

---

# **carreralib**

***Release 1.0.2***

**Jan 15, 2023**



---

## Contents

---

<b>1</b>	<b>Demo RMS</b>	<b>3</b>
<b>2</b>	<b>Control Unit Firmware</b>	<b>5</b>
<b>3</b>	<b>API</b>	<b>7</b>
<b>4</b>	<b>Connection Module</b>	<b>11</b>
<b>5</b>	<b>Protocol Module</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



This module provides a Python interface to Carrera® DIGITAL 124/132 slotcar systems connected via a serial (cable) connection.

```
>>> from carreralib import ControlUnit
>>> cu = ControlUnit('/dev/ttyUSB0')
>>> cu.version()
'5337'
>>> cu.poll()
Status(fuel=(14, 14, 14, 14, 14, 14, 0, 0), start=0, mode=6,
      pit=(False, False, False, False, False, False, False, False),
      display=8)
>>> cu.start()
>>> cu.poll()
Status(fuel=(14, 14, 14, 14, 14, 14, 0, 0), start=1, mode=6,
      pit=(False, False, False, False, False, False, False, False),
      display=8)
>>> cu.start()
>>> cu.poll()
Timer(address=1, timestamp=105295, sector=1)
>>> cu.poll()
Timer(address=0, timestamp=105410, sector=1)
```

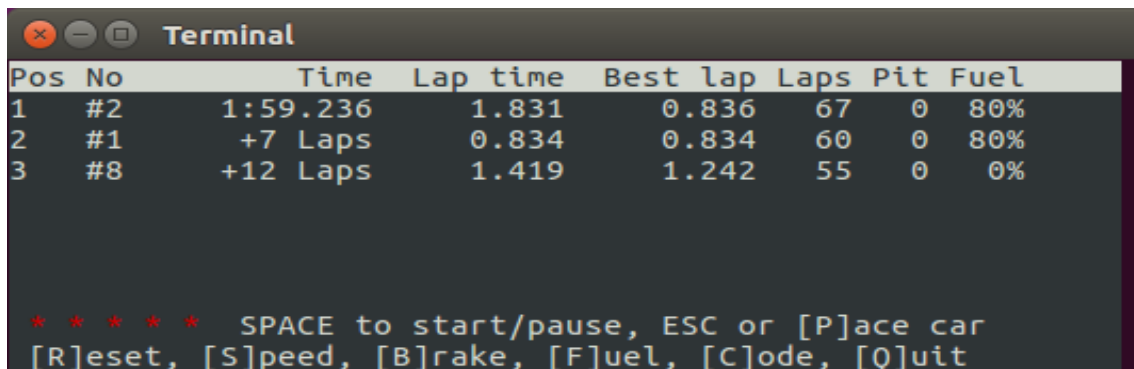


# CHAPTER 1

## Demo RMS

For demonstration purposes, the *carreralib* module can be used from the command line to run a simple curses-based race management system (RMS):

```
python -m carreralib /dev/ttyUSB0
```



Pos	No	Time	Lap time	Best lap	Laps	Pit	Fuel
1	#2	1:59.236	1.831	0.836	67	0	80%
2	#1	+7 Laps	0.834	0.834	60	0	80%
3	#8	+12 Laps	1.419	1.242	55	0	0%

\* \* \* \* \* SPACE to start/pause, ESC or [P]ace car  
[R]eset, [S]peed, [B]rake, [F]uel, [C]ode, [Q]uit

Within the RMS, use the space key to start or pause a race, R to reset a race, and Q to quit.

When called without a device name or path, this will list command line options and the serial or Bluetooth devices that a Control Unit may be connected to, e.g. on Linux:

```
$ python -m carreralib
usage: python -m carreralib [-h] [-l LOGFILE] [-t TIMEOUT] [-v] [DEVICE]

positional arguments:
  DEVICE                the Control Unit device, e.g. a serial port or MAC address

options:
  -h, --help            show this help message and exit
  -l LOGFILE, --logfile LOGFILE
                        where to write log messages
  -t TIMEOUT, --timeout TIMEOUT
```

(continues on next page)

(continued from previous page)

```
-v, --verbose          maximum time in seconds to wait for Control Unit
                        write more log messages

devices:
  /dev/ttyUSB0          USB-Serial Controller
```

On Windows, this will show the respective COM port:

```
devices:
  COM3                  Prolific USB-to-Serial Comm Port (COM3)
```

If a Carrera AppConnect® Bluetooth device is found, this will show the Bluetooth MAC address of the device instead:

```
devices:
  C6:34:FA:1D:1D:5D    Control_Unit
```

So instead of `/dev/ttyUSB0`, specify the respective COM port on Windows, or the Bluetooth MAC address (e.g. `C6:34:FA:1D:1D:5D`) when using Carrera AppConnect®.



## CHAPTER 2

---

### Control Unit Firmware

---

To show the current firmware version of your Control Unit, use:

```
python -m carreralib.fw /dev/ttyUSB0
```

---

**Note:** Trying to update the firmware of your Control Unit is a potentially dangerous operation that may wreck your hardware. Use at your own risk!

---

To upgrade (or downgrade) your Control Unit's firmware, given an ASCII firmware file, use:

```
python -m carreralib.fw /dev/ttyUSB0 digital_blackbox_NF_V337.HMF
```

---

**Note:** Control Unit firmware are the intellectual property of Carrera Toys GmbH, and are only provided by the copyright holders. Please do *not* ask for firmware files here!

---



## CHAPTER 3

---

### API

---

The `ControlUnit` class encapsulates a connection to a Carrera® DIGITAL 124/132 Control Unit (CU) and provides all the features needed to implement a custom race management system (RMS).

Note that `ControlUnit` uses zero-based controller addresses, so 0 corresponds to controller #1, 6 is the address the autonomous car, and 7 the address of the pace car.

**class** carreralib.**ControlUnit** (*device*, *\*\*kwargs*)

Interface to a Carrera Digital 124/132 Control Unit.

*device* should name a serial port, e.g. `/dev/ttyUSB0` on GNU/Linux or `COM3` on Windows. Additional keyword arguments will be passed to the underlying `Connection` object.

**BRAKE\_BUTTON\_ID = 6**

The Control Unit's BRAKE button ID.

**CODE\_BUTTON\_ID = 8**

The Control Unit's CODE button ID.

**FUEL\_BUTTON\_ID = 7**

The Control Unit's FUEL button ID.

**PACE\_CAR\_ESC\_BUTTON\_ID = 1**

The Control Unit's PACE CAR/ESC button ID.

**SPEED\_BUTTON\_ID = 5**

The Control Unit's SPEED button ID.

**START\_ENTER\_BUTTON\_ID = 2**

The Control Unit's START/ENTER button ID.

**class** **Status**

Response type returned if no timer events are pending.

This is a `collections.namedtuple` subclass with the following read-only attributes:

Attribute	Index	Value
fuel	0	Eight-item list of fuel levels (0..15)
start	1	Start light indicator (0..9)
mode	2	4-bit mode bit mask
pit	3	8-bit pit lane bit mask
display	4	Number of drivers to display (6 or 8)

**FUEL\_MODE = 1**

Mode bit mask indicating fule mode is enabled.

**LAP\_COUNTER\_MODE = 8**

Mode bit mask indicating a lap counter is connected.

**PIT\_LANE\_MODE = 4**

Mode bit mask indicating a pit lane adapter is connected.

**REAL\_MODE = 2**

Mode bit mask indicating real fuel mode is enabled.

**class Timer**

Response type for timer events.

This is a `collections.namedtuple` subclass with the following read-only attributes:

Attribute	Index	Value
address	0	Controller address (0..7)
timestamp	1	32-bit time stamp in milleseconds
sector	2	Sector (1 for start/finish, 2 or 3 for times reported by Check Lanes)

**close()**

Close the connection to the CU.

**clrpos()**

Clear/reset the Position Tower display.

**fwu\_start()**

Initiate a CU firmware update.

**fwu\_write(data)**

Write CU firmware update data.

**ignore(mask)**

Ignore the controllers represented by bitmask *mask*.

**poll()**

Poll the CU for pending messages.

The returned value will be an instance of either `ControlUnit.Timer` or `ControlUnit.Status`, depending on whether any timer events are pending.

**press(button\_id)**

Simulate pressing the CU button with the given ID.

**request(buf, maxlength=None)**

Send a message to the CU and wait for a response.

**reset()**

Reset the CU timer.

**setbrake** (*address, value*)  
Set the brake value for controller *address*.

**setfuel** (*address, value*)  
Set the fuel value for controller *address*.

**setlap** (*value*)  
Set the current lap displayed by the Position Tower.

**setlap\_hi** (*value*)  
Set the high nibble of the current lap.

**setlap\_lo** (*value*)  
Set the low nibble of the current lap.

**setpos** (*address, position*)  
Set the controller's position displayed by the Position Tower.

**setspeed** (*address, value*)  
Set the speed value for controller address.

**start** ()  
Initiate the CU start sequence.

**version** ()  
Retrieve the CU version as a string.



## CHAPTER 4

---

### Connection Module

---

This module is mostly of interest to developers who want to create their own connection implementation, for example to use a Bluetooth implementation.

**exception** carreralib.connection.**BufferTooShort**

Raised when the supplied buffer is too small a message.

**class** carreralib.connection.**Connection** (*device*, *\*\*kwargs*)

Base class for connections to a Carrera digital slotcar system.

**close** ()

Close the connection.

**max\_fwu\_block\_size** = None

Maximum number of bytes in one firmware update frame.

**recv** (*maxlength=None*)

Return a complete message of byte data sent from the other end of the connection as a bytes object.

**send** (*buf*, *offset=0*, *size=None*)

Send byte data from an object supporting the buffer interface as a complete message.

**exception** carreralib.connection.**ConnectionError**

The base class of all connection exceptions.

**exception** carreralib.connection.**TimeoutError**

Raised when a timeout expires.

carreralib.connection.**open** (*device*, *\*\*kwargs*)

Open a connection to the given device.

carreralib.connection.**scan** ()

Search for potential devices.





Protocol Module

---

This module provides utility functions for dealing with the Carrera® DIGITAL 124/132 protocol.

`carreralib.protocol.pack(fmt, *args)`

Return a bytes object containing the arguments packed according to the format string *fmt*.

Similar to `struct.pack()`, this function performs conversion between Python values and binary protocol data. Format strings may contain the following characters:

Format	Python type	Value
B	int	unsigned byte
c	bytes of length 1	a single ASCII character
C	no value	check sum
I	int	32-bit unsigned integer
r	int	“raw” unsigned byte (BLE only)
s	bytes	ASCII string
x	no value	padding/ignored
Y	int	nibble or <i>nybble</i>

As with `struct.pack()`, a format character may be preceded by an integral count. For the `s` format character, the count is interpreted as the length of the bytes. For the `C` format character, the count is interpreted as an *offset* in the generated buffer from which the checksum should be calculated. For the other format characters, the count is simply interpreted as a repeat count.

`carreralib.protocol.unpack(fmt, buf)`

Unpack from the buffer *buf* according to the format string *fmt*.

`carreralib.protocol.chksum(buf, offset=0, size=None)`

Compute the protocol checksum for the buffer *buf*.



### C

carreralib, [??](#)

carreralib.connection, [11](#)

carreralib.protocol, [13](#)



## B

BRAKE\_BUTTON\_ID (*carreralib.ControlUnit* attribute), 7

BufferTooShort, 11

## C

carreralib (*module*), 1

carreralib.connection (*module*), 11

carreralib.protocol (*module*), 13

chksum () (*in module carreralib.protocol*), 13

close () (*carreralib.connection.Connection* method), 11

close () (*carreralib.ControlUnit* method), 8

clrpos () (*carreralib.ControlUnit* method), 8

CODE\_BUTTON\_ID (*carreralib.ControlUnit* attribute), 7

Connection (*class in carreralib.connection*), 11

ConnectionError, 11

ControlUnit (*class in carreralib*), 7

ControlUnit.Status (*class in carreralib*), 7

ControlUnit.Timer (*class in carreralib*), 8

## F

FUEL\_BUTTON\_ID (*carreralib.ControlUnit* attribute), 7

FUEL\_MODE (*carreralib.ControlUnit.Status* attribute), 8

fwu\_start () (*carreralib.ControlUnit* method), 8

fwu\_write () (*carreralib.ControlUnit* method), 8

## I

ignore () (*carreralib.ControlUnit* method), 8

## L

LAP\_COUNTER\_MODE (*carreralib.ControlUnit.Status* attribute), 8

## M

max\_fwu\_block\_size (*carreralib.connection.Connection* attribute), 11

## O

open () (*in module carreralib.connection*), 11

## P

PACE\_CAR\_ESC\_BUTTON\_ID (*carreralib.ControlUnit* attribute), 7

pack () (*in module carreralib.protocol*), 13

PIT\_LANE\_MODE (*carreralib.ControlUnit.Status* attribute), 8

poll () (*carreralib.ControlUnit* method), 8

press () (*carreralib.ControlUnit* method), 8

## R

REAL\_MODE (*carreralib.ControlUnit.Status* attribute), 8

recv () (*carreralib.connection.Connection* method), 11

request () (*carreralib.ControlUnit* method), 8

reset () (*carreralib.ControlUnit* method), 8

## S

scan () (*in module carreralib.connection*), 11

send () (*carreralib.connection.Connection* method), 11

setbrake () (*carreralib.ControlUnit* method), 8

setfuel () (*carreralib.ControlUnit* method), 9

setlap () (*carreralib.ControlUnit* method), 9

setlap\_hi () (*carreralib.ControlUnit* method), 9

setlap\_lo () (*carreralib.ControlUnit* method), 9

setpos () (*carreralib.ControlUnit* method), 9

setspeed () (*carreralib.ControlUnit* method), 9

SPEED\_BUTTON\_ID (*carreralib.ControlUnit* attribute), 7

start () (*carreralib.ControlUnit* method), 9

START\_ENTER\_BUTTON\_ID (*carreralib.ControlUnit* attribute), 7

## T

TimeoutError, 11

## U

unpack () (*in module carreralib.protocol*), 13

## V

`version()` (*carreralib.ControlUnit* method), [9](#)