
Adafruit HT16K33 Library Documentation

Release 1.0

Radomir Dopieralski

Mar 03, 2021

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	24
6.3	Matrix Displays	24
6.4	Segment Displays	26
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  # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2  # SPDX-License-Identifier: MIT
3
4  # Basic example of clearing and drawing a pixel on a LED matrix display.
5  # This example and library is meant to work with Adafruit CircuitPython API.
6  # Author: Tony DiCola
7  # License: Public Domain
8
9  # Import all board pins.
10 import time
11 import board
12 import busio
13
14 # Import the HT16K33 LED matrix module.
15 from adafruit_ht16k33 import matrix
16
17
18 # Create the I2C interface.
19 i2c = busio.I2C(board.SCL, board.SDA)
20
21 # Create the matrix class.
22 # This creates a 16x8 matrix:
23 matrix = matrix.Matrix16x8(i2c)
24 # Or this creates a 16x8 matrix backpack:
25 # matrix = matrix.MatrixBackpack16x8(i2c)
26 # Or this creates a 8x8 matrix:
27 # matrix = matrix.Matrix8x8(i2c)
```

(continues on next page)

(continued from previous page)

```

28 # Or this creates a 8x8 bicolor matrix:
29 # matrix = matrix.Matrix8x8x2(i2c)
30 # Finally you can optionally specify a custom I2C address of the HT16k33 like:
31 # matrix = matrix.Matrix16x8(i2c, address=0x70)
32
33 # Clear the matrix.
34 matrix.fill(0)
35
36 # Set a pixel in the origin 0, 0 position.
37 matrix[0, 0] = 1
38 # Set a pixel in the middle 8, 4 position.
39 matrix[8, 4] = 1
40 # Set a pixel in the opposite 15, 7 position.
41 matrix[15, 7] = 1
42
43 time.sleep(2)
44
45 # Draw a Smiley Face
46 matrix.fill(0)
47
48 for row in range(2, 6):
49     matrix[row, 0] = 1
50     matrix[row, 7] = 1
51
52 for column in range(2, 6):
53     matrix[0, column] = 1
54     matrix[7, column] = 1
55
56 matrix[1, 1] = 1
57 matrix[1, 6] = 1
58 matrix[6, 1] = 1
59 matrix[6, 6] = 1
60 matrix[2, 5] = 1
61 matrix[5, 5] = 1
62 matrix[2, 3] = 1
63 matrix[5, 3] = 1
64 matrix[3, 2] = 1
65 matrix[4, 2] = 1
66
67 # Move the Smiley Face Around
68 while True:
69     for frame in range(0, 8):
70         matrix.shift_right(True)
71         time.sleep(0.05)
72     for frame in range(0, 8):
73         matrix.shift_down(True)
74         time.sleep(0.05)
75     for frame in range(0, 8):
76         matrix.shift_left(True)
77         time.sleep(0.05)
78     for frame in range(0, 8):
79         matrix.shift_up(True)
80         time.sleep(0.05)

```

Listing 2: examples/ht16k33_segments_simpletest.py

```

1  # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2  # SPDX-License-Identifier: MIT
3
4  # Basic example of setting digits on a LED segment display.
5  # This example and library is meant to work with Adafruit CircuitPython API.
6  # Author: Tony DiCola
7  # License: Public Domain
8
9  import time
10
11 # Import all board pins.
12 import board
13 import busio
14
15 # Import the HT16K33 LED segment module.
16 from adafruit_ht16k33 import segments
17
18 # Create the I2C interface.
19 i2c = busio.I2C(board.SCL, board.SDA)
20
21 # Create the LED segment class.
22 # This creates a 7 segment 4 character display:
23 display = segments.Seg7x4(i2c)
24 # Or this creates a 14 segment alphanumeric 4 character display:
25 # display = segments.Seg14x4(i2c)
26 # Or this creates a big 7 segment 4 character display
27 # display = segments.BigSeg7x4(i2c)
28 # Finally you can optionally specify a custom I2C address of the HT16k33 like:
29 # display = segments.Seg7x4(i2c, address=0x70)
30
31 # Clear the display.
32 display.fill(0)
33
34 # Can just print a number
35 display.print(42)
36 time.sleep(2)
37
38 # Or, can print a hexadecimal value
39 display.print_hex(0xFF23)
40 time.sleep(2)
41
42 # Or, print the time
43 display.print("12:30")
44 time.sleep(2)
45
46 display.colon = False
47
48 # Or, can set individual digits / characters
49 # Set the first character to '1':
50 display[0] = "1"
51 # Set the second character to '2':
52 display[1] = "2"
53 # Set the third character to 'A':
54 display[2] = "A"
55 # Set the forth character to 'B':

```

(continues on next page)

(continued from previous page)

```

56 display[3] = "B"
57 time.sleep(2)
58
59 # Or, can even set the segments to make up characters
60 if isinstance(display, segments.Seg7x4):
61     # 7-segment raw digits
62     display.set_digit_raw(0, 0xFF)
63     display.set_digit_raw(1, 0b11111111)
64     display.set_digit_raw(2, 0x79)
65     display.set_digit_raw(3, 0b01111001)
66 else:
67     # 14-segment raw digits
68     display.set_digit_raw(0, 0x2D3F)
69     display.set_digit_raw(1, 0b0010110100111111)
70     display.set_digit_raw(2, (0b00101101, 0b00111111))
71     display.set_digit_raw(3, [0x2D, 0x3F])
72 time.sleep(2)
73
74 # Show a looping marquee
75 display.marquee("Deadbeef 192.168.100.102... ", 0.2)

```

Listing 3: examples/ht16k33_bicolor24_simpletest.py

```

1  # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2  # SPDX-License-Identifier: MIT
3
4  # Basic example of using the Bi-color 24 segment bargraph display.
5  # This example and library is meant to work with Adafruit CircuitPython API.
6  # Author: Carter Nelson
7  # License: Public Domain
8
9  import time
10
11 # Import board related modules
12 import board
13 import busio
14
15 # Import the Bicolor24 driver from the HT16K33 module
16 from adafruit_ht16k33.bargraph import Bicolor24
17
18 # Create the I2C interface
19 i2c = busio.I2C(board.SCL, board.SDA)
20
21 # Create the LED bargraph class.
22 bc24 = Bicolor24(i2c)
23
24 # Set individual segments of bargraph
25 bc24[0] = bc24.LED_RED
26 bc24[1] = bc24.LED_GREEN
27 bc24[2] = bc24.LED_YELLOW
28
29 time.sleep(2)
30
31 # Turn them all off
32 bc24.fill(bc24.LED_OFF)
33

```

(continues on next page)

(continued from previous page)

```

34 # Turn them on in a loop
35 for i in range(24):
36     bc24[i] = bc24.LED_RED
37     time.sleep(0.1)
38     bc24[i] = bc24.LED_OFF
39
40 time.sleep(1)
41
42 # Fill the entire bargraph
43 bc24.fill(bc24.LED_GREEN)

```

Listing 4: examples/ht16k33_matrix_pillow_image.py

```

1  # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2  # SPDX-License-Identifier: MIT
3
4  # Basic example of drawing an image
5  # This example and library is meant to work with Adafruit CircuitPython API.
6  #
7  # This example is for use on (Linux) computers that are using CPython with
8  # Adafruit Blinka to support CircuitPython libraries. CircuitPython does
9  # not support PIL/pillow (python imaging library)!
10 #
11 # Author: Melissa LeBlanc-Williams
12 # License: Public Domain
13
14 # Import all board pins.
15 import board
16 import busio
17 from PIL import Image
18
19 # Import the HT16K33 LED matrix module.
20 from adafruit_ht16k33 import matrix
21
22 # Create the I2C interface.
23 i2c = busio.I2C(board.SCL, board.SDA)
24
25 # Create the matrix class.
26 # This creates a 16x8 matrix:
27 mtrx = matrix.Matrix16x8(i2c)
28 # Or this creates a 16x8 matrix backpack:
29 # mtrx = matrix.MatrixBackpack16x8(i2c)
30 # Or this creates a 8x8 matrix:
31 # mtrx = matrix.Matrix8x8(i2c)
32 # Or this creates a 8x8 bicolor matrix:
33 # mtrx = matrix.Matrix8x8x2(i2c)
34 # Finally you can optionally specify a custom I2C address of the HT16k33 like:
35 # mtrx = matrix.Matrix16x8(i2c, address=0x70)
36
37 if isinstance(mtrx, matrix.Matrix8x8x2):
38     image = Image.open("squares-color.png")
39 elif isinstance(mtrx, matrix.Matrix16x8):
40     image = Image.open("squares-mono-16x8.png")
41 else:
42     image = Image.open("squares-mono-8x8.png")
43

```

(continues on next page)

(continued from previous page)

```
44 # Clear the matrix
45 mtrx.fill(0)
46 mtrx.image(image)
```

Listing 5: examples/ht16k33_animation_demo.py

```
1  # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2  # SPDX-License-Identifier: MIT
3
4  """
5      Test script for display animations on an HT16K33 with alphanumeric display
6
7      The display must be initialized with auto_write=False.
8  """
9
10 from time import sleep
11 import board
12 import busio
13 from adafruit_ht16k33.segments import Seg14x4
14
15 #
16 #   Segment bits on the HT16K33 with alphanumeric display.
17 #
18 #   Add the values of the segments you need to create a bitmask
19 #
20
21 N = 16384
22 M = 8192
23 L = 4096
24 K = 2048
25 J = 1024
26 I = 512
27 H = 256
28 G2 = 128
29 G1 = 64
30 F = 32
31 E = 16
32 D = 8
33 C = 4
34 B = 2
35 A = 1
36
37 #   The number of seconds to delay between writing segments
38 DEFAULT_CHAR_DELAY_SEC = 0.2
39
40 #   The number of cycles to go for each animation
41 DEFAULT_CYCLES = 5
42
43 #   Brightness of the display (0 to 15)
44 DEFAULT_DISPLAY_BRIGHTNESS = 0.3
45
46 #   Initialize the I2C bus
47 i2c = busio.I2C(board.SCL, board.SDA)
48
49 #   Initialize the HT16K33 with alphanumeric display featherwing.
50 #
```

(continues on next page)

(continued from previous page)

```

51 # You MUST set auto_write=False
52 display = Seg14x4(i2c, auto_write=False)
53 display.brightness = DEFAULT_DISPLAY_BRIGHTNESS
54
55
56 def animate(digits, bitmasks, delay=DEFAULT_CHAR_DELAY_SEC, auto_write=True):
57     """
58     Main driver for all alphanumeric display animations (WIP!!!)
59     Param: digits - a list of the digits to write to, in order, like [0, 1, 3].
60     ↳ The digits are
61         0 to 3 starting at the left most digit.
62     Param: bitmasks - a list of the bitmasks to write, in sequence, to the
63     ↳ specified digits.
64     Param: delay - The delay, in seconds (or fractions of), between writing
65     ↳ bitmasks to a digit.
66     Param: auto_write - Whether to actually write to the display immediately or
67     ↳ not.
68
69     Returns: Nothing
70     """
71     if not isinstance(digits, list):
72         raise ValueError("The first parameter MUST be a list!")
73     if not isinstance(bitmasks, list):
74         raise ValueError("The second parameter MUST be a list!")
75     if delay < 0:
76         raise ValueError("The delay between frames must be positive!")
77     for dig in digits:
78         if not 0 <= dig <= 3:
79             raise ValueError(
80                 "Digit value must be \
81                 an integer in the range: 0-3"
82             )
83
84         for bits in bitmasks:
85             if not 0 <= bits <= 0xFFFF:
86                 raise ValueError(
87                     "Bitmask value must be an \
88                     integer in the range: 0-65535"
89                 )
90
91             display.set_digit_raw(dig, bits)
92
93             if auto_write:
94                 display.show()
95                 sleep(delay)
96
97 def chase_forward_and_reverse(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
98     cy = 0
99
100     while cy < cycles:
101         animate([0, 1, 2, 3], [A, 0], delay)
102         animate([3], [B, C, D, 0], delay)
103         animate([2, 1, 0], [D, 0], delay)
104         animate([0], [E, F, H, G2, 0], delay)
105         animate([1, 2], [G1, G2, 0], delay)
106         animate([3], [G1, J, A, 0], delay)

```

(continues on next page)

(continued from previous page)

```

104     animate([2, 1], [A, 0], delay)
105     animate([0], [A, F, E, D, 0], delay)
106     animate([1, 2], [D, 0], delay)
107     animate([3], [D, C, B, J, G1, 0], delay)
108     animate([2, 1], [G2, G1, 0], delay)
109     animate([0], [H, 0], delay)
110
111     cy += 1
112
113
114 def prelude_to_spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
115     cy = 0
116     auto_write = False
117
118     while cy < cycles:
119         animate([1, 2], [A], 0, auto_write)
120         display.show()
121         sleep(delay)
122
123         animate([0, 3], [A], 0, auto_write)
124         display.show()
125         sleep(delay)
126
127         animate([0], [A + F], 0, auto_write)
128         animate([3], [A + B], 0, auto_write)
129         display.show()
130         sleep(delay)
131
132         animate([0], [A + E + F], 0, auto_write)
133         animate([3], [A + B + C], 0, auto_write)
134         display.show()
135         sleep(delay)
136
137         animate([0], [A + D + E + F], 0, auto_write)
138         animate([3], [A + B + C + D], 0, auto_write)
139         display.show()
140         sleep(delay)
141
142         animate([1], [A + D], 0, auto_write)
143         animate([2], [A + D], 0, auto_write)
144         display.show()
145         sleep(delay)
146
147         animate([1], [A + D + M], 0, auto_write)
148         animate([2], [A + D + K], 0, auto_write)
149         display.show()
150         sleep(delay)
151
152         animate([1], [A + D + M + H], 0, auto_write)
153         animate([2], [A + D + K + J], 0, auto_write)
154         display.show()
155         sleep(delay)
156
157         animate([0], [A + E + F + J + D], 0, auto_write)
158         animate([3], [A + B + C + H + D], 0, auto_write)
159         display.show()
160         sleep(delay)

```

(continues on next page)

(continued from previous page)

```

161     animate([0], [A + E + F + J + K + D], 0, auto_write)
162     animate([3], [A + B + C + H + M + D], 0, auto_write)
163     display.show()
164     sleep(delay)
165
166     display.fill(0)
167     display.show()
168     sleep(delay)
169
170     cy += 1
171
172
173
174 def spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
175     cy = 0
176     auto_write = False
177
178     while cy < cycles:
179         animate([0], [H + M], 0, auto_write)
180         animate([1], [J + K], 0, auto_write)
181         animate([2], [H + M], 0, auto_write)
182         animate([3], [J + K], 0, auto_write)
183         display.show()
184         sleep(delay)
185
186         animate([0], [G1 + G2], 0, auto_write)
187         animate([1], [G1 + G2], 0, auto_write)
188         animate([2], [G1 + G2], 0, auto_write)
189         animate([3], [G1 + G2], 0, auto_write)
190         display.show()
191         sleep(delay)
192
193         animate([0], [J + K], 0, auto_write)
194         animate([1], [H + M], 0, auto_write)
195         animate([2], [J + K], 0, auto_write)
196         animate([3], [H + M], 0, auto_write)
197         display.show()
198         sleep(delay)
199
200         cy += 1
201
202     display.fill(0)
203
204
205 def enclosed_spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
206     cy = 0
207     auto_write = False
208
209     while cy < cycles:
210         animate([0], [A + D + E + F + H + M], 0, auto_write)
211         animate([1], [A + D + J + K], 0, auto_write)
212         animate([2], [A + D + H + M], 0, auto_write)
213         animate([3], [A + B + C + D + J + K], 0, auto_write)
214         display.show()
215         sleep(delay)
216
217         animate([0], [A + D + E + F + G1 + G2], 0, auto_write)

```

(continues on next page)

(continued from previous page)

```

218     animate([1], [A + D + G1 + G2], 0, auto_write)
219     animate([2], [A + D + G1 + G2], 0, auto_write)
220     animate([3], [A + B + C + D + G1 + G2], 0, auto_write)
221     display.show()
222     sleep(delay)
223
224     animate([0], [A + D + E + F + J + K], 0, auto_write)
225     animate([1], [A + D + H + M], 0, auto_write)
226     animate([2], [A + D + J + K], 0, auto_write)
227     animate([3], [A + B + C + D + H + M], 0, auto_write)
228     display.show()
229     sleep(delay)
230
231     cy += 1
232
233     display.fill(0)
234
235
236 def count_down():
237     auto_write = False
238     numbers = [
239         [A + B + C + D + G1 + G2 + N],
240         [A + B + D + E + G1 + G2 + N],
241         [B + C + N],
242     ]
243     index = 0
244
245     display.fill(0)
246
247     while index < len(numbers):
248         animate([index], numbers[index], 0, auto_write)
249         display.show()
250         sleep(1)
251         display.fill(0)
252         sleep(0.5)
253
254         index += 1
255
256     sleep(1)
257     display.fill(0)
258
259
260 try:
261     text = "Init"
262
263     display.fill(1)
264     display.show()
265     sleep(1)
266     display.fill(0)
267     display.show()
268
269     display.print(text)
270     display.show()
271     sleep(2)
272     display.fill(0)
273     display.show()
274     sleep(1)

```

(continues on next page)

(continued from previous page)

```

275
276     count_down()
277     sleep(0.2)
278
279     text = "Go!!"
280
281     display.print(text)
282     display.show()
283     sleep(1.5)
284     display.fill(0)
285     display.show()
286     sleep(0.5)
287     print()
288
289     while True:
290         # Arrow
291         print("Arrow")
292         animate([0, 1, 2], [G1 + G2], 0.1)
293         animate([3], [G1 + H + K], 0.1)
294         sleep(1.0)
295         display.fill(0)
296         sleep(1.0)
297
298         # Flying
299         print("Flying")
300         cyc = 0
301
302         while cyc < DEFAULT_CYCLES:
303             animate([0], [H + J, G1 + G2, K + M, G1 + G2], DEFAULT_CHAR_DELAY_SEC)
304
305             cyc += 1
306
307             animate([0], [0])
308             sleep(1.0)
309             display.fill(0)
310             sleep(1.0)
311
312             # Chase forward and reverse.
313             print("Chase forward and reverse")
314             chase_forward_and_reverse(0.01, 5)
315             sleep(1.0)
316             display.fill(0)
317             sleep(1.0)
318
319             # Testing writing to more than one segment simultaneously
320             print("Prelude to Spinners")
321             prelude_to_spinners(0.1, 5)
322             sleep(1.0)
323             display.fill(0)
324             display.show()
325             sleep(1.0)
326
327             print("Spinners")
328             spinners(0.1, 20)
329             sleep(1.0)
330             display.fill(0)
331             display.show()

```

(continues on next page)

(continued from previous page)

```
332     sleep(1.0)
333
334     print("Enclosed Spinners")
335     enclosed_spinners(0.1, 20)
336     sleep(1.0)
337     display.fill(0)
338     display.show()
339     sleep(1.0)
340
341     print()
342 except KeyboardInterrupt:
343     display.fill(0)
344     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, bright-
                                         ness=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, bright-
                                         ness=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`, [24](#)
`adafruit_ht16k33.matrix`, [24](#)
`adafruit_ht16k33.segments`, [25](#)

A

`adafruit_ht16k33.ht16k33` (module), 24
`adafruit_ht16k33.matrix` (module), 24
`adafruit_ht16k33.segments` (module), 25
`ampm` (`adafruit_ht16k33.segments.BigSeg7x4` attribute), 26
`auto_write` (`adafruit_ht16k33.ht16k33.HT16K33` attribute), 24

B

`BigSeg7x4` (class in `adafruit_ht16k33.segments`), 26
`blink_rate` (`adafruit_ht16k33.ht16k33.HT16K33` attribute), 24
`bottom_left_dot` (`adafruit_ht16k33.segments.BigSeg7x4` attribute), 26
`brightness` (`adafruit_ht16k33.ht16k33.HT16K33` attribute), 24

C

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

F

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

H

`HT16K33` (class in `adafruit_ht16k33.ht16k33`), 24

I

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

M

`marquee()` (`adafruit_ht16k33.segments.Seg14x4` method), 26
`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), 25
`pixel()` (`adafruit_ht16k33.matrix.Matrix8x8x2` method), 25
`pixel()` (`adafruit_ht16k33.matrix.MatrixBackpack16x8` method), 25
`print()` (`adafruit_ht16k33.segments.Seg14x4` method), 26
`print_hex()` (`adafruit_ht16k33.segments.Seg14x4` method), 26

R

`rows` (`adafruit_ht16k33.matrix.Matrix8x8` attribute), 25

S

`scroll()` (`adafruit_ht16k33.segments.Seg14x4` method), 26
`scroll()` (`adafruit_ht16k33.segments.Seg7x4` method), 26
`Seg14x4` (class in `adafruit_ht16k33.segments`), 26
`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), 25

`shift_down()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [25](#)
`shift_left()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [25](#)
`shift_right()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [25](#)
`shift_up()` (*adafruit_ht16k33.matrix.Matrix8x8*
method), [25](#)
`show()` (*adafruit_ht16k33.ht16k33.HT16K33* *method*),
[24](#)

T

`top_left_dot` (*adafruit_ht16k33.segments.BigSeg7x4*
attribute), [26](#)