python-ev3dev Documentation

Release 1.0.0

Ralph Hempel et al

Sep 05, 2017

Contents

A Python3 library implementing an interface for ev3dev devices, letting you control motors, sensors, hardware buttons, LCD displays and more from Python code.

If you haven't written code in Python before, you'll need to learn the language before you can use this library.

CHAPTER 1

Getting Started

This library runs on ev3dev. Before continuing, make sure that you have set up your EV3 or other ev3dev device as explained in the ev3dev Getting Started guide. Make sure that you have a kernel version that includes `-10-ev3dev` or higher (a larger number). You can check the kernel version by selecting "About" in Brickman and scrolling down to the "kernel version". If you don't have a compatible version, upgrade the kernel before continuing. Also note that if the ev3dev image you downloaded was created before September 2016, you probably don't have the most recent version of this library installed: see *Upgrading this Library* to upgrade it.

Once you have booted ev3dev and connected to your EV3 (or Raspberry Pi / BeagleBone) via SSH, you should be ready to start using ev3dev with Python: this library is included out-of-the-box. If you want to go through some basic usage examples, check out the *Usage Examples* section to try out motors, sensors and LEDs. Then look at *Writing Python Programs for Ev3dev* to see how you can save your Python code to a file.

Make sure that you look at the *User Resources* section as well for links to documentation and larger examples.

CHAPTER 2

Usage Examples

To run these minimal examples, run the Python3 interpreter from the terminal using the `python3` command:

```
$ python3
Python 3.4.2 (default, Oct  8 2014, 14:47:30)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The >>> characters are the default prompt for Python. In the examples below, we have removed these characters so it's easier to cut and paste the code into your session.

Required: Import the library

If you are using an EV3 brick (which is the case for most users), add the following to the top of your file:

```
import ev3dev.ev3 as ev3
```

If you are using a BrickPi, use this line:

```
import ev3dev.brickpi as ev3
```

Controlling the LEDs with a touch sensor

This code will turn the left LED red whenever the touch sensor is pressed, and back to green when it's released. Plug a touch sensor into any sensor port and then paste in this code - you'll need to hit `Enter` after pasting to complete the loop and start the program. Hit `Ctrl-C` to exit the loop.

```
ts = ev3.TouchSensor()
while True:
    ev3.Leds.set_color(ev3.Leds.LEFT, (ev3.Leds.GREEN, ev3.Leds.RED)[ts.value()])
```

Running a motor

Now plug a motor into the `A` port and paste this code into the Python prompt. This little program will run the motor at 500 ticks per second, which on the EV3 "large" motors equates to around 1.4 rotations per second, for three seconds (3000 milliseconds).

```
m = ev3.LargeMotor('outA')
m.run_timed(time_sp=3000, speed_sp=500)
```

The units for `speed_sp` that you see above are in "tacho ticks" per second. On the large EV3 motor, these equate to one tick per degree, so this is 500 degress per second.

Using text-to-speech

If you want to make your robot speak, you can use the *Sound.speak* method:

```
ev3.Sound.speak('Welcome to the E V 3 dev project!').wait()
```

To quit the Python REPL, just type `exit()` or press `Ctrl-D`.

Make sure to check out the *User Resources* section for more detailed information on these features and many others.

CHAPTER 3

Writing Python Programs for Ev3dev

Every Python program should have a few basic parts. Use this template to get started:

```
#!/usr/bin/env python3
from ev3dev.ev3 import *

# TODO: Add code here
```

The first two lines should be included in every Python program you write for ev3dev. The first allows you to run this program from Brickman, while the second imports this library.

When saving Python files, it is best to use the `.py` extension, e.g. `my-file.py`. To be able to run your Python code, **your program must be executable**. To mark a program as executable run `chmod +x my-file.py`. You can then run `my-file.py` via the Brickman File Browser or you can run it from the command line via `$ ./my-file.py`

CHAPTER 4

User Resources

Library Documentation **Class documentation for this library can be found on** our Read the Docs page **.** You can always go there to get information on how you can use this library's functionality.

ev3python.com One of our community members, @ndward, has put together a great website with detailed guides on using this library which are targeted at beginners. If you are just getting started with programming, we highly recommend that you check it out at ev3python.com!

Frequently-Asked Questions Experiencing an odd error or unsure of how to do something that seems simple? Check our our FAQ to see if there's an existing answer.

ev3dev.org ev3dev.org is a great resource for finding guides and tutorials on using ev3dev, straight from the maintainers.

Support If you are having trouble using this library, please open an issue at our Issues tracker so that we can help you. When opening an issue, make sure to include as much information as possible about what you are trying to do and what you have tried. The issue template is in place to guide you through this process.

Demo Robot Laurens Valk of robot-square has been kind enough to allow us to reference his excellent EXPLOR3R robot. Consider building the EXPLOR3R and running the demo programs referenced below to get familiar with what Python programs using this binding look like.

Demo Code There are demo programs that you can run to get acquainted with this language binding. The programs are designed to work with the EXPLOR3R robot.

CHAPTER 5

Upgrading this Library

You can upgrade this library from the command line as follows. Make sure to type the password (the default is `maker`) when prompted.

```
sudo apt-get update
sudo apt-get install --only-upgrade python3-ev3dev
```

CHAPTER 6

Developer Resources

Python Package Index The Python language has a package repository where you can find libraries that others have written, including the latest version of this package.

The ev3dev Binding Specification Like all of the language bindings for ev3dev supported hardware, the Python binding follows the minimal API that must be provided per this document.

The ev3dev-lang Project on GitHub The source repository for the generic API and the scripts to automatically generate the binding. Only developers of the ev3dev-lang-python binding would normally need to access this information.

CHAPTER 7

Python 2.x and Python 3.x Compatibility

Some versions of the ev3dev distribution come with both Python 2.x and Python 3.x installed but this library is compatible only with Python 3.

As of the 2016-10-17 ev3dev image, the version of this library which is included runs on Python 3 and this is the only version that will be supported from here forward.

Contents

API reference

Each class in ev3dev module inherits from the base `Device` class.

Contents:

Motor classes

Tacho motor

Large EV3 Motor

Medium EV3 Motor

DC Motor

Servo Motor

Sensor classes

Sensor

This is the base class all the other sensor classes are derived from.

Special sensor classes

The classes derive from `Sensor` and provide helper functions specific to the corresponding sensor type. Each of the functions makes sure the sensor is in the required mode and then returns the specified value.

Touch Sensor

Color Sensor

Ultrasonic Sensor

Gyro Sensor

Infrared Sensor

Sound Sensor

Light Sensor

Other classes

Remote Control

Beacon Seeker

Button

Leds

Power Supply

Sound

Screen

Bitmap fonts

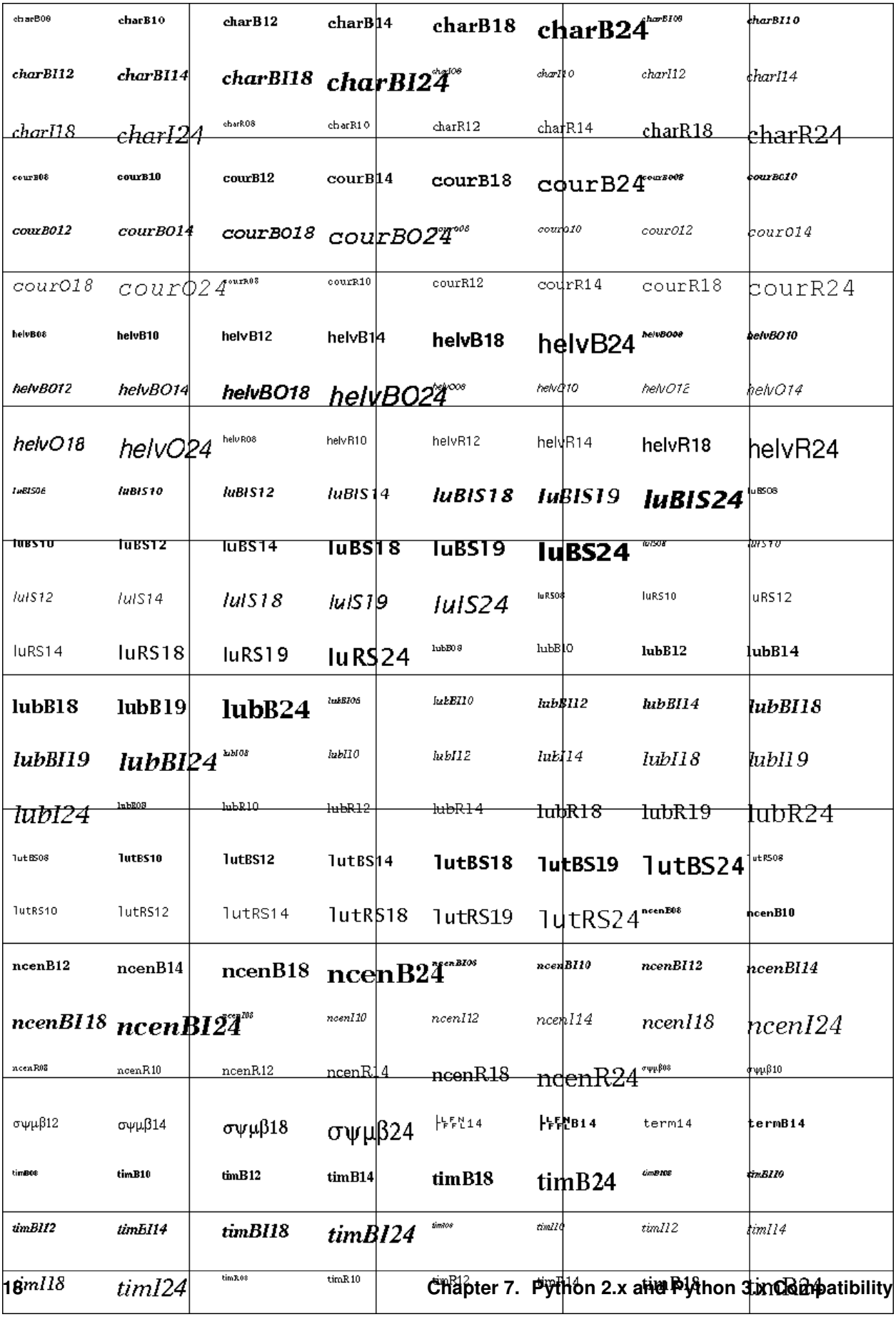
The `Screen` class allows to write text on the LCD using python imaging library (PIL) interface (see description of the `text()` method here). The `ev3dev.fonts` module contains bitmap fonts in PIL format that should look good on a tiny EV3 screen:

```
import ev3dev.fonts as fonts
screen.draw.text((10,10), 'Hello World!', font=fonts.load('luBS14'))
```

`ev3dev.fonts.`**`available`**`()`
Returns list of available font names.

ev3dev.fonts.**load**(*name*)
Loads the font specified by name and returns it as an instance of PIL.ImageFont class.

The following image lists all available fonts. The grid lines correspond to EV3 screen size:


charB08 charB10 charB12 charB14 charB18 charB24 charBI08 charBI10
charBI12 charBI14 charBI18 charBI24 charI08 charI10 charI12 charI14
charI18 charI24 charR08 charR10 charR12 charR14 charR18 charR24
courB08 courB10 courB12 courB14 courB18 courB24 courBO08 courBO10
courBO12 courBO14 courBO18 courBO24 courO08 courO10 courO12 courO14
courO18 courO24 courR08 courR10 courR12 courR14 courR18 courR24
helvB08 helvB10 helvB12 helvB14 helvB18 helvB24 helvBO08 helvBO10
helvBO12 helvBO14 helvBO18 helvBO24 helvO08 helvO10 helvO12 helvO14
helvO18 helvO24 helvR08 helvR10 helvR12 helvR14 helvR18 helvR24
luBIS08 luBIS10 luBIS12 luBIS14 luBIS18 luBIS19 luBIS24 luBS08
luBS10 luBS12 luBS14 luBS18 luBS19 luBS24 luIS08 luIS10
luIS12 luIS14 luIS18 luIS19 luIS24 luRS08 luRS10 uRS12
luRS14 luRS18 luRS19 luRS24 lubB08 lubB10 lubB12 lubB14
lubB18 lubB19 lubB24 lubBI08 lubBI10 lubBI12 lubBI14 lubBI18
lubBI19 lubBI24 lubI08 lubI10 lubI12 lubI14 lubI18 lubI19
lubI24 lubR08 lubR10 lubR12 lubR14 lubR18 lubR19 lubR24
lutBS08 lutBS10 lutBS12 lutBS14 lutBS18 lutBS19 lutBS24 lutRS08
lutRS10 lutRS12 lutRS14 lutRS18 lutRS19 lutRS24 ncenB08 ncenB10
ncenB12 ncenB14 ncenB18 ncenB24 ncenBI08 ncenBI10 ncenBI12 ncenBI14
ncenBI18 ncenBI24 ncenI08 ncenI10 ncenI12 ncenI14 ncenI18 ncenI24
ncenR08 ncenR10 ncenR12 ncenR14 ncenR18 ncenR24 σψμβ08 σψμβ10
σψμβ12 σψμβ14 σψμβ18 σψμβ24 TERM14 TERMB14 term14 termB14
timB08 timB10 timB12 timB14 timB18 timB24 timBI08 timBI10
timBI12 timBI14 timBI18 timBI24 timI08 timI10 timI12 timI14
timI18 timI24 timR08 timR10 timR12 timR14 timR18 timR24

Lego Port

Working with ev3dev remotely using RPyC

RPyC (pronounced as are-pie-see), or Remote Python Call, is a transparent python library for symmetrical remote procedure calls, clustering and distributed-computing. RPyC makes use of object-proxying, a technique that employs python's dynamic nature, to overcome the physical boundaries between processes and computers, so that remote objects can be manipulated as if they were local. Here are simple steps you need to follow in order to install and use RPyC with ev3dev:

1. Install RPyC both on the EV3 and on your desktop PC. For the EV3, enter the following command at the command prompt (after you connect with SSH):

   ```
   sudo easy_install3 rpyc
   ```

   On the desktop PC, it really depends on your operating system. In case it is some flavor of linux, you should be able to do

   ```
   sudo pip3 install rpyc
   ```

   In case it is Windows, there is a win32 installer on the project's sourceforge page. Also, have a look at the Download and Install page on their site.

2. Create file `rpyc_server.sh` with the following contents on the EV3:

   ```
   #!/bin/bash
   python3 `which rpyc_classic.py`
   ```

   and make the file executable:

   ```
   chmod +x rpyc_server.sh
   ```

   Launch the created file either from SSH session (with `./rpyc_server.sh` command), or from brickman. It should output something like

   ```
   INFO:SLAVE/18812:server started on [0.0.0.0]:18812
   ```

   and keep running.

3. Now you are ready to connect to the RPyC server from your desktop PC. The following python script should make a large motor connected to output port `A` spin for a second.

   ```
   import rpyc
   conn = rpyc.classic.connect('ev3dev') # host name or IP address of the EV3
   ev3 = conn.modules['ev3dev.ev3']      # import ev3dev.ev3 remotely
   m = ev3.LargeMotor('outA')
   m.run_timed(time_sp=1000, speed_sp=600)
   ```

You can run scripts like this from any interactive python environment, like ipython shell/notebook, spyder, pycharm, etc.

Some *advantages* of using RPyC with ev3dev are:

- It uses much less resources than running ipython notebook on EV3; RPyC server is lightweight, and only requires an IP connection to the EV3 once set up (no ssh required).
- The scripts you are working with are actually stored and edited on your desktop PC, with your favorite editor/IDE.

- Some robots may need much more computational power than what EV3 can give you. A notable example is the Rubics cube solver: there is an algorithm that provides almost optimal solution (in terms of number of cube rotations), but it takes more RAM than is available on EV3. With RPYC, you could run the heavy-duty computations on your desktop.

The most obvious *disadvantage* is latency introduced by network connection. This may be a show stopper for robots where reaction speed is essential.

Frequently-Asked Questions

My script works when launched as `python3 script.py` but exits immediately or throws an error when launched from Brickman or as `./script.py`

This may occur if your file includes Windows-style line endings, which are often inserted by editors on Windows. To resolve this issue, open an SSH session and run the following command, replacing `<file>` with the name of the Python file you're using:

```
sed -i 's/\r//g' <file>
```

This will fix it for the copy of the file on the brick, but if you plan to edit it again from Windows you should configure your editor to use Unix-style endings. For PyCharm, you can find a guide on doing this here. Most other editors have similar options; there may be an option for it in the status bar at the bottom of the window or in the menu bar at the top.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index