# PIL Package (autodoc of remaining modules)

Reference for modules whose documentation has not yet been ported or written can be found here.

## `BdfFontFile` Module

***class* PIL.BdfFontFile.BdfFontFile(*fp*)**

Bases: `PIL.FontFile.FontFile`

**PIL.BdfFontFile.bdf_char(*f*)**

## `ContainerIO` Module

***class* PIL.ContainerIO.ContainerIO(*file*, *offset*, *length*)**

Bases: `object`

**isatty()**

**read(*n=0*)**

Read data.

@def read(bytes=0) :param bytes: Number of bytes to read. If omitted or zero,

read until end of region.

**Returns:** An 8-bit string

### readline()

Read a line of text.

**Returns:** An 8-bit string.

### readlines()

Read multiple lines of text.

**Returns:** A list of 8-bit strings.

### seek(*offset*, *mode=0*)

Move file pointer.

**Parameters:**
- **offset** – Offset in bytes.
- **mode** – Starting position. Use 0 for beginning of region, 1 for current offset, and 2 for end of region. You cannot move the pointer outside the defined region.

### tell()

Get current file pointer.

**Returns:** Offset from start of region, in bytes.

## FontFile Module

### *class* PIL.FontFile.FontFile

Bases: `object`

### bitmap= *None*

### compile()

Create metrics and bitmap

**PIL.FontFile.puti16(*fp*, *values*)**

# GdImageFile Module

***class* PIL.GdImageFile.GdImageFile(*fp=None*, *filename=None*)**

Bases: `PIL.ImageFile.ImageFile`

**format= '*GD*'**

**format_description= '*GD uncompressed images*'**

**PIL.GdImageFile.open(*fp*, *mode='r'*)**

Load texture from a GD image file.

**Parameters:**
- **filename** – GD file name, or an opened file handle.
- **mode** – Optional mode. In this version, if the mode argument is given, it must be “r”.

**Returns:** An image instance.

**Raises:** **IOError** – If the image could not be read.

# GimpGradientFile Module

***class* PIL.GimpGradientFile.GimpGradientFile(*fp*)**

Bases: `PIL.GimpGradientFile.GradientFile`

***class* PIL.GimpGradientFile.GradientFile**

Bases: `object`

**gradient= *None***

---

**PIL.GimpGradientFile.curved(*middle*, *pos*)**

---

**PIL.GimpGradientFile.linear(*middle*, *pos*)**

---

**PIL.GimpGradientFile.sine(*middle*, *pos*)**

---

**PIL.GimpGradientFile.sphere_decreasing(*middle*, *pos*)**

---

**PIL.GimpGradientFile.sphere_increasing(*middle*, *pos*)**

## GimpPaletteFile Module

---

***class* PIL.GimpPaletteFile.GimpPaletteFile(*fp*)**

Bases: `object`

**getpalette()**

**rawmode= '*RGB*'**

## ImageDraw2 Module

---

***class* PIL.ImageDraw2.Brush(*color*, *opacity=255*)**

Bases: `object`

---

***class* PIL.ImageDraw2.Draw(*image*, *size=None*, *color=None*)**

Bases: `object`

**arc(*xy*, *start*, *end*, **options*)**

**chord(*xy*, *start*, *end*, **options*)**

**flush()**

**line(*xy*, **options*)**

**pieslice(*xy*, *start*, *end*, **options*)**

**polygon(*xy*, **options*)**

**rectangle(*xy*, **options*)**

**render(*op*, *xy*, *pen*, *brush=None*)**

**settransform(*offset*)**

**symbol(*xy*, *symbol*, **options*)**

**text(*xy*, *text*, *font*)**

**textsize(*text*, *font*)**

---

***class* PIL.ImageDraw2.Font(*color*, *file*, *size=12*)**

Bases: `object`

---

***class* PIL.ImageDraw2.Pen(*color*, *width=1*, *opacity=255*)**

Bases: `object`

## ImageShow Module

---

***class* PIL.ImageShow.DisplayViewer**

Bases: `PIL.ImageShow.UnixViewer`

**get_command_ex(*file*, ***options*)**

Bases: `PIL.ImageShow.Viewer`

**show_file(*file*, ***options*)**

## *class* PIL.ImageShow.Viewer

Bases: `object`

Base class for viewers.

**format= *None***

**get_command(*file*, ***options*)**

**get_format(*image*)**

Return format name, or None to save as PGM/PPM

**save_image(*image*)**

Save to temporary file, and return filename

**show(*image*, ***options*)**

**show_file(*file*, ***options*)**

Display given file

**show_image(*image*, ***options*)**

Display given image

## *class* PIL.ImageShow.XVViewer

Bases: `PIL.ImageShow.UnixViewer`

**get_command_ex(*file*, *title=None*, ***options*)**

## PIL.ImageShow.register(*viewer*, *order=1*)

**Parameters:**
- **image** – An image object.
- **title** – Optional title. Not all viewers can display the title.
- ****options** – Additional viewer options.

**Returns:** True if a suitable viewer was found, false otherwise.

---

**PIL.ImageShow.which(*executable*)**

# `ImageTransform` Module

---

***class* PIL.ImageTransform.AffineTransform(*data*)**

Bases: `PIL.ImageTransform.Transform`

Define an affine image transform.

This function takes a 6-tuple (a, b, c, d, e, f) which contain the first two rows from an affine transform matrix. For each pixel (x, y) in the output image, the new value is taken from a position (a x + b y + c, d x + e y + f) in the input image, rounded to nearest pixel.

This function can be used to scale, translate, rotate, and shear the original image.

See `transform()`

**Parameters:** **matrix** – A 6-tuple (a, b, c, d, e, f) containing the first two rows from an affine transform matrix.

> **method= *0***

---

***class* PIL.ImageTransform.ExtentTransform(*data*)**

Bases: `PIL.ImageTransform.Transform`

Define a transform to extract a subregion from an image.

Maps a rectangle (defined by two corners) from the image to a rectangle of the given size. The resulting image will contain data sampled from between the corners, such that (x0, y0) in the input image will end up at (0,0) in the

This method can be used to crop, stretch, shrink, or mirror an arbitrary rectangle in the current image. It is slightly slower than crop, but about as fast as a corresponding resize operation.

See `transform()`

**Parameters:** **bbox** – A 4-tuple (x0, y0, x1, y1) which specifies two points in the input image's coordinate system.

**method= *1***

---

## *class* PIL.ImageTransform.MeshTransform(*data*)

Bases: `PIL.ImageTransform.Transform`

Define a mesh image transform. A mesh transform consists of one or more individual quad transforms.

See `transform()`

**Parameters:** **data** – A list of (bbox, quad) tuples.

**method= *4***

---

## *class* PIL.ImageTransform.QuadTransform(*data*)

Bases: `PIL.ImageTransform.Transform`

Define a quad image transform.

Maps a quadrilateral (a region defined by four corners) from the image to a rectangle of the given size.

See `transform()`

**Parameters:** **xy** – An 8-tuple (x0, y0, x1, y1, x2, y2, y3, y3) which contain the upper left, lower left, lower right, and upper right corner of the source quadrilateral.

**method= *3***

Base: **`PIL.ImageTransform.ImageTransformHandler`**

**`getdata()`**

**`transform(`*`size`*`, `*`image`*`, **`*`options`*`)`**

## `JpegPresets` Module

JPEG quality settings equivalent to the Photoshop settings.

More presets can be added to the presets dict if needed.

Can be use when saving JPEG file.

To apply the preset, specify:

```
quality="preset_name"
```

To apply only the quantization table:

```
qtables="preset_name"
```

To apply only the subsampling setting:

```
subsampling="preset_name"
```

Example:

```
im.save("image_name.jpg", quality="web_high")
```

## Subsampling

for chroma information than for luma information (ref.: https://en.wikipedia.org/wiki/Chroma_subsampling)

Possible subsampling values are 0, 1 and 2 that correspond to 4:4:4, 4:2:2 and 4:1:1 (or 4:2:0?).

You can get the subsampling of a JPEG with the *JpegImagePlugin.get_subsampling(im)* function.

## Quantization tables

They are values use by the DCT (Discrete cosine transform) to remove *unnecessary* information from the image (the lossy part of the compression). (ref.: https://en.wikipedia.org/wiki/Quantization_matrix#Quantization_matrices, https://en.wikipedia.org/wiki/JPEG#Quantization)

You can get the quantization tables of a JPEG with:

```
im.quantization
```

This will return a dict with a number of arrays. You can pass this dict directly as the qtables argument when saving a JPEG.

The tables format between im.quantization and quantization in presets differ in 3 ways:

1. The base container of the preset is a list with sublists instead of dict. dict[0] -> list[0], dict[1] -> list[1], ...
2. Each table in a preset is a list instead of an array.
3. The zigzag order is remove in the preset (needed by libjpeg >= 6a).

You can convert the dict format to the preset format with the *JpegImagePlugin.convert_dict_qtables(dict_qtables)* function.

Libjpeg ref.: http://web.archive.org/web/20120328125543/http://www.jpegcameras.com/libjpeg/libjpeg-3.html

### *class* PIL.PaletteFile.PaletteFile(*fp*)

Bases: `object`

**getpalette()**

**rawmode= '*RGB*'**

## PcfFontFile Module

### *class* PIL.PcfFontFile.PcfFontFile(*fp*)

Bases: `PIL.FontFile.FontFile`

**name= '*name*'**

### PIL.PcfFontFile.sz(*s*, *o*)

## PngImagePlugin.iTXt Class

### *class* PIL.PngImagePlugin.iTXt

Bases: `str`

Subclass of string to allow iTXt chunks to look like strings while keeping their extra information

**__new__(*cls*, *text*, *lang*, *tkey*)**

**Parameters:**
- **value** – value for this key
- **lang** – language code
- **tkey** – UTF-8 version of the key name

## PngImagePlugin.PngInfo Class

### *class* PIL.PngImagePlugin.PngInfo

Bases: `object`

**add(*cid*, *data*)**

Appends an arbitrary chunk. Use with caution.

**Parameters:**
- **cid** – a byte string, 4 bytes long.
- **data** – a byte string of the encoded data

**add_itxt(*key*, *value*, *lang*='', *tkey*='', *zip*=*False*)**

Appends an iTXt chunk.

**Parameters:**
- **key** – latin-1 encodable text key name
- **value** – value for this key
- **lang** – language code
- **tkey** – UTF-8 version of the key name
- **zip** – compression flag

**add_text(*key*, *value*, *zip*=*0*)**

Appends a text chunk.

**Parameters:**
- **key** – latin-1 encodable text key name
- **value** – value for this key, text or an `PIL.PngImagePlugin.iTXt` instance
- **zip** – compression flag

# `TarIO` Module

***class* PIL.TarIO.TarIO(*tarfile*, *file*)**

Bases: `PIL.ContainerIO.ContainerIO`

# `WalImageFile` Module

**PIL.WalImageFile.open(*filename*)**

Load texture from a Quake2 WAL texture file.

**Parameters:** **filename** – WAL file name, or an opened file handle.

**Returns:** An image instance.

## `_binary` Module

**PIL._binary.i16be(*c*, *o=0*)**

**PIL._binary.i16le(*c*, *o=0*)**

Converts a 2-bytes (16 bits) string to an unsigned integer.

c: string containing bytes to convert o: offset of bytes to convert in string

**PIL._binary.i32be(*c*, *o=0*)**

**PIL._binary.i32le(*c*, *o=0*)**

Converts a 4-bytes (32 bits) string to an unsigned integer.

c: string containing bytes to convert o: offset of bytes to convert in string

**PIL._binary.i8(*c*)**

**PIL._binary.o16be(*i*)**

**PIL._binary.o16le(*i*)**

**PIL._binary.o32be(*i*)**

**PIL._binary.o32le(*i*)**

**PIL._binary.o8(*i*)**

**PIL._binary.si16le(*c*, *o=0*)**

c: string containing bytes to convert o: offset of bytes to convert in string

---

### PIL._binary.si32le(*c*, *o=0*)

Converts a 4-bytes (32 bits) string to a signed integer.

c: string containing bytes to convert o: offset of bytes to convert in string