Adafruit HT16K33 Library Documentation

Release 1.0

Radomir Dopieralski

Mar 10, 2021

Contents

This is a library for using the I^2C-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.

CHAPTER 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/ht16k33_matrix_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Basic example of clearing and drawing a pixel on a LED matrix display.
# This example and library is meant to work with Adafruit CircuitPython API.
# Author: Tony DiCola
# License: Public Domain

# Import all board pins.
import time
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 16x8 matrix backpack:
# matrix = matrix.MatrixBackpack16x8(i2c)
# Or this creates a 8x8 matrix:
# matrix = matrix.Matrix8x8(i2c)
```

(continues on next page)

(continued from previous page)

```
# Or this creates a 8x8 bicolor matrix:
# matrix = matrix.Matrix8x8x2(i2c)
# 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

time.sleep(2)

# Draw a Smiley Face
matrix.fill(0)

for row in range(2, 6):
    matrix[row, 0] = 1
    matrix[row, 7] = 1

for column in range(2, 6):
    matrix[0, column] = 1
    matrix[7, column] = 1

matrix[1, 1] = 1
matrix[1, 6] = 1
matrix[6, 1] = 1
matrix[6, 6] = 1
matrix[2, 5] = 1
matrix[5, 5] = 1
matrix[2, 3] = 1
matrix[5, 3] = 1
matrix[3, 2] = 1
matrix[4, 2] = 1

# Move the Smiley Face Around
while True:
    for frame in range(0, 8):
        matrix.shift_right(True)
        time.sleep(0.05)
    for frame in range(0, 8):
        matrix.shift_down(True)
        time.sleep(0.05)
    for frame in range(0, 8):
        matrix.shift_left(True)
        time.sleep(0.05)
    for frame in range(0, 8):
        matrix.shift_up(True)
        time.sleep(0.05)
```

Listing 2: examples/ht16k33_segments_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

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

import time

# Import all board pins.
import board
import busio

# Import the HT16K33 LED segment module.
from adafruit_ht16k33 import segments

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

# Create the LED segment class.
# This creates a 7 segment 4 character display:
display = segments.Seg7x4(i2c)
# Or this creates a 14 segment alphanumeric 4 character display:
# display = segments.Seg14x4(i2c)
# Or this creates a big 7 segment 4 character display
# display = segments.BigSeg7x4(i2c)
# Finally you can optionally specify a custom I2C address of the HT16k33 like:
# display = segments.Seg7x4(i2c, address=0x70)

# Clear the display.
display.fill(0)

# Can just print a number
display.print(42)
time.sleep(2)

# Or, can print a hexadecimal value
display.print_hex(0xFF23)
time.sleep(2)

# Or, print the time
display.print("12:30")
time.sleep(2)

display.colon = False

# Or, can set indivdual digits / characters
# Set the first character to '1':
display[0] = "1"
# Set the second character to '2':
display[1] = "2"
# Set the third character to 'A':
display[2] = "A"
# Set the forth character to 'B':
```

(continues on next page)

(continued from previous page)

```
display[3] = "B"
time.sleep(2)

# Or, can even set the segments to make up characters
if isinstance(display, segments.Seg7x4):
    # 7-segment raw digits
    display.set_digit_raw(0, 0xFF)
    display.set_digit_raw(1, 0b11111111)
    display.set_digit_raw(2, 0x79)
    display.set_digit_raw(3, 0b01111001)
else:
    # 14-segment raw digits
    display.set_digit_raw(0, 0x2D3F)
    display.set_digit_raw(1, 0b0010110100111111)
    display.set_digit_raw(2, (0b00101101, 0b00111111))
    display.set_digit_raw(3, [0x2D, 0x3F])
time.sleep(2)

# Show a looping marquee
display.marquee("Deadbeef 192.168.100.102... ", 0.2)
```

Listing 3: examples/ht16k33_bicolor24_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Basic example of using the Bi-color 24 segment bargraph display.
# This example and library is meant to work with Adafruit CircuitPython API.
# Author: Carter Nelson
# License: Public Domain

import time

# Import board related modules
import board
import busio

# Import the Bicolor24 driver from the HT16K33 module
from adafruit_ht16k33.bargraph import Bicolor24

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

# Create the LED bargraph class.
bc24 = Bicolor24(i2c)

# Set individual segments of bargraph
bc24[0] = bc24.LED_RED
bc24[1] = bc24.LED_GREEN
bc24[2] = bc24.LED_YELLOW

time.sleep(2)

# Turn them all off
bc24.fill(bc24.LED_OFF)

```

(continues on next page)

(continued from previous page)

```
# Turn them on in a loop
for i in range(24):
    bc24[i] = bc24.LED_RED
    time.sleep(0.1)
    bc24[i] = bc24.LED_OFF

time.sleep(1)

# Fill the entrire bargraph
bc24.fill(bc24.LED_GREEN)
```

Listing 4: examples/ht16k33_matrix_pillow_image.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Basic example of drawing an image
# This example and library is meant to work with Adafruit CircuitPython API.
#
# This example is for use on (Linux) computers that are using CPython with
# Adafruit Blinka to support CircuitPython libraries. CircuitPython does
# not support PIL/pillow (python imaging library)!
#
# Author: Melissa LeBlanc-Williams
# License: Public Domain

# Import all board pins.
import board
import busio
from PIL import Image

# 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:
mtrx = matrix.Matrix16x8(i2c)
# Or this creates a 16x8 matrix backpack:
# mtrx = matrix.MatrixBackpack16x8(i2c)
# Or this creates a 8x8 matrix:
# mtrx = matrix.Matrix8x8(i2c)
# Or this creates a 8x8 bicolor matrix:
# mtrx = matrix.Matrix8x8x2(i2c)
# Finally you can optionally specify a custom I2C address of the HT16k33 like:
# mtrx = matrix.Matrix16x8(i2c, address=0x70)

if isinstance(mtrx, matrix.Matrix8x8x2):
    image = Image.open("squares-color.png")
elif isinstance(mtrx, matrix.Matrix16x8):
    image = Image.open("squares-mono-16x8.png")
else:
    image = Image.open("squares-mono-8x8.png")

```

(continues on next page)

(continued from previous page)

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

Listing 5: examples/ht16k33_animation_demo.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
    Test script for display animations on an HT16K33 with alphanumeric display

    The display must be initialized with auto_write=False.
"""

from time import sleep
import board
import busio
from adafruit_ht16k33.segments import Seg14x4

#
#   Segment bits on the HT16K33 with alphanumeric display.
#
#   Add the values of the segments you need to create a bitmask
#

N = 16384
M = 8192
L = 4096
K = 2048
J = 1024
I = 512
H = 256
G2 = 128
G1 = 64
F = 32
E = 16
D = 8
C = 4
B = 2
A = 1

#   The number of seconds to delay between writing segments
DEFAULT_CHAR_DELAY_SEC = 0.2

#   The number of cycles to go for each animation
DEFAULT_CYCLES = 5

#   Brightness of the display (0 to 15)
DEFAULT_DISPLAY_BRIGHTNESS = 0.3

#   Initialize the I2C bus
i2c = busio.I2C(board.SCL, board.SDA)

#   Initialize the HT16K33 with alphanumeric display featherwing.
#
```

(continues on next page)

(continued from previous page)

```
#   You MUST set auto_write=False
display = Seg14x4(i2c, auto_write=False)
display.brightness = DEFAULT_DISPLAY_BRIGHTNESS


def animate(digits, bitmasks, delay=DEFAULT_CHAR_DELAY_SEC, auto_write=True):
    """
    Main driver for all alphanumeric display animations (WIP!!!)
        Param: digits - a list of the digits to write to, in order, like [0, 1, 3]. The digits are
            0 to 3 starting at the left most digit.
        Param: bitmasks - a list of the bitmasks to write, in sequence, to the specified digits.
        Param: delay - The delay, in seconds (or fractions of), between writing bitmasks to a digit.
        Param: auto_write - Whether to actually write to the display immediately or not.

        Returns: Nothing
    """
    if not isinstance(digits, list):
        raise ValueError("The first parameter MUST be a list!")
    if not isinstance(bitmasks, list):
        raise ValueError("The second parameter MUST be a list!")
    if delay < 0:
        raise ValueError("The delay between frames must be positive!")
    for dig in digits:
        if not 0 <= dig <= 3:
            raise ValueError(
                "Digit value must be \
            an integer in the range: 0-3"
            )

        for bits in bitmasks:
            if not 0 <= bits <= 0xFFFF:
                raise ValueError(
                    "Bitmask value must be an \
                integer in the range: 0-65535"
                )

            display.set_digit_raw(dig, bits)

            if auto_write:
                display.show()
                sleep(delay)


def chase_forward_and_reverse(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
    cy = 0

    while cy < cycles:
        animate([0, 1, 2, 3], [A, 0], delay)
        animate([3], [B, C, D, 0], delay)
        animate([2, 1, 0], [D, 0], delay)
        animate([0], [E, F, H, G2, 0], delay)
        animate([1, 2], [G1, G2, 0], delay)
        animate([3], [G1, J, A, 0], delay)
```

(continues on next page)

(continued from previous page)

```
        animate([2, 1], [A, 0], delay)
        animate([0], [A, F, E, D, 0], delay)
        animate([1, 2], [D, 0], delay)
        animate([3], [D, C, B, J, G1, 0], delay)
        animate([2, 1], [G2, G1, 0], delay)
        animate([0], [H, 0], delay)

        cy += 1


def prelude_to_spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
    cy = 0
    auto_write = False

    while cy < cycles:
        animate([1, 2], [A], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0, 3], [A], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [A + F], 0, auto_write)
        animate([3], [A + B], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [A + E + F], 0, auto_write)
        animate([3], [A + B + C], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [A + D + E + F], 0, auto_write)
        animate([3], [A + B + C + D], 0, auto_write)
        display.show()
        sleep(delay)

        animate([1], [A + D], 0, auto_write)
        animate([2], [A + D], 0, auto_write)
        display.show()
        sleep(delay)

        animate([1], [A + D + M], 0, auto_write)
        animate([2], [A + D + K], 0, auto_write)
        display.show()
        sleep(delay)

        animate([1], [A + D + M + H], 0, auto_write)
        animate([2], [A + D + K + J], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [A + E + F + J + D], 0, auto_write)
        animate([3], [A + B + C + H + D], 0, auto_write)
        display.show()
        sleep(delay)
```

(continues on next page)

(continued from previous page)

```
        animate([0], [A + E + F + J + K + D], 0, auto_write)
        animate([3], [A + B + C + H + M + D], 0, auto_write)
        display.show()
        sleep(delay)

        display.fill(0)
        display.show()
        sleep(delay)

        cy += 1


def spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
    cy = 0
    auto_write = False

    while cy < cycles:
        animate([0], [H + M], 0, auto_write)
        animate([1], [J + K], 0, auto_write)
        animate([2], [H + M], 0, auto_write)
        animate([3], [J + K], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [G1 + G2], 0, auto_write)
        animate([1], [G1 + G2], 0, auto_write)
        animate([2], [G1 + G2], 0, auto_write)
        animate([3], [G1 + G2], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [J + K], 0, auto_write)
        animate([1], [H + M], 0, auto_write)
        animate([2], [J + K], 0, auto_write)
        animate([3], [H + M], 0, auto_write)
        display.show()
        sleep(delay)

        cy += 1

    display.fill(0)


def enclosed_spinners(delay=DEFAULT_CHAR_DELAY_SEC, cycles=DEFAULT_CYCLES):
    cy = 0
    auto_write = False

    while cy < cycles:
        animate([0], [A + D + E + F + H + M], 0, auto_write)
        animate([1], [A + D + J + K], 0, auto_write)
        animate([2], [A + D + H + M], 0, auto_write)
        animate([3], [A + B + C + D + J + K], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [A + D + E + F + G1 + G2], 0, auto_write)
```

(continues on next page)

(continued from previous page)

```
        animate([1], [A + D + G1 + G2], 0, auto_write)
        animate([2], [A + D + G1 + G2], 0, auto_write)
        animate([3], [A + B + C + D + G1 + G2], 0, auto_write)
        display.show()
        sleep(delay)

        animate([0], [A + D + E + F + J + K], 0, auto_write)
        animate([1], [A + D + H + M], 0, auto_write)
        animate([2], [A + D + J + K], 0, auto_write)
        animate([3], [A + B + C + D + H + M], 0, auto_write)
        display.show()
        sleep(delay)

        cy += 1

    display.fill(0)


def count_down():
    auto_write = False
    numbers = [
        [A + B + C + D + G1 + G2 + N],
        [A + B + D + E + G1 + G2 + N],
        [B + C + N],
    ]
    index = 0

    display.fill(0)

    while index < len(numbers):
        animate([index], numbers[index], 0, auto_write)
        display.show()
        sleep(1)
        display.fill(0)
        sleep(0.5)

        index += 1

    sleep(1)
    display.fill(0)


try:
    text = "Init"

    display.fill(1)
    display.show()
    sleep(1)
    display.fill(0)
    display.show()

    display.print(text)
    display.show()
    sleep(2)
    display.fill(0)
    display.show()
    sleep(1)
```

(continues on next page)

(continued from previous page)

```
    count_down()
    sleep(0.2)

    text = "Go!!"

    display.print(text)
    display.show()
    sleep(1.5)
    display.fill(0)
    display.show()
    sleep(0.5)
    print()

    while True:
        #   Arrow
        print("Arrow")
        animate([0, 1, 2], [G1 + G2], 0.1)
        animate([3], [G1 + H + K], 0.1)
        sleep(1.0)
        display.fill(0)
        sleep(1.0)

        #   Flying
        print("Flying")
        cyc = 0

        while cyc < DEFAULT_CYCLES:
            animate([0], [H + J, G1 + G2, K + M, G1 + G2], DEFAULT_CHAR_DELAY_SEC)

            cyc += 1

        animate([0], [0])
        sleep(1.0)
        display.fill(0)
        sleep(1.0)

        #   Chase forward and reverse.
        print("Chase forward and reverse")
        chase_forward_and_reverse(0.01, 5)
        sleep(1.0)
        display.fill(0)
        sleep(1.0)

        #   Testing writing to more than one segment simultaneously
        print("Prelude to Spinners")
        prelude_to_spinners(0.1, 5)
        sleep(1.0)
        display.fill(0)
        display.show()
        sleep(1.0)

        print("Spinners")
        spinners(0.1, 20)
        sleep(1.0)
        display.fill(0)
        display.show()
```

(continues on next page)

(continued from previous page)

```
        sleep(1.0)

        print("Enclosed Spinners")
        enclosed_spinners(0.1, 20)
        sleep(1.0)
        display.fill(0)
        display.show()
        sleep(1.0)

        print()
except KeyboardInterrupt:
    display.fill(0)
    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 addess 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 hexidecimal 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

Python Module Index

a

Index

T