 2048
22 J = 1024
23 I = 512
24 H = 256
25 G2 = 128
26 G1 = 64
27 F = 32
28 E = 16
29 D = 8
30 C = 4
31 B = 2
32 A = 1
33
34 #   The number of seconds to delay between writing segments
35 DEFAULT_CHAR_DELAY_SEC = 0.2
36
37 #   The number of cycles to go for each animation
38 DEFAULT_CYCLES = 5
39
40 #   Brightness of the display (0 to 15)
41 DEFAULT_DISPLAY_BRIGHTNESS = 0.3
42
43 #   Initialize the I2C bus
44 i2c = busio.I2C(board.SCL, board.SDA)
45
46 #   Initialize the HT16K33 with alphanumeric display featherwing.
47 #
48 #   You MUST set auto_write=False
49 display = Seg14x4(i2c, auto_write=False)
50 display.brightness = DEFAULT_DISPLAY_BRIGHTNESS
51
52 def animate(digits, bitmasks, delay=DEFAULT_CHAR_DELAY_SEC, auto_write=True):
53     '''
54     Main driver for all alphanumeric display animations (WIP!!!)
55     Param: digits - a list of the digits to write to, in order, like [0, 1, 3].
56     ↳ The digits are
57         0 to 3 starting at the left most digit.
58     Param: bitmasks - a list of the bitmasks to write, in sequence, to the
59     ↳ specified digits.
60     Param: delay - The delay, in seconds (or fractions of), between writing
61     ↳ bitmasks to a digit.
62     Param: auto_write - Whether to actually write to the display immediately or
63     ↳ not.
64
65     Returns: Nothing
66     '''

```

(continues on next page)

(continued from previous page)

```

63     if not isinstance(digits, list):
64         raise ValueError("The first parameter MUST be a list!")
65     elif not isinstance(bitmasks, list):
66         raise ValueError("The second parameter MUST be a list!")
67     elif delay < 0:
68         raise ValueError("The delay between frames must be positive!")
69     else:
70         for dig in digits:
71             if not 0 <= dig <= 3:
72                 raise ValueError('Digit value must be \
73                     an integer in the range: 0-3')
74
75             for bits in bitmasks:
76                 if not 0 <= bits <= 0xFFFF:
77                     raise ValueError('Bitmask value must be an \
78                         integer in the range: 0-65535')
79
80                 display.set_digit_raw(dig, bits)
81
82                 if auto_write:
83                     display.show()
84                     sleep(delay)
85
86 def chase_forward_and_reverse(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
87     cy = 0
88
89     while cy < cycles:
90         animate([0, 1, 2, 3], [A, 0], delay)
91         animate([3], [B, C, D, 0], delay)
92         animate([2, 1, 0], [D, 0], delay)
93         animate([0], [E, F, H, G2, 0], delay)
94         animate([1, 2], [G1, G2, 0], delay)
95         animate([3], [G1, J, A, 0], delay)
96         animate([2, 1], [A, 0], delay)
97         animate([0], [A, F, E, D, 0], delay)
98         animate([1, 2], [D, 0], delay)
99         animate([3], [D, C, B, J, G1, 0], delay)
100        animate([2, 1], [G2, G1, 0], delay)
101        animate([0], [H, 0], delay)
102
103        cy += 1
104
105 def prelude_to_spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
106     cy = 0
107     auto_write = False
108
109     while cy < cycles:
110         animate([1, 2], [A], 0, auto_write)
111         display.show()
112         sleep(delay)
113
114         animate([0, 3], [A], 0, auto_write)
115         display.show()
116         sleep(delay)
117
118         animate([0], [A+F], 0, auto_write)
119         animate([3], [A+B], 0, auto_write)

```

(continues on next page)

(continued from previous page)

```

120     display.show()
121     sleep(delay)
122
123     animate([0], [A+E+F], 0, auto_write)
124     animate([3], [A+B+C], 0, auto_write)
125     display.show()
126     sleep(delay)
127
128     animate([0], [A+D+E+F], 0, auto_write)
129     animate([3], [A+B+C+D], 0, auto_write)
130     display.show()
131     sleep(delay)
132
133     animate([1], [A+D], 0, auto_write)
134     animate([2], [A+D], 0, auto_write)
135     display.show()
136     sleep(delay)
137
138     animate([1], [A+D+M], 0, auto_write)
139     animate([2], [A+D+K], 0, auto_write)
140     display.show()
141     sleep(delay)
142
143     animate([1], [A+D+M+H], 0, auto_write)
144     animate([2], [A+D+K+J], 0, auto_write)
145     display.show()
146     sleep(delay)
147
148     animate([0], [A+E+F+J+D], 0, auto_write)
149     animate([3], [A+B+C+H+D], 0, auto_write)
150     display.show()
151     sleep(delay)
152
153     animate([0], [A+E+F+J+K+D], 0, auto_write)
154     animate([3], [A+B+C+H+M+D], 0, auto_write)
155     display.show()
156     sleep(delay)
157
158     display.fill(0)
159     display.show()
160     sleep(delay)
161
162     cy += 1
163
164 def spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
165     cy = 0
166     auto_write = False
167
168     while cy < cycles:
169         animate([0], [H+M], 0, auto_write)
170         animate([1], [J+K], 0, auto_write)
171         animate([2], [H+M], 0, auto_write)
172         animate([3], [J+K], 0, auto_write)
173         display.show()
174         sleep(delay)
175
176         animate([0], [G1+G2], 0, auto_write)

```

(continues on next page)

(continued from previous page)

```

177     animate([1], [G1+G2], 0, auto_write)
178     animate([2], [G1+G2], 0, auto_write)
179     animate([3], [G1+G2], 0, auto_write)
180     display.show()
181     sleep(delay)
182
183     animate([0], [J+K], 0, auto_write)
184     animate([1], [H+M], 0, auto_write)
185     animate([2], [J+K], 0, auto_write)
186     animate([3], [H+M], 0, auto_write)
187     display.show()
188     sleep(delay)
189
190     cy += 1
191
192     display.fill(0)
193
194 def enclosed_spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
195     cy = 0
196     auto_write = False
197
198     while cy < cycles:
199         animate([0], [A+D+E+F+H+M], 0, auto_write)
200         animate([1], [A+D+J+K], 0, auto_write)
201         animate([2], [A+D+H+M], 0, auto_write)
202         animate([3], [A+B+C+D+J+K], 0, auto_write)
203         display.show()
204         sleep(delay)
205
206         animate([0], [A+D+E+F+G1+G2], 0, auto_write)
207         animate([1], [A+D+G1+G2], 0, auto_write)
208         animate([2], [A+D+G1+G2], 0, auto_write)
209         animate([3], [A+B+C+D+G1+G2], 0, auto_write)
210         display.show()
211         sleep(delay)
212
213         animate([0], [A+D+E+F+J+K], 0, auto_write)
214         animate([1], [A+D+H+M], 0, auto_write)
215         animate([2], [A+D+J+K], 0, auto_write)
216         animate([3], [A+B+C+D+H+M], 0, auto_write)
217         display.show()
218         sleep(delay)
219
220         cy += 1
221
222     display.fill(0)
223
224 def count_down():
225     auto_write = False
226     numbers = [ [A+B+C+D+G1+G2+N], [A+B+D+E+G1+G2+N], [B+C+N] ]
227     index = 0
228
229     display.fill(0)
230
231     while index < len(numbers):
232         animate([index], numbers[index], 0, auto_write)
233         display.show()

```

(continues on next page)

(continued from previous page)

```

234     sleep(1)
235     display.fill(0)
236     sleep(0.5)
237
238     index += 1
239
240     sleep(1)
241     display.fill(0)
242
243 try:
244     text = "Init"
245
246     display.fill(1)
247     display.show()
248     sleep(1)
249     display.fill(0)
250     display.show()
251
252     display.print(text)
253     display.show()
254     sleep(2)
255     display.fill(0)
256     display.show()
257     sleep(1)
258
259     count_down()
260     sleep(0.2)
261
262     text = "Go!!"
263
264     display.print(text)
265     display.show()
266     sleep(1.5)
267     display.fill(0)
268     display.show()
269     sleep(0.5)
270     print()
271
272 while True:
273     # Arrow
274     print("Arrow")
275     animate([0, 1, 2], [G1+G2], 0.1)
276     animate([3], [G1+H+K], 0.1)
277     sleep(1.0)
278     display.fill(0)
279     sleep(1.0)
280
281     # Flying
282     print("Flying")
283     cyc = 0
284
285     while cyc < DEFAULT_CYCLES:
286         animate([0], [H+J, G1+G2, K+M, G1+G2], DEFAULT_CHAR_DELAY_SEC)
287
288         cyc += 1
289
290     animate([0], [0])

```

(continues on next page)

(continued from previous page)

```

291     sleep(1.0)
292     display.fill(0)
293     sleep(1.0)
294
295     # Chase forward and reverse.
296     print("Chase forward and reverse")
297     chase_forward_and_reverse(0.01, 5)
298     sleep(1.0)
299     display.fill(0)
300     sleep(1.0)
301
302     # Testing writing to more than one segment simultaneously
303     print("Prelude to Spinners")
304     prelude_to_spinners(0.1, 5)
305     sleep(1.0)
306     display.fill(0)
307     display.show()
308     sleep(1.0)
309
310     print("Spinners")
311     spinners(0.1, 20)
312     sleep(1.0)
313     display.fill(0)
314     display.show()
315     sleep(1.0)
316
317
318     print("Enclosed Spinners")
319     enclosed_spinners(0.1, 20)
320     sleep(1.0)
321     display.fill(0)
322     display.show()
323     sleep(1.0)
324
325     print()
326 except KeyboardInterrupt:
327     display.fill(0)
328     display.show()

```

6.2 adafruit_ht16k33.ht16k33

- Authors: Radomir Dopieralski & Tony DiCola for Adafruit Industries

class adafruit_ht16k33.ht16k33.**HT16K33** (*i2c*, *address=112*, *auto_write=True*, *brightness=1.0*)

The base class for all displays. Contains common methods.

Parameters

- **address** (*int*) – The I2C address of the HT16K33.
- **auto_write** (*bool*) – True if the display should immediately change when set. If False, *show* must be called explicitly.
- **brightness** (*float*) – 0.0 - 1.0 default brightness level.

auto_write
Auto write updates to the display.

blink_rate
The blink rate. Range 0-3.

brightness
The brightness. Range 0.0-1.0

fill (*color*)
Fill the whole display with the given color.

show ()
Refresh the display and show the changes.

6.3 Matrix Displays

class adafruit_ht16k33.matrix.**Matrix16x8** (*i2c*, *address=112*, *auto_write=True*, *brightness=1.0*)
The matrix wing.

pixel (*x*, *y*, *color=None*)
Get or set the color of a given pixel.

class adafruit_ht16k33.matrix.**Matrix8x8** (*i2c*, *address=112*, *auto_write=True*, *brightness=1.0*)
A single matrix.

columns
Read-only property for number of columns

image (*img*)
Set buffer to value of Python Imaging Library image. The image should be in 1 bit mode and a size equal to the display size.

pixel (*x*, *y*, *color=None*)
Get or set the color of a given pixel.

rows
Read-only property for number of rows

shift (*x*, *y*, *rotate=False*)
Shift pixels by x and y

Parameters rotate – (Optional) Rotate the shifted pixels to the left side (default=False)

shift_down (*rotate=False*)
Shift all pixels down

Parameters rotate – (Optional) Rotate the shifted pixels to top (default=False)

shift_left (*rotate=False*)
Shift all pixels left

Parameters rotate – (Optional) Rotate the shifted pixels to the right side (default=False)

shift_right (*rotate=False*)
Shift all pixels right

Parameters rotate – (Optional) Rotate the shifted pixels to the left side (default=False)

shift_up (*rotate=False*)
Shift all pixels up

Parameters **rotate** – (Optional) Rotate the shifted pixels to bottom (default=False)

class adafruit_ht16k33.matrix.**Matrix8x8x2** (*i2c, address=112, auto_write=True, brightness=1.0*)

A bi-color matrix.

fill (*color*)
Fill the whole display with the given color.

image (*img*)
Set buffer to value of Python Imaging Library image. The image should be a size equal to the display size.

pixel (*x, y, color=None*)
Get or set the color of a given pixel.

class adafruit_ht16k33.matrix.**MatrixBackpack16x8** (*i2c, address=112, auto_write=True, brightness=1.0*)

A double matrix backpack.

pixel (*x, y, color=None*)
Get or set the color of a given pixel.

6.4 Segment Displays

class adafruit_ht16k33.segments.**BigSeg7x4** (*i2c, address=112, auto_write=True*)

Numeric 7-segment display. It has the same methods as the alphanumeric display, but only supports displaying a limited set of characters.

ampm
The AM/PM indicator.

bottom_left_dot
The bottom-left dot indicator.

top_left_dot
The top-left dot indicator.

class adafruit_ht16k33.segments.**Colon** (*disp, num_of_colons=1*)

Helper class for controlling the colons. Not intended for direct use.

class adafruit_ht16k33.segments.**Seg14x4** (*i2c, address=112, auto_write=True, brightness=1.0*)

Alpha-numeric, 14-segment display.

marquee (*text, delay=0.25, loop=True*)
Automatically scroll the text at the specified delay between characters

Parameters

- **text** (*str*) – The text to display
- **delay** (*float*) – (optional) The delay in seconds to pause before scrolling to the next character (default=0.25)
- **loop** (*bool*) – (optional) Whether to endlessly loop the text (default=True)

print (*value, decimal=0*)
Print the value to the display.

print_hex (*value*)

Print the value as a hexadecimal string to the display.

scroll (*count=1*)

Scroll the display by specified number of places.

set_digit_raw (*index, bitmask*)

Set digit at position to raw bitmask value. Position should be a value of 0 to 3 with 0 being the left most character on the display.

bitmask should be 2 bytes such as: 0xFFFF If can be passed as an integer, list, or tuple

class `adafruit_ht16k33.segments.Seg7x4` (*i2c, address=112, auto_write=True*)

Numeric 7-segment display. It has the same methods as the alphanumeric display, but only supports displaying a limited set of characters.

colon

Simplified colon accessor

scroll (*count=1*)

Scroll the display by specified number of places.

set_digit_raw (*index, bitmask*)

Set digit at position to raw bitmask value. Position should be a value of 0 to 3 with 0 being the left most digit on the display.

CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`

a

`adafruit_ht16k33.ht16k33`, [23](#)
`adafruit_ht16k33.matrix`, [24](#)
`adafruit_ht16k33.segments`, [25](#)

A

adafruit_ht16k33.ht16k33 (module), 23
adafruit_ht16k33.matrix (module), 24
adafruit_ht16k33.segments (module), 25
ampm (adafruit_ht16k33.segments.BigSeg7x4 attribute), 25
auto_write (adafruit_ht16k33.ht16k33.HT16K33 attribute), 23

B

BigSeg7x4 (class in adafruit_ht16k33.segments), 25
blink_rate (adafruit_ht16k33.ht16k33.HT16K33 attribute), 24
bottom_left_dot (adafruit_ht16k33.segments.BigSeg7x4 attribute), 25
brightness (adafruit_ht16k33.ht16k33.HT16K33 attribute), 24

C

colon (adafruit_ht16k33.segments.Seg7x4 attribute), 26
Colon (class in adafruit_ht16k33.segments), 25
columns (adafruit_ht16k33.matrix.Matrix8x8 attribute), 24

F

fill() (adafruit_ht16k33.ht16k33.HT16K33 method), 24
fill() (adafruit_ht16k33.matrix.Matrix8x8x2 method), 25

H

HT16K33 (class in adafruit_ht16k33.ht16k33), 23

I

image() (adafruit_ht16k33.matrix.Matrix8x8 method), 24
image() (adafruit_ht16k33.matrix.Matrix8x8x2 method), 25

M

marquee() (adafruit_ht16k33.segments.Seg14x4 method), 25
Matrix16x8 (class in adafruit_ht16k33.matrix), 24
Matrix8x8 (class in adafruit_ht16k33.matrix), 24
Matrix8x8x2 (class in adafruit_ht16k33.matrix), 25
MatrixBackpack16x8 (class in adafruit_ht16k33.matrix), 25

P

pixel() (adafruit_ht16k33.matrix.Matrix16x8 method), 24
pixel() (adafruit_ht16k33.matrix.Matrix8x8 method), 24
pixel() (adafruit_ht16k33.matrix.Matrix8x8x2 method), 25
pixel() (adafruit_ht16k33.matrix.MatrixBackpack16x8 method), 25
print() (adafruit_ht16k33.segments.Seg14x4 method), 25
print_hex() (adafruit_ht16k33.segments.Seg14x4 method), 25

R

rows (adafruit_ht16k33.matrix.Matrix8x8 attribute), 24

S

scroll() (adafruit_ht16k33.segments.Seg14x4 method), 26
scroll() (adafruit_ht16k33.segments.Seg7x4 method), 26
Seg14x4 (class in adafruit_ht16k33.segments), 25
Seg7x4 (class in adafruit_ht16k33.segments), 26
set_digit_raw() (adafruit_ht16k33.segments.Seg14x4 method), 26
set_digit_raw() (adafruit_ht16k33.segments.Seg7x4 method), 26
shift() (adafruit_ht16k33.matrix.Matrix8x8 method), 24

`shift_down()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [24](#)
`shift_left()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [24](#)
`shift_right()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [24](#)
`shift_up()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [24](#)
`show()` (*adafruit_ht16k33.ht16k33.HT16K33* *method*),
[24](#)

T

`top_left_dot` (*adafruit_ht16k33.segments.BigSeg7x4*
attribute), [25](#)