# GoPiGo3 Documentation

***Release 1.0.0***

**Robert Lucian Chiriac**

**May 01, 2018**

# Contents:

DEXTER INDUSTRIES

CHAPTER 1

# About GoPiGo3

## 1.1 Who are we and what we do.


Dexter Industries is an American educational robotics company that develops robot kits that make programming accessible for everyone.

## 1.2 What's this documentation about.

This documentation is all about the GoPiGo3 robot. Within this, you will find instructions on:

- How to get started with the GoPiGo3 robot - assembling, setting up the environment, etc.
- How to get started with the example programs found in our repo.
- How to operate the GoPiGo3 with our API. The user has a comprehensive documentation of all the modules/functions/classes that are needed for controlling the robot.
- How to troubleshoot the GoPiGo3 in case of unsuspected situations.

CHAPTER 2

# Getting Started

## 2.1 Buying a GoPiGo3

To buy a GoPiGo3 robot, please head over to our online shop and search for the GoPiGo3 robot. From our shop, you can get sensors for your robot such as the Distance Sensor, the Grove Light Sensor, etc.

## 2.2 Assembling GoPiGo3

For assembling your GoPiGo3 robot, read the instructions from the following page: assembling instructions.

## 2.3 Connecting to GoPiGo3

For connecting to your GoPiGo3 robot with a computer or laptop, read the instructions on the following page: connecting to robot.

## 2.4 Program your GoPiGo3

For programming your GoPiGo3 to do anything you want, read the instructions found here: programming your robot.

CHAPTER 3

# Tutorials - Basic

This chapter revolves around the `easygopigo3` module.

Please make sure you have followed all the instructions found in *Getting Started* before jumping into tutorials. In all these tutorials, you will need:

1. A GoPiGo3 robot.
2. Sensor/Actuator specific for the tutorial : i.e.: a Grove Buzzer, a Line Follower, etc.

## 3.1 Flashing an LED

### 3.1.1 Our goal

In this tutorial, we are making a Grove Led flash continuously, while it's being connected to a GoPiGo3 robot.

### 3.1.2 The code we analyse

The code we're analyzing in this tutorial is the following one.

```
# import the EasyGoPiGo3 drivers
import time
import easygopigo3 as easy

# Create an instance of the GoPiGo3 class.
# GPG will be the GoPiGo3 object.
gpg = easy.EasyGoPiGo3()


# create the LED instance, passing the port and GPG
my_led = gpg.init_led("AD1")
# or
# my_LED = easy.Led("AD1", GPG)
```

```
# loop 100 times
for i in range(100):
    my_led.light_max() # turn LED at max power
    time.sleep(0.5)

    my_led.light_on(30)  # 30% power
    time.sleep(0.5)

    my_led.light_off() # turn LED off
    time.sleep(0.5)
```

The source code for this example program can be found here on github.

### 3.1.3 The modules

Start by importing 2 important modules:

```
import time
import easygopigo3 as easy
```

The `easygopigo3` module is used for interacting with the GoPiGo3 robot, whilst the `time` module is generally used for delaying actions, commands, setting timers etc.

### 3.1.4 The objects

After this, we need to instantiate an `easygopigo3.EasyGoPiGo3` object. We are using the `EasyGoPiGo3` object for creating an instance of `Led` class, which is necessary for controlling the Grove Led device.

```
gpg = easy.EasyGoPiGo3()
```

Now that we have an `EasyGoPiGo3` object, we can instantiate a `Led` object. The argument of the initializer method is the port to which we connect the Grove Led and it's set to `"AD1"`.

```
my_led = gpg.init_led("AD1")
```

**Note:** See the following *graphical representation* as a reference to where the ports are.

### 3.1.5 Main part

In this section of the tutorial we are focusing on 3 methods of the `easygopigo3.Led` class.

- The `light_max()` method - which turns the LED at the maximum brightness.
- The `light_on()` method - used for turning the LED at a certain percent of the maximum brightness.
- The `light_off()` method - used for turning off the LED.

All in all, the following code snippet turns on the LED to the maximum brightness, then it sets the LED's brightness at 30% and in the last it turns off the LED. The delay between all these 3 commands is set at half a second.

```
for i in range(100):
    my_led.light_max() # turn LED at max power
    time.sleep(0.5)

    my_led.light_on(30)  # 30% power
    time.sleep(0.5)

    my_led.light_off() # turn LED off
    time.sleep(0.5)
```

### 3.1.6 Running it

Connect the Grove Led to your GoPiGo3 robot to port "AD1" and then let's crank up the Raspberry Pi. For running the analyzed example program, within a terminal on your Raspberry Pi, type the following 2 commands:

```
cd ~/Desktop/GoPiGo3/Software/Python/Examples
python easy_LED.py
```

## 3.2 Pushing a Button

### 3.2.1 Our goal

In this tutorial, we are going to control GoPiGo3 Dex's eyes with a Grove Button.

- When the Grove Button is pressed, Dex's eyes turn on.
- When the Grove Button is released, Dex's eyes turn off.

### 3.2.2 The code we analyse

In the end the code should look like this.

```
# import the time library for the sleep function
import time

# import the GoPiGo3 drivers
import easygopigo3 as easy

# Create an instance of the GoPiGo3 class.
# GPG will be the GoPiGo3 object.
gpg = easy.EasyGoPiGo3()

# Put a grove button in port AD1
my_button = gpg.init_button_sensor("AD1")

print("Ensure there's a button in port AD1")
print("Press and release the button as often as you want")
print("the program will run for 2 minutes or")
print("Ctrl-C to interrupt it")
```

```
start = time.time()
RELEASED = 0
PRESSED = 1
state = RELEASED

while time.time() - start < 120:

    if state == RELEASED and my_button.read() == 1:
        print("PRESSED")
        gpg.open_eyes()
        state = PRESSED
    if state == PRESSED and my_button.read() == 0:
        print("RELEASED")
        gpg.close_eyes()
        state = RELEASED
    time.sleep(0.05)

print("All done!")
```

The source code for this example program can be found here on github.

### 3.2.3 The modules

Start by importing 2 important modules:

```
import time
import easygopigo3 as easy
```

The `easygopigo3` module is used for interacting with the GoPiGo3 robot, whilst the `time` module is generally used for delaying actions, commands, setting timers etc.

### 3.2.4 The objects

After this, we need to instantiate an `easygopigo3.EasyGoPiGo3` object. The `EasyGoPiGo3` object is used for 2 things:

- For turning *ON* and *OFF* the GoPiGo3 Dex's eyes.
- For instantiating a `ButtonSensor` object for reading the Grove Button's state.

```
gpg = easy.EasyGoPiGo3()
```

Now that we have an `EasyGoPiGo3` object, we can instantiate a `ButtonSensor` object. The argument of the initializer method is the port to which we connect the Grove Button and it's set to `"AD1"`.

```
my_button = gpg.init_button_sensor("AD1")
```

**Note:** See the following *graphical representation* as a reference to where the ports are.

### 3.2.5 Setting variables

Define 2 states for the button we're using. We are setting the default state to `"RELEASED"`.

```
start = time.time()
RELEASED = 0
PRESSED = 1
state = RELEASED
```

There's also a variable called `start` to which we assign the clock time of that moment. We use it to limit for how long the script runs.

### 3.2.6 Main part

The main part is basically a while loop that's going to run for 120 seconds. Within the while loop, we have 2 `if / else` blocks that define a simple algorithm: whenever the previous state is different from the current one, we either turn on or close Dex's eyes. Here's the logic:

- If in the previous iteration of the while loop the button was **released** and now the button is **1** (aka **pressed**), then we turn **on** the LEDs and save the new state in `state` variable.
- If in the previous iteration of the while loop the button was **pressed** and now the button is **0** (aka **released**), then we turn **off** the LEDs and save the new state in `state` variable.

This way, we don't call `gpg.open_eyes()` all the time when the button is pressed or `gpg.close_eyes()` when the button is released. It only needs to call one of these 2 functions once.

```
while time.time() - start < 120:

  if state == RELEASED and my_button.read() == 1:
    print("PRESSED")
    gpg.open_eyes()
    state = PRESSED
  if state == PRESSED and my_button.read() == 0:
    print("RELEASED")
    gpg.close_eyes()
    state = RELEASED

  time.sleep(0.05)
```

`time.sleep(0.05)` was added to limit the CPU time. 50 mS is more than enough.

### 3.2.7 Running it

Make sure you have connected the Grove Button to your GoPiGo3 robot to port `"AD1"`. Then, on the Rasperry Pi, from within a terminal, type the following commands.

```
cd ~/Desktop/GoPiGo3/Software/Python/Examples
python easy_Button.py
```

## 3.3 Ringing a Buzzer

### 3.3.1 Our goal

In this tutorial, we are making a Grove Buzzer play different musical tones on our GoPiGo3 robot. We start off with 3 musical notes and finish by playing the well-known *"Twinkle Twinklle Little Star"* song.

### 3.3.2 The code we analyse

The code we're analyzing in this tutorial is this.

```
# import the time library for the sleep function
import time

# import the GoPiGo3 drivers
import easygopigo3 as easy

# Create an instance of the GoPiGo3 class.
# GPG will be the GoPiGo3 object.
gpg = easy.EasyGoPiGo3()

# Create an instance of the Buzzer
# connect a buzzer to port AD2
my_buzzer = gpg.init_buzzer("AD2")

twinkle = ["C4","C4","G4","G4","A4","A4","G4"]

print("Expecting a buzzer on Port AD2")
print("A4")
my_buzzer.sound(440)
time.sleep(1)
print("A5")
my_buzzer.sound(880)
time.sleep(1)
print("A3")
my_buzzer.sound(220)
time.sleep(1)

for note in twinkle:
    print(note)
    my_buzzer.sound(my_buzzer.scale[note])
    time.sleep(0.5)
    my_buzzer.sound_off()
    time.sleep(0.25)

my_buzzer.sound_off()
```

The source code for this example program can be found here on github.

### 3.3.3 The modules

Start by importing 2 important modules:

```
import time
import easygopigo3 as easy
```

The `easygopigo3` module is used for interacting with the GoPiGo3 robot, whilst the `time` module is generally used for delaying actions, commands, setting timers etc.

### 3.3.4 The objects

After this, we need to instantiate an `easygopigo3.EasyGoPiGo3` object. We will be using the `EasyGoPiGo3` object for creating an instance of `Buzzer` class, which is necessary for controlling the Grove Buzzer device.

```
gpg = easy.EasyGoPiGo3()
```

Now that we have an `EasyGoPiGo3` object, we can instantiate a `Buzzer` object. The argument of the initializer method is the port to which we connect the Grove Buzzer and it's set to `"AD2"`.

```
my_buzzer = gpg.init_buzzer("AD2")
```

**Note:** See the following *graphical representation* as a reference to where the ports are.

### 3.3.5 Setting variables

To play the *"Twinkle Twinkle Little Star"* song, we need to have a sequence of musical notes that describe this song. We're encoding the musical notes into a list (called `twinkle`) of strings, where each string represents a musical note.

```
twinkle = ["C4","C4","G4","G4","A4","A4","G4"]
```

### 3.3.6 Main part

The main zone of the code is divided into 2 sections:

1. The 1st section, where we only play 3 musical notes with a 1 second delay.
2. The 2nd section, where we play the lovely *"Twinkle Twinkle Little Start"* song.

In the 1st section, we use the `easygopigo3.Buzzer.sound()` method, which takes as a paramater, an integer that represents the frequency of the emitted sound. As you can see in the following code snippet, each musical note corresponds to a certain frequency:

- The frequency of *A4* musical note is *440Hz*.
- The frequency of *A5* musical note is *880Hz*.
- The frequency of *A3* musical note is *220Hz*.

```
print("A4")
my_buzzer.sound(440)
time.sleep(1)

print("A5")
my_buzzer.sound(880)
time.sleep(1)

print("A3")
```

```
my_buzzer.sound(220)
time.sleep(1)
```

In the 2nd section we are using the `scale` dictionary. In this dictionary there are stored the frequencies of each musical note. So, when using the `twinkle` list in conjuction with `scale` attribute, we're basically retrieving the frequency of a musical note (found in `twinkle` attribute) from the `scale` dictionary.

```
for note in twinkle:
    print(note)
    my_buzzer.sound(buzzer.scale[note])
    time.sleep(0.5)
    my_buzzer.sound_off()
    time.sleep(0.25)
```

### 3.3.7 Running it

The only thing left to do is to connect the Grove Buzzer to your GoPiGo3 robot to port `"AD2"`. Then, on your Raspberry Pi, from within a terminal, type the following commands:

```
cd ~/Desktop/GoPiGo3/Software/Python/Examples
python easy_Buzzer.py
```

**Tip:** Please don't expect to hear a symphony, because the buzzer wasn't made for playing tones. We use the buzzer within this context to only demonstrate that it's a nice feature.

## 3.4 Detecting Light

### 3.4.1 Our goal

In this tutorial, we are making a Grove Light Sensor light up a Grove Led depending on how strong the intensity of the light is. The Grove Light Sensor and the Grove Led are both connected to the GoPiGo3 and use the following ports.

- Port `"AD1"` for the light sensor.
- Port `"AD2"` for the LED.

**Important:** Since this tutorial is based on *Led tutorial*, we recommend following that one before going through the current one.

### 3.4.2 The code we analyse

The code we're analyzing in this tutorial is the following one.

```
# import the time library for the sleep function
import time

# import the GoPiGo3 drivers
import easygopigo3 as easy
```

```
# Create an instance of the GoPiGo3 class.
# GPG will be the GoPiGo3 object.
gpg = easy.EasyGoPiGo3()

# Create an instance of the Light sensor
my_light_sensor = gpg.init_light_sensor("AD1")
my_led = gpg.init_led("AD2")

# loop forever while polling the sensor
while(True):
    # get absolute value
    reading = my_light_sensor.read()
    # scale the reading to a 0-100 scale
    percent_reading = my_light_sensor.percent_read()

    # check if the light's intensity is above 50%
    if percent_reading >= 50:
      my_led.light_off()
    else:
      my_led.light_max()
    print("{}, {:.1f}%".format(reading, percent_reading))

    time.sleep(0.05)
```

The source code for this tutorial can also be found here on github.

### 3.4.3 The modules

Start by importing 2 important modules:

```
import time
import easygopigo3 as easy
```

The `easygopigo3` module is used for interacting with the GoPiGo3 robot, whilst the `time` module is generally used for delaying actions, commands, setting timers etc.

### 3.4.4 The objects

After this, we need to instantiate an `easygopigo3.EasyGoPiGo3` object. We are using the `EasyGoPiGo3` object for creating an instance of `Led` class, which is necessary for controlling the Grove Led and for reading off of the Grove Light Sensor.

```
gpg = easy.EasyGoPiGo3()
```

Now that we have an `EasyGoPiGo3` object, we can instantiate a `LightSensor` and `Led` objects. The argument of each of the 2 initializer methods represents the port to which a device is connected.

```
my_light_sensor = gpg.init_light_sensor("AD1")
my_led = gpg.init_led("AD2")
```

**Note:** See the following *graphical representation* as a reference to where the ports are.

### 3.4.5 Main part

Let's make the LED behave in the following way.

- When the light's intensity is below 50%, turn on the LED.
- When the light's intensity is above 50%, turn off the LED.

To do this, we need to read the percentage value off of the light sensor - the variable responsible for holding the value is called `percent_reading`. Depending on the determined percentage, we turn the LED on or off.

To do all this, check out the following code snippet.

```
while(True):
    # get absolute value
    reading = my_light_sensor.read()
    # scale the reading to a 0-100 scale
    percent_reading = my_light_sensor.percent_read()

    # check if the light's intensity is above 50%
    if percent_read >= 50:
      my_led.light_off()
    else:
      my_led.light_max()
    print("{}, {:.1f}%".format(reading, percent_reading))

    time.sleep(0.05)
```

### 3.4.6 Running it

Here's the fun part. Let's run the python script.

Connect the Grove Light Sensor to your GoPiGo3 robot to port `"AD1"` and Grove Led to port `"AD2"`. Within a terminal on your Raspberry Pi, type the following 2 commands:

```
cd ~/Desktop/GoPiGo3/Software/Python/Examples
python easy_Light_Sensor.py
```

## 3.5 Measuring with the Distance Sensor

### 3.5.1 Our goal

In this tutorial, we are using a Distance Sensor for measuring the distance to a target with the GoPiGo3 robot. We are going to print the values on a terminal.

### 3.5.2 The code we analyse

The code we're analyzing in this tutorial is the following one.

```
# import the GoPiGo3 drivers
import time
import easygopigo3 as easy
```

```
# This example shows how to read values from the Distance Sensor

# Create an instance of the GoPiGo3 class.
# GPG will be the GoPiGo3 object.
gpg = easy.EasyGoPiGo3()

# Create an instance of the Distance Sensor class.
# I2C1 and I2C2 are just labels used for identifyng the port on the GoPiGo3 board.
# But technically, I2C1 and I2C2 are the same thing, so we don't have to pass any␣
↪port to the constructor.
my_distance_sensor = gpg.init_distance_sensor()

while True:
    # Directly print the values of the sensor.
    print("Distance Sensor Reading (mm): " + str(my_distance_sensor.read_mm()))
```

The source code for this example program can be found here on github.

### 3.5.3 The modules

Start by importing 2 important modules:

```
import time
import easygopigo3 as easy
```

The `easygopigo3` module is used for interacting with the GoPiGo3 robot, whilst the `time` module is generally used for delaying actions, commands, setting timers etc.

### 3.5.4 The objects

For interfacing with the Distance Sensor we need to instantiate an object of the `easygopigo3.EasyGoPiGo3` class so in return, we can instantiate an object of the `easygopigo3.DistanceSensor` class. We do it like in the following code snippet.

```
gpg = easy.EasyGoPiGo3() # this is an EasyGoPiGo3 object
my_distance_sensor = gpg.init_distance_sensor() # this is a DistanceSensor object
```

### 3.5.5 Main part

There's a single while loop in the entire script. The loop is for printing the values that we're reading repeatedly. We will be using the `read_mm()` method for reading the distance in millimeters to the target.

```
while True:

    # Directly print the values of the sensor.
    print("Distance Sensor Reading (mm): " + str(my_distance_sensor.read_mm()))
```

**See also:**

Check out `easygopigo3.DistanceSensor`'s API for more details.

### 3.5.6 Running it

Connect the Distance Sensor to any of the 2 "I2C" ports on the GoPiGo3 robot. After the sensor is connected, on your Raspberry Pi, open up a terminal and type in the following 2 commands.

```
cd ~/Desktop/GoPiGo3/Software/Python/Examples
python easy_Distance_Sensor.py
```

**Note:** See the following *graphical representation* as a reference to where the ports are.

CHAPTER 4

# Tutorials - Advanced

**Note:** Coming soon!

CHAPTER 5

# API Reference Point - Basic

## 5.1 Requirements

Before using this chapter's classes, you need to be able to import the following module.

```
import easygopigo3
```

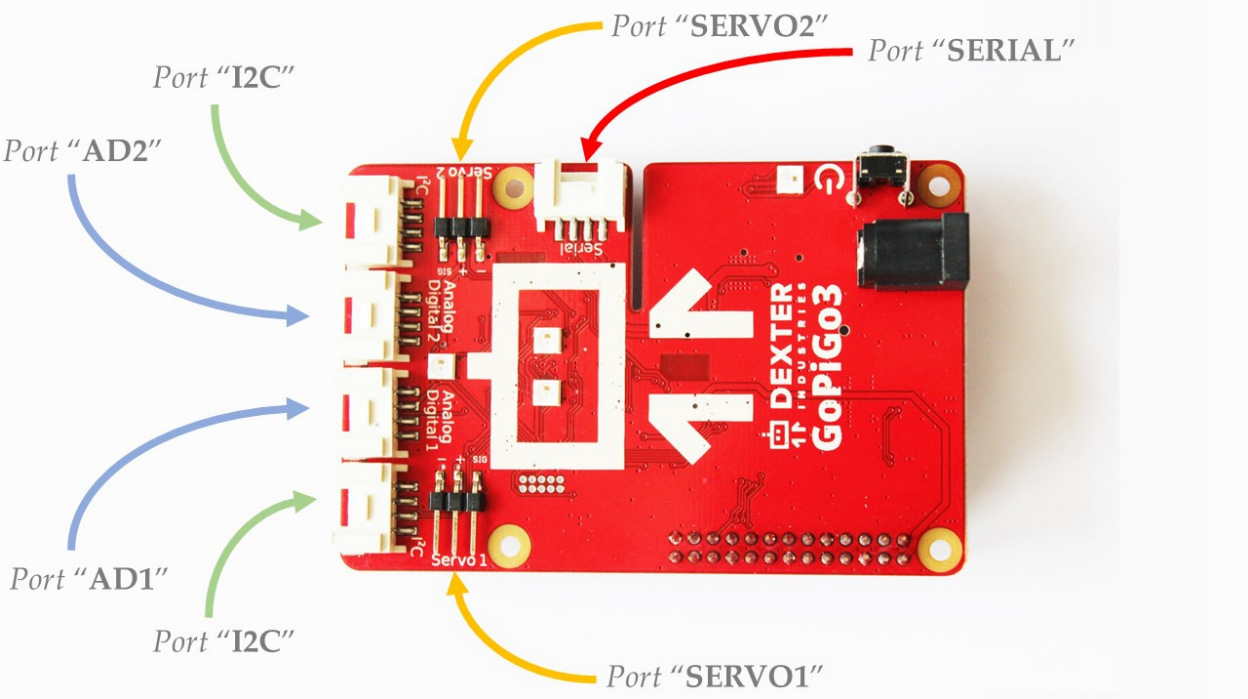
If you have issues importing these two modules, then make sure:

- You have followed the steps found in *Getting Started* guide.
- You have installed either Raspbian For Robots, the GoPiGo3 repository or the GoPiGo3 package (the pip package).
- You have the `gopigo3` package installed by typing the command `pip freeze | grep gopigo3` on your Raspberry Pi's terminal. If the package is installed, then a string with the `GoPiGo3==[x.y.z]` format will show up.

If you encounter issues that aren't covered by our *Getting Started* guide or *FAQ* chapter, please head over to our forum.

## 5.2 Hardware Ports

In this graphical representation, the GoPiGo3 board has the following ports available for use. The quoted literals are to be used as pin identifiers inside the python scripts.




These ports have the following functionalities:

- Ports `"AD1"` and `"AD2"` - general purpose input/output ports.
- Ports `"SERVO1"` and `"SERVO2"` - servo controller ports.
- Ports `"I2C"` - ports to which you can connect I2C-enabled devices.
- Port `"SERIAL"` - port to which you can connect UART-enabled device.

**Note:** Use the quoted port names when referencing them inside a python script like in the following example.

```
# we need an EasyGoPiGo3 object for instantiating sensor / actuator objects
gpg3_obj = EasyGoPiGo3()

# we're using the quoted port names from the above graphical representation

# here's a LightSensor object binded on port AD2
light_obj = gpg3_obj.init_light_sensor("AD2")

# here's a UltraSonicSensor object binded on port AD1
us_obj = gpg3_obj.init_ultrasonic_sensor("AD1")

# here's a LineFollower object binded on port I2C
line_follower_obj = gpg3_obj.init_line_follower("I2C")

# and so on
```

**See also:**

For more technical details on the GoPiGo3 robot, please check our technical specs page.

## 5.3 EasyGoPiGo3

## 5.4 LightSensor

## 5.5 SoundSensor

## 5.6 LoudnessSensor

## 5.7 UltrasonicSensor

## 5.8 Buzzer

## 5.9 Led

## 5.10 MotionSensor

## 5.11 ButtonSensor

## 5.12 LineFollower

## 5.13 Servo

## 5.14 DistanceSensor

## 5.15 DHTSensor

**Warning:** Coming soon!

## 5.16 Remote

CHAPTER 6

# API Reference Point - Advanced

## 6.1 Requirements

Before using this chapter's classes, you need to be able to import the following modules.

```
import easygopigo3
import gopigo3
```

If you have issues importing these 2 modules, then make sure that:

- You've followed the steps found in *Getting Started* guide.
- You have installed either Raspbian For Robots, the GoPiGo3 repository or the GoPiGo3 package (the pip package).
- You have the `gopigo3` package installed by typing the command `pip freeze | grep gopigo3` on your Raspberry Pi's terminal. If the package is installed, then a string with the `GoPiGo3==[x.y.z]` format will show up.

If you encounter issues that aren't covered by our *Getting Started* guide or *FAQ* chapter, please head over to our forum.

## 6.2 Sensor

## 6.3 DigitalSensor

**Note:** Coming soon!

## 6.4 AnalogSensor

CHAPTER 7

# Developer's Guide

**Note:** Coming soon!

## 7.1 Our contributors

1. Matt Richardson - Github Account
2. Nicole Parrot - Github Account
3. Robert Lucian Chiriac - Github Account
4. John Cole - Github Account

CHAPTER 8

# Frequently Asked Questions

**Note:** Coming soon!

For more questions, please head over to our Dexter Industries forum.

CHAPTER 9

# Indices and tables

- genindex
- modindex
- search