`media` – Sounds and Images

This module describes media such as sound and images that you can use in your projects. Media are divided into submodules that indicate on which platform they are available.

`media.ev3dev` – Sounds and Images

EV3 MicroPython is built on top of ev3dev, which comes with a variety of image and sound files. You can access them using the classes below.

You can also use your own sound and image files by placing them in your project folder.

Image Files

class **`ImageFile`**

Paths to standard EV3 images.

Information ▲

`ACCEPT`

`BACKWARD`

`DECLINE`

`FORWARD`

LEFT

←

NO_GO

⊘

QUESTION_MARK

?

RIGHT

→

STOP_1

STOP

STOP_2

STOP

THUMBS_DOWN

👎

THUMBS_UP

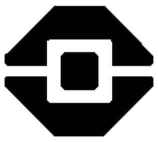
👍

WARNING

LEGO ▲

> `EV3`

> `EV3_ICON`



Objects ▲

> `TARGET`

Eyes ▲

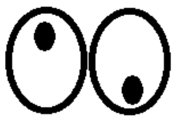
> `ANGRY`

> `AWAKE`

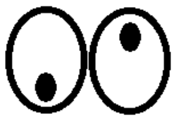
> `BOTTOM_LEFT`

> `BOTTOM_RIGHT`

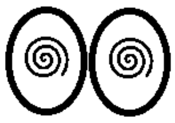
CRAZY_1



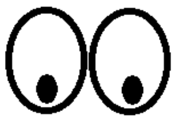
CRAZY_2



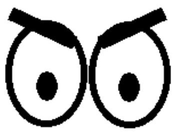
DIZZY



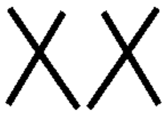
DOWN



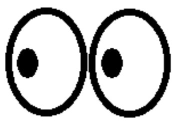
EVIL



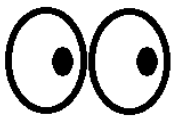
KNOCKED_OUT



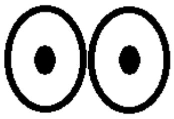
MIDDLE_LEFT



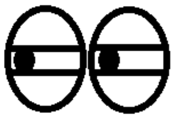
MIDDLE_RIGHT



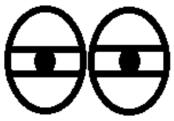
NEUTRAL



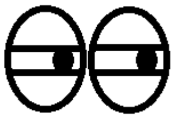
PINCHED_LEFT



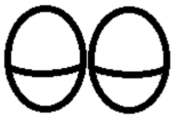
PINCHED_MIDDLE



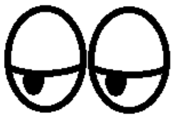
PINCHED_RIGHT



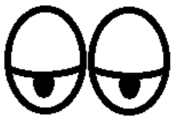
SLEEPING



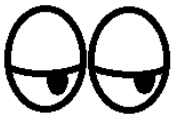
TIRED_LEFT



TIRED_MIDDLE

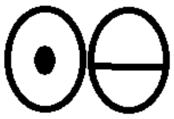


TIRED_RIGHT



UP

`WINKING`



Sound Files

class `SoundFile`

Paths to standard EV3 sounds.

Expressions ▲

`BOING`

0:00 / 0:00

Download

`BOO`

0:00 / 0:01

Download

`CHEERING`

0:00 / 0:03

Download

`CRUNCHING`

0:00 / 0:02

Download

`CRYING`

0:00 / 0:01

Download

FANFARE

0:00 / 0:02

Download

KUNG_FU

0:00 / 0:00

Download

LAUGHING_1

0:00 / 0:00

Download

LAUGHING_2

0:00 / 0:00

Download

MAGIC_WAND

0:00 / 0:01

Download

OUCH

0:00 / 0:00

Download

SHOUTING

0:00 / 0:00

Download

SMACK

0:00 / 0:00

Download

`SNEEZING`

0:00 / 0:00

Download

`SNORING`

0:00 / 0:01

Download

`UH_OH`

0:00 / 0:00

Download

Information ▲

`ACTIVATE`

0:00 / 0:01

Download

`ANALYZE`

0:00 / 0:01

Download

`BACKWARDS`

0:00 / 0:01

Download

`COLOR`

0:00 / 0:01

Download

> `DETECTED`

0:00 / 0:01

Download

> `DOWN`

0:00 / 0:00

Download

> `ERROR`

0:00 / 0:01

Download

> `ERROR_ALARM`

0:00 / 0:01

Download

> `FLASHING`

0:00 / 0:01

Download

> `FORWARD`

0:00 / 0:01

Download

> `LEFT`

0:00 / 0:01

Download

> **OBJECT**

0:00 / 0:01

Download

> **RIGHT**

0:00 / 0:00

Download

> **SEARCHING**

0:00 / 0:01

Download

> **START**

0:00 / 0:01

Download

> **STOP**

0:00 / 0:00

Download

> **TOUCH**

0:00 / 0:01

Download

> **TURN**

0:00 / 0:00

Download

UP

0:00 / 0:00

Download

Communication ▲

BRAVO

0:00 / 0:01

Download

EV3

0:00 / 0:01

Download

FANTASTIC

0:00 / 0:01

Download

GAME_OVER

0:00 / 0:01

Download

GO

0:00 / 0:00

Download

GOOD_JOB

0:00 / 0:01

Download

GOOD

0:00 / 0:00

Download

> **GOODBYE**

0:00 / 0:01

Download

> **HELLO**

0:00 / 0:00

Download

> **HI**

0:00 / 0:00

Download

> **LEGO**

0:00 / 0:01

Download

> **MINDSTORMS**

0:00 / 0:01

Download

> **MORNING**

0:00 / 0:01

Download

> **NO**

0:00 / 0:00

Download

> `OKAY`

0:00 / 0:01

Download

> `OKEY_DOKEY`

0:00 / 0:01

Download

> `SORRY`

0:00 / 0:01

Download

> `THANK_YOU`

0:00 / 0:01

Download

> `YES`

0:00 / 0:01

Download

Movement sounds ▲

> `SPEED_DOWN`

0:00 / 0:01

Download

> `SPEED_IDLE`

0:00 / 0:00

Download

`SPEED_UP`

0:00 / 0:01

Download

Colors ▲

`BLACK`

0:00 / 0:01

Download

`BLUE`

0:00 / 0:00

Download

`BROWN`

0:00 / 0:01

Download

`GREEN`

0:00 / 0:00

Download

`RED`

0:00 / 0:00

Download

`WHITE`

0:00 / 0:00

Download

`YELLOW`

0:00 / 0:01

Download

Mechanical ▲

`AIR_RELEASE`

0:00 / 0:01

Download

`AIRBRAKE`

0:00 / 0:00

Download

`BACKING_ALERT`

0:00 / 0:02

Download

`HORN_1`

0:00 / 0:00

Download

`HORN_2`

0:00 / 0:01

Download

`LASER`

0:00 / 0:00

Download

`MOTOR_IDLE`

0:00 / 0:00

Download

> `MOTOR_START`

0:00 / 0:01

Download

> `MOTOR_STOP`

0:00 / 0:00

Download

> `RATCHET`

0:00 / 0:00

Download

> `SONAR`

0:00 / 0:02

Download

> `TICK_TACK`

0:00 / 0:01

Download

Animal sounds ▲

> `CAT_PURR`

0:00 / 0:03

Download

> `DOG_BARK_1`

0:00 / 0:02

Download

`DOG_BARK_2`

0:00 / 0:00

Download

`DOG_GROWL`

0:00 / 0:02

Download

`DOG_SNIFF`

0:00 / 0:01

Download

`DOG_WHINE`

0:00 / 0:01

Download

`ELEPHANT_CALL`

0:00 / 0:01

Download

`INSECT_BUZZ_1`

0:00 / 0:02

Download

`INSECT_BUZZ_2`

0:00 / 0:02

Download

`INSECT_CHIRP`

0:00 / 0:02

Download

`SNAKE_HISS`

0:00 / 0:01

Download

`SNAKE_RATTLE`

0:00 / 0:01

Download

`T_REX_ROAR`

0:00 / 0:01

Download

Numbers ▲

`ZERO`

0:00 / 0:01

Download

`ONE`

0:00 / 0:00

Download

`TWO`

0:00 / 0:00

Download

THREE

0:00 / 0:00

Download

FOUR

0:00 / 0:00

Download

FIVE

0:00 / 0:01

Download

SIX

0:00 / 0:01

Download

SEVEN

0:00 / 0:01

Download

EIGHT

0:00 / 0:00

Download

NINE

0:00 / 0:00

Download

TEN

0:00 / 0:00

Download

System sounds ▲

CLICK

0:00 / 0:00

Download

CONFIRM

0:00 / 0:01

Download

GENERAL_ALERT

0:00 / 0:00

Download

OVERPOWER

0:00 / 0:01

Download

READY

0:00 / 0:00

Download

Fonts

class `Font`*(family=None, size=12, bold=False, monospace=False, lang=None, script=None)*

Object that represents a font for writing text.

The font object will be a font that is the “best” match based on the parameters given and available fonts installed.

Parameters:

- **family** (*str*) – The preferred font family or `None` to use the default value.
- **size** (*int*) – The preferred font size. Most fonts have sizes between 6 and 24. This is the "point" size and not the same as `height`.
- **bold** (*bool*) – When `True`, prefer bold fonts.
- **monospace** (*bool*) – When `True` prefer monospaced fonts. This is useful for aligning multiple rows of text.
- **lang** (*str*) – A language code, such as `'en'` or `'zh-cn'` or `None` to use the default language. [1]
- **script** (*str*) – A unicode script identifier such as `'Runr'` or `None`.

[1] Language codes ▼

`DEFAULT`= *Font('Lucida', 12)*

The default font.

`family`

Gets the family name of the font.

`style`

Gets a string describing the font style.

Can be "Regular" or "Bold".

`width`

Gets the width of the widest character of the font.

`height`

Gets the height of the font.

`text_width`(*text*)

Gets the width of the text when the text is drawn using this font.

Parameters:	**text** (*str*) – The text.
Returns:	The width in pixels.
Return type:	int

`text_height`(*text*)

Gets the height of the text when the text is drawn using this font.

Parameters: **text** (*str*) – The text.

Returns: The height in pixels.

Return type: int

Exploring more fonts

Behind the scenes, Pybricks uses Fontconfig for fonts. The Fontconfig command line tools can be used to explore available fonts in more detail. To do so, go to the ev3dev device browser, right click on your EV3 brick, and click *Open SSH Terminal*. Then you can enter one of these commands:

```
# List available font families.
fc-list :scalable=false family
# Perform lookup similar to Font.DEFAULT
fc-match :scalable=false:dpi=119:family=Lucida:size=12
# Perform lookup similar to Font(size=24,lang=zh-cn)
fc-match :scalable=false:dpi=119:size=24:lang=zh-cn
```

Pybricks only allows the use of bitmap fonts (`scalable=false`) and the screen on the EV3 has 119 pixels per inch (`dpi=119`).

Image Manipulation

Instead of drawing directly on the EV3 screen, you can make and interact with image files using the `Image` class given below.

class **`Image`**(*source*, *sub=False*)

Object representing a graphics image. This can either be an in-memory copy of an image or the image displayed on a screen.

Parameters:

- **source** (*str or Image*) –
  The source of the image.

  If `source` is a string, then the image will be loaded from the file path given by the string. Only `.png` files are supported. As a special case, if the string is `_screen_`, the image will be configured to draw directly on the screen.

  If an `Image` is given, the new object will contain a copy of the `source` image object.

- **sub** (*bool*) –
  If `sub` is `True`, then the image object will act as a sub-image of the `source` image (this only works if the type of `source` is `Image` and not when it is a `str`).

  Additional keyword arguments `x1`, `y1`, `x2`, `y2` are needed when `sub=True`. These specify the top-left and bottom-right coordinates in the `source` image that will be used as the bounds for the sub-image.

static **`empty`**(*width=<screen width>*, *height=<screen height>*)

Creates a new empty `Image` object.

Parameters:

- **width** (*int*) – The width of the image in pixels.
- **height** (*int*) – The height of the image in pixels.

Returns: A new image with all pixels set to `Color.WHITE`.

Return type: Image

Raises:

- `TypeError` – `width` or `height` is not a number.
- `ValueError` – `width` or `height` is less than 1.
- `RuntimeError` – There was a problem allocating a new image.

Drawing text

There are two ways to draw text on images. `draw_text()` lets text be placed precisely on the image or `print()` can be used to automatically print text on a new line.

`draw_text`(*x*, *y*, *text*, *text_color=Color.BLACK*, *background_color=None*)

Draws text on this image.

The most recent font set using `set_font()` will be used or `Font.DEFAULT` if no font has been set yet.

Parameters:

- **x** (*int*) – The x-axis value where the left side of the text will start.
- **y** (*int*) – The y-axis value where the top of the text will start.
- **text** (*str*) – The text to draw.
- **text_color** (*Color*) – The color used for drawing the text.
- **background_color** (*Color*) – The color used to fill the rectangle behind the text or `None` for transparent background.

`print`(**args*, *sep=' '*, *end='\n'*)

Prints a line of text on this image.

This method works like the builtin `print()` function, but it writes on this image instead.

You can set the font using `set_font()`. If no font has been set, `Font.DEFAULT` will be used. The text is always printed used black text with a white background.

Unlike the builtin `print()`, the text does not wrap if it is too wide to fit on this image. It just gets cut off. But if the text would go off of the bottom of this image, the entire image is scrolled up and the text is printed in the new blank area at the bottom of this image.

Parameters:

- * (*object*) – Zero or more objects to print.
- **sep** (*str*) – Separator that will be placed between each object that is printed.
- **end** (*str*) – End of line that will be printed after the last object.

`set_font`(*font*)

Sets the font used for writing on this image.

The font is used for both `draw_text()` and `print()`.

Parameters: **font** (`Font`) – The font to use.

Drawing images

A copy of another image can be drawn on an image. Also consider using sub-images to copy part of an image.

`draw_image`(*x*, *y*, *source*, *transparent=None*)

Draws the `source` image on this image.

Parameters:

- **x** (*int*) – The x-axis value where the left side of the image will start.
- **y** (*int*) – The y-axis value where the top of the image will start.
- **source** (*Image or str*) – The source `Image`. If the argument is a string, then the `source` image is loaded from file.
- **transparent** (*Color*) – The color of `image` to treat as transparent or `None` for no transparency.

Drawing shapes

These are the methods to draw basic shapes, including points, lines, rectangles and circles.

`draw_pixel`(*x*, *y*, *color=Color.BLACK*)

Draws a single pixel on this image.

Parameters:

- **x** (*int*) – The x coordinate of the pixel.
- **y** (*int*) – The y coordinate of the pixel.
- **color** (*Color*) – The color of the pixel.

`draw_line`(*x1*, *y1*, *x2*, *y2*, *width=1*, *color=Color.BLACK*)

Draws a line on this image.

Parameters:

- **x1** (*int*) – The x coordinate of the starting point of the line.
- **y1** (*int*) – The y coordinate of the starting point of the line.
- **x2** (*int*) – The x coordinate of the ending point of the line.
- **y2** (*int*) – The y coordinate of the ending point of the line.
- **width** (*int*) – The width of the line in pixels.
- **color** (*Color*) – The color of the line.

`draw_box`(*x1*, *y1*, *x2*, *y2*, *r=0*, *fill=False*, *color=Color.BLACK*)

Draws a box on this image.

Parameters:

- **x1** (*int*) – The x coordinate of the left side of the box.
- **y1** (*int*) – The y coordinate of the top of the box.
- **x2** (*int*) – The x coordinate of the right side of the box.
- **y2** (*int*) – The y coordinate of the bottom of the box.
- **r** (*int*) – The radius of the corners of the box.
- **fill** (*bool*) – If `True`, the box will be filled with `color`, otherwise only the outline of the box will be drawn.
- **color** (*Color*) – The color of the box.

`draw_circle`(*x, y, r, fill=False, color=Color.BLACK*)

Draws a circle on this image.

Parameters:
- **x** (*int*) – The x coordinate of the center of the circle.
- **y** (*int*) – The y coordinate of the center of the circle.
- **r** (*int*) – The radius of the circle.
- **fill** (*bool*) – If `True`, the circle will be filled with `color`, otherwise only the circumference will be drawn.
- **color** (*Color*) – The color of the circle.

Image properties

`width`

Gets the width of this image in pixels.

`height`

Gets the height of this image in pixels.

Replacing the entire image

`clear`()

Clears this image. All pixels on this image will be set to `Color.WHITE`.

`load_image`(*source*)

Clears this image, then draws the `source` image centered in this image.

Parameters: **source** (*Image or str*) – The source `Image`. If the argument is a string, then the `source` image is loaded from file.

Saving the image

`save`(*filename*)

Saves this image as a `.png` file.

Parameters: **filename** (*str*) – The path to the file to be saved.

Raises:
- `TypeError` – `filename` is not a string.
- `OSError` – There was a problem saving the file.