Pillow (PIL Fork)

Release 12.3.0

Fredrik Lundh (PIL), Jeffrey 'Alex' Clark (Pillow)

Jul 01, 2026

CONTENTS

1 Overview **3**
- 1.1 Installation . . . 3
- 1.2 Handbook . . . 11
- 1.3 Reference . . . 64
- 1.4 Porting . . . 232
- 1.5 About . . . 232
- 1.6 Release notes . . . 233
- 1.7 Deprecations and removals . . . 317

2 Indices and tables **331**

Python Module Index **333**

Index **335**

Pillow is the friendly PIL fork by Jeffrey 'Alex' Clark and contributors. PIL is the Python Imaging Library by Fredrik Lundh and contributors.

Pillow for enterprise is available via the Tidelift Subscription. Learn more.

CHAPTER

ONE

OVERVIEW

The Python Imaging Library adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

1.1 Installation

1.1.1 Basic installation

Note

The following instructions will install Pillow with support for most common image formats. See *External libraries* for a full list of external libraries supported.

Install Pillow with **pip**:

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
```

Optionally, install defusedxml for Pillow to read XMP data, and olefile for Pillow to read FPX and MIC images:

```
python3 -m pip install --upgrade defusedxml olefile
```

We provide binaries for Linux for each of the supported Python versions in the manylinux wheel format. These include support for all optional libraries except libimagequant. Raqm support requires FriBiDi to be installed separately:

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
```

Most major Linux distributions, including Fedora, Ubuntu and ArchLinux also include Pillow in packages that previously contained PIL e.g. `python-imaging`. Debian splits it into two packages, `python3-pil` and `python3-pil.imagetk`.

We provide binaries for macOS for each of the supported Python versions in the wheel format. These include support for all optional libraries except libimagequant. Raqm support requires FriBiDi to be installed separately:

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
```

While we provide binaries for both x86-64 and arm64, we do not provide universal2 binaries. However, it is simple to combine our current binaries to create one:

```
python3 -m pip download --only-binary=:all: --platform macosx_10_10_x86_64 Pillow
python3 -m pip download --only-binary=:all: --platform macosx_11_0_arm64 Pillow
python3 -m pip install delocate
```

Then, with the names of the downloaded wheels, use Python to combine them:

```
from delocate.fuse import fuse_wheels
fuse_wheels('Pillow-9.4.0-2-cp39-cp39-macosx_10_10_x86_64.whl', 'Pillow-9.4.0-cp39-cp39-
↪macosx_11_0_arm64.whl', 'Pillow-9.4.0-cp39-cp39-macosx_11_0_universal2.whl')
```

We provide Pillow binaries for Windows compiled for the matrix of supported Pythons in the wheel format. These include x86, x86-64 and arm64 versions. These binaries include support for all optional libraries except libimagequant and libxcb. Raqm support requires FriBiDi to be installed separately:

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
```

To install Pillow in MSYS2, see *Building from source*.

Pillow can be installed on FreeBSD via the official Ports or Packages systems:

Ports:

```
cd /usr/ports/graphics/py-pillow && make install clean
```

Packages:

```
pkg install py38-pillow
```

> **Note**
>
> The Pillow FreeBSD port and packages are tested by the ports team with all supported FreeBSD versions.

1.1.2 Python support

Pillow supports these Python versions.

Table 1: Newer versions

Python	3.14	3.13	3.12	3.11	3.10	3.9	3.8	3.7	3.6	3.5
Pillow 12	Yes	Yes	Yes	Yes	Yes					
Pillow 11		Yes	Yes	Yes	Yes	Yes				
Pillow 10.1 - 10.4			Yes	Yes	Yes	Yes	Yes			
Pillow 10.0				Yes	Yes	Yes	Yes			
Pillow 9.3 - 9.5				Yes	Yes	Yes	Yes	Yes		
Pillow 9.0 - 9.2					Yes	Yes	Yes	Yes		
Pillow 8.3.2 - 8.4					Yes	Yes	Yes	Yes	Yes	
Pillow 8.0 - 8.3.1						Yes	Yes	Yes	Yes	
Pillow 7.0 - 7.2							Yes	Yes	Yes	Yes

Table 2: Older versions

Python	3.8	3.7	3.6	3.5	3.4	3.3	3.2	2.7	2.6	2.5	2.4
Pillow 6.2.1 - 6.2.2	Yes	Yes	Yes	Yes				Yes			
Pillow 6.0 - 6.2.0		Yes	Yes	Yes				Yes			
Pillow 5.2 - 5.4		Yes	Yes	Yes	Yes			Yes			
Pillow 5.0 - 5.1			Yes	Yes	Yes			Yes			
Pillow 4			Yes	Yes	Yes	Yes		Yes			
Pillow 2 - 3				Yes	Yes	Yes	Yes	Yes	Yes		
Pillow < 2								Yes	Yes	Yes	Yes

1.1.3 Platform support

Current platform support for Pillow. Binary distributions are contributed for each release on a volunteer basis, but the source should compile and run everywhere platform support is listed. In general, we aim to support all current versions of Linux, macOS, and Windows.

Continuous integration targets

These platforms are built and tested for every change.

Operating system	Tested Python versions	Tested architecture
Alpine	3.12	x86-64
Amazon Linux 2023	3.11	x86-64
Arch	3.14	x86-64
CentOS Stream 9	3.10	x86-64
CentOS Stream 10	3.12	x86-64
Debian 13 Trixie	3.13	x86, x86-64
Fedora 43	3.14	x86-64
Fedora 44	3.14	x86-64
Gentoo	3.13	x86-64
macOS 15 Sequoia	3.11, 3.12, 3.13, 3.14, 3.15, PyPy3	arm64
macOS 26 Tahoe	3.10	x86-64
Ubuntu Linux 22.04 LTS (Jammy)	3.10	x86-64
Ubuntu Linux 24.04 LTS (Noble)	3.10, 3.11, 3.12, 3.13, 3.14, 3.15, PyPy3	x86-64
Ubuntu Linux 26.04 LTS (Resolute)	3.14	x86-64, arm64v8, ppc64le, s390x
Windows Server 2022	3.10	x86
Windows Server 2025	3.11, 3.12, 3.13, 3.14, 3.15, PyPy3	x86-64
	3.14 (MinGW)	x86-64

Other platforms

These platforms have been reported to work at the versions mentioned.

Note

Contributors please test Pillow on your platform then update this document and send a pull request.

Operating system	Tested Python versions	Latest tested Pillow version	Tested processors
macOS 26 Tahoe	3.10, 3.11, 3.12, 3.13, 3.14	12.2.0	arm
	3.9	11.3.0	
macOS 15 Sequoia	3.9, 3.10, 3.11, 3.12, 3.13	11.3.0	arm
	3.8	10.4.0	
macOS 14 Sonoma	3.8, 3.9, 3.10, 3.11, 3.12	10.4.0	arm
macOS 13 Ventura	3.8, 3.9, 3.10, 3.11	10.0.1	arm
	3.7	9.5.0	
macOS 12 Monterey	3.7, 3.8, 3.9, 3.10, 3.11	9.3.0	arm
macOS 11 Big Sur	3.7, 3.8, 3.9, 3.10	8.4.0	arm
	3.7, 3.8, 3.9, 3.10, 3.11	9.4.0	x86-64
	3.6	8.4.0	
macOS 10.15 Catalina	3.6, 3.7, 3.8, 3.9	8.3.2	x86-64
	3.5	7.2.0	
macOS 10.14 Mojave	3.5, 3.6, 3.7, 3.8	7.2.0	x86-64
	2.7	6.0.0	
	3.4	5.4.1	
macOS 10.13 High Sierra	2.7, 3.4, 3.5, 3.6	4.2.1	x86-64
macOS 10.12 Sierra	2.7, 3.4, 3.5, 3.6	4.1.1	x86-64
Mac OS X 10.11 El Capitan	2.7, 3.4, 3.5, 3.6, 3.7	5.4.1	x86-64
	3.3	4.1.0	
Mac OS X 10.9 Mavericks	2.7, 3.2, 3.3, 3.4	3.0.0	x86-64
Mac OS X 10.8 Mountain Lion	2.6, 2.7, 3.2, 3.3		x86-64
Redhat Linux 6	2.6		x86
CentOS 6.3	2.7, 3.3		x86
CentOS 8	3.9	9.0.0	x86-64
Fedora 23	2.7, 3.4	3.1.0	x86-64
Ubuntu Linux 12.04 LTS (Precise)	2.6, 3.2, 3.3, 3.4, 3.5 PyPy5.3.1, PyPy3 v2.4.0	3.4.1	x86,x86-64
	2.7	4.3.0	x86-64
	2.7, 3.2	3.4.1	ppc
Ubuntu Linux 10.04 LTS (Lucid)	2.6	2.3.0	x86,x86-64
Debian 8.2 Jessie	2.7, 3.4	3.1.0	x86-64
Raspbian Jessie	2.7, 3.4	3.1.0	arm
Raspbian Stretch	2.7, 3.5	4.0.0	arm
Raspberry Pi OS	3.6, 3.7, 3.8, 3.9	8.2.0	arm
	2.7	6.2.2	
Gentoo Linux	2.7, 3.2	2.1.0	x86-64
FreeBSD 11.1	2.7, 3.4, 3.5, 3.6	4.3.0	x86-64
FreeBSD 10.3	2.7, 3.4, 3.5	4.2.0	x86-64
FreeBSD 10.2	2.7, 3.4	3.1.0	x86-64
Windows 11 23H2	3.9, 3.10, 3.11, 3.12, 3.13	11.0.0	arm64
Windows 11 Pro	3.11, 3.12	10.2.0	x86-64
Windows 10	3.7	7.1.0	x86-64
Windows 10/Cygwin 3.3	3.6, 3.7, 3.8, 3.9	8.4.0	x86-64
Windows 8.1 Pro	2.6, 2.7, 3.2, 3.3, 3.4	2.4.0	x86,x86-64
Windows 8 Pro	2.6, 2.7, 3.2, 3.3, 3.4a3	2.2.0	x86,x86-64

continues on next page

Table 3 – continued from previous page

Operating system	Tested Python versions	Latest tested Pillow version	Tested processors
Windows 7 Professional	3.7	7.0.0	x86,x86-64
Windows Server 2008 R2 Enterprise	3.3		x86-64

1.1.4 Building from source

External libraries

> **Note**
>
> You **do not need to install all supported external libraries** to use Pillow's basic features. **Zlib** and **libjpeg** are required by default.

> **Note**
>
> There are Dockerfiles in our Docker images repo to install the dependencies for some operating systems.

Many of Pillow's features require external libraries:

- **libjpeg** provides JPEG functionality.
  - Pillow has been tested with libjpeg versions **6b**, **8**, **9-9d** and libjpeg-turbo version **8**.
  - Starting with Pillow 3.0.0, libjpeg is required by default. It can be disabled with the `-C jpeg=disable` flag.
- **zlib** provides access to compressed PNGs
  - Starting with Pillow 3.0.0, zlib is required by default. It can be disabled with the `-C zlib=disable` flag.
- **libtiff** provides compressed TIFF functionality
  - Pillow has been tested with libtiff versions **4.0-4.7.1**
- **libfreetype** provides type related services
- **littlecms** provides color management
  - Pillow version 2.2.1 and below uses liblcms1, Pillow 2.3.0 and above uses liblcms2. Tested with **1.19** and **2.7-2.19.1**.
- **libwebp** provides the WebP format.
- **openjpeg** provides JPEG 2000 functionality.
  - Pillow has been tested with openjpeg **2.0.0**, **2.1.0**, **2.3.1**, **2.4.0**, **2.5.0**, **2.5.2**, **2.5.3** and **2.5.4**.
  - Pillow does **not** support the earlier **1.5** series which ships with Debian Jessie.
- **libimagequant** provides improved color quantization
  - Pillow has been tested with libimagequant **2.6-4.4.1**
  - Libimagequant is licensed GPLv3, which is more restrictive than the Pillow license, therefore we will not be distributing binaries with libimagequant support enabled.

- **libraqm** provides complex text layout support.
  - libraqm provides bidirectional text support (using FriBiDi), shaping (using HarfBuzz), and proper script itemization. As a result, Raqm can support most writing systems covered by Unicode.
  - libraqm depends on the following libraries: FreeType, HarfBuzz, FriBiDi, make sure that you install them before installing libraqm if not available as package in your system.
  - Setting text direction or font features is not supported without libraqm.
  - Pillow wheels since version 8.2.0 include a modified version of libraqm that loads libfribidi at runtime if it is installed. On Windows this requires compiling FriBiDi and installing `fribidi.dll` into a directory listed in the Dynamic-link library search order (Microsoft Learn) (`fribidi-0.dll` or `libfribidi-0.dll` are also detected). See *Build Options* to see how to build this version.
  - Previous versions of Pillow (5.0.0 to 8.1.2) linked libraqm dynamically at runtime.
- **libxcb** provides X11 screengrab support.
- **libavif** provides support for the AVIF format.
  - Pillow requires libavif version **1.0.0** or greater.
  - libavif is merely an API that wraps AVIF codecs. If you are compiling libavif from source, you will also need to install both an AVIF encoder and decoder, such as rav1e and dav1d, or libaom, which both encodes and decodes AVIF images.

If you didn't build Python from source, make sure you have Python's development libraries installed.

In Debian or Ubuntu:

```
sudo apt-get install python3-dev python3-setuptools
```

In Fedora, the command is:

```
sudo dnf install python3-devel redhat-rpm-config
```

In Alpine, the command is:

```
sudo apk add python3-dev py3-setuptools
```

Note

`redhat-rpm-config` is required on Fedora 23, but not earlier versions.

Prerequisites for **Ubuntu 16.04 LTS - 26.04 LTS** are installed with:

```
sudo apt-get install libtiff5-dev libjpeg8-dev libopenjp2-7-dev zlib1g-dev \
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.6-dev tk8.6-dev python3-tk \
    libharfbuzz-dev libfribidi-dev libxcb1-dev
```

To install libraqm, `sudo apt-get install meson` and then see `depends/install_raqm.sh`.

Build prerequisites for libavif on Ubuntu are installed with:

```
sudo apt-get install cmake ninja-build nasm
```

Then see `depends/install_libavif.sh` to build and install libavif.

Prerequisites are installed on recent **Red Hat**, **CentOS** or **Fedora** with:

```
sudo dnf install libtiff-devel libjpeg-devel openjpeg2-devel zlib-devel \
    freetype-devel lcms2-devel libwebp-devel tcl-devel tk-devel \
    harfbuzz-devel fribidi-devel libraqm-devel libimagequant-devel libxcb-devel
```

Note that the package manager may be yum or DNF, depending on the exact distribution.

Prerequisites are installed for **Alpine** with:

```
sudo apk add tiff-dev jpeg-dev openjpeg-dev zlib-dev freetype-dev lcms2-dev \
    libwebp-dev tcl-dev tk-dev harfbuzz-dev fribidi-dev libimagequant-dev \
    libxcb-dev libpng-dev
```

See also the `Dockerfiles` in the Test Infrastructure repo (https://github.com/python-pillow/docker-images) for a known working install process for other tested distros.

The Xcode command line tools are required to compile portions of Pillow. The tools are installed by running `xcode-select --install` from the command line. The command line tools are required even if you have the full Xcode package installed. It may be necessary to run `sudo xcodebuild -license` to accept the license prior to using the tools.

The easiest way to install external libraries is via Homebrew. After you install Homebrew, run:

```
brew install libavif libjpeg libraqm libtiff little-cms2 openjpeg webp
```

If you would like to use libavif with more codecs than just aom, then instead of installing libavif through Homebrew directly, you can use Homebrew to install libavif's build dependencies:

```
brew install aom dav1d rav1e svt-av1
```

Then see `depends/install_libavif.sh` to install libavif.

We recommend you use prebuilt wheels from PyPI. If you wish to compile Pillow manually, you can use the build scripts in the `winbuild` directory used for CI testing and development. These scripts require Visual Studio 2017 or newer and NASM.

The scripts also install Pillow from the local copy of the source code, so the *Installing* instructions will not be necessary afterwards.

To build Pillow using MSYS2, make sure you run the **MSYS2 MinGW 32-bit** or **MSYS2 MinGW 64-bit** console, *not* **MSYS2** directly.

The following instructions target the 64-bit build, for 32-bit replace all occurrences of `mingw-w64-x86_64-` with `mingw-w64-i686-`.

Make sure you have Python and GCC installed:

```
pacman -S \
    mingw-w64-x86_64-gcc \
    mingw-w64-x86_64-python \
    mingw-w64-x86_64-python-pip \
    mingw-w64-x86_64-python-setuptools
```

Prerequisites are installed on **MSYS2 MinGW 64-bit** with:

```
pacman -S \
    mingw-w64-x86_64-libjpeg-turbo \
    mingw-w64-x86_64-zlib \
    mingw-w64-x86_64-libtiff \
```

(continues on next page)

(continued from previous page)

```
    mingw-w64-x86_64-freetype \
    mingw-w64-x86_64-lcms2 \
    mingw-w64-x86_64-libwebp \
    mingw-w64-x86_64-openjpeg2 \
    mingw-w64-x86_64-libimagequant \
    mingw-w64-x86_64-libraqm \
    mingw-w64-x86_64-libavif
```

Note

Only FreeBSD 10 and 11 tested

Make sure you have Python's development libraries installed:

```
sudo pkg install python3
```

Prerequisites are installed on **FreeBSD 10 or 11** with:

```
sudo pkg install jpeg-turbo tiff webp lcms2 freetype2 openjpeg harfbuzz fribidi libxcb␣
↪libavif
```

Then see `depends/install_raqm_cmake.sh` to install libraqm.

Basic Android support has been added for compilation within the Termux environment. The dependencies can be installed by:

```
pkg install -y python ndk-sysroot clang make \
    libjpeg-turbo
```

This has been tested within the Termux app on ChromeOS, on x86.

Installing

Once you have installed the prerequisites, to install Pillow from the source code on PyPI, run:

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow --no-binary :all:
```

If the prerequisites are installed in the standard library locations for your machine (e.g. `/usr` or `/usr/local`), no additional configuration should be required. If they are installed in a non-standard location, you may need to configure setuptools to use those locations by editing `setup.py` or `pyproject.toml`, or by adding environment variables on the command line:

```
CFLAGS="-I/usr/pkg/include" python3 -m pip install --upgrade Pillow --no-binary :all:
```

If Pillow has been previously built without the required prerequisites, it may be necessary to manually clear the pip cache or build without cache using the `--no-cache-dir` option to force a build with newly installed external libraries.

If you would like to install from a local copy of the source code instead, you can clone from GitHub with `git clone https://github.com/python-pillow/Pillow` or download and extract the compressed archive from PyPI.

After navigating to the Pillow directory, run:

```
python3 -m pip install --upgrade pip
python3 -m pip install .
```

Build options

- Config setting: `-C parallel=n`. Can also be given with environment variable: `MAX_CONCURRENCY=n`. Pillow can use multiprocessing to build the extensions. Setting `-C parallel=n` sets the number of CPUs to use to `n`, or can disable parallel building by using a setting of 1. By default, it uses as many CPUs as are present.
- Config settings: `-C zlib=disable`, `-C jpeg=disable`, `-C tiff=disable`, `-C freetype=disable`, `-C raqm=disable`, `-C lcms=disable`, `-C webp=disable`, `-C jpeg2000=disable`, `-C imagequant=disable`, `-C xcb=disable`, `-C avif=disable`. Disable building the corresponding feature even if the development libraries are present on the building machine.
- Config settings: `-C zlib=enable`, `-C jpeg=enable`, `-C tiff=enable`, `-C freetype=enable`, `-C raqm=enable`, `-C lcms=enable`, `-C webp=enable`, `-C jpeg2000=enable`, `-C imagequant=enable`, `-C xcb=enable`, `-C avif=enable`. Require that the corresponding feature is built. The build will raise an exception if the libraries are not found. Tcl and Tk must be used together.
- Config settings: `-C raqm=vendor`, `-C fribidi=vendor`. These flags are used to compile a modified version of libraqm and a shim that dynamically loads libfribidi at runtime. These are used to compile the standard Pillow wheels. Compiling libraqm requires a C99-compliant compiler.
- Config setting: `-C platform-guessing=disable`. Skips all of the platform dependent guessing of include and library directories for automated build systems that configure the proper paths in the environment variables (e.g. Buildroot).
- Config setting: `-C debug=true`. Adds a debugging flag to the include and library search process to dump all paths searched for and found to stdout.

Sample usage:

```
python3 -m pip install --upgrade Pillow -C [feature]=enable
```

1.1.5 Old versions

You can download old distributions from the release history at PyPI and by direct URL access eg. https://pypi.org/project/pillow/1.0/.

1.2 Handbook

1.2.1 Overview

The **Python Imaging Library** adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

Let's look at a few possible uses of this library.

Image archives

The Python Imaging Library is ideal for image archival and batch processing applications. You can use the library to create thumbnails, convert between file formats, print images, etc.

The current version identifies and reads a large number of formats. Write support is intentionally restricted to the most commonly used interchange and presentation formats.

Image display

The current release includes Tk *PhotoImage* and *BitmapImage* interfaces, as well as a *Windows DIB interface* that can be used with PythonWin and other Windows-based toolkits. Many other GUI toolkits come with some kind of PIL support.

For debugging, there's also a *show()* method which saves an image to disk, and calls an external display utility.

Image processing

The library contains basic image processing functionality, including point operations, filtering with a set of built-in convolution kernels, and colour space conversions.

The library also supports image resizing, rotation and arbitrary affine transforms.

There's a histogram method allowing you to pull some statistics out of an image. This can be used for automatic contrast enhancement, and for global statistical analysis.

1.2.2 Tutorial

Using the Image class

The most important class in the Python Imaging Library is the *Image* class, defined in the module with the same name. You can create instances of this class in several ways; either by loading images from files, processing other images, or creating images from scratch.

To load an image from a file, use the *open()* function in the *Image* module:

```
>>> from PIL import Image
>>> im = Image.open("hopper.ppm")
```

If successful, this function returns an *Image* object. You can now use instance attributes to examine the file contents:

```
>>> print(im.format, im.size, im.mode)
PPM (512, 512) RGB
```

The *format* attribute identifies the source of an image. If the image was not read from a file, it is set to None. The size attribute is a 2-tuple containing width and height (in pixels). The *mode* attribute defines the number and names of the bands in the image, and also the pixel type and depth. Common modes are "L" (luminance) for grayscale images, "RGB" for true color images, and "CMYK" for pre-press images.

If the file cannot be opened, an `OSError` exception is raised.

Once you have an instance of the *Image* class, you can use the methods defined by this class to process and manipulate the image. For example, let's display the image we just loaded:

```
>>> im.show()
```

> **Note**
>
> The standard version of *show()* is not very efficient, since it saves the image to a temporary file and calls a utility to display the image. If you don't have an appropriate utility installed, it won't even work. When it does work though, it is very handy for debugging and tests.

The following sections provide an overview of the different functions provided in this library.

Reading and writing images

The Python Imaging Library supports a wide variety of image file formats. To read files from disk, use the *open()* function in the *Image* module. You don't have to know the file format to open a file. The library automatically determines the format based on the contents of the file.

To save a file, use the *save()* method of the *Image* class. When saving files, the name becomes important. Unless you specify the format, the library uses the filename extension to discover which file storage format to use.

Convert files to JPEG

```
import os, sys
from PIL import Image

for infile in sys.argv[1:]:
    f, e = os.path.splitext(infile)
    outfile = f + ".jpg"
    if infile != outfile:
        try:
            with Image.open(infile) as im:
                im.save(outfile)
        except OSError:
            print("cannot convert", infile)
```



A second argument can be supplied to the *save()* method which explicitly specifies a file format. If you use a non-standard extension, you must always specify the format this way:

Create JPEG thumbnails

```
import os, sys
from PIL import Image

size = (128, 128)

for infile in sys.argv[1:]:
    outfile = os.path.splitext(infile)[0] + ".thumbnail"
```

(continues on next page)

(continued from previous page)

```
    if infile != outfile:
        try:
            with Image.open(infile) as im:
                im.thumbnail(size)
                im.save(outfile, "JPEG")
        except OSError:
            print("cannot create thumbnail for", infile)
```



It is important to note that the library doesn't decode or load the raster data unless it really has to. When you open a file, the file header is read to determine the file format and extract things like mode, size, and other properties required to decode the file, but the rest of the file is not processed until later.

This means that opening an image file is a fast operation, which is independent of the file size and compression type. Here's a simple script to quickly identify a set of image files:

Identify image files

```
import sys
from PIL import Image

for infile in sys.argv[1:]:
    try:
        with Image.open(infile) as im:
            print(infile, im.format, f"{im.size}x{im.mode}")
    except OSError:
        pass
```

Cutting, pasting, and merging images

The *Image* class contains methods allowing you to manipulate regions within an image. To extract a sub-rectangle from an image, use the *crop()* method.

Copying a subrectangle from an image

```
box = (0, 0, 64, 64)
region = im.crop(box)
```

The region is defined by a 4-tuple, where coordinates are (left, upper, right, lower). The Python Imaging Library uses a coordinate system with (0, 0) in the upper left corner. Also note that coordinates refer to positions between the pixels, so the region in the above example is exactly 64x64 pixels.

The region could now be processed in a certain manner and pasted back.

Processing a subrectangle, and pasting it back

```
region = region.transpose(Image.Transpose.ROTATE_180)
im.paste(region, box)
```

When pasting regions back, the size of the region must match the given region exactly. In addition, the region cannot extend outside the image. However, the modes of the original image and the region do not need to match. If they don't, the region is automatically converted before being pasted (see the section on *Color transforms* below for details).

Here's an additional example:

Rolling an image

```
def roll(im: Image.Image, delta: int) -> Image.Image:
    """Roll an image sideways."""
    xsize, ysize = im.size

    delta = delta % xsize
    if delta == 0:
        return im

    part1 = im.crop((0, 0, delta, ysize))
    part2 = im.crop((delta, 0, xsize, ysize))
    im.paste(part1, (xsize - delta, 0, xsize, ysize))
    im.paste(part2, (0, 0, xsize - delta, ysize))

    return im
```

Or if you would like to merge two images into a wider image:

Merging images

```
def merge(im1: Image.Image, im2: Image.Image) -> Image.Image:
    w = im1.size[0] + im2.size[0]
    h = max(im1.size[1], im2.size[1])
    im = Image.new("RGBA", (w, h))

    im.paste(im1)
    im.paste(im2, (im1.size[0], 0))

    return im
```

For more advanced tricks, the paste method can also take a transparency mask as an optional argument. In this mask, the value 255 indicates that the pasted image is opaque in that position (that is, the pasted image should be used as is). The value 0 means that the pasted image is completely transparent. Values in-between indicate different levels of transparency. For example, pasting an RGBA image and also using it as the mask would paste the opaque portion of the image but not its transparent background.

The Python Imaging Library also allows you to work with the individual bands of an multi-band image, such as an RGB image. The split method creates a set of new images, each containing one band from the original multi-band image.

The merge function takes a mode and a tuple of images, and combines them into a new image. The following sample swaps the three bands of an RGB image:

Splitting and merging bands

```
r, g, b = im.split()
im = Image.merge("RGB", (b, g, r))
```

Note that for a single-band image, *split()* returns the image itself. To work with individual color bands, you may want to convert the image to "RGB" first.

Geometrical transforms

The *PIL.Image.Image* class contains methods to *resize()* and *rotate()* an image. The former takes a tuple giving the new size, the latter the angle in degrees counter-clockwise.

Simple geometry transforms

```
out = im.resize((128, 128))
out = im.rotate(45) # degrees counter-clockwise
```

To rotate the image in 90 degree steps, you can either use the *rotate()* method or the *transpose()* method. The latter can also be used to flip an image around its horizontal or vertical axis.

Transposing an image

```
out = im.transpose(Image.Transpose.FLIP_LEFT_RIGHT)
```

```
out = im.transpose(Image.Transpose.FLIP_TOP_BOTTOM)
```

```
out = im.transpose(Image.Transpose.ROTATE_90)
```

```
out = im.transpose(Image.Transpose.ROTATE_180)
```

```
out = im.transpose(Image.Transpose.ROTATE_270)
```

`transpose(ROTATE)` operations can also be performed identically with *rotate()* operations, provided the `expand` flag is true, to provide for the same changes to the image's size.

A more general form of image transformations can be carried out via the *transform()* method.

Relative resizing

Instead of calculating the size of the new image when resizing, you can also choose to resize relative to a given size.

```
from PIL import Image, ImageOps
size = (100, 150)
with Image.open("hopper.webp") as im:
    ImageOps.contain(im, size).save("imageops_contain.webp")
    ImageOps.cover(im, size).save("imageops_cover.webp")
    ImageOps.fit(im, size).save("imageops_fit.webp")
    ImageOps.pad(im, size, color="#f00").save("imageops_pad.webp")

    # thumbnail() can also be used,
    # but will modify the image object in place
    im.thumbnail(size)
    im.save("image_thumbnail.webp")
```

	thumbnail()	*contain()*	*cover()*	*fit()*	*pad()*
Given size	(100, 150)	(100, 150)	(100, 150)	(100, 150)	(100, 150)
Resulting image					
Resulting size	100×100	100×100	150×150	100×150	100×150

Color transforms

The Python Imaging Library allows you to convert images between different pixel representations using the *convert()* method.

Converting between modes

```
from PIL import Image

with Image.open("hopper.ppm") as im:
    im = im.convert("L")
```

The library supports transformations between each supported mode and the “L” and “RGB” modes. To convert between other modes, you may have to use an intermediate image (typically an “RGB” image).

Image enhancement

The Python Imaging Library provides a number of methods and modules that can be used to enhance images.

Filters

The *ImageFilter* module contains a number of pre-defined enhancement filters that can be used with the *filter()* method.

Applying filters

```
from PIL import ImageFilter
out = im.filter(ImageFilter.DETAIL)
```

Point operations

The *point()* method can be used to translate the pixel values of an image (e.g. image contrast manipulation). In most cases, a function object expecting one argument can be passed to this method. Each pixel is processed according to that function:

Applying point transforms

```
# multiply each pixel by 20
out = im.point(lambda i: i * 20)
```

Using the above technique, you can quickly apply any simple expression to an image. You can also combine the *point()* and *paste()* methods to selectively modify an image:

Processing individual bands

```
# split the image into individual bands
source = im.split()

R, G, B = 0, 1, 2

# select regions where red is less than 100
mask = source[R].point(lambda i: i < 100 and 255)

# process the green band
out = source[G].point(lambda i: i * 0.7)

# paste the processed band back, but only where red was < 100
source[G].paste(out, None, mask)

# build a new multiband image
im = Image.merge(im.mode, source)
```

Note the syntax used to create the mask:

```
imout = im.point(lambda i: expression and 255)
```

Python only evaluates the portion of a logical expression as is necessary to determine the outcome, and returns the last value examined as the result of the expression. So if the expression above is false (0), Python does not look at the second operand, and thus returns 0. Otherwise, it returns 255.

Enhancement

For more advanced image enhancement, you can use the classes in the *ImageEnhance* module. Once created from an image, an enhancement object can be used to quickly try out different settings.

You can adjust contrast, brightness, color balance and sharpness in this way.

Enhancing images

```
from PIL import ImageEnhance

enh = ImageEnhance.Contrast(im)
enh.enhance(1.3).show("30% more contrast")
```



Image sequences

The Python Imaging Library contains some basic support for image sequences (also called animation formats). Supported sequence formats include FLI/FLC, GIF, and a few experimental formats. TIFF files can also contain more than one frame.

When you open a sequence file, PIL automatically loads the first frame in the sequence. You can use the seek and tell methods to move between different frames:

Reading sequences

```
from PIL import Image

with Image.open("animation.gif") as im:
    im.seek(1)  # skip to the second frame

    try:
        while 1:
            im.seek(im.tell() + 1)
            # do something to im
    except EOFError:
        pass  # end of sequence
```

As seen in this example, you'll get an `EOFError` exception when the sequence ends.

Writing sequences

You can create animated GIFs with Pillow, e.g.

```
from PIL import Image

# List of image filenames
image_filenames = [
    "hopper.jpg",
    "rotated_hopper_270.jpg",
    "rotated_hopper_180.jpg",
    "rotated_hopper_90.jpg",
]
```

(continues on next page)

(continued from previous page)

```
# Open images and create a list
images = [Image.open(filename) for filename in image_filenames]

# Save the images as an animated GIF
images[0].save(
    "animated_hopper.gif",
    append_images=images[1:],
    duration=500,  # duration of each frame in milliseconds
    loop=0,  # loop forever
)
```

The following class lets you use the for-statement to loop over the sequence:

Using the *Iterator* class

```
from PIL import ImageSequence
for frame in ImageSequence.Iterator(im):
    # ...do something to frame...
```

PostScript printing

The Python Imaging Library includes functions to print images, text and graphics on PostScript printers. Here's a simple example:

Drawing PostScript

```
from PIL import Image, PSDraw
import os

# Define the PostScript file
ps_file = open("hopper.ps", "wb")

# Create a PSDraw object
ps = PSDraw.PSDraw(ps_file)

# Start the document
ps.begin_document()

# Set the text to be drawn
text = "Hopper"

# Define the PostScript font
font_name = "Helvetica-Narrow-Bold"
font_size = 36

# Calculate text size (approximation as PSDraw doesn't provide direct method)
# Assuming average character width as 0.6 of the font size
text_width = len(text) * font_size * 0.6
text_height = font_size
```

(continues on next page)

(continued from previous page)

```
# Set the position (top-center)
page_width, page_height = 595, 842  # A4 size in points
text_x = (page_width - text_width) // 2
text_y = page_height - text_height - 50  # Distance from the top of the page

# Load the image
image_path = "hopper.ppm"  # Update this with your image path
with Image.open(image_path) as im:
    # Resize the image if it's too large
    im.thumbnail((page_width - 100, page_height // 2))

    # Define the box where the image will be placed
    img_x = (page_width - im.width) // 2
    img_y = text_y + text_height - 200  # 200 points below the text

    # Draw the image (75 dpi)
    ps.image((img_x, img_y, img_x + im.width, img_y + im.height), im, 75)

# Draw the text
ps.setfont(font_name, font_size)
ps.text((text_x, text_y), text)

# End the document
ps.end_document()
ps_file.close()
```

Note

PostScript converted to PDF for display purposes

More on reading images

As described earlier, the *open()* function of the *Image* module is used to open an image file. In most cases, you simply pass it the filename as an argument. `Image.open()` can be used as a context manager:

```
from PIL import Image
with Image.open("hopper.ppm") as im:
    ...
```

If everything goes well, the result is an *PIL.Image.Image* object. Otherwise, an `OSError` exception is raised.

You can use a file-like object instead of the filename. The object must implement `file.read`, `file.seek` and `file.tell` methods, and be opened in binary mode.

Reading from an open file

```
from PIL import Image

with open("hopper.ppm", "rb") as fp:
    im = Image.open(fp)
```

To read an image from binary data, use the `BytesIO` class:

Reading from binary data

```
from PIL import Image
import io

im = Image.open(io.BytesIO(buffer))
```

Note that the library rewinds the file (using `seek(0)`) before reading the image header. In addition, seek will also be used when the image data is read (by the load method). If the image file is embedded in a larger file, such as a tar file, you can use the *ContainerIO* or *TarIO* modules to access it.

Reading from URL

```
from PIL import Image
from urllib.request import urlopen
url = "https://python-pillow.github.io/assets/images/pillow-logo.png"
img = Image.open(urlopen(url))
```

Reading from a tar archive

```
from PIL import Image, TarIO

fp = TarIO.TarIO("hopper.tar", "hopper.jpg")
im = Image.open(fp)
```

Batch processing

Operations can be applied to multiple image files. For example, all PNG images in the current directory can be saved as JPEGs at reduced quality.

```
import glob
from PIL import Image

def compress_image(source_path: str, dest_path: str) -> None:
    with Image.open(source_path) as img:
        if img.mode != "RGB":
            img = img.convert("RGB")
        img.save(dest_path, "JPEG", optimize=True, quality=80)


paths = glob.glob("*.png")
for path in paths:
    compress_image(path, path[:-4] + ".jpg")
```

Since images can also be opened from a `Path` from the `pathlib` module, the example could be modified to use `pathlib` instead of the `glob` module.

```
from pathlib import Path

paths = Path(".").glob("*.png")
```

(continues on next page)

(continued from previous page)

```
for path in paths:
    compress_image(path, path.stem + ".jpg")
```

Controlling the decoder

Some decoders allow you to manipulate the image while reading it from a file. This can often be used to speed up decoding when creating thumbnails (when speed is usually more important than quality) and printing to a monochrome laser printer (when only a grayscale version of the image is needed).

The *draft()* method manipulates an opened but not yet loaded image so it as closely as possible matches the given mode and size. This is done by reconfiguring the image decoder.

Reading in draft mode

This is only available for JPEG and MPO files.

```
from PIL import Image

with Image.open(file) as im:
    print("original =", im.mode, im.size)

    im.draft("L", (100, 100))
    print("draft =", im.mode, im.size)
```

This prints something like:

```
original = RGB (512, 512)
draft = L (128, 128)
```

Note that the resulting image may not exactly match the requested mode and size. To make sure that the image is not larger than the given size, use the thumbnail method instead.

1.2.3 Concepts

The Python Imaging Library handles *raster images*; that is, rectangles of pixel data.

Bands

An image can consist of one or more bands of data. The Python Imaging Library allows you to store several bands in a single image, provided they all have the same dimensions and depth. For example, a PNG image might have 'R', 'G', 'B', and 'A' bands for the red, green, blue, and alpha transparency values. Many operations act on each band separately, e.g., histograms. It is often useful to think of each pixel as having one value per band.

To get the number and names of bands in an image, use the *getbands()* method.

Modes

The `mode` of an image is a string which defines the type and depth of a pixel in the image. Each pixel uses the full range of the bit depth. So a 1-bit pixel has a range of 0-1, an 8-bit pixel has a range of 0-255, a 32-signed integer pixel has the range of INT32 and a 32-bit floating point pixel has the range of FLOAT32. The current release supports the following standard modes:

- 1 (1-bit pixels, black and white, stored with one pixel per byte)
- L (8-bit pixels, grayscale)

- `P` (8-bit pixels, mapped to any other mode using a color palette)
- `RGB` (3x8-bit pixels, true color)
- `RGBA` (4x8-bit pixels, true color with transparency mask)
- `CMYK` (4x8-bit pixels, color separation)
- `YCbCr` (3x8-bit pixels, color video format)
  - Note that this refers to the JPEG, and not the ITU-R BT.2020, standard
- `LAB` (3x8-bit pixels, the L*a*b color space)
- `HSV` (3x8-bit pixels, Hue, Saturation, Value color space)
  - Hue's range of 0-255 is a scaled version of 0 degrees <= Hue < 360 degrees
- `I` (32-bit signed integer pixels)
- `F` (32-bit floating point pixels)

Pillow also provides limited support for a few additional modes, including:

- `LA` (L with alpha)
- `PA` (P with alpha)
- `RGBX` (true color with padding)
- `RGBa` (true color with premultiplied alpha)
- `La` (L with premultiplied alpha)
- `I;16` (16-bit unsigned integer pixels)
- `I;16L` (16-bit little endian unsigned integer pixels)
- `I;16B` (16-bit big endian unsigned integer pixels)
- `I;16N` (16-bit native endian unsigned integer pixels)

Premultiplied alpha is where the values for each other channel have been multiplied by the alpha. For example, an RGBA pixel of `(10, 20, 30, 127)` would convert to an RGBa pixel of `(5, 10, 15, 127)`. The values of the R, G and B channels are halved as a result of the half transparency in the alpha channel.

Apart from these additional modes, Pillow doesn't yet support multichannel images with a depth of more than 8 bits per channel.

Pillow also doesn't support user-defined modes; if you need to handle band combinations that are not listed above, use a sequence of Image objects.

You can read the mode of an image through the *mode* attribute. This is a string containing one of the above values.

Size

You can read the image size through the *size* attribute. This is a 2-tuple, containing the horizontal and vertical size in pixels.

Coordinate system

The Python Imaging Library uses a Cartesian pixel coordinate system, with (0,0) in the upper left corner. Note that the coordinates refer to the implied pixel corners; the centre of a pixel addressed as (0, 0) actually lies at (0.5, 0.5).

Coordinates are usually passed to the library as 2-tuples (x, y). Rectangles are represented as 4-tuples, (x1, y1, x2, y2), with the upper left corner given first.

Palette

The palette mode (P) uses a color palette to define the actual color for each pixel.

Colors

To specify colors, you can use tuples with a value for each channel in the image, e.g. `Image.new("RGB", (1, 1), (255, 0, 0))`.

If an image has a single channel, you can use a single number instead, e.g. `Image.new("L", (1, 1), 255)`. For "F" mode images, floating point values are also accepted. In the case of "P" mode images, these will be indexes for the color palette.

If a single value is used for an image with more than one channel, it will still be parsed:

```
>>> from PIL import Image
>>> im = Image.new("RGBA", (1, 1), 0x04030201)
>>> im.getpixel((0, 0))
(1, 2, 3, 4)
```

Some methods accept other forms, such as color names. See *Color names*.

Info

You can attach auxiliary information to an image using the *info* attribute. This is a dictionary object.

How such information is handled when loading and saving image files is up to the file format handler (see the chapter on *Image file formats*). Most handlers add properties to the *info* attribute when loading an image, but ignore it when saving images.

Transparency

If an image does not have an alpha band, transparency may be specified in the *info* attribute with a "transparency" key.

Most of the time, the "transparency" value is a single integer, describing which pixel value is transparent in a "1", "L", "I" or "P" mode image. However, PNG images may have three values, one for each channel in an "RGB" mode image, or can have a byte string for a "P" mode image, to specify the alpha value for each palette entry.

Orientation

A common element of the *info* attribute for JPG and TIFF images is the EXIF orientation tag. This is an instruction for how the image data should be oriented. For example, it may instruct an image to be rotated by 90 degrees, or to be mirrored. To apply this information to an image, *exif_transpose()* can be used.

Filters

For geometry operations that may map multiple input pixels to a single output pixel, the Python Imaging Library provides different resampling *filters*.

`Resampling.`**`NEAREST`**

Pick one nearest pixel from the input image. Ignore all other input pixels.

`Resampling.`**`BOX`**

Each pixel of source image contributes to one pixel of the destination image with identical weights. For upscaling is equivalent of *Resampling.NEAREST*. This filter can only be used with the *resize()* and *thumbnail()* methods.

Added in version 3.4.0.

`Resampling.`**`BILINEAR`**

For resize calculate the output pixel value using linear interpolation on all pixels that may contribute to the output value. For other transformations linear interpolation over a 2x2 environment in the input image is used.

`Resampling.`**`HAMMING`**

Produces a sharper image than *Resampling.BILINEAR*, doesn't have dislocations on local level like with *Resampling.BOX*. This filter can only be used with the *resize()* and *thumbnail()* methods.

Added in version 3.4.0.

`Resampling.`**`BICUBIC`**

For resize calculate the output pixel value using cubic interpolation on all pixels that may contribute to the output value. For other transformations cubic interpolation over a 4x4 environment in the input image is used.

`Resampling.`**`LANCZOS`**

Calculate the output pixel value using a high-quality Lanczos filter (a truncated sinc) on all pixels that may contribute to the output value. This filter can only be used with the *resize()* and *thumbnail()* methods.

Added in version 1.1.3.

Filters comparison table

Filter	Downscaling quality	Upscaling quality	Performance
Resampling.NEAREST			
Resampling.BOX			
Resampling.BILINEAR			
Resampling.HAMMING			
Resampling.BICUBIC			
Resampling.LANCZOS			

1.2.4 Appendices

Note

Contributors please include appendices as needed or appropriate with your bug fixes, feature additions and tests.

Image file formats

The Python Imaging Library supports a wide variety of raster file formats. Over 30 different file formats can be identified and read by the library. Write support is less extensive, but most common interchange and presentation formats are supported.

The *open()* function identifies files from their contents, not their names, but the *save()* method looks at the name to determine which format to use, unless the format is given explicitly.

When an image is opened from a file, only that instance of the image is considered to have the format. Copies of the image will contain data loaded from the file, but not the file itself, meaning that it can no longer be considered to be in the original format. So if *copy()* is called on an image, or another method internally creates a copy of the image, then any methods or attributes specific to the format will no longer be present. The `fp` (file pointer) attribute will no longer be present, and the *format* attribute will be `None`.

Fully supported formats

AVIF

Pillow reads and writes AVIF files, including AVIF sequence images. It is only possible to save 8-bit AVIF images, and all AVIF images are decoded as 8-bit RGB(A).

The *save()* method supports the following options:

quality
: Integer, 0-100, defaults to 75. 0 gives the smallest size and poorest quality, 100 the largest size and best quality.

subsampling
: If present, sets the subsampling for the encoder. If absent, and all frames are in grayscale mode without alpha, `4:0:0` is used. Otherwise defaults to `4:2:0`. Options include:

- `4:0:0`
- `4:2:0`
- `4:2:2`
- `4:4:4`

speed
: Quality/speed trade-off (0=slower/better, 10=fastest). Defaults to 6.

max_threads
: Limit the number of active threads used. By default, there is no limit. If the aom codec is used, there is a maximum of 64.

range
: YUV range, either "full" or "limited". Defaults to "full".

codec
: AV1 codec to use for encoding. Specific values are "aom", "rav1e", and "svt", presuming the chosen codec is available. Defaults to "auto", which will choose the first available codec in the order of the preceding list.

tile_rows / tile_cols
: For tile encoding, the (log 2) number of tile rows and columns to use. Valid values are 0-6, default 0. Ignored if "autotiling" is set to true.

autotiling
: Split the image up to allow parallelization. Enabled automatically if "tile_rows" and "tile_cols" both have their default values of zero.

alpha_premultiplied
: Encode the image with premultiplied alpha. Defaults to `False`.

advanced
: Codec specific options.

icc_profile
: The ICC Profile to include in the saved file.

exif
: The exif data to include in the saved file.

xmp
: The XMP data to include in the saved file.

Saving sequences

When calling *save()* to write an AVIF file, by default only the first frame of a multiframe image will be saved. If the `save_all` argument is present and true, then all frames will be saved, and the following options will also be available.

append_images
: A list of images to append as additional frames. Each of the images in the list can be single or multiframe images.

duration
: The display duration of each frame, in milliseconds. Pass a single integer for a constant duration, or a list or tuple to set the duration for each frame separately.

BLP

BLP is the Blizzard Mipmap Format, a texture format used in World of Warcraft. Pillow supports reading `JPEG` Compressed or raw `BLP1` images, and all types of `BLP2` images.

Saving

Pillow supports writing BLP images. The *save()* method can take the following keyword arguments:

blp_version
: If present and set to "BLP1", images will be saved as BLP1. Otherwise, images will be saved as BLP2.

BMP

Pillow reads and writes Windows and OS/2 BMP files containing `1`, `L`, `P`, or `RGB` data. 16-colour images are read as `P` images. Support for reading 8-bit run-length encoding was added in Pillow 9.1.0. Support for reading 4-bit run-length encoding was added in Pillow 9.3.0.

Opening

The *open()* method sets the following *info* properties:

compression
: Set to 1 if the file is a 256-color run-length encoded image. Set to 2 if the file is a 16-color run-length encoded image.

DDS

DDS is a popular container texture format used in video games and natively supported by DirectX.

DXT1 and DXT5 pixel formats can be read, only in `RGBA` mode.

Added in version 3.4.0: DXT3 images can be read in `RGBA` mode and DX10 images can be read in `RGB` and `RGBA` mode.

Added in version 6.0.0: Uncompressed `RGBA` images can be read.

Added in version 8.3.0: BC5S images can be opened in `RGB` mode, and uncompressed `RGB` images can be read. Uncompressed data can also be saved to image files.

Added in version 9.3.0: ATI1 images can be opened in `L` mode and ATI2 images can be opened in `RGB` mode.

Added in version 9.4.0: Uncompressed `L` ("luminance") and `LA` images can be opened and saved.

Added in version 10.1.0: BC5U can be read in `RGB` mode, and 8-bit color indexed images can be read in `P` mode.

Added in version 11.2.1: DXT1, DXT3, DXT5, BC2, BC3 and BC5 pixel formats can be saved::

    im.save(out, pixel_format="DXT1")

DIB

Pillow reads and writes DIB files. DIB files are similar to BMP files, so see above for more information.

Added in version 6.0.0.

EPS

Pillow identifies EPS files containing image data, and can read files that contain embedded raster images (ImageData descriptors). If Ghostscript is available, other EPS files can be read as well. The EPS driver can also write EPS images. The EPS driver can read EPS images in `L`, `LAB`, `RGB` and `CMYK` mode, but Ghostscript may convert the images to `RGB` mode rather than leaving them in the original color space. The EPS driver can write images in `L`, `RGB` and `CMYK` modes.

Loading

To use Ghostscript, Pillow searches for the "gs" executable. On Windows, it also searches for "gswin32c" and "gswin64c". To customise this behaviour, `EpsImagePlugin.gs_binary = "gswin64"` will set the name of the executable to use. `EpsImagePlugin.gs_binary = False` will prevent Ghostscript use.

If Ghostscript is available, you can call the *load()* method with the following parameters to affect how Ghostscript renders the EPS.

scale
: Affects the scale of the resultant rasterized image. If the EPS suggests that the image be rendered at 100px x 100px, setting this parameter to 2 will make the Ghostscript render a 200px x 200px image instead. The relative position of the bounding box is maintained:

```
im = Image.open(...)
im.size  # (100,100)
im.load(scale=2)
im.size  # (200,200)
```

transparency
: If true, generates an RGBA image with a transparent background, instead of the default behaviour of an RGB image with a white background.

GIF

Pillow reads GIF87a and GIF89a versions of the GIF file format. The library writes files in GIF87a by default, unless GIF89a features are used or GIF89a is already in use. Files are written with LZW encoding.

GIF files are initially read as grayscale (`L`) or palette mode (`P`) images. Seeking to later frames in a `P` image will change the image to `RGB` (or `RGBA` if the first frame had transparency).

`P` mode images are changed to `RGB` because each frame of a GIF may contain its own individual palette of up to 256 colors. When a new frame is placed onto a previous frame, those colors may combine to exceed the `P` mode limit of 256 colors. Instead, the image is converted to `RGB` handle this.

If you would prefer the first `P` image frame to be `RGB` as well, so that every `P` frame is converted to `RGB` or `RGBA` mode, there is a setting available:

```
from PIL import GifImagePlugin
GifImagePlugin.LOADING_STRATEGY = GifImagePlugin.LoadingStrategy.RGB_ALWAYS
```

GIF frames do not always contain individual palettes however. If there is only a global palette, then all of the colors can fit within `P` mode. If you would prefer the frames to be kept as `P` in that case, there is also a setting available:

```
from PIL import GifImagePlugin
GifImagePlugin.LOADING_STRATEGY = GifImagePlugin.LoadingStrategy.RGB_AFTER_DIFFERENT_
↪PALETTE_ONLY
```

To restore the default behavior, where `P` mode images are only converted to `RGB` or `RGBA` after the first frame:

```
from PIL import GifImagePlugin
GifImagePlugin.LOADING_STRATEGY = GifImagePlugin.LoadingStrategy.RGB_AFTER_FIRST
```

Opening

The *open()* method sets the following *info* properties:

background
: Default background color (a palette color index).

transparency
: Transparency color index. This key is omitted if the image is not transparent.

version
: Version (either `GIF87a` or `GIF89a`).

duration
: May not be present. The time to display the current frame of the GIF, in milliseconds.

loop
: May not be present. The number of times the GIF should loop. 0 means that it will loop forever.

comment
: May not be present. A comment about the image. This is the last comment found before the current frame's image.

extension
: May not be present. Contains application specific information.

Reading sequences

The GIF loader supports the *seek()* and *tell()* methods. You can combine these methods to seek to the next frame (`im.seek(im.tell() + 1)`).

`im.seek()` raises an `EOFError` if you try to seek after the last frame.

Saving

When calling *save()* to write a GIF file, the following options are available:

```
im.save(out, save_all=True, append_images=[im1, im2, ...])
```

save_all
: If present and true, or if `append_images` is not empty, all frames of the image will be saved. Otherwise, only the first frame of a multiframe image will be saved.

append_images
: A list of images to append as additional frames. Each of the images in the list can be single or multiframe images. This is supported for AVIF, GIF, PDF, PNG, TIFF and WebP.

  It is also supported for ICO and ICNS. If images are passed in of relevant sizes, they will be used instead of scaling down the main image.

include_color_table
Whether or not to include local color table.

interlace
Whether or not the image is interlaced. By default, it is, unless the image is less than 16 pixels in width or height.

disposal
Indicates the way in which the graphic is to be treated after being displayed.

- 0 - No disposal specified.
- 1 - Do not dispose.
- 2 - Restore to background color.
- 3 - Restore to previous content.

Pass a single integer for a constant disposal, or a list or tuple to set the disposal for each frame separately.

palette
Use the specified palette for the saved image. The palette should be a bytes or bytearray object containing the palette entries in RGBRGB... form. It should be no more than 768 bytes. Alternately, the palette can be passed in as an *PIL.ImagePalette.ImagePalette* object.

optimize
Whether to attempt to compress the palette by eliminating unused colors (this is only useful if the palette can be compressed to the next smaller power of 2 elements) and whether to mark all pixels that are not new in the next frame as transparent.

This is attempted by default, unless a palette is specified as an option or as part of the first image's *info* dictionary.

Note that if the image you are saving comes from an existing GIF, it may have the following properties in its *info* dictionary. For these options, if you do not pass them in, they will default to their *info* values.

transparency
Transparency color index.

duration
The display duration of each frame of the multiframe gif, in milliseconds. Pass a single integer for a constant duration, or a list or tuple to set the duration for each frame separately.

loop
Integer number of times the GIF should loop. 0 means that it will loop forever. If omitted or `None`, the image will not loop.

comment
A comment about the image.

Reading local images

The GIF loader creates an image memory the same size as the GIF file's *logical screen size*, and pastes the actual pixel data (the *local image*) into this image. If you only want the actual pixel rectangle, you can crop the image:

```
im = Image.open(...)

if im.tile[0][0] == "gif":
    # only read the first "local image" from this GIF file
    box = im.tile[0][1]
    im = im.crop(box)
```

ICNS

Pillow reads and writes macOS `.icns` files. By default, the largest available icon is read, though you can override this by setting the *size* property before calling *load()*. The *open()* method sets the following *info* property:

> **Note**
>
> Prior to version 8.3.0, Pillow could only write ICNS files on macOS.

sizes
: A list of supported sizes found in this icon file; these are a 3-tuple, `(width, height, scale)`, where `scale` is 2 for a retina icon and 1 for a standard icon.

Loading

You can call the *load()* method with the following parameter.

scale
: Affects the scale of the resultant image. If the size is set to `(512, 512)`, after loading at scale 2, the final value of *size* will be `(1024, 1024)`.

Saving

The *save()* method can take the following keyword arguments:

append_images
: A list of images to replace the scaled down versions of the image. The order of the images does not matter, as their use is determined by the size of each image.

  Added in version 5.1.0.

ICO

ICO is used to store icons on Windows. The largest available icon is read.

Saving

The *save()* method supports the following options:

sizes
: A list of sizes included in this ico file; these are a 2-tuple, `(width, height)`; Default to `[(16, 16), (24, 24), (32, 32), (48, 48), (64, 64), (128, 128), (256, 256)]`. Any sizes bigger than the original size or 256 will be ignored.

The *save()* method can take the following keyword arguments:

append_images
: A list of images to replace the scaled down versions of the image. The order of the images does not matter, as their use is determined by the size of each image.

  Added in version 8.1.0.

bitmap_format
: By default, the image data will be saved in PNG format. With a bitmap format of "bmp", image data will be saved in BMP format instead.

  Added in version 8.3.0.

IM

IM is a format used by LabEye and other applications based on the IFUNC image processing library. The library reads and writes most uncompressed interchange versions of this format.

IM is the only format that can store all internal Pillow formats.

JPEG

Pillow reads JPEG, JFIF, and Adobe JPEG files containing `L`, `RGB`, or `CMYK` data. It writes standard and progressive JFIF files.

Using the *draft()* method, you can speed things up by converting `RGB` images to `L`, and resize images to 1/2, 1/4 or 1/8 of their original size while loading them.

By default Pillow doesn't allow loading of truncated JPEG files, set *ImageFile.LOAD_TRUNCATED_IMAGES* to override this.

Opening

The *open()* method may set the following *info* properties if available:

jfif
: JFIF application marker found. If the file is not a JFIF file, this key is not present.

jfif_version
: A tuple representing the jfif version, (major version, minor version).

jfif_density
: A tuple representing the pixel density of the image, in units specified by jfif_unit.

jfif_unit
: Units for the jfif_density:

  - 0 - No Units
  - 1 - Pixels per Inch
  - 2 - Pixels per Centimeter

dpi
: A tuple representing the reported pixel density in pixels per inch, if the file is a jfif file and the units are in inches.

adobe
: Adobe application marker found. If the file is not an Adobe JPEG file, this key is not present.

adobe_transform
: Vendor Specific Tag.

progression
: Indicates that this is a progressive JPEG file.

icc_profile
: The ICC color profile for the image.

exif
: Raw EXIF data from the image.

comment
: A comment about the image, from the COM marker. This is separate from the UserComment tag that may be stored in the EXIF data.

  Added in version 7.1.0.

Saving

The *save()* method supports the following options:

quality
: The image quality, on a scale from 0 (worst) to 95 (best), or the string `keep`. The default is 75. Values above 95 should be avoided; 100 disables portions of the JPEG compression algorithm, and results in large files with hardly any gain in image quality. The value `keep` is only valid for JPEG files and will retain the original image quality level, subsampling, and qtables. For more information on how qtables are modified based on the quality parameter, see the qtables section.

optimize
: If present and true, indicates that the encoder should make an extra pass over the image in order to select optimal encoder settings.

progressive
: If present and true, indicates that this image should be stored as a progressive JPEG file.

dpi
: A tuple of integers representing the pixel density, `(x,y)`.

icc_profile
: If present and true, the image is stored with the provided ICC profile. If this parameter is not provided, the image will be saved with no profile attached. To preserve the existing profile:

```
im.save(filename, 'jpeg', icc_profile=im.info.get('icc_profile'))
```

exif
: If present, the image will be stored with the provided raw EXIF data.

keep_rgb
: By default, libjpeg converts images with an RGB color space to YCbCr. If this option is present and true, those images will be stored as RGB instead.

  When this option is enabled, attempting to chroma-subsample RGB images with the `subsampling` option will raise an `OSError`.

  Added in version 10.2.0.

subsampling
: If present, sets the subsampling for the encoder.

  - `keep`: Only valid for JPEG files, will retain the original image setting.
  - `4:4:4`, `4:2:2`, `4:2:0`: Specific sampling values
  - `0`: equivalent to `4:4:4`
  - `1`: equivalent to `4:2:2`
  - `2`: equivalent to `4:2:0`

  If absent, the setting will be determined by libjpeg or libjpeg-turbo.

restart_marker_blocks
: If present, emit a restart marker whenever the specified number of MCU blocks has been produced.

  Added in version 10.2.0.

restart_marker_rows
: If present, emit a restart marker whenever the specified number of MCU rows has been produced.

  Added in version 10.2.0.

qtables
If present, sets the qtables for the encoder. This is listed as an advanced option for wizards in the JPEG documentation. Use with caution. `qtables` can be one of several types of values:

- a string, naming a preset, e.g. `keep`, `web_low`, or `web_high`
- a list, tuple, or dictionary (with integer keys = range(len(keys))) of lists of 64 integers. There must be between 2 and 4 tables.

If a quality parameter is provided, the qtables will be adjusted accordingly. By default, the qtables are based on a standard JPEG table with a quality of 50. The qtable values will be reduced if the quality is higher than 50 and increased if the quality is lower than 50.

Added in version 2.5.0.

streamtype
Allows storing images without quantization and Huffman tables, or with these tables but without image data. This is useful for container formats or network protocols that handle tables separately and share them between images.

- `0` (default): interchange datastream, with tables and image data
- `1`: abbreviated table specification (tables-only) datastream

  Added in version 10.2.0.
- `2`: abbreviated image (image-only) datastream

comment
A comment about the image.

Added in version 9.4.0.

> **Note**
>
> To enable JPEG support, you need to build and install the IJG JPEG library before building the Python Imaging Library. See the distribution README for details.

JPEG 2000

Added in version 2.4.0.

Pillow reads and writes JPEG 2000 files containing `L`, `LA`, `RGB`, `RGBA`, or `YCbCr` data. When reading, `YCbCr` data is converted to `RGB` or `RGBA` depending on whether or not there is an alpha channel.

Added in version 8.3.0: Pillow can read (but not write) `RGB`, `RGBA`, and `YCbCr` images with subsampled components.

Added in version 10.4.0: Pillow can read `CMYK` images with OpenJPEG 2.5.1 and later.

Added in version 11.1.0: Pillow can write `CMYK` images with OpenJPEG 2.5.3 and later.

Pillow supports JPEG 2000 raw codestreams (`.j2k` files), as well as boxed JPEG 2000 files (`.jp2` or `.jpx` files).

When loading, if you set the `mode` on the image prior to the *load()* method being invoked, you can ask Pillow to convert the image to either `RGB` or `RGBA` rather than choosing for itself. It is also possible to set `reduce` to the number of resolutions to discard (each one reduces the size of the resulting image by a factor of 2), and `layers` to specify the number of quality layers to load.

Saving

The *save()* method supports the following options:

offset
The image offset, as a tuple of integers, e.g. (16, 16)

tile_offset
The tile offset, again as a 2-tuple of integers.

tile_size
The tile size as a 2-tuple. If not specified, or if set to None, the image will be saved without tiling.

quality_mode
Either `"rates"` or `"dB"` depending on the units you want to use to specify image quality.

quality_layers
A sequence of numbers, each of which represents either an approximate size reduction (if quality mode is `"rates"`) or a signal to noise ratio value in decibels. If not specified, defaults to a single layer of full quality.

num_resolutions
The number of different image resolutions to be stored (which corresponds to the number of Discrete Wavelet Transform decompositions plus one).

codeblock_size
The code-block size as a 2-tuple. Minimum size is 4 x 4, maximum is 1024 x 1024, with the additional restriction that no code-block may have more than 4096 coefficients (i.e. the product of the two numbers must be no greater than 4096).

precinct_size
The precinct size as a 2-tuple. Must be a power of two along both axes, and must be greater than the code-block size.

irreversible
If `True`, use the lossy discrete waveform transformation DWT 9-7. Defaults to `False`, which uses the lossless DWT 5-3.

mct
If `1` then enable multiple component transformation when encoding, otherwise use `0` for no component transformation (default). If MCT is enabled and `irreversible` is `True` then the Irreversible Color Transformation will be applied, otherwise encoding will use the Reversible Color Transformation. MCT works best with a `mode` of `RGB` and is only applicable when the image data has 3 components.

Added in version 9.1.0.

progression
Controls the progression order; must be one of `"LRCP"`, `"RLCP"`, `"RPCL"`, `"PCRL"`, `"CPRL"`. The letters stand for Component, Position, Resolution and Layer respectively and control the order of encoding, the idea being that e.g. an image encoded using LRCP mode can have its quality layers decoded as they arrive at the decoder, while one encoded using RLCP mode will have increasing resolutions decoded as they arrive, and so on.

signed
If true, then tell the encoder to save the image as signed.

Added in version 9.4.0.

cinema_mode
Set the encoder to produce output compliant with the digital cinema specifications. The options here are `"no"` (the default), `"cinema2k-24"` for 24fps 2K, `"cinema2k-48"` for 48fps 2K, and `"cinema4k-24"` for 24fps 4K. Note that for compliant 2K files, *at least one* of your image dimensions must match 2048 x 1080, while for compliant 4K files, *at least one* of the dimensions must match 4096 x 2160.

no_jp2
: If `True` then don't wrap the raw codestream in the JP2 file format when saving, otherwise the extension of the filename will be used to determine the format (default).

  Added in version 9.1.0.

comment
: Adds a custom comment to the file, replacing the default "Created by OpenJPEG version" comment.

  Added in version 9.5.0.

plt
: If `True` and OpenJPEG 2.4.0 or later is available, then include a PLT (packet length, tile-part header) marker in the produced file. Defaults to `False`.

  Added in version 9.5.0.

> **Note**
>
> To enable JPEG 2000 support, you need to build and install the OpenJPEG library, version 2.0.0 or higher, before building the Python Imaging Library.
>
> Windows users can install the OpenJPEG binaries available on the OpenJPEG website, but must add them to their PATH in order to use Pillow (if you fail to do this, you will get errors about not being able to load the `_imaging` DLL).

MPO

Pillow reads and writes Multi Picture Object (MPO) files. When first opened, it loads the primary image. The *seek()* and *tell()* methods may be used to read other pictures from the file. The pictures are zero-indexed and random access is supported.

Saving

When calling *save()* to write an MPO file, by default only the first frame of a multiframe image will be saved. If the `save_all` argument is present and true, or if `append_images` is not empty, all frames will be saved.

append_images
: A list of images to append as additional pictures. Each of the images in the list can be single or multiframe images.

  Added in version 9.3.0.

MSP

Pillow identifies and reads MSP files from Windows 1 and 2. The library writes uncompressed (Windows 1) versions of this format.

PCX

Pillow reads and writes PCX files containing `1`, `L`, `P`, or `RGB` data.

Opening

The *open()* function sets the following *info* properties:

scale
: The absolute value of the number stored in the *Scale Factor / Endianness* line.

PNG

Pillow identifies, reads, and writes PNG files containing `1`, `L`, `LA`, `I`, `P`, `RGB` or `RGBA` data. Interlaced files are supported as of v1.1.7.

As of Pillow 6.0, EXIF data can be read from PNG images. However, unlike other image formats, EXIF data is not guaranteed to be present in *info* until *load()* has been called.

By default Pillow doesn't allow loading of truncated PNG files, set *ImageFile.LOAD_TRUNCATED_IMAGES* to override this.

Opening

The *open()* function sets the following *info* properties, when appropriate:

chromaticity
: The chromaticity points, as an 8 tuple of floats. (`White Point X`, `White Point Y`, `Red X`, `Red Y`, `Green X`, `Green Y`, `Blue X`, `Blue Y`)

gamma
: Gamma, given as a floating point number.

srgb
: The sRGB rendering intent as an integer.

  - 0 Perceptual
  - 1 Relative Colorimetric
  - 2 Saturation
  - 3 Absolute Colorimetric

transparency
: For `P` images: Either the palette index for full transparent pixels, or a byte string with alpha values for each palette entry.

  For `1`, `L`, `I` and `RGB` images, the color that represents full transparent pixels in this image.

  This key is omitted if the image is not a transparent palette image.

`open` also sets `Image.text` to a dictionary of the values of the `tEXt`, `zTXt`, and `iTXt` chunks of the PNG image. Individual compressed chunks are limited to a decompressed size of *PngImagePlugin.MAX_TEXT_CHUNK*, by default 1MB, to prevent decompression bombs. Additionally, the total size of all of the text chunks is limited to *PngImagePlugin.MAX_TEXT_MEMORY*, defaulting to 64MB.

Saving

The *save()* method supports the following options:

optimize
: If present and true, instructs the PNG writer to make the output file as small as possible. This includes extra processing in order to find optimal encoder settings.

transparency
For P, 1, L, I, and RGB images, this option controls what color from the image to mark as transparent.

For P images, this can be a either the palette index, or a byte string with alpha values for each palette entry.

dpi
A tuple of two numbers corresponding to the desired dpi in each direction.

pnginfo
A *PIL.PngImagePlugin.PngInfo* instance containing chunks.

compress_level
ZLIB compression level, a number between 0 and 9: 1 gives best speed, 9 gives best compression, 0 gives no compression at all. Default is 6. When `optimize` option is True `compress_level` has no effect (it is set to 9 regardless of a value passed).

icc_profile
The ICC Profile to include in the saved file.

exif
The exif data to include in the saved file.

Added in version 6.0.0.

bits (experimental)
For P images, this option controls how many bits to store. If omitted, the PNG writer uses 8 bits (256 colors).

dictionary (experimental)
Set the ZLIB encoder dictionary.

Note

To enable PNG support, you need to build and install the ZLIB compression library before building the Python Imaging Library. See the installation documentation for details.

APNG sequences

The PNG loader includes limited support for reading and writing Animated Portable Network Graphics (APNG) files. When an APNG file is loaded, *get_format_mimetype()* will return `"image/apng"`. The value of the *is_animated* property will be `True` when the *n_frames* property is greater than 1. For APNG files, the `n_frames` property depends on both the animation frame count as well as the presence or absence of a default image. See the `default_image` property documentation below for more details. The *seek()* and *tell()* methods are supported.

`im.seek()` raises an `EOFError` if you try to seek after the last frame.

These *info* properties will be set for APNG frames, where applicable:

default_image
Specifies whether or not this APNG file contains a separate default image, which is not a part of the actual APNG animation.

When an APNG file contains a default image, the initially loaded image (i.e. the result of `seek(0)`) will be the default image. To account for the presence of the default image, the *n_frames* property will be set to `frame_count + 1`, where `frame_count` is the actual APNG animation frame count. To load the first APNG animation frame, `seek(1)` must be called.

- `True` - The APNG contains default image, which is not an animation frame.

- `False` - The APNG does not contain a default image. The `n_frames` property will be set to the actual APNG animation frame count. The initially loaded image (i.e. `seek(0)`) will be the first APNG animation frame.

loop
The number of times to loop this APNG, 0 indicates infinite looping.

duration
The time to display this APNG frame (in milliseconds), given as a float.

> **Note**
>
> The APNG loader returns images the same size as the APNG file's logical screen size. The returned image contains the pixel data for a given frame, after applying any APNG frame disposal and frame blend operations (i.e. it contains what a web browser would render for this frame - the composite of all previous frames and this frame).
>
> Any APNG file containing sequence errors is treated as an invalid image. The APNG loader will not attempt to repair and reorder files containing sequence errors.

Saving

When calling *save()*, by default only a single frame PNG file will be saved. To save an APNG file (including a single frame APNG), the `save_all` parameter should be set to `True` or `append_images` should not be empty. The following parameters can also be set:

default_image
Boolean value, specifying whether or not the base image is a default image. If `True`, the base image will be used as the default image, and the first image from the `append_images` sequence will be the first APNG animation frame. If `False`, the base image will be used as the first APNG animation frame. Defaults to `False`.

append_images
A list or tuple of images to append as additional frames. Each of the images in the list can be single or multiframe images. The size of each frame should match the size of the base image. Also note that if a frame's mode does not match that of the base image, the frame will be converted to the base image mode.

loop
Integer number of times to loop this APNG, 0 indicates infinite looping. Defaults to 0.

duration
The length of time (or list or tuple of lengths of time) to display this APNG frame (in milliseconds). Defaults to 0.

disposal
An integer (or list or tuple of integers) specifying the APNG disposal operation to be used for this frame before rendering the next frame. Defaults to 0.

- 0 (*OP_NONE*, default) - No disposal is done on this frame before rendering the next frame.
- 1 (*PIL.PngImagePlugin.Disposal.OP_BACKGROUND*) - This frame's modified region is cleared to fully transparent black before rendering the next frame.
- 2 (*OP_PREVIOUS*) - This frame's modified region is reverted to the previous frame's contents before rendering the next frame.

blend
An integer (or list or tuple of integers) specifying the APNG blend operation to be used for this frame before rendering the next frame. Defaults to 0.

- 0 (*OP_SOURCE*) - All color components of this frame, including alpha, overwrite the previous output image contents.
- 1 (*OP_OVER*) - This frame should be alpha composited with the previous output image contents.

Note

The `duration`, `disposal` and `blend` parameters can be set to lists or tuples to specify values for each individual frame in the animation. The length of the list or tuple must be identical to the total number of actual frames in the APNG animation. If the APNG contains a default image (i.e. `default_image` is set to `True`), these list or tuple parameters should not include an entry for the default image.

PPM

Pillow reads and writes PBM, PGM, PPM, PNM and PFM files containing `1`, `L`, `I`, `RGB` or `F` data.

"Raw" (P4 to P6) formats can be read, and are used when writing.

Added in version 9.2.0: "Plain" (P1 to P3) formats can be read.

Added in version 10.3.0: Grayscale (Pf format) Portable FloatMap (PFM) files containing `F` data can be read and used when writing.

Color (PF format) PFM files are not supported.

QOI

Added in version 9.5.0.

Pillow reads and writes images in Quite OK Image format using a Python codec. If you wish to write code specifically for this format, qoi is an alternative library that uses C to decode the image and interfaces with NumPy.

Saving

The *save()* method can take the following keyword arguments:

colorspace
: If set to "sRGB", the colorspace will be written as sRGB with linear alpha, instead of all channels being linear.

SGI

Pillow reads and writes uncompressed `L`, `RGB`, and `RGBA` files.

SPIDER

Pillow reads and writes SPIDER image files of 32-bit floating point data ("F;32F").

Pillow also reads SPIDER stack files containing sequences of SPIDER images. The *seek()* and *tell()* methods are supported, and random access is allowed.

Opening

The *open()* method sets the following attributes:

format
: Set to `SPIDER`

istack
: Set to 1 if the file is an image stack, else 0.

n_frames
: Set to the number of images in the stack.

A convenience method, *convert2byte()*, is provided for converting floating point data to byte data (mode L):

```
im = Image.open("image001.spi").convert2byte()
```

Saving

The extension of SPIDER files may be any 3 alphanumeric characters. Therefore the output format must be specified explicitly:

```
im.save('newimage.spi', format='SPIDER')
```

For more information about the SPIDER image processing package, see https://github.com/spider-em/SPIDER

TGA

Pillow reads and writes TGA images containing L, LA, P, RGB, and RGBA data. Pillow can read and write both uncompressed and run-length encoded TGAs.

Saving

The *save()* method can take the following keyword arguments:

compression
: If set to "tga_rle", the file will be run-length encoded.

  Added in version 5.3.0.

id_section
: The identification field.

  Added in version 5.3.0.

orientation
: If present and a positive number, the first pixel is for the top left corner, rather than the bottom left corner.

  Added in version 5.3.0.

TIFF

Pillow reads and writes TIFF files. It can read both striped and tiled images, pixel and plane interleaved multi-band images. If you have libtiff and its headers installed, Pillow can read and write many kinds of compressed TIFF files. If not, Pillow will only read and write uncompressed files.

> **Note**
>
> Beginning in version 5.0.0, Pillow requires libtiff to read or write compressed files. Prior to that release, Pillow had buggy support for reading Packbits, LZW and JPEG compressed TIFFs without using libtiff.

Opening

The *open()* method sets the following *info* properties:

compression
Compression mode.

Added in version 2.0.0.

dpi
Image resolution as an `(xdpi, ydpi)` tuple, where applicable. You can use the *tag* attribute to get more detailed information about the image resolution.

Added in version 1.1.5.

resolution
Image resolution as an `(xres, yres)` tuple, where applicable. This is a measurement in whichever unit is specified by the file.

Added in version 1.1.5.

The *tag_v2* attribute contains a dictionary of TIFF metadata. The keys are numerical indexes from *TiffTags.TAGS_V2*. Values are strings or numbers for single items, multiple values are returned in a tuple of values. Rational numbers are returned as a *IFDRational* object.

Added in version 3.0.0.

For compatibility with legacy code, the *tag* attribute contains a dictionary of decoded TIFF fields as returned prior to version 3.0.0. Values are returned as either strings or tuples of numeric values. Rational numbers are returned as a tuple of `(numerator, denominator)`.

Deprecated since version 3.0.0.

Reading multi-frame TIFF images

The TIFF loader supports the *seek()* and *tell()* methods, taking and returning frame numbers within the image file. You can combine these methods to seek to the next frame (`im.seek(im.tell() + 1)`). Frames are numbered from 0 to `im.n_frames - 1`, and can be accessed in any order.

`im.seek()` raises an `EOFError` if you try to seek after the last frame.

Saving

The *save()* method can take the following keyword arguments:

save_all
If true, or if `append_images` is not empty, Pillow will save all frames of the image to a multiframe tiff document.

Added in version 3.4.0.

append_images
A list of images to append as additional frames. Each of the images in the list can be single or multiframe images.

Added in version 4.2.0.

tiffinfo
A *ImageFileDirectory_v2* object or dict object containing tiff tags and values. The TIFF field type is autodetected for Numeric and string values, any other types require using an *ImageFileDirectory_v2* object and setting the type in *tagtype* with the appropriate numerical value from *TiffTags.TYPES*.

Added in version 2.3.0.

Metadata values that are of the rational type should be passed in using a *IFDRational* object.

Added in version 3.1.0.

For compatibility with legacy code, a *ImageFileDirectory_v1* object may be passed in this field. However, this is deprecated.

Added in version 5.4.0.

Previous versions only supported some tags when writing using libtiff. The supported list is found in *TiffTags.LIBTIFF_CORE*.

Added in version 6.1.0.

Added support for signed types (e.g. `TIFF_SIGNED_LONG`) and multiple values. Multiple values for a single tag must be to *ImageFileDirectory_v2* as a tuple and require a matching type in *tagtype* tagtype.

exif
Alternate keyword to "tiffinfo", for consistency with other formats.

Added in version 8.4.0.

big_tiff
If true, the image will be saved as a BigTIFF.

Added in version 11.1.0.

compression
A string containing the desired compression method for the file. (valid only with libtiff installed) Valid compression methods are: `None`, `"group3"`, `"group4"`, `"jpeg"`, `"lzma"`, `"packbits"`, `"tiff_adobe_deflate"`, `"tiff_ccitt"`, `"tiff_lzw"`, `"tiff_raw_16"`, `"tiff_sgilog"`, `"tiff_sgilog24"`, `"tiff_thunderscan"`, `"webp"`, `"zstd"`

quality
The image quality for JPEG compression, on a scale from 0 (worst) to 100 (best). The default is 75.

Added in version 6.1.0.

These arguments to set the tiff header fields are an alternative to using the general tags available through tiffinfo.

description

software

date_time

artist

copyright
Strings

icc_profile
The ICC Profile to include in the saved file.

resolution_unit
An integer. 1 for no unit, 2 for inches and 3 for centimeters.

resolution
Either an integer or a float, used for both the x and y resolution.

x_resolution
Either an integer or a float.

y_resolution
Either an integer or a float.

dpi
A tuple of `(x_resolution, y_resolution)`, with inches as the resolution unit. For consistency with other image formats, the x and y resolutions of the dpi will be rounded to the nearest integer.

WebP

Pillow reads and writes WebP files. Requires libwebp v0.5.0 or later.

Saving

The *save()* method supports the following options:

lossless
If present and true, instructs the WebP writer to use lossless compression.

quality
Integer, 0-100, defaults to 80. For lossy, 0 gives the smallest size and 100 the largest. For lossless, this parameter is the amount of effort put into the compression: 0 is the fastest, but gives larger files compared to the slowest, but best, 100.

alpha_quality
Integer, 0-100, defaults to 100. For lossy compression only. 0 gives the smallest size and 100 is lossless.

method
Quality/speed trade-off (0=fast, 6=slower-better). Defaults to 4.

exact
If true, preserve the transparent RGB values. Otherwise, discard invisible RGB values for better compression. Defaults to false.

icc_profile
The ICC Profile to include in the saved file.

exif
The exif data to include in the saved file.

xmp
The XMP data to include in the saved file.

Saving sequences

When calling *save()* to write a WebP file, by default only the first frame of a multiframe image will be saved. If the `save_all` argument is present and true, or if `append_images` is not empty, all frames will be saved, and the following options will also be available.

append_images
A list of images to append as additional frames. Each of the images in the list can be single or multiframe images.

duration
The display duration of each frame, in milliseconds. Pass a single integer for a constant duration, or a list or tuple to set the duration for each frame separately.

loop
Number of times to repeat the animation. Defaults to [0 = infinite].

background
Background color of the canvas, as an RGBA tuple with values in the range of (0-255).

minimize_size
If true, minimize the output size (slow). Implicitly disables key-frame insertion.

kmin, kmax
: Minimum and maximum distance between consecutive key frames in the output. The library may insert some key frames as needed to satisfy this criteria. Note that these conditions should hold: kmax > kmin and kmin >= kmax / 2 + 1. Also, if kmax <= 0, then key-frame insertion is disabled; and if kmax == 1, then all frames will be key-frames (kmin value does not matter for these special cases).

allow_mixed
: If true, use mixed compression mode; the encoder heuristically chooses between lossy and lossless for each frame.

XBM

Pillow reads and writes X bitmap files (mode 1).

Read-only formats

CUR

CUR is used to store cursors on Windows. The CUR decoder reads the largest available cursor. Animated cursors are not supported.

DCX

DCX is a container file format for PCX files, defined by Intel. The DCX format is commonly used in fax applications. The DCX decoder can read files containing `1`, `L`, `P`, or `RGB` data.

When the file is opened, only the first image is read. You can use *`seek()`* or *`ImageSequence`* to read other images.

FITS

Added in version 9.1.0.

Pillow identifies and reads FITS files, commonly used for astronomy. Uncompressed and GZIP_1 compressed images can be read.

FLI, FLC

Pillow reads Autodesk FLI and FLC animations.

The *`open()`* method sets the following *`info`* properties:

duration
: The delay (in milliseconds) between each frame.

FPX

Pillow reads Kodak FlashPix files. Only the highest resolution image is read from the file, and the viewing transform is not taken into account.

To enable FPX support, you must install olefile.

> **Note**
>
> To enable full FlashPix support, you need to build and install the IJG JPEG library before building the Python Imaging Library. See the distribution README for details.

FTEX

Added in version 3.2.0.

The FTEX decoder reads textures used for 3D objects in Independence War 2: Edge Of Chaos. The plugin reads a single texture per file, in the compressed and uncompressed formats.

GBR

The GBR decoder reads GIMP brush files, version 1 and 2.

Opening

The *open()* method sets the following *info* properties:

comment
: The brush name.

spacing
: The spacing between the brushes, in pixels. Version 2 only.

GD

Pillow reads uncompressed GD2 files. Note that you must use *PIL.GdImageFile.open()* to read such a file.

Opening

The *open()* method sets the following *info* properties:

transparency
: Transparency color index. This key is omitted if the image is not transparent.

IMT

Pillow reads Image Tools images containing L data.

IPTC/NAA

Pillow provides limited read support for IPTC/NAA newsphoto files.

MCIDAS

Pillow identifies and reads 8-bit McIdas area files.

MIC

Pillow identifies and reads Microsoft Image Composer (MIC) files. When opened, the first sprite in the file is loaded. You can use *seek()* and *tell()* to read other sprites from the file.

Note that there may be an embedded gamma of 2.2 in MIC files.

To enable MIC support, you must install olefile.

PCD

Pillow reads PhotoCD files containing `RGB` data. This only reads the 768x512 resolution image from the file. Higher resolutions are encoded in a proprietary encoding.

PIXAR

Pillow provides limited support for PIXAR raster files. The library can identify and read "dumped" RGB files.

The format code is `PIXAR`.

PSD

Pillow identifies and reads PSD files written by Adobe Photoshop 2.5 and 3.0.

SUN

Pillow identifies and reads Sun raster files.

WAL

Added in version 1.1.4.

Pillow reads Quake2 WAL texture files.

Note that this file format cannot be automatically identified, so you must use the open function in the *WalImageFile* module to read files in this format.

By default, a Quake2 standard palette is attached to the texture. To override the palette, use the *PIL.Image.Image.putpalette()* method.

WMF, EMF

Pillow can identify WMF and EMF files.

On Windows, it can read WMF and EMF files. By default, it will load the image at 72 dpi. To load it at another resolution:

```
from PIL import Image

with Image.open("drawing.wmf") as im:
    im.load(dpi=144)
```

To add other read or write support, use *PIL.WmfImagePlugin.register_handler()* to register a WMF and EMF handler.

```
from typing import IO

from PIL import Image, ImageFile
from PIL import WmfImagePlugin


class WmfHandler(ImageFile.StubHandler):
    def open(self, im: ImageFile.StubImageFile) -> None:
        ...
```

(continues on next page)

(continued from previous page)

```
    def load(self, im: ImageFile.StubImageFile) -> Image.Image:
        ...
        return image

    def save(self, im: Image.Image, fp: IO[bytes], filename: str) -> None:
        ...


wmf_handler = WmfHandler()

WmfImagePlugin.register_handler(wmf_handler)

im = Image.open("sample.wmf")
```

XPM

Pillow reads X pixmap files as P mode images if there are 256 colors or less, and as RGB images otherwise.

Opening

The *open()* method sets the following *info* properties:

transparency
: Transparency color index. This key is omitted if the image is not transparent.

XV thumbnails

Pillow can read XV thumbnail files.

Write-only formats

PALM

Pillow provides write-only support for PALM pixmap files.

The format code is `Palm`, the extension is `.palm`.

PDF

Pillow can write PDF (Acrobat) images. Such images are written as binary PDF 1.4 files. Different encoding methods are used, depending on the image mode.

- 1 mode images are saved using TIFF encoding, or JPEG encoding if libtiff support is unavailable
- L, RGB and CMYK mode images use JPEG encoding
- P mode images use HEX encoding
- LA and RGBA mode images use JPEG2000 encoding

Saving

The *save()* method can take the following keyword arguments:

save_all
: If a multiframe image is used, by default, only the first image will be saved. To save all frames, each frame to a

separate page of the PDF, the `save_all` parameter should be present and set to `True` or `append_images` should not be empty.

Added in version 3.0.0.

append_images
A list of *PIL.Image.Image* objects to append as additional pages. Each of the images in the list can be single or multiframe images.

Added in version 4.2.0.

append
Set to True to append pages to an existing PDF file. If the file doesn't exist, an `OSError` will be raised.

Added in version 5.1.0.

resolution
Image resolution in DPI. This, together with the number of pixels in the image, will determine the physical dimensions of the page that will be saved in the PDF.

dpi
A tuple of `(x_resolution, y_resolution)`, with inches as the resolution unit. If both the `resolution` parameter and the `dpi` parameter are present, `resolution` will be ignored.

title
The document's title. If not appending to an existing PDF file, this will default to the filename.

Added in version 5.1.0.

author
The name of the person who created the document.

Added in version 5.1.0.

subject
The subject of the document.

Added in version 5.1.0.

keywords
Keywords associated with the document.

Added in version 5.1.0.

creator
If the document was converted to PDF from another format, the name of the conforming product that created the original document from which it was converted.

Added in version 5.1.0.

producer
If the document was converted to PDF from another format, the name of the conforming product that converted it to PDF.

Added in version 5.1.0.

creationDate
The creation date of the document. If not appending to an existing PDF file, this will default to the current time.

Added in version 5.3.0.

modDate
The modification date of the document. If not appending to an existing PDF file, this will default to the current time.

Added in version 5.3.0.

Identify-only formats

BUFR

Added in version 1.1.3.

Pillow provides a stub driver for BUFR files.

To add read or write support to your application, use *PIL.BufrStubImagePlugin.register_handler()*.

GRIB

Added in version 1.1.5.

Pillow provides a stub driver for GRIB files.

The driver requires the file to start with a GRIB header. If you have files with embedded GRIB data, or files with multiple GRIB fields, your application has to seek to the header before passing the file handle to Pillow.

To add read or write support to your application, use *PIL.GribStubImagePlugin.register_handler()*.

HDF5

Added in version 1.1.5.

Pillow provides a stub driver for HDF5 files.

To add read or write support to your application, use *PIL.Hdf5StubImagePlugin.register_handler()*.

MPEG

Pillow identifies MPEG files.

Text anchors

The `anchor` parameter determines the alignment of drawn text relative to the `xy` parameter. The default alignment is top left, specifically `la` (left-ascender) for horizontal text and `lt` (left-top) for vertical text.

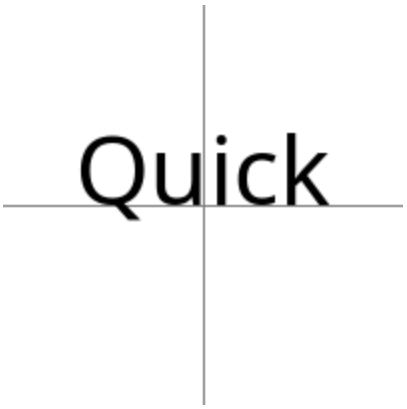
This parameter is only supported by OpenType/TrueType fonts. Other fonts may ignore the parameter and use the default (top left) alignment.

Specifying an anchor

An anchor is specified with a two-character string. The first character is the horizontal alignment, the second character is the vertical alignment. For example, the default value of `la` for horizontal text means left-ascender aligned text.

When drawing text with *PIL.ImageDraw.ImageDraw.text()* with a specific anchor, the text will be placed such that the specified anchor point is at the `xy` coordinates.

For example, in the following image, the text is `ms` (middle-baseline) aligned, with `xy` at the intersection of the two lines:




```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/NotoSans-Regular.ttf", 48)
im = Image.new("RGB", (200, 200), "white")
d = ImageDraw.Draw(im)
d.line(((0, 100), (200, 100)), "gray")
d.line(((100, 0), (100, 200)), "gray")
d.text((100, 100), "Quick", fill="black", anchor="ms", font=font)
```

Quick reference

Horizontal anchor alignment

`l` — left
: Anchor is to the left of the text.

  For *horizontal* text this is the origin of the first glyph, as shown in the FreeType tutorial.

`m` — middle
: Anchor is horizontally centered with the text.

  For *vertical* text it is recommended to use `s` (baseline) alignment instead, as it does not change based on the specific glyphs of the given text.

`r` — right
: Anchor is to the right of the text.

  For *horizontal* text this is the advanced origin of the last glyph, as shown in the FreeType tutorial.

`s` — baseline *(vertical text only)*
: Anchor is at the baseline (middle) of the text. The exact alignment depends on the font.

  For *vertical* text this is the recommended alignment, as it does not change based on the specific glyphs of the given text (see image for vertical text above).

Vertical anchor alignment

a — ascender / top ***(horizontal text only)***
Anchor is at the ascender line (top) of the first line of text, as defined by the font.

See Font metrics on Wikipedia for more information.

t — top ***(single-line text only)***
Anchor is at the top of the text.

For *vertical* text this is the origin of the first glyph, as shown in the FreeType tutorial.

For *horizontal* text it is recommended to use a (ascender) alignment instead, as it does not change based on the specific glyphs of the given text.

m — middle
Anchor is vertically centered with the text.

For *horizontal* text this is the midpoint of the first ascender line and the last descender line.

s — baseline ***(horizontal text only)***
Anchor is at the baseline (bottom) of the first line of text, only descenders extend below the anchor.

See Font metrics on Wikipedia for more information.

b — bottom ***(single-line text only)***
Anchor is at the bottom of the text.

For *vertical* text this is the advanced origin of the last glyph, as shown in the FreeType tutorial.

For *horizontal* text it is recommended to use d (descender) alignment instead, as it does not change based on the specific glyphs of the given text.

d — descender / bottom ***(horizontal text only)***
Anchor is at the descender line (bottom) of the last line of text, as defined by the font.

See Font metrics on Wikipedia for more information.

Examples

The following image shows several examples of anchors for horizontal text. In each section the xy parameter was set to the center shown by the intersection of the two lines.

Third-party plugins

Pillow uses a plugin model which allows users to add their own decoders and encoders to the library, without any changes to the library itself.

Here is a list of PyPI projects that offer additional plugins:

- amigainfo: Adds support for Amiga Workbench .info icon files.
- amos-abk: AMOS BASIC sprite and image banks.
- DjvuRleImagePlugin: Plugin for the DjVu RLE image format as defined in the DjVuLibre docs.
- heif-image-plugin: Simple HEIF/HEIC images plugin, based on the pyheif library.
- jxlpy: Introduces reading and writing support for JPEG XL.
- pillow-degas: Adds reading Atari ST Degas image files.
- pillow-heif: Python bindings to libheif for working with HEIF images.

- pillow-jpls: Plugin for the JPEG-LS codec, based on the Charls JPEG-LS implementation. Python bindings implemented using pybind11.
- pillow-jxl-plugin: Plugin for JPEG-XL, using Rust for bindings.
- pillow-mbm: Adds support for KSP's proprietary MBM texture format.
- pillow-netpbm: Adds .pam support, and loads images using Netpbm's converter collection.
- pillow-svg: Implements basic SVG read support. Supports basic paths, shapes, and text.
- raw-pillow-opener: Simple camera raw opener, based on the rawpy library.

Writing your own image plugin

Pillow uses a plugin model which allows you to add your own decoders and encoders to the library, without any changes to the library itself. Such plugins usually have names like `XxxImagePlugin.py`, where `Xxx` is a unique format name (usually an abbreviation).

> **Warning**
>
> Pillow >= 2.1.0 no longer automatically imports any file in the Python path with a name ending in `ImagePlugin.py`. You will need to import your image plugin manually.

Pillow decodes files in two stages:

1. It loops over the available image plugins in the loaded order, and calls the plugin's `_accept` function with the first 16 bytes of the file. If the `_accept` function returns true, the plugin's `_open` method is called to set up the image metadata and image tiles. The `_open` method is not for decoding the actual image data.
2. When the image data is requested, the `ImageFile.load` method is called, which sets up a decoder for each tile and feeds the data to it.

An image plugin should contain a format handler derived from the *PIL.ImageFile.ImageFile* base class. This class should provide an `_open` method, which reads the file header and set at least the internal `_size` and `_mode` attributes so that *mode* and *size* are populated. To be able to load the file, the method must also create a list of `tile` descriptors, which contain a decoder name, extents of the tile, and any decoder-specific data. The format handler class must be explicitly registered, via a call to the *Image* module.

> **Note**
>
> For performance reasons, it is important that the `_open` method quickly rejects files that do not have the appropriate contents.

Example

The following plugin supports a simple format, which has a 128-byte header consisting of the words "SPAM" followed by the width, height, and pixel size in bits. The header fields are separated by spaces. The image data follows directly after the header, and can be either bi-level, grayscale, or 24-bit true color.

SpamImagePlugin.py:

```
from PIL import Image, ImageFile


def _accept(prefix: bytes) -> bool:
```

(continues on next page)

(continued from previous page)

```
    return prefix.startswith(b"SPAM")


class SpamImageFile(ImageFile.ImageFile):

    format = "SPAM"
    format_description = "Spam raster image"

    def _open(self) -> None:

        header = self.fp.read(128).split()

        # size in pixels (width, height)
        self._size = int(header[1]), int(header[2])

        # mode setting
        bits = int(header[3])
        if bits == 1:
            self._mode = "1"
        elif bits == 8:
            self._mode = "L"
        elif bits == 24:
            self._mode = "RGB"
        else:
            msg = "unknown number of bits"
            raise SyntaxError(msg)

        # data descriptor
        self.tile = [ImageFile._Tile("raw", (0, 0) + self.size, 128, (self.mode, 0, 1))]


Image.register_open(SpamImageFile.format, SpamImageFile, _accept)

Image.register_extensions(
    SpamImageFile.format,
    [
        ".spam",
        ".spa",  # DOS version
    ],
)
```

The format handler must always set the internal `_size` and `_mode` attributes so that *`size`* and *`mode`* are populated. If these are not set, the file cannot be opened. To simplify the plugin, the calling code considers exceptions like `SyntaxError`, `KeyError`, `IndexError`, `EOFError` and `struct.error` as a failure to identify the file.

Note that the image plugin must be explicitly registered using *`PIL.Image.register_open()`*. Although not required, it is also a good idea to register any extensions used by this format.

Once the plugin has been imported, it can be used:

```
from PIL import Image
import SpamImagePlugin
```

(continues on next page)

(continued from previous page)

```
with Image.open("hopper.spam") as im:
    pass
```

The `tile` attribute

To be able to read the file as well as just identifying it, the `tile` attribute must also be set. This attribute consists of a list of tile descriptors, where each descriptor specifies how data should be loaded to a given region in the image.

In most cases, only a single descriptor is used, covering the full image. *PsdImagePlugin.PsdImageFile* uses multiple tiles to combine channels within a single layer, given that the channels are stored separately, one after the other.

The tile descriptor is a 4-tuple with the following contents:

```
(decoder, region, offset, parameters)
```

The fields are used as follows:

decoder
: Specifies which decoder to use. The `raw` decoder used here supports uncompressed data, in a variety of pixel formats. For more information on this decoder, see the description below.

  A list of C decoders can be seen under codecs section of the function array in `_imaging.c`. Python decoders are registered within the relevant plugins.

region
: A 4-tuple specifying where to store data in the image.

offset
: Byte offset from the beginning of the file to image data.

parameters
: Parameters to the decoder. The contents of this field depends on the decoder specified by the first field in the tile descriptor tuple. If the decoder doesn't need any parameters, use `None` for this field.

Note that the `tile` attribute contains a list of tile descriptors, not just a single descriptor.

Decoders

The raw decoder

The `raw` decoder is used to read uncompressed data from an image file. It can be used with most uncompressed file formats, such as PPM, BMP, uncompressed TIFF, and many others. To use the raw decoder with the *PIL.Image.frombytes()* function, use the following syntax:

```
image = Image.frombytes(
    mode, size, data, "raw",
    raw_mode, stride, orientation
    )
```

When used in a tile descriptor, the parameter field should look like:

```
(raw_mode, stride, orientation)
```

The fields are used as follows:

raw_mode
: The pixel layout used in the file, and is used to properly convert data to PIL's internal layout. For a summary of the available formats, see the table below.

stride
: The distance in bytes between two consecutive lines in the image. If 0, the image is assumed to be packed (no padding between lines). If omitted, the stride defaults to 0.

orientation
: Whether the first line in the image is the top line on the screen (1), or the bottom line (-1). If omitted, the orientation defaults to 1.

The **raw mode** field is used to determine how the data should be unpacked to match PIL's internal pixel layout. PIL supports a large set of raw modes; for a complete list, see the table in the `Unpack.c` module. The following table describes some commonly used **raw modes**:

mode	description
`1`	1-bit bilevel, stored with the leftmost pixel in the most significant bit. 0 means black, 1 means white.
`1;I`	1-bit inverted bilevel, stored with the leftmost pixel in the most significant bit. 0 means white, 1 means black.
`1;R`	1-bit reversed bilevel, stored with the leftmost pixel in the least significant bit. 0 means black, 1 means white.
`L`	8-bit grayscale. 0 means black, 255 means white.
`L;I`	8-bit inverted grayscale. 0 means white, 255 means black.
`P`	8-bit palette-mapped image.
`RGB`	24-bit true colour, stored as (red, green, blue).
`BGR`	24-bit true colour, stored as (blue, green, red).
`RGBX`	24-bit true colour, stored as (red, green, blue, pad). The pad pixels may vary.
`RGB;L`	24-bit true colour, line interleaved (first all red pixels, then all green pixels, finally all blue pixels).

Note that for the most common cases, the raw mode is simply the same as the mode.

The Python Imaging Library supports many other decoders, including JPEG, PNG, and PackBits. For details, see the `decode.c` source file, and the standard plugin implementations provided with the library.

Decoding floating point data

PIL provides some special mechanisms to allow you to load a wide variety of formats into a mode `F` (floating point) image memory.

You can use the `raw` decoder to read images where data is packed in any standard machine data type, using one of the following raw modes:

mode	description
F	32-bit native floating point.
F;8	8-bit unsigned integer.
F;8S	8-bit signed integer.
F;16	16-bit little endian unsigned integer.
F;16S	16-bit little endian signed integer.
F;16B	16-bit big endian unsigned integer.
F;16BS	16-bit big endian signed integer.
F;16N	16-bit native unsigned integer.
F;16NS	16-bit native signed integer.
F;32	32-bit little endian unsigned integer.
F;32S	32-bit little endian signed integer.
F;32B	32-bit big endian unsigned integer.
F;32BS	32-bit big endian signed integer.
F;32N	32-bit native unsigned integer.
F;32NS	32-bit native signed integer.
F;32F	32-bit little endian floating point.
F;32BF	32-bit big endian floating point.
F;32NF	32-bit native floating point.
F;64F	64-bit little endian floating point.
F;64BF	64-bit big endian floating point.
F;64NF	64-bit native floating point.

The bit decoder

If the raw decoder cannot handle your format, PIL also provides a special "bit" decoder that can be used to read various packed formats into a floating point image memory.

To use the bit decoder with the *PIL.Image.frombytes()* function, use the following syntax:

```
image = Image.frombytes(
    mode, size, data, "bit",
    bits, pad, fill, sign, orientation
    )
```

When used in a tile descriptor, the parameter field should look like:

```
(bits, pad, fill, sign, orientation)
```

The fields are used as follows:

bits
: Number of bits per pixel (2-32). No default.

pad
: Padding between lines, in bits. This is either 0 if there is no padding, or 8 if lines are padded to full bytes. If omitted, the pad value defaults to 8.

fill
: Controls how data are added to, and stored from, the decoder bit buffer.

fill=0
: Add bytes to the LSB end of the decoder buffer; store pixels from the MSB end.

fill=1
: Add bytes to the MSB end of the decoder buffer; store pixels from the MSB end.

fill=2
: Add bytes to the LSB end of the decoder buffer; store pixels from the LSB end.

fill=3
: Add bytes to the MSB end of the decoder buffer; store pixels from the LSB end.

If omitted, the fill order defaults to 0.

sign
: If non-zero, bit fields are sign extended. If zero or omitted, bit fields are unsigned.

orientation
: Whether the first line in the image is the top line on the screen (1), or the bottom line (-1). If omitted, the orientation defaults to 1.

Writing your own file codec in C

There are 3 stages in a file codec's lifetime:

1. Setup: Pillow looks for a function in the decoder or encoder registry, falling back to a function named `[codecname]_decoder` or `[codecname]_encoder` on the internal core image object. That function is called with the `args` tuple from the `tile`.
2. Transforming: The codec's `decode` or `encode` function is repeatedly called with chunks of image data.
3. Cleanup: If the codec has registered a cleanup function, it will be called at the end of the transformation process, even if there was an exception raised.

Setup

The current conventions are that the codec setup function is named `PyImaging_[codecname]DecoderNew` or `PyImaging_[codecname]EncoderNew` and defined in `decode.c` or `encode.c`. The Python binding for it is named `[codecname]_decoder` or `[codecname]_encoder` and is set up from within the `_imaging.c` file in the codecs section of the function array.

The setup function needs to call `PyImaging_DecoderNew` or `PyImaging_EncoderNew` and at the very least, set the `decode` or `encode` function pointer. The fields of interest in this object are:

decode/encode
: Function pointer to the decode or encode function, which has access to `im`, `state`, and the buffer of data to be transformed.

cleanup
: Function pointer to the cleanup function, has access to `state`.

im
: The target image, will be set by Pillow.

state
: An ImagingCodecStateInstance, will be set by Pillow. The `context` member is an opaque struct that can be used by the codec to store any format specific state or options.

pulls_fd/pushes_fd
: If the decoder has `pulls_fd` or the encoder has `pushes_fd` set to 1, `state->fd` will be a pointer to the Python file like object. The codec may use the functions in `codec_fd.c` to read or write directly with the file like object rather than have the data pushed through a buffer.

  Added in version 3.3.0.

Transforming

The decode or encode function is called with the target (core) image, the codec state structure, and a buffer of data to be transformed.

It is the codec's responsibility to pull as much data as possible out of the buffer and return the number of bytes consumed. The next call to the codec will include the previous unconsumed tail. The codec function will be called multiple times as the data is processed.

Alternatively, if `pulls_fd` or `pushes_fd` is set, then the decode or encode function is called once, with an empty buffer. It is the codec's responsibility to transform the entire tile in that one call. Using this will provide a codec with more freedom, but that freedom may mean increased memory usage if the entire tile is held in memory at once by the codec.

If an error occurs, set `state->errcode` and return -1.

Return -1 on success, without setting the errcode.

Cleanup

The cleanup function is called after the codec returns a negative value, or if there is an error. This function should free any allocated memory and release any resources from external libraries.

Writing your own file codec in Python

Python file decoders and encoders should derive from *PIL.ImageFile.PyDecoder* and *PIL.ImageFile.PyEncoder* respectively, and should at least override the decode or encode method. They should be registered using *PIL.Image.register_decoder()* and *PIL.Image.register_encoder()*. As in the C implementation of the file codecs, there are three stages in the lifetime of a Python-based file codec:

1. Setup: Pillow looks for the codec in the decoder or encoder registry, then instantiates the class.
2. Transforming: The instance's `decode` method is repeatedly called with a buffer of data to be interpreted, or the `encode` method is repeatedly called with the size of data to be output.

   Alternatively, if the decoder's `_pulls_fd` property (or the encoder's `_pushes_fd` property) is set to `True`, then `decode` and `encode` will only be called once. In the decoder, `self.fd` can be used to access the file-like object. Using this will provide a codec with more freedom, but that freedom may mean increased memory usage if entire file is held in memory at once by the codec.

   In `decode`, once the data has been interpreted, `set_as_raw` can be used to populate the image.
3. Cleanup: The instance's `cleanup` method is called once the transformation is complete. This can be used to clean up any resources used by the codec.

   If you set `_pulls_fd` or `_pushes_fd` to `True` however, then you probably chose to perform any cleanup tasks at the end of `decode` or `encode`.

For an example *PIL.ImageFile.PyDecoder*, see DdsImagePlugin. For a plugin that uses both *PIL.ImageFile.PyDecoder* and *PIL.ImageFile.PyEncoder*, see BlpImagePlugin

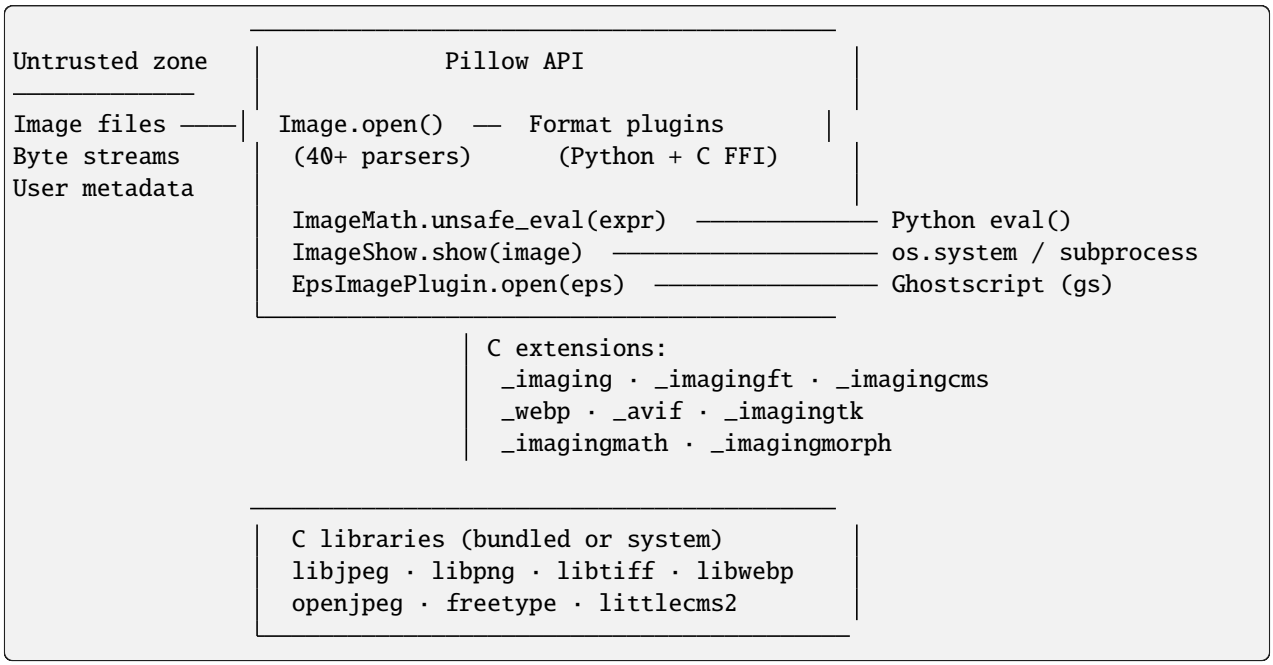
1.2.5 Security

Pillow's primary attack surface is **parsing untrusted image data**. This page documents the threat model for developers integrating Pillow into applications that handle images from untrusted sources, along with recommended mitigations.

To report a vulnerability see *Reporting a vulnerability*.

Threat model (STRIDE)

The analysis below follows the STRIDE framework and covers the boundary between untrusted image input and the Pillow API.



Spoofing

S-1 — Format sniffing bypass

`Image.open()` detects format by magic bytes, not file extension or MIME type. An attacker can name a file `safe.png` while its content is TIFF, JPEG 2000, or EPS, causing a different — potentially more dangerous — parser to run.

Mitigations: validate MIME type and magic bytes independently before calling `Image.open()`; pass the `formats` argument with an allowlist of accepted formats.

S-2 — Plugin registry spoofing

Pillow's format registry is a global mutable dictionary. A malicious package installed in the same environment could register a replacement parser for a well-known format.

Mitigations: use isolated virtual environments with pinned, hash-verified dependencies; audit `Image.registered_extensions()` at startup.

Tampering

T-1 — Malicious metadata propagation

Pillow preserves EXIF, XMP, IPTC, ICC profiles, and comments when round-tripping images. Applications that store or render metadata without sanitisation are vulnerable to second-order injection (SQLi, XSS, command injection).

Mitigations: treat all values from `image.info`, `image._getexif()`, `image.getexif()`, and `image.text` as untrusted; sanitise before storing or rendering; strip metadata when it is not required.

T-2 — Covert data channel (steganography)

Pillow does not remove hidden data (JPEG comments, PNG text chunks) when re-saving. An attacker can embed data that survives the encode-decode cycle invisibly.

Mitigations: to guarantee a clean output when saving, create a new image instance via `image.copy()` and delete the `image.info` contents.

T-3 — Supply chain tampering

Pre-compiled wheels bundle libjpeg-turbo, libpng, libtiff, libwebp, openjpeg, freetype, littlecms2, and other libraries. A compromised PyPI release or build pipeline could ship malicious binaries.

Mitigations: pin with hash verification (`python3 -m pip install --require-hashes`); monitor Pillow security advisories; use Dependabot or OSV-Scanner for bundled C library CVEs.

Repudiation

R-1 — No structured audit trail

Without application-level logging there is no record of which images were opened, what formats were detected, or what operations were performed, making forensic investigation harder after an incident.

Mitigations: log the filename/hash, detected format, and dimensions of every image processed; log and alert on `Image.DecompressionBombWarning`, `Image.DecompressionBombError`, and `PIL.UnidentifiedImageError`.

Information disclosure

I-1 — Metadata in saved images

GPS coordinates, author names, software version strings, and ICC profiles can be inadvertently included in output images served publicly.

Mitigations: explicitly strip EXIF and XMP on save (set `exif=b""`, `icc_profile=None`, omit `pnginfo`); verify output with `exiftool` in CI.

I-2 — Temporary file exposure

Several code paths write pixel data to temporary files via `tempfile.mkstemp()`. Exception paths can leave these files behind on shared filesystems.

Mitigations: files are created with mode `0o600`; mount `/tmp` as a per-container `tmpfs`; ensure `try/finally` cleanup is in place.

Denial of service

D-1 — Decompression bomb

A small compressed image can expand to gigabytes in memory. *PIL.Image.MAX_IMAGE_PIXELS* raises `Image.DecompressionBombError` at 2× the limit and `Image.DecompressionBombWarning` at 1×. PNG text chunks are separately capped by `PngImagePlugin.MAX_TEXT_CHUNK` and `MAX_TEXT_MEMORY`. Check the values in your installed Pillow version at runtime or in the reference/source for the current defaults.

Mitigations: **never** set `Image.MAX_IMAGE_PIXELS = None` in production; treat `Image.DecompressionBombWarning` as an error; set OS/container memory limits per worker.

D-2 — CPU exhaustion

Large-but-legal images (within `MAX_IMAGE_PIXELS`) can still saturate CPU through high-quality resampling, convolution filters, or complex draw operations.

Mitigations: apply per-request CPU time limits; set a practical dimension ceiling below `MAX_IMAGE_PIXELS`; rate-limit processing requests.

D-3 — Algorithmic complexity in parsers

Formats such as TIFF (nested IFD chains), animated GIF/WebP (many frames), and PNG (many text chunks) can exhaust CPU or memory before pixel data is decoded.

Mitigations: restrict accepted formats to the minimum required; enforce a file-size limit before passing data to Pillow; use per-request timeouts.

Elevation of privilege

E-1 — C extension memory corruption (RCE)

Pillow's ~87 C source files and its bundled C libraries process attacker-controlled bytes. Historical CVEs include buffer overflows, integer overflows, and use-after-free vulnerabilities that allow arbitrary code execution.

Mitigations: keep Pillow and all C libraries up to date; compile with hardening flags (ASLR, stack canaries, PIE, `_FORTIFY_SOURCE=2`); run image processing in a sandboxed subprocess (seccomp-bpf, AppArmor, or a restricted container).

E-2 — Ghostscript exploitation via EPS (RCE)

Opening an EPS file invokes the system Ghostscript binary (`gs`) via `subprocess`. Ghostscript has a long history of sandbox-escape CVEs permitting arbitrary code execution from malicious PostScript.

Mitigations: **block EPS files** at the application input layer before passing files to Pillow; if EPS must be supported, run Ghostscript in a fully isolated sandbox with no network and no sensitive mounts. Pillow does not provide a stable public API for unregistering individual format plugins, so do not rely on mutating internal registries such as `Image.OPEN` as a security control.

E-3 — ImageMath.unsafe_eval() code injection

unsafe_eval() calls Python's built-in `eval()` with only a minimal `__builtins__` restriction, which can be bypassed via introspection. Any user-controlled string passed to this function results in arbitrary code execution.

Mitigations: **never** pass user-controlled strings to `ImageMath.unsafe_eval()`; use *lambda_eval()* instead, which accepts a Python callable and never calls `eval`.

E-4 — Font path traversal via ImageFont

`ImageFont.truetype(font, size)` passes the filename to the FreeType C library. If font paths are constructed from user input without canonicalisation, an attacker may supply a path like `../../../../etc/passwd`.

Mitigations: never construct font paths from user input; if font selection must be user-driven, resolve names against an explicit allowlist of pre-validated absolute paths.

Recommendations

The following mitigations are listed in priority order.

1. **Sandbox image processing** — run Pillow workers in a seccomp/AppArmor restricted subprocess, isolated from the main application process.
2. **Block or sandbox EPS** — reject EPS at the application boundary, or run Ghostscript in an isolated container.
3. **Never use `ImageMath.unsafe_eval()` with user input** — migrate all callers to *lambda_eval()*.
4. **Keep all dependencies current** — Pillow and its C library dependencies (including libjpeg, libpng, libtiff, libwebp, openjpeg, freetype, littlecms2, Ghostscript, and others). Subscribe to Pillow security advisories.
5. **Enforce `MAX_IMAGE_PIXELS`** — never set it to `None`; treat `Image.DecompressionBombWarning` as an error.
6. **Allowlist image formats** — restrict accepted formats when opening images, for example with `Image.open(.., formats=...)`, and isolate installs/environments if you need to minimise supported formats.
7. **Strip metadata on output** — never pass through EXIF/XMP/ICC from user uploads to publicly served images.
8. **Sanitise all metadata** returned by Pillow before using it downstream.
9. **Pin dependencies with hash verification** — use `pip install --require-hashes` and lockfiles.

10. **Log and alert** on `Image.DecompressionBombWarning`, `Image.DecompressionBombError`, `PIL.UnidentifiedImageError`, and all exceptions from `Image.open()`.

Reporting a vulnerability

To report sensitive vulnerability information, report it privately on GitHub.

If you cannot use GitHub, use the Tidelift security contact. Tidelift will coordinate the fix and disclosure.

Do not report sensitive vulnerability information in public.

Additionally:

1. Please ensure that your issue is reproducable in **main**. We only support the latest version. The one exception – if your issue is exploitable in the latest public release, but not in main, and the pull request does not mention a security implication this may be an unknown security issue that was inadvertely fixed.
2. Demonstrating a memory overflow is enough. Please do not weaponize the reproducer to do remote code execution.
3. Please do not report unexpected Python exceptions as a DoS or a memory safety bug. An issue that raises a Python exception in a library is unlikely to be considered a security issue. This may or may not be an ordinary bug depending on the context.

1.3 Reference

1.3.1 *Image* module

The *Image* module provides a class with the same name which is used to represent a PIL image. The module also provides a number of factory functions, including functions to load images from files, and to create new images.

Examples

Open, rotate, and display an image (using the default viewer)

The following script loads an image, rotates it 45 degrees, and displays it using an external viewer (usually xv on Unix, and the Paint program on Windows).

```
from PIL import Image
with Image.open("hopper.jpg") as im:
    im.rotate(45).show()
```

Create thumbnails

The following script creates nice thumbnails of all JPEG images in the current directory preserving aspect ratios with 128x128 max resolution.

```
from PIL import Image
import glob, os

size = 128, 128

for infile in glob.glob("*.jpg"):
    file, ext = os.path.splitext(infile)
    with Image.open(infile) as im:
        im.thumbnail(size)
        im.save(file + ".thumbnail", "JPEG")
```

Functions

PIL.Image.**open**(*fp:* StrOrBytesPath | *IO[bytes]*, *mode: Literal['r'] = 'r'*, *formats: list[str] | tuple[str, ...] | None = None*) → *ImageFile.ImageFile*

Opens and identifies the given image file.

This is a lazy operation; this function identifies the file, but the file remains open and the actual image data is not read from the file until you try to process the data (or call the *load()* method). See *new()*. See *File handling in Pillow*.

Parameters

- **fp** – A filename (string), os.PathLike object or a file object. The file object must implement `file.read`, `file.seek`, and `file.tell` methods, and be opened in binary mode. The file object will also seek to zero before reading.
- **mode** – The mode. If given, this argument must be "r".
- **formats** – A list or tuple of formats to attempt to load the file in. This can be used to restrict the set of formats checked. Pass `None` to try all supported formats. You can print the set of available formats by running `python3 -m PIL` or using the *PIL.features.pilinfo()* function.

Returns

An *Image* object.

Raises

- **FileNotFoundError** – If the file cannot be found.
- ***PIL.UnidentifiedImageError*** – If the image cannot be opened and identified.
- **ValueError** – If the `mode` is not "r", or if a `StringIO` instance is used for `fp`.
- **TypeError** – If `formats` is not `None`, a list or a tuple.

> **Warning**
>
> To protect against potential DOS attacks caused by "decompression bombs" (i.e. malicious files which decompress into a huge amount of data and are designed to crash or cause disruption by using up a lot of memory), Pillow will issue a `DecompressionBombWarning` if the number of pixels in an image is over a certain limit, *MAX_IMAGE_PIXELS*.
>
> This threshold can be changed by setting *MAX_IMAGE_PIXELS*. It can be disabled by setting `Image.MAX_IMAGE_PIXELS = None`.
>
> If desired, the warning can be turned into an error with `warnings.simplefilter('error', Image.DecompressionBombWarning)` or suppressed entirely with `warnings.simplefilter('ignore', Image.DecompressionBombWarning)`. See also the logging documentation to have warnings output to the logging facility instead of stderr.
>
> If the number of pixels is greater than twice *MAX_IMAGE_PIXELS*, then a `DecompressionBombError` will be raised instead.

Image processing

PIL.Image.**alpha_composite**(*im1:* Image, *im2:* Image) → *Image*

Alpha composite im2 over im1.

Parameters

- **im1** – The first image. Must have mode RGBA or LA.
- **im2** – The second image. Must have the same mode and size as the first image.

Returns
An *Image* object.

PIL.Image.**blend**(*im1:* Image, *im2:* Image, *alpha: float*) → *Image*

Creates a new image by interpolating between two input images, using a constant alpha:

```
out = image1 * (1.0 - alpha) + image2 * alpha
```

Parameters

- **im1** – The first image.
- **im2** – The second image. Must have the same mode and size as the first image.
- **alpha** – The interpolation alpha factor. If alpha is 0.0, a copy of the first image is returned. If alpha is 1.0, a copy of the second image is returned. There are no restrictions on the alpha value. If necessary, the result is clipped to fit into the allowed output range.

Returns
An *Image* object.

PIL.Image.**composite**(*image1:* Image, *image2:* Image, *mask:* Image) → *Image*

Create composite image by blending images using a transparency mask.

Parameters

- **image1** – The first image.
- **image2** – The second image. Must have the same mode and size as the first image.
- **mask** – A mask image. This image can have mode "1", "L", or "RGBA", and must have the same size as the other two images.

PIL.Image.**eval**(*image:* Image, **args: Callable[[int], float]*) → *Image*

Applies the function (which should take one argument) to each pixel in the given image. If the image has more than one band, the same function is applied to each band. Note that the function is evaluated once for each possible pixel value, so you cannot use random components or other generators.

Parameters

- **image** – The input image.
- **function** – A function object, taking one integer argument.

Returns
An *Image* object.

PIL.Image.**merge**(*mode: str*, *bands: Sequence[*Image*]*) → *Image*

Merge a set of single band images into a new multiband image.

Parameters

- **mode** – The mode to use for the output image. See: *Modes*.
- **bands** – A sequence containing one single-band image for each band in the output image. All bands must have the same size.

Returns
An *Image* object.

Constructing images

PIL.Image.**new**(*mode: str*, *size: tuple[int, int] | list[int]*, *color: float | tuple[float, ...] | str | None = 0*) → *Image*

Creates a new image with the given mode and size.

Parameters

- **mode** – The mode to use for the new image. See: *Modes*.
- **size** – A 2-tuple, containing (width, height) in pixels.
- **color** – What color to use for the image. Default is black. If given, this should be a single integer or floating point value for single-band modes, and a tuple for multi-band modes (one value per band). When creating RGB or HSV images, you can also use color strings as supported by the ImageColor module. See *Colors* for more information. If the color is None, the image is not initialised.

Returns

An *Image* object.

PIL.Image.**fromarray**(*obj: SupportsArrayInterface*, *mode: str | None = None*) → *Image*

Creates an image memory from an object exporting the array interface (using the buffer protocol):

```
from PIL import Image
import numpy as np
a = np.zeros((5, 5))
im = Image.fromarray(a)
```

If `obj` is not contiguous, then the `tobytes` method is called and *frombuffer()* is used.

In the case of NumPy, be aware that Pillow modes do not always correspond to NumPy dtypes. Pillow modes only offer 1-bit pixels, 8-bit pixels, 32-bit signed integer pixels, and 32-bit floating point pixels.

Pillow images can also be converted to arrays:

```
from PIL import Image
import numpy as np
im = Image.open("hopper.jpg")
a = np.asarray(im)
```

When converting Pillow images to arrays however, only pixel values are transferred. This means that P and PA mode images will lose their palette.

Parameters

- **obj** – Object with array interface
- **mode** – Optional mode to use when reading `obj`. Since pixel values do not contain information about palettes or color spaces, this can be used to place grayscale L mode data within a P mode image, or read RGB data as YCbCr for example.

  See: *Modes* for general information about modes.

Returns

An image object.

Added in version 1.1.6.

PIL.Image.**fromarrow**(*obj: SupportsArrowArrayInterface*, *mode: str*, *size: tuple[int, int]*) → *Image*

Creates an image with zero-copy shared memory from an object exporting the arrow_c_array interface protocol:

```
from PIL import Image
import pyarrow as pa
arr = pa.array([0]*(5*5*4), type=pa.uint8())
im = Image.fromarrow(arr, 'RGBA', (5, 5))
```

If the data representation of the `obj` is not compatible with Pillow internal storage, a ValueError is raised.

Pillow images can also be converted to Arrow objects:

```
from PIL import Image
import pyarrow as pa
im = Image.open('hopper.jpg')
arr = pa.array(im)
```

As with array support, when converting Pillow images to arrays, only pixel values are transferred. This means that P and PA mode images will lose their palette.

Parameters

- **obj** – Object with an arrow_c_array interface
- **mode** – Image mode.
- **size** – Image size. This must match the storage of the arrow object.

Returns

An Image object

Note that according to the Arrow spec, both the producer and the consumer should consider the exported array to be immutable, as unsynchronized updates will potentially cause inconsistent data.

See: *Arrow support* for more detailed information

Added in version 11.2.1.

PIL.Image.**frombytes**(*mode: str*, *size: tuple[int, int]*, *data:* DecoderInput, *decoder_name: str = 'raw'*, **args: Any*) → *Image*

Creates a copy of an image memory from pixel data in a buffer.

In its simplest form, this function takes three arguments (mode, size, and unpacked pixel data).

You can also use any pixel decoder supported by PIL. For more information on available decoders, see the section *Writing Your Own File Codec*.

Note that this function decodes pixel data only, not entire images. If you have an entire image in a string, wrap it in a `BytesIO` object, and use *open()* to load it.

Parameters

- **mode** – The image mode. See: *Modes*.
- **size** – The image size.
- **data** – A byte buffer containing raw data for the given mode.
- **decoder_name** – What decoder to use.
- **args** – Additional parameters for the given decoder.

Returns

An *Image* object.

PIL.Image.**frombuffer**(*mode: str*, *size: tuple[int, int]*, *data: bytes | SupportsArrayInterface*, *decoder_name: str = 'raw'*, **args: Any*) → *Image*

Creates an image memory referencing pixel data in a byte buffer.

This function is similar to *frombytes()*, but uses data in the byte buffer, where possible. This means that changes to the original buffer object are reflected in this image). Not all modes can share memory; supported modes include "L", "RGBX", "RGBA", and "CMYK".

Note that this function decodes pixel data only, not entire images. If you have an entire image file in a string, wrap it in a BytesIO object, and use *open()* to load it.

The default parameters used for the "raw" decoder differs from that used for *frombytes()*. This is a bug, and will probably be fixed in a future release. The current release issues a warning if you do this; to disable the warning, you should provide the full set of parameters. See below for details.

Parameters

- **mode** – The image mode. See: *Modes*.
- **size** – The image size.
- **data** – A bytes or other buffer object containing raw data for the given mode.
- **decoder_name** – What decoder to use.
- **args** – Additional parameters for the given decoder. For the default encoder ("raw"), it's recommended that you provide the full set of parameters:

```
frombuffer(mode, size, data, "raw", mode, 0, 1)
```

Returns

An *Image* object.

Added in version 1.1.4.

Generating images

PIL.Image.**effect_mandelbrot**(*size: tuple[int, int]*, *extent: tuple[float, float, float, float]*, *quality: int*) → *Image*

Generate a Mandelbrot set covering the given extent.

Parameters

- **size** – The requested size in pixels, as a 2-tuple: (width, height).
- **extent** – The extent to cover, as a 4-tuple: (x0, y0, x1, y1).
- **quality** – Quality.

PIL.Image.**effect_noise**(*size: tuple[int, int]*, *sigma: float*) → *Image*

Generate Gaussian noise centered around 128.

Parameters

- **size** – The requested size in pixels, as a 2-tuple: (width, height).
- **sigma** – Standard deviation of noise.

PIL.Image.**linear_gradient**(*mode: str*) → *Image*

Generate 256x256 linear gradient from black to white, top to bottom.

Parameters

mode – Input mode.

PIL.Image.**radial_gradient**(*mode: str*) → *Image*

Generate 256x256 radial gradient from black to white, centre to edge.

Parameters
mode – Input mode.

Registering plugins

PIL.Image.**preinit**() → None

Explicitly loads BMP, GIF, JPEG, PPM and PNG file format drivers.

It is called when opening or saving images.

PIL.Image.**init**() → bool

Explicitly initializes the Python Imaging Library. This function loads all available file format drivers.

It is called when opening or saving images if *preinit()* is insufficient, and by *pilinfo()*.

Note

These functions are for use by plugin authors. They are called when a plugin is loaded as part of *preinit()* or *init()*. Application authors can ignore them.

PIL.Image.**register_open**(*id: str*, *factory: Callable[[IO[bytes], str | bytes], ImageFile.ImageFile] | type[ImageFile.ImageFile]*, *accept: Callable[[bytes], bool | str] | None = None*) → None

Register an image file plugin. This function should not be used in application code.

Parameters

- **id** – An image format identifier.
- **factory** – An image file factory method.
- **accept** – An optional function that can be used to quickly reject images having another format.

PIL.Image.**register_mime**(*id: str*, *mimetype: str*) → None

Registers an image MIME type by populating `Image.MIME`. This function should not be used in application code.

`Image.MIME` provides a mapping from image format identifiers to mime formats, but *get_format_mimetype()* can provide a different result for specific images.

Parameters

- **id** – An image format identifier.
- **mimetype** – The image MIME type for this format.

PIL.Image.**register_save**(*id: str*, *driver: Callable[[Image, IO[bytes], str | bytes], None]*) → None

Registers an image save function. This function should not be used in application code.

Parameters

- **id** – An image format identifier.
- **driver** – A function to save images in this format.

PIL.Image.**register_save_all**(*id: str*, *driver: Callable[[*Image, *IO[bytes], str | bytes], None]*) → None

Registers an image function to save all the frames of a multiframe format. This function should not be used in application code.

Parameters

- **id** – An image format identifier.
- **driver** – A function to save images in this format.

PIL.Image.**register_extension**(*id: str*, *extension: str*) → None

Registers an image extension. This function should not be used in application code.

Parameters

- **id** – An image format identifier.
- **extension** – An extension used for this format.

PIL.Image.**register_extensions**(*id: str*, *extensions: list[str]*) → None

Registers image extensions. This function should not be used in application code.

Parameters

- **id** – An image format identifier.
- **extensions** – A list of extensions used for this format.

PIL.Image.**registered_extensions**() → dict[str, str]

Returns a dictionary containing all file extensions belonging to registered plugins

PIL.Image.**register_decoder**(*name: str*, *decoder: type[*ImageFile.PyDecoder*]*) → None

Registers an image decoder. This function should not be used in application code.

Parameters

- **name** – The name of the decoder
- **decoder** – An ImageFile.PyDecoder object

Added in version 4.1.0.

PIL.Image.**register_encoder**(*name: str*, *encoder: type[*ImageFile.PyEncoder*]*) → None

Registers an image encoder. This function should not be used in application code.

Parameters

- **name** – The name of the encoder
- **encoder** – An ImageFile.PyEncoder object

Added in version 4.1.0.

The Image class

class PIL.Image.**Image**

This class represents an image object. To create *Image* objects, use the appropriate factory functions. There's hardly ever any reason to call the Image constructor directly.

- *open()*
- *new()*
- *frombytes()*

An instance of the *Image* class has the following methods. Unless otherwise stated, all methods return a new instance of the *Image* class, holding the resulting image.

Image.**alpha_composite**(*im:* Image, *dest: Sequence[int] = (0, 0)*, *source: Sequence[int] = (0, 0)*) → None

'In-place' analog of Image.alpha_composite. Composites an image onto this image.

Parameters

- **im** – image to composite over this one
- **dest** – Optional 2 tuple (left, top) specifying the upper left corner in this (destination) image.
- **source** – Optional 2 (left, top) tuple for the upper left corner in the overlay source image, or 4 tuple (left, top, right, bottom) for the bounds of the source rectangle

Performance Note: Not currently implemented in-place in the core layer.

Image.**apply_transparency**() → None

If a P mode image has a "transparency" key in the info dictionary, remove the key and instead apply the transparency to the palette. Otherwise, the image is unchanged.

Image.**convert**(*mode: str | None = None*, *matrix: tuple[float, ...] | None = None*, *dither:* Dither | *None = None*, *palette:* Palette *= Palette.WEB*, *colors: int = 256*) → *Image*

Returns a converted copy of this image. For the "P" mode, this method translates pixels through the palette. If mode is omitted, a mode is chosen so that all information in the image and the palette can be represented without a palette.

This supports all possible conversions between "L", "RGB" and "CMYK". The `matrix` argument only supports "L" and "RGB".

When translating a color image to grayscale (mode "L"), the library uses the ITU-R 601-2 luma transform:

```
L = R * 299/1000 + G * 587/1000 + B * 114/1000
```

The default method of converting a grayscale ("L") or "RGB" image into a bilevel (mode "1") image uses Floyd-Steinberg dither to approximate the original image luminosity levels. If dither is `None`, all values larger than 127 are set to 255 (white), all other values to 0 (black). To use other thresholds, use the *point()* method.

When converting from "RGBA" to "P" without a `matrix` argument, this passes the operation to *quantize()*, and `dither` and `palette` are ignored.

When converting from "PA", if an "RGBA" palette is present, the alpha channel from the image will be used instead of the values from the palette.

Parameters

- **mode** – The requested mode. See: *Modes*.
- **matrix** – An optional conversion matrix. If given, this should be 4- or 12-tuple containing floating point values.
- **dither** – Dithering method, used when converting from mode "RGB" to "P" or from "RGB" or "L" to "1". Available methods are *Dither.NONE* or *Dither.FLOYDSTEINBERG* (default). Note that this is not used when `matrix` is supplied.
- **palette** – Palette to use when converting from mode "RGB" to "P". Available palettes are *Palette.WEB* or *Palette.ADAPTIVE*.
- **colors** – Number of colors to use for the *Palette.ADAPTIVE* palette. Defaults to 256.

Return type
Image

Returns
An *Image* object.

The following example converts an RGB image (linearly calibrated according to ITU-R 709, using the D65 luminant) to the CIE XYZ color space:

```
rgb2xyz = (
    0.412453, 0.357580, 0.180423, 0,
    0.212671, 0.715160, 0.072169, 0,
    0.019334, 0.119193, 0.950227, 0)
out = im.convert("RGB", rgb2xyz)
```

Image.**copy**() → *Image*

Copies this image. Use this method if you wish to paste things into an image, but still retain the original.

Return type
Image

Returns
An *Image* object.

Image.**crop**(*box: tuple[float, float, float, float] | None = None*) → *Image*

Returns a rectangular region from this image. The box is a 4-tuple defining the left, upper, right, and lower pixel coordinate. See *Coordinate system*.

Note: Prior to Pillow 3.4.0, this was a lazy operation.

Parameters
box – The crop rectangle, as a (left, upper, right, lower)-tuple.

Return type
Image

Returns
An *Image* object.

This crops the input image with the provided coordinates:

```
from PIL import Image

with Image.open("hopper.jpg") as im:

    # The crop method from the Image module takes four coordinates as input.
    # The right can also be represented as (left+width)
    # and lower can be represented as (upper+height).
    (left, upper, right, lower) = (20, 20, 100, 100)

    # Here the image "im" is cropped and assigned to new variable im_crop
    im_crop = im.crop((left, upper, right, lower))
```

Image.**draft**(*mode: str | None, size: tuple[int, int] | None*) → tuple[str, tuple[int, int, float, float]] | None

Configures the image file loader so it returns a version of the image that as closely as possible matches the given mode and size. For example, you can use this method to convert a color JPEG to grayscale while loading it.

If any changes are made, returns a tuple with the chosen `mode` and `box` with coordinates of the original image within the altered one.

Note that this method modifies the *Image* object in place. If the image has already been loaded, this method has no effect.

Note: This method is not implemented for most images. It is currently implemented only for JPEG and MPO images.

Parameters

- **mode** – The requested mode.
- **size** – The requested size in pixels, as a 2-tuple: (width, height).

Image.**effect_spread**(*distance: int*) → *Image*

Randomly spread pixels in an image.

Parameters
distance – Distance to spread pixels.

Image.**entropy**(*mask:* Image | *None* = *None*, *extrema: tuple[float, float]* | *None* = *None*) → float

Calculates and returns the entropy for the image.

A bilevel image (mode "1") is treated as a grayscale ("L") image by this method.

If a mask is provided, the method employs the histogram for those parts of the image where the mask image is non-zero. The mask image must have the same size as the image, and be either a bi-level image (mode "1") or a grayscale image ("L").

Parameters

- **mask** – An optional mask.
- **extrema** – An optional tuple of manually-specified extrema.

Returns
A float value representing the image entropy

Image.**filter**(*filter:* ImageFilter.Filter | *type[*ImageFilter.Filter*]*) → *Image*

Filters this image using the given filter. For a list of available filters, see the *ImageFilter* module.

Parameters
filter – Filter kernel.

Returns
An *Image* object.

This blurs the input image using a filter from the `ImageFilter` module:

```
from PIL import Image, ImageFilter

with Image.open("hopper.jpg") as im:

    # Blur the input image using the filter ImageFilter.BLUR
    im_blurred = im.filter(filter=ImageFilter.BLUR)
```

Image.**frombytes**(*data:* DecoderInput, *decoder_name: str* = *'raw'*, **args: Any*) → None

Loads this image with pixel data from a bytes object.

This method is similar to the *frombytes()* function, but loads data into this image instead of creating a new image object.

Image.**getbands**() → tuple[str, ...]

Returns a tuple containing the name of each band in this image. For example, `getbands` on an RGB image returns ("R", "G", "B").

Returns
A tuple containing band names.

Return type
tuple

This helps to get the bands of the input image:

```
from PIL import Image

with Image.open("hopper.jpg") as im:
    print(im.getbands())  # Returns ('R', 'G', 'B')
```

Image.**getbbox**(** (Keyword-only parameters separator (PEP 3102))*, *alpha_only: bool = True*) → tuple[int, int, int, int] | None

Calculates the bounding box of the non-zero regions in the image.

Parameters
alpha_only – Optional flag, defaulting to `True`. If `True` and the image has an alpha channel, trim transparent pixels. Otherwise, trim pixels when all channels are zero. Keyword-only argument.

Returns
The bounding box is returned as a 4-tuple defining the left, upper, right, and lower pixel coordinate. See *Coordinate system*. If the image is completely empty, this method returns None.

This helps to get the bounding box coordinates of the input image:

```
from PIL import Image

with Image.open("hopper.jpg") as im:
    print(im.getbbox())
    # Returns four coordinates in the format (left, upper, right, lower)
```

Image.**getchannel**(*channel: int | str*) → *Image*

Returns an image containing a single channel of the source image.

Parameters
channel – What channel to return. Could be index (0 for "R" channel of "RGB") or channel name ("A" for alpha channel of "RGBA").

Returns
An image in "L" mode.

Added in version 4.3.0.

Image.**getcolors**(*maxcolors: int = 256*) → list[tuple[int, tuple[int, ...]]] | list[tuple[int, float]] | None

Returns a list of colors used in this image.

The colors will be in the image's mode. For example, an RGB image will return a tuple of (red, green, blue) color values, and a P image will return the index of the color in the palette.

Parameters
maxcolors – Maximum number of colors. If this number is exceeded, this method returns None. The default limit is 256 colors.

Returns
An unsorted list of (count, pixel) values.

Image.**getdata**(*band: int | None = None*) → *core.ImagingCore*

Returns the contents of this image as a sequence object containing pixel values. The sequence object is flattened, so that values for line one follow directly after the values of line zero, and so on.

Note that the sequence object returned by this method is an internal PIL data type, which only supports certain sequence operations. To convert it to an ordinary sequence (e.g. for printing), use `list(im.getdata())`.

Parameters
band – What band to return. The default is to return all bands. To return a single band, pass in the index value (e.g. 0 to get the "R" band from an "RGB" image).

Returns
A sequence-like object.

`Image.get_flattened_data(band: int | None = None) → tuple[tuple[int, ...], ...] | tuple[float, ...]`

Returns the contents of this image as a tuple containing pixel values. The sequence object is flattened, so that values for line one follow directly after the values of line zero, and so on.

Parameters
band – What band to return. The default is to return all bands. To return a single band, pass in the index value (e.g. 0 to get the "R" band from an "RGB" image).

Returns
A tuple containing pixel values.

`Image.getexif() → Exif`

Gets EXIF data from the image.

Returns
an *Exif* object.

`Image.getextrema() → tuple[float, float] | tuple[tuple[int, int], ...]`

Gets the minimum and maximum pixel values for each band in the image.

Returns
For a single-band image, a 2-tuple containing the minimum and maximum pixel value. For a multi-band image, a tuple containing one 2-tuple for each band.

`Image.getpalette(rawmode: str | None = 'RGB') → list[int] | None`

Returns the image palette as a list.

Parameters
rawmode – The mode in which to return the palette. `None` will return the palette in its current mode.

Added in version 9.1.0.

Returns
A list of color values [r, g, b, ...], or None if the image has no palette.

`Image.getpixel(xy: tuple[int, int] | list[int]) → float | tuple[int, ...] | None`

Returns the pixel value at a given position.

Parameters
xy – The coordinate, given as (x, y). See *Coordinate system*.

Returns
The pixel value. If the image is a multi-layer image, this method returns a tuple.

`Image.getprojection() → tuple[list[int], list[int]]`

Get projection to x and y axes

Returns
Two sequences, indicating where there are non-zero pixels along the X-axis and the Y-axis, respectively.

Image.**getxmp**() → dict[str, Any]

Returns a dictionary containing the XMP tags. Requires defusedxml to be installed.

Returns
XMP tags in a dictionary.

Image.**histogram**(*mask:* Image | *None* = *None*, *extrema:* *tuple[float, float]* | *None* = *None*) → list[int]

Returns a histogram for the image. The histogram is returned as a list of pixel counts, one for each pixel value in the source image. Counts are grouped into 256 bins for each band, even if the image has more than 8 bits per band. If the image has more than one band, the histograms for all bands are concatenated (for example, the histogram for an "RGB" image contains 768 values).

A bilevel image (mode "1") is treated as a grayscale ("L") image by this method.

If a mask is provided, the method returns a histogram for those parts of the image where the mask image is non-zero. The mask image must have the same size as the image, and be either a bi-level image (mode "1") or a grayscale image ("L").

Parameters

- **mask** – An optional mask.
- **extrema** – An optional tuple of manually-specified extrema.

Returns
A list containing pixel counts.

Image.**paste**(*im:* Image | *str* | *float* | *tuple[float, ...]*, *box:* Image | *tuple[int, int, int, int]* | *tuple[int, int]* | *None* = *None*, *mask:* Image | *None* = *None*) → None

Pastes another image into this image. The box argument is either a 2-tuple giving the upper left corner, a 4-tuple defining the left, upper, right, and lower pixel coordinate, or None (same as (0, 0)). See *Coordinate system*. If a 4-tuple is given, the size of the pasted image must match the size of the region.

If the modes don't match, the pasted image is converted to the mode of this image (see the *convert()* method for details).

Instead of an image, the source can be a integer or tuple containing pixel values. The method then fills the region with the given color. When creating RGB images, you can also use color strings as supported by the ImageColor module. See *Colors* for more information.

If a mask is given, this method updates only the regions indicated by the mask. You can use either "1", "L", "LA", "RGBA" or "RGBa" images (if present, the alpha band is used as mask). Where the mask is 255, the given image is copied as is. Where the mask is 0, the current value is preserved. Intermediate values will mix the two images together, including their alpha channels if they have them.

See *alpha_composite()* if you want to combine images with respect to their alpha channels.

Parameters

- **im** – Source image or pixel value (integer, float or tuple).
- **box** – An optional 4-tuple giving the region to paste into. If a 2-tuple is used instead, it's treated as the upper left corner. If omitted or None, the source is pasted into the upper left corner.

  If an image is given as the second argument and there is no third, the box defaults to (0, 0), and the second argument is interpreted as a mask image.
- **mask** – An optional mask image.

Image.**point**(*lut: Sequence[float] | NumpyArray | Callable[[int], float] | Callable[[ImagePointTransform], ImagePointTransform | float] | ImagePointHandler*, *mode: str | None = None*) → *Image*

Maps this image through a lookup table or function.

Parameters

- **lut** – A lookup table, containing 256 (or 65536 if self.mode==”I” and mode == “L”) values per band in the image. A function can be used instead, it should take a single argument. The function is called once for each possible pixel value, and the resulting table is applied to all bands of the image.

  It may also be an *ImagePointHandler* object:

  ```
  class Example(Image.ImagePointHandler):
    def point(self, im: Image) -> Image:
      # Return result
  ```

- **mode** – Output mode (default is same as input). This can only be used if the source image has mode “L” or “P”, and the output has mode “1” or the source image mode is “I” and the output mode is “L”.

Returns

An *Image* object.

Image.**putalpha**(*alpha:* Image | *int*) → None

Adds or replaces the alpha layer in this image. If the image does not have an alpha layer, it’s converted to “LA” or “RGBA”. The new layer must be either “L” or “1”.

Parameters

alpha – The new alpha layer. This can either be an “L” or “1” image having the same size as this image, or an integer.

Image.**putdata**(*data: Sequence[float] | Sequence[Sequence[int]] | core.ImagingCore | NumpyArray*, *scale: float = 1.0*, *offset: float = 0.0*) → None

Copies pixel data from a flattened sequence object into the image. The values should start at the upper left corner (0, 0), continue to the end of the line, followed directly by the first value of the second line, and so on. Data will be read until either the image or the sequence ends. The scale and offset values are used to adjust the sequence values: **pixel = value*scale + offset**.

Parameters

- **data** – A flattened sequence object. See *Colors* for more information about values.
- **scale** – An optional scale value. The default is 1.0.
- **offset** – An optional offset value. The default is 0.0.

Image.**putpalette**(*data:* ImagePalette.ImagePalette | *bytes* | *Sequence[int]*, *rawmode: str = 'RGB'*) → None

Attaches a palette to this image. The image must be a “P”, “PA”, “L” or “LA” image.

The palette sequence must contain at most 256 colors, made up of one integer value for each channel in the raw mode. For example, if the raw mode is “RGB”, then it can contain at most 768 values, made up of red, green and blue values for the corresponding pixel index in the 256 colors. If the raw mode is “RGBA”, then it can contain at most 1024 values, containing red, green, blue and alpha values.

Alternatively, an 8-bit string may be used instead of an integer sequence.

Parameters

- **data** – A palette sequence (either a list or a string).

- **rawmode** – The raw mode of the palette. Either "RGB", "RGBA", "CMYK", or a mode that can be transformed to one of those modes (e.g. "R", "RGBA;L").

Image.**putpixel**(*xy: tuple[int, int] | list[int]*, *value: float | tuple[int, ...] | list[int]*) → None

Modifies the pixel at the given position. The color is given as a single numerical value for single-band images, and a tuple for multi-band images. In addition to this, RGB and RGBA tuples are accepted for P and PA images. See *Colors* for more information.

Note that this method is relatively slow. For more extensive changes, use *paste()* or the *ImageDraw* module instead.

See:

- *paste()*
- *putdata()*
- *ImageDraw*

Parameters

- **xy** – The pixel coordinate, given as (x, y). See *Coordinate system*.
- **value** – The pixel value.

Image.**quantize**(*colors: int = 256*, *method: int | None = None*, *kmeans: int = 0*, *palette:* Image | *None = None*, *dither:* Dither = *Dither.FLOYDSTEINBERG*) → *Image*

Convert the image to 'P' mode with the specified number of colors.

Parameters

- **colors** – The desired number of colors, <= 256
- **method** – *Quantize.MEDIANCUT* (median cut), *Quantize.MAXCOVERAGE* (maximum coverage), *Quantize.FASTOCTREE* (fast octree), *Quantize.LIBIMAGEQUANT* (libimagequant; check support using *PIL.features.check_feature()* with `feature="libimagequant"`).

  By default, *Quantize.MEDIANCUT* will be used.

  The exception to this is RGBA images. *Quantize.MEDIANCUT* and *Quantize.MAXCOVERAGE* do not support RGBA images, so *Quantize.FASTOCTREE* is used by default instead.
- **kmeans** – Integer greater than or equal to zero.
- **palette** – Quantize to the palette of given *PIL.Image.Image*.
- **dither** – Dithering method, used when converting from mode "RGB" to "P" or from "RGB" or "L" to "1". Available methods are *Dither.NONE* or *Dither.FLOYDSTEINBERG* (default).

Returns

A new image

Image.**reduce**(*factor: int | tuple[int, int]*, *box: tuple[int, int, int, int] | None = None*) → *Image*

Returns a copy of the image reduced `factor` times. If the size of the image is not dividable by `factor`, the resulting size will be rounded up.

Parameters

- **factor** – A greater than 0 integer or tuple of two integers for width and height separately.

- **box** – An optional 4-tuple of ints providing the source image region to be reduced. The values must be within `(0, 0, width, height)` rectangle. If omitted or `None`, the entire source is used.

`Image.`**`remap_palette`**(*dest_map: list[int]*, *source_palette: bytes | bytearray | None = None*) → *Image*

Rewrites the image to reorder the palette.

Parameters

- **dest_map** – A list of indexes into the original palette. e.g. `[1,0]` would swap a two item palette, and `list(range(256))` is the identity transform.
- **source_palette** – Bytes or None.

Returns
An *Image* object.

`Image.`**`resize`**(*size: tuple[int, int] | list[int] | NumpyArray*, *resample: int | None = None*, *box: tuple[float, float, float, float] | None = None*, *reducing_gap: float | None = None*) → *Image*

Returns a resized copy of this image.

Parameters

- **size** – The requested size in pixels, as a tuple or array: (width, height).
- **resample** – An optional resampling filter. This can be one of *Resampling.NEAREST*, *Resampling.BOX*, *Resampling.BILINEAR*, *Resampling.HAMMING*, *Resampling.BICUBIC* or *Resampling.LANCZOS*. If the image has mode "1" or "P", it is always set to *Resampling.NEAREST*. Otherwise, the default filter is *Resampling.BICUBIC*. See: *Filters*.
- **box** – An optional 4-tuple of floats providing the source image region to be scaled. The values must be within (0, 0, width, height) rectangle. If omitted or None, the entire source is used.
- **reducing_gap** – Apply optimization by resizing the image in two steps. First, reducing the image by integer times using *reduce()*. Second, resizing using regular resampling. The last step changes size no less than by `reducing_gap` times. `reducing_gap` may be None (no first step is performed) or should be greater than 1.0. The bigger `reducing_gap`, the closer the result to the fair resampling. The smaller `reducing_gap`, the faster resizing. With `reducing_gap` greater or equal to 3.0, the result is indistinguishable from fair resampling in most cases. The default value is None (no optimization).

Returns
An *Image* object.

This resizes the given image from `(width, height)` to `(width/2, height/2)`:

```
from PIL import Image

with Image.open("hopper.jpg") as im:

    # Provide the target width and height of the image
    (width, height) = (im.width // 2, im.height // 2)
    im_resized = im.resize((width, height))
```

`Image.`**`rotate`**(*angle: float*, *resample: Resampling = Resampling.NEAREST*, *expand: int | bool = False*, *center: tuple[float, float] | None = None*, *translate: tuple[int, int] | None = None*, *fillcolor: float | tuple[float, ...] | str | None = None*) → *Image*

Returns a rotated copy of this image. This method returns a copy of this image, rotated the given number of degrees counter clockwise around its centre.

Parameters

- **angle** – In degrees counter clockwise.
- **resample** – An optional resampling filter. This can be one of *Resampling.NEAREST* (use nearest neighbour), *Resampling.BILINEAR* (linear interpolation in a 2x2 environment), or *Resampling.BICUBIC* (cubic spline interpolation in a 4x4 environment). If omitted, or if the image has mode "1" or "P", it is set to *Resampling.NEAREST*. See *Filters*.
- **expand** – Optional expansion flag. If true, expands the output image to make it large enough to hold the entire rotated image. If false or omitted, make the output image the same size as the input image. Note that the expand flag assumes rotation around the center and no translation.
- **center** – Optional center of rotation (a 2-tuple). Origin is the upper left corner. Default is the center of the image.
- **translate** – An optional post-rotate translation (a 2-tuple).
- **fillcolor** – An optional color for area outside the rotated image.

Returns

An *Image* object.

This rotates the input image by `theta` degrees counter clockwise:

```
from PIL import Image

with Image.open("hopper.jpg") as im:

    # Rotate the image by 60 degrees counter clockwise
    theta = 60
    # Angle is in degrees counter clockwise
    im_rotated = im.rotate(angle=theta)
```

`Image.save`(*fp:* StrOrBytesPath | *IO[bytes]*, *format: str* | *None* = *None*, ***params: Any*) → None

Saves this image under the given filename. If no format is specified, the format to use is determined from the filename extension, if possible.

Keyword options can be used to provide additional instructions to the writer. If a writer doesn't recognise an option, it is silently ignored. The available options are described in the *image format documentation* for each writer.

You can use a file object instead of a filename. In this case, you must always specify the format. The file object must implement the `seek`, `tell`, and `write` methods, and be opened in binary mode.

Parameters

- **fp** – A filename (string), os.PathLike object or file object.
- **format** – Optional format override. If omitted, the format to use is determined from the filename extension. If a file object was used instead of a filename, this parameter should always be used.
- **params** – Extra parameters to the image writer. These can also be set on the image itself through `encoderinfo`. This is useful when saving multiple images:

```
# Saving XMP data to a single image
from PIL import Image
red = Image.new("RGB", (1, 1), "#f00")
red.save("out.mpo", xmp=b"test")

# Saving XMP data to the second frame of an image
from PIL import Image
black = Image.new("RGB", (1, 1))
red = Image.new("RGB", (1, 1), "#f00")
red.encoderinfo = {"xmp": b"test"}
black.save("out.mpo", save_all=True, append_images=[red])
```

Returns
None

Raises

- **ValueError** – If the output format could not be determined from the file name. Use the format option to solve this.
- **OSError** – If the file could not be written. The file may have been created, and may contain partial data.

Image.**seek**(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

Image.**show**(*title: str | None = None*) → None

Displays this image. This method is mainly intended for debugging purposes.

This method calls *PIL.ImageShow.show()* internally. You can use *PIL.ImageShow.register()* to override its default behaviour.

The image is first saved to a temporary file. By default, it will be in PNG format.

On Unix, the image is then opened using the **xdg-open**, **display**, **gm**, **eog** or **xv** utility, depending on which one can be found.

On macOS, the image is opened with the native Preview application.

On Windows, the image is opened with the standard PNG display utility.

Parameters
title – Optional title to use for the image window, where possible.

Image.**split**() → tuple[*Image*, ...]

Split this image into individual bands. This method returns a tuple of individual image bands from an image. For example, splitting an "RGB" image creates three new images each containing a copy of one of the original bands (red, green, blue).

If you need only one band, *getchannel()* method can be more convenient and faster.

Returns
A tuple containing bands.

Image.**tell**() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

Image.**thumbnail**(*size: tuple[float, float]*, *resample:* Resampling = *Resampling.BICUBIC*, *reducing_gap: float | None = 2.0*) → None

Make this image into a thumbnail. This method modifies the image to contain a thumbnail version of itself, no larger than the given size. This method calculates an appropriate thumbnail size to preserve the aspect of the image, calls the *draft()* method to configure the file reader (where applicable), and finally resizes the image.

Note that this function modifies the *Image* object in place. If you need to use the full resolution image as well, apply this method to a *copy()* of the original image.

Parameters

- **size** – The requested size in pixels, as a 2-tuple: (width, height).
- **resample** – Optional resampling filter. This can be one of *Resampling.NEAREST*, *Resampling.BOX*, *Resampling.BILINEAR*, *Resampling.HAMMING*, *Resampling.BICUBIC* or *Resampling.LANCZOS*. If omitted, it defaults to *Resampling.BICUBIC*. (was *Resampling.NEAREST* prior to version 2.5.0). See: *Filters*.
- **reducing_gap** – Apply optimization by resizing the image in two steps. First, reducing the image by integer times using *reduce()* or *draft()* for JPEG images. Second, resizing using regular resampling. The last step changes size no less than by `reducing_gap` times. `reducing_gap` may be None (no first step is performed) or should be greater than 1.0. The bigger `reducing_gap`, the closer the result to the fair resampling. The smaller `reducing_gap`, the faster resizing. With `reducing_gap` greater or equal to 3.0, the result is indistinguishable from fair resampling in most cases. The default value is 2.0 (very close to fair resampling while still being faster in many cases).

Returns
None

Image.**tobitmap**(*name: str = 'image'*) → bytes

Returns the image converted to an X11 bitmap.

> **Note**
>
> This method only works for mode "1" images.

Parameters
name – The name prefix to use for the bitmap variables.

Returns
A string containing an X11 bitmap.

Raises
ValueError – If the mode is not "1"

Image.**tobytes**(*encoder_name: str = 'raw'*, **args: Any*) → bytes

Return image as a bytes object.

> **Warning**
>
> This method returns raw image data derived from Pillow's internal storage. For compressed image data (e.g. PNG, JPEG) use *save()*, with a BytesIO parameter for in-memory data.

Parameters

- **encoder_name** – What encoder to use.

  The default is to use the standard "raw" encoder. To see how this packs pixel data into the returned bytes, see `libImaging/Pack.c`.

  A list of C encoders can be seen under codecs section of the function array in `_imaging.c`. Python encoders are registered within the relevant plugins.

- **args** – Extra arguments to the encoder.

Returns

A `bytes` object.

Image.**transform**(*size: tuple[int, int]*, *method: Transform | ImageTransformHandler | SupportsGetData*, *data: Sequence[Any] | None = None*, *resample: int = Resampling.NEAREST*, *fill: int = 1*, *fillcolor: float | tuple[float, ...] | str | None = None*) → *Image*

Transforms this image. This method creates a new image with the given size, and the same mode as the original, and copies data to the new image using the given transform.

Parameters

- **size** – The output size in pixels, as a 2-tuple: (width, height).
- **method** – The transformation method. This is one of *Transform.EXTENT* (cut out a rectangular subregion), *Transform.AFFINE* (affine transform), *Transform.PERSPECTIVE* (perspective transform), *Transform.QUAD* (map a quadrilateral to a rectangle), or *Transform.MESH* (map a number of source quadrilaterals in one operation).

  It may also be an *ImageTransformHandler* object:

  ```
  class Example(Image.ImageTransformHandler):
      def transform(self, size, data, resample, fill=1):
          # Return result
  ```

  Implementations of *ImageTransformHandler* for some of the *Transform* methods are provided in *ImageTransform*.

  It may also be an object with a `method.getdata` method that returns a tuple supplying new `method` and `data` values:

  ```
  class Example:
      def getdata(self):
          method = Image.Transform.EXTENT
          data = (0, 0, 100, 100)
          return method, data
  ```

- **data** – Extra data to the transformation method.

- **resample** – Optional resampling filter. It can be one of *Resampling.NEAREST* (use nearest neighbour), *Resampling.BILINEAR* (linear interpolation in a 2x2 environment), or *Resampling.BICUBIC* (cubic spline interpolation in a 4x4 environment). If omitted, or if the image has mode "1" or "P", it is set to *Resampling.NEAREST*. See: *Filters*.
- **fill** – If `method` is an *ImageTransformHandler* object, this is one of the arguments passed to it. Otherwise, it is unused.
- **fillcolor** – Optional fill color for the area outside the transform in the output image.

Returns
An *Image* object.

`Image.`**`transpose`**`(`*method:* Transpose`)` → *Image*

Transpose image (flip or rotate in 90 degree steps)

Parameters
method – One of *Transpose.FLIP_LEFT_RIGHT*, *Transpose.FLIP_TOP_BOTTOM*, *Transpose.ROTATE_90*, *Transpose.ROTATE_180*, *Transpose.ROTATE_270*, *Transpose.TRANSPOSE* or *Transpose.TRANSVERSE*.

Returns
Returns a flipped or rotated copy of this image.

This flips the input image by using the *Transpose.FLIP_LEFT_RIGHT* method.

```
from PIL import Image

with Image.open("hopper.jpg") as im:

    # Flip the image from left to right
    im_flipped = im.transpose(method=Image.Transpose.FLIP_LEFT_RIGHT)
    # To flip the image from top to bottom,
    # use the method "Image.Transpose.FLIP_TOP_BOTTOM"
```

`Image.`**`verify`**`()` → None

Verifies the contents of a file. For data read from a file, this method attempts to determine if the file is broken, without actually decoding the image data. If this method finds any problems, it raises suitable exceptions. If you need to load the image after using this method, you must reopen the image file.

`Image.`**`load`**`()` → *core.PixelAccess* | None

Allocates storage for the image and loads the pixel data. In normal cases, you don't need to call this method, since the Image class automatically loads an opened image when it is accessed for the first time.

If the file associated with the image was opened by Pillow, then this method will close it. The exception to this is if the image has multiple frames, in which case the file will be left open for seek operations. See *File handling in Pillow* for more information.

Returns
An image access object.

Return type
PixelAccess

`Image.`**`close`**`()` → None

This operation will destroy the image core and release its memory. The image data will be unusable afterward.

This function is required to close images that have multiple frames or have not had their file read and closed by the *load()* method. See *File handling in Pillow* for more information.

Image attributes

Instances of the *Image* class have the following attributes:

`Image.filename: str`

The filename or path of the source file. Only images created with the factory function `open` have a filename attribute. If the input is a file like object, the filename attribute is set to an empty string.

`Image.format: str | None`

The file format of the source file. For images created by the library itself (via a factory function, or by running a method on an existing image), this attribute is set to `None`.

`Image.mode: str`

Image mode. This is a string specifying the pixel format used by the image. Typical values are "1", "L", "RGB", or "CMYK." See *Modes* for a full list.

`Image.size: tuple[int]`

Image size, in pixels. The size is given as a 2-tuple (width, height).

`Image.width: int`

Image width, in pixels.

`Image.height: int`

Image height, in pixels.

`Image.palette: PIL.ImagePalette.ImagePalette | None`

Colour palette table, if any. If mode is "P" or "PA", this should be an instance of the *ImagePalette* class. Otherwise, it should be set to `None`.

`Image.info: dict`

A dictionary holding data associated with the image. This dictionary is used by file handlers to pass on various non-image information read from the file. See documentation for the various file handlers for details.

Most methods ignore the dictionary when returning new images; since the keys are not standardized, it's not possible for a method to know if the operation affects the dictionary. If you need the information later on, keep a reference to the info dictionary returned from the open method.

Unless noted elsewhere, this dictionary does not affect saving files.

`Image.is_animated: bool`

`True` if this image has more than one frame, or `False` otherwise.

This attribute is only defined by image plugins that support animated images. Plugins may leave this attribute undefined if they don't support loading animated images, even if the given format supports animated images.

Given that this attribute is not present for all images use `getattr(image, "is_animated", False)` to check if Pillow is aware of multiple frames in an image regardless of its format.

> **See also**
>
> *n_frames*, *seek()* and *tell()*

`Image.n_frames: int`

The number of frames in this image.

This attribute is only defined by image plugins that support animated images. Plugins may leave this attribute undefined if they don't support loading animated images, even if the given format supports animated images.

Given that this attribute is not present for all images use `getattr(image, "n_frames", 1)` to check the number of frames that Pillow is aware of in an image regardless of its format.

See also

is_animated, *seek()* and *tell()*

`Image.`**has_transparency_data**

Determine if an image has transparency data, whether in the form of an alpha channel, a palette with an alpha channel, or a "transparency" key in the info dictionary.

Note the image might still appear solid, if all of the values shown within are opaque.

Returns
A boolean.

Classes

class `PIL.Image.`**Exif**

Bases: `MutableMapping`

This class provides read and write access to EXIF image data:

```
from PIL import Image
im = Image.open("exif.png")
exif = im.getexif()  # Returns an instance of this class
```

Information can be read and written, iterated over or deleted:

```
print(exif[274])  # 1
exif[274] = 2
for k, v in exif.items():
  print("Tag", k, "Value", v)  # Tag 274 Value 2
del exif[274]
```

To access information beyond IFD0, *get_ifd()* returns a dictionary:

```
from PIL import ExifTags
im = Image.open("exif_gps.jpg")
exif = im.getexif()
gps_ifd = exif.get_ifd(ExifTags.IFD.GPSInfo)
print(gps_ifd)
```

Other IFDs include `ExifTags.IFD.Exif`, `ExifTags.IFD.MakerNote`, `ExifTags.IFD.Interop` and `ExifTags.IFD.IFD1`.

ExifTags also has enum classes to provide names for data:

```
print(exif[ExifTags.Base.Software])  # PIL
print(gps_ifd[ExifTags.GPS.GPSDateStamp])  # 1999:99:99 99:99:99
```

bigtiff = False

endian: str | None = None

get_ifd(*tag: int*) → dict[int, Any]

hide_offsets() → None

load(*data: bytes*) → None

load_from_fp(*fp: IO[bytes]*, *offset: int | None = None*) → None

tobytes(*offset: int = 8*) → bytes

class PIL.Image.**ImagePointHandler**

Used as a mixin by point transforms (for use with *point()*)

class PIL.Image.**ImagePointTransform**(*scale: float*, *offset: float*)

Used with *point()* for single band images with more than 8 bits, this represents an affine transformation, where the value is multiplied by `scale` and `offset` is added.

class PIL.Image.**ImageTransformHandler**

Used as a mixin by geometry transforms (for use with *transform()*)

Protocols

class PIL.Image.**SupportsArrayInterface**(**args*, ***kwargs*)

Bases: Protocol

An object that has an `__array_interface__` dictionary.

class PIL.Image.**SupportsArrowArrayInterface**(**args*, ***kwargs*)

Bases: Protocol

An object that has an `__arrow_c_array__` method corresponding to the arrow c data interface.

class PIL.Image.**SupportsGetData**(**args*, ***kwargs*)

Bases: Protocol

PIL.Image.**DecoderInput**

alias of bytes | bytearray | memoryview | *SupportsArrayInterface*

Constants

PIL.Image.**NONE**

PIL.Image.**MAX_IMAGE_PIXELS**

Set to 89,478,485, approximately 0.25GB for a 24-bit (3 bpp) image. See *open()* for more information about how this is used.

PIL.Image.**WARN_POSSIBLE_FORMATS**

Set to false. If true, when an image cannot be identified, warnings will be raised from formats that attempted to read the data.

Transpose methods

Used to specify the *Image.transpose()* method to use.

class PIL.Image.**Transpose**(**values*)

FLIP_LEFT_RIGHT = 0

FLIP_TOP_BOTTOM = 1

ROTATE_180 = 3

ROTATE_270 = 4

ROTATE_90 = 2

TRANSPOSE = 5

TRANSVERSE = 6

Transform methods

Used to specify the *Image.transform()* method to use.

class PIL.Image.**Transform**

AFFINE
Affine transform

EXTENT
Cut out a rectangular subregion

PERSPECTIVE
Perspective transform

QUAD
Map a quadrilateral to a rectangle

MESH
Map a number of source quadrilaterals in one operation

Resampling filters

See *Filters* for details.

class PIL.Image.**Resampling**(**values*)

BICUBIC = 3

BILINEAR = 2

BOX = 4

HAMMING = 5

LANCZOS = 1

NEAREST = 0

Dither modes

Used to specify the dithering method to use for the *convert()* and *quantize()* methods.

class PIL.Image.**Dither**

NONE
No dither

ORDERED

Not implemented

RASTERIZE

Not implemented

FLOYDSTEINBERG

Floyd-Steinberg dither

Palettes

Used to specify the palette to use for the *convert()* method.

class PIL.Image.**Palette**(**values*)

ADAPTIVE = 1

WEB = 0

Quantization methods

Used to specify the quantization method to use for the *quantize()* method.

class PIL.Image.**Quantize**

MEDIANCUT

Median cut. Default method, except for RGBA images. This method does not support RGBA images.

MAXCOVERAGE

Maximum coverage. This method does not support RGBA images.

FASTOCTREE

Fast octree. Default method for RGBA images.

LIBIMAGEQUANT

libimagequant

Check support using *PIL.features.check_feature()* with `feature="libimagequant"`.

1.3.2 *ImageChops* ("channel operations") module

The *ImageChops* module contains a number of arithmetical image operations, called channel operations ("chops"). These can be used for various purposes, including special effects, image compositions, algorithmic painting, and more.

For more pre-made operations, see *ImageOps*.

At this time, most channel operations are only implemented for 8-bit images (e.g. "L" and "RGB").

Functions

Most channel operations take one or two image arguments and return a new image. Unless otherwise noted, the result of a channel operation is always clipped to the range 0 to MAX (which is 255 for all modes supported by the operations in this module).

PIL.ImageChops.**add**(*image1:* Image, *image2:* Image, *scale: float = 1.0*, *offset: float = 0*) → *Image*

Adds two images, dividing the result by scale and adding the offset. If omitted, scale defaults to 1.0, and offset to 0.0.

```
out = ((image1 + image2) / scale + offset)
```

Return type
Image

PIL.ImageChops.**add_modulo**(*image1:* Image, *image2:* Image) → *Image*

Add two images, without clipping the result.

```
out = ((image1 + image2) % MAX)
```

Return type
Image

PIL.ImageChops.**blend**(*image1:* Image, *image2:* Image, *alpha:* *float*) → *Image*

Blend images using constant transparency weight. Alias for *PIL.Image.blend()*.

Return type
Image

PIL.ImageChops.**composite**(*image1:* Image, *image2:* Image, *mask:* Image) → *Image*

Create composite using transparency mask. Alias for *PIL.Image.composite()*.

Return type
Image

PIL.ImageChops.**constant**(*image:* Image, *value:* *int*) → *Image*

Fill a channel with a given gray level.

Return type
Image

PIL.ImageChops.**darker**(*image1:* Image, *image2:* Image) → *Image*

Compares the two images, pixel by pixel, and returns a new image containing the darker values.

```
out = min(image1, image2)
```

Return type
Image

PIL.ImageChops.**difference**(*image1:* Image, *image2:* Image) → *Image*

Returns the absolute value of the pixel-by-pixel difference between the two images.

```
out = abs(image1 - image2)
```

Return type
Image

PIL.ImageChops.**duplicate**(*image:* Image) → *Image*

Copy a channel. Alias for *PIL.Image.Image.copy()*.

Return type
Image

PIL.ImageChops.**invert**(*image:* Image) → *Image*

Invert an image (channel).

```
out = MAX - image
```

Return type
Image

PIL.ImageChops.**lighter**(*image1:* Image, *image2:* Image) → *Image*

Compares the two images, pixel by pixel, and returns a new image containing the lighter values.

```
out = max(image1, image2)
```

Return type
Image

PIL.ImageChops.**logical_and**(*image1:* Image, *image2:* Image) → *Image*

Logical AND between two images.

Both of the images must have mode "1". If you would like to perform a logical AND on an image with a mode other than "1", try *multiply()* instead, using a black-and-white mask as the second image.

```
out = ((image1 and image2) % MAX)
```

Return type
Image

PIL.ImageChops.**logical_or**(*image1:* Image, *image2:* Image) → *Image*

Logical OR between two images.

Both of the images must have mode "1".

```
out = ((image1 or image2) % MAX)
```

Return type
Image

PIL.ImageChops.**logical_xor**(*image1:* Image, *image2:* Image) → *Image*

Logical XOR between two images.

Both of the images must have mode "1".

```
out = ((bool(image1) != bool(image2)) % MAX)
```

Return type
Image

PIL.ImageChops.**multiply**(*image1:* Image, *image2:* Image) → *Image*

Superimposes two images on top of each other.

If you multiply an image with a solid black image, the result is black. If you multiply with a solid white image, the image is unaffected.

```
out = image1 * image2 / MAX
```

Return type
Image

PIL.ImageChops.**soft_light**(*image1:* Image, *image2:* Image) → *Image*

Superimposes two images on top of each other using the Soft Light algorithm

Return type
Image

PIL.ImageChops.**hard_light**(*image1:* Image, *image2:* Image) → *Image*

Superimposes two images on top of each other using the Hard Light algorithm

Return type
Image

PIL.ImageChops.**overlay**(*image1:* Image, *image2:* Image) → *Image*

Superimposes two images on top of each other using the Overlay algorithm

Return type
Image

PIL.ImageChops.**offset**(*image:* Image, *xoffset:* *int*, *yoffset:* *int* | *None* = *None*) → *Image*

Returns a copy of the image where data has been offset by the given distances. Data wraps around the edges. If `yoffset` is omitted, it is assumed to be equal to `xoffset`.

Parameters

- **image** – Input image.
- **xoffset** – The horizontal distance.
- **yoffset** – The vertical distance. If omitted, both distances are set to the same value.

Return type
Image

PIL.ImageChops.**screen**(*image1:* Image, *image2:* Image) → *Image*

Superimposes two inverted images on top of each other.

```
out = MAX - ((MAX - image1) * (MAX - image2) / MAX)
```

Return type
Image

PIL.ImageChops.**subtract**(*image1:* Image, *image2:* Image, *scale:* *float* = *1.0*, *offset:* *float* = *0*) → *Image*

Subtracts two images, dividing the result by scale and adding the offset. If omitted, scale defaults to 1.0, and offset to 0.0.

```
out = ((image1 - image2) / scale + offset)
```

Return type
Image

PIL.ImageChops.**subtract_modulo**(*image1:* Image, *image2:* Image) → *Image*

Subtract two images, without clipping the result.

```
out = ((image1 - image2) % MAX)
```

Return type
Image

1.3.3 *ImageCms* module

The *ImageCms* module provides color profile management support using the LittleCMS2 color management engine, based on Kevin Cazabon's PyCMS library.

class PIL.ImageCms.**ImageCmsProfile**(*profile: str* | SupportsRead[*bytes*] | CmsProfile)

__init__(*profile: str* | SupportsRead[*bytes*] | CmsProfile) → None

Parameters
profile – Either a string representing a filename, a file like object containing a profile or a low-level profile object

tobytes() → bytes

Returns the profile in a format suitable for embedding in saved images.

Returns
a bytes object containing the ICC profile.

class PIL.ImageCms.**ImageCmsTransform**(*input:* ImageCmsProfile, *output:* ImageCmsProfile, *input_mode: str*, *output_mode: str*, *intent:* Intent = *Intent.PERCEPTUAL*, *proof:* ImageCmsProfile | *None* = *None*, *proof_intent:* Intent = *Intent.ABSOLUTE_COLORIMETRIC*, *flags:* Flags = *<Flags.NONE: 0>*)

Bases: *ImagePointHandler*

Transform. This can be used with the procedural API, or with the standard *point()* method.

Will return the output profile in the `output.info['icc_profile']`.

apply(*im:* Image, *imOut:* Image | *None* = *None*) → *Image*

apply_in_place(*im:* Image) → *Image*

point(*im:* Image) → *Image*

exception PIL.ImageCms.**PyCMSError**

(pyCMS) Exception class. This is used for all errors in the pyCMS API.

Constants

class PIL.ImageCms.**Intent**(**values*)

Bases: IntEnum

PERCEPTUAL = 0

RELATIVE_COLORIMETRIC = 1

SATURATION = 2

ABSOLUTE_COLORIMETRIC = 3

class PIL.ImageCms.**Direction**(**values*)

Bases: IntEnum

INPUT = 0

OUTPUT = 1

PROOF = 2

class PIL.ImageCms.**Flags**(**values*)

Bases: IntFlag

Flags and documentation are taken from `lcms2.h`.

NONE = 0

NOCACHE = 64

Inhibit 1-pixel cache

NOOPTIMIZE = 256

Inhibit optimizations

NULLTRANSFORM = 512

Don't transform anyway

GAMUTCHECK = 4096

Out of Gamut alarm

SOFTPROOFING = 16384

Do softproofing

BLACKPOINTCOMPENSATION = 8192

NOWHITEONWHITEFIXUP = 4

Don't fix scum dot

HIGHRESPRECALC = 1024

Use more memory to give better accuracy

LOWRESPRECALC = 2048

Use less memory to minimize resources

USE_8BITS_DEVICELINK = 8

Create 8 bits devicelinks

GUESSDEVICECLASS = 32

Guess device class (for `transform2devicelink`)

KEEP_SEQUENCE = 128

Keep profile sequence for devicelink creation

FORCE_CLUT = 2

Force CLUT optimization

CLUT_POST_LINEARIZATION = 1

create postlinearization tables if possible

CLUT_PRE_LINEARIZATION = 16

create prelinearization tables if possible

NONEGATIVES = 32768

Prevent negative numbers in floating point transforms

COPY_ALPHA = 67108864

Alpha channels are copied on `cmsDoTransform()`

NODEFAULTRESOURCEDEF = 16777216

static GRIDPOINTS(*n: int*) → *Flags*

Fine-tune control over number of gridpoints

Parameters
n – `int` in range `0 <= n <= 255`

Functions

PIL.ImageCms.**applyTransform**(*im:* Image, *transform:* ImageCmsTransform, *inPlace: bool = False*) → *Image* | None

(pyCMS) Applies a transform to a given image.

If `im.mode != transform.input_mode`, a *PyCMSError* is raised.

If `inPlace` is `True` and `transform.input_mode != transform.output_mode`, a *PyCMSError* is raised.

If `im.mode`, `transform.input_mode` or `transform.output_mode` is not supported by pyCMSdll or the profiles you used for the transform, a *PyCMSError* is raised.

If an error occurs while the transform is being applied, a *PyCMSError* is raised.

This function applies a pre-calculated transform (from ImageCms.buildTransform() or ImageCms.buildTransformFromOpenProfiles()) to an image. The transform can be used for multiple images, saving considerable calculation time if doing the same conversion multiple times.

If you want to modify im in-place instead of receiving a new image as the return value, set `inPlace` to `True`. This can only be done if `transform.input_mode` and `transform.output_mode` are the same, because we can't change the mode in-place (the buffer sizes for some modes are different). The default behavior is to return a new *Image* object of the same dimensions in mode `transform.output_mode`.

Parameters

- **im** – An *Image* object, and `im.mode` must be the same as the `input_mode` supported by the transform.
- **transform** – A valid CmsTransform class object
- **inPlace** – Bool. If `True`, `im` is modified in place and `None` is returned, if `False`, a new *Image* object with the transform applied is returned (and `im` is not changed). The default is `False`.

Returns
Either `None`, or a new *Image* object, depending on the value of `inPlace`. The profile will be returned in the image's `info['icc_profile']`.

Raises
PyCMSError –

PIL.ImageCms.**buildProofTransform**(*inputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *outputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *proofProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *inMode: str*, *outMode: str*, *renderingIntent: Intent = Intent.PERCEPTUAL*, *proofRenderingIntent: Intent = Intent.ABSOLUTE_COLORIMETRIC*, *flags: Flags = <Flags.SOFTPROOFING: 16384>*) → *ImageCmsTransform*

(pyCMS) Builds an ICC transform mapping from the `inputProfile` to the `outputProfile`, but tries to simulate the result that would be obtained on the `proofProfile` device.

If the input, output, or proof profiles specified are not valid filenames, a *PyCMSError* will be raised.

If an error occurs during creation of the transform, a *PyCMSError* will be raised.

If `inMode` or `outMode` are not a mode supported by the `outputProfile` (or by pyCMS), a *PyCMSError* will be raised.

This function builds and returns an ICC transform from the `inputProfile` to the `outputProfile`, but tries to simulate the result that would be obtained on the `proofProfile` device using `renderingIntent` and `proofRenderingIntent` to determine what to do with out-of-gamut colors. This is known as "soft-proofing". It will ONLY work for converting images that are in `inMode` to images that are in outMode color format (PIL mode, i.e. "RGB", "RGBA", "CMYK", etc.).

Usage of the resulting transform object is exactly the same as with ImageCms.buildTransform().

Proof profiling is generally used when using an output device to get a good idea of what the final printed/displayed image would look like on the `proofProfile` device when it's quicker and easier to use the output device for judging color. Generally, this means that the output device is a monitor, or a dye-sub printer (etc.), and the simulated device is something more expensive, complicated, or time consuming (making it difficult to make a real print for color judgement purposes).

Soft-proofing basically functions by adjusting the colors on the output device to match the colors of the device being simulated. However, when the simulated device has a much wider gamut than the output device, you may obtain marginal results.

Parameters

- **inputProfile** – String, as a valid filename path to the ICC input profile you wish to use for this transform, or a profile object
- **outputProfile** – String, as a valid filename path to the ICC output (monitor, usually) profile you wish to use for this transform, or a profile object
- **proofProfile** – String, as a valid filename path to the ICC proof profile you wish to use for this transform, or a profile object
- **inMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. "RGB", "RGBA", "CMYK", etc.)
- **outMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. "RGB", "RGBA", "CMYK", etc.)
- **renderingIntent** – Integer (0-3) specifying the rendering intent you wish to use for the input->proof (simulated) transform

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what they do.

- **proofRenderingIntent** – Integer (0-3) specifying the rendering intent you wish to use for proof->output transform

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what they do.

- **flags** – Integer (0-...) specifying additional flags

Returns
A CmsTransform class object.

Raises
PyCMSError –

PIL.ImageCms.**buildProofTransformFromOpenProfiles**(*inputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *outputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *proofProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *inMode: str*, *outMode: str*, *renderingIntent: Intent = Intent.PERCEPTUAL*, *proofRenderingIntent: Intent = Intent.ABSOLUTE_COLORIMETRIC*, *flags: Flags = <Flags.SOFTPROOFING: 16384>*) → *ImageCmsTransform*

(pyCMS) Builds an ICC transform mapping from the `inputProfile` to the `outputProfile`, but tries to simulate the result that would be obtained on the `proofProfile` device.

If the input, output, or proof profiles specified are not valid filenames, a *PyCMSError* will be raised.

If an error occurs during creation of the transform, a *PyCMSError* will be raised.

If `inMode` or `outMode` are not a mode supported by the `outputProfile` (or by pyCMS), a *PyCMSError* will be raised.

This function builds and returns an ICC transform from the `inputProfile` to the `outputProfile`, but tries to simulate the result that would be obtained on the `proofProfile` device using `renderingIntent` and `proofRenderingIntent` to determine what to do with out-of-gamut colors. This is known as "soft-proofing". It will ONLY work for converting images that are in `inMode` to images that are in outMode color format (PIL mode, i.e. "RGB", "RGBA", "CMYK", etc.).

Usage of the resulting transform object is exactly the same as with ImageCms.buildTransform().

Proof profiling is generally used when using an output device to get a good idea of what the final printed/displayed image would look like on the `proofProfile` device when it's quicker and easier to use the output device for judging color. Generally, this means that the output device is a monitor, or a dye-sub printer (etc.), and the simulated device is something more expensive, complicated, or time consuming (making it difficult to make a real print for color judgement purposes).

Soft-proofing basically functions by adjusting the colors on the output device to match the colors of the device being simulated. However, when the simulated device has a much wider gamut than the output device, you may obtain marginal results.

Parameters

- **inputProfile** – String, as a valid filename path to the ICC input profile you wish to use for this transform, or a profile object

- **outputProfile** – String, as a valid filename path to the ICC output (monitor, usually) profile you wish to use for this transform, or a profile object
- **proofProfile** – String, as a valid filename path to the ICC proof profile you wish to use for this transform, or a profile object
- **inMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. "RGB", "RGBA", "CMYK", etc.)
- **outMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. "RGB", "RGBA", "CMYK", etc.)
- **renderingIntent** – Integer (0-3) specifying the rendering intent you wish to use for the input->proof (simulated) transform

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what they do.
- **proofRenderingIntent** – Integer (0-3) specifying the rendering intent you wish to use for proof->output transform

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what they do.
- **flags** – Integer (0-...) specifying additional flags

Returns
A CmsTransform class object.

Raises
PyCMSError –

PIL.ImageCms.**buildTransform**(*inputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *outputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *inMode: str*, *outMode: str*, *renderingIntent:* Intent = *Intent.PERCEPTUAL*, *flags:* Flags = *<Flags.NONE: 0>*) → *ImageCmsTransform*

(pyCMS) Builds an ICC transform mapping from the `inputProfile` to the `outputProfile`. Use applyTransform to apply the transform to a given image.

If the input or output profiles specified are not valid filenames, a *PyCMSError* will be raised. If an error occurs during creation of the transform, a *PyCMSError* will be raised.

If `inMode` or `outMode` are not a mode supported by the `outputProfile` (or by pyCMS), a *PyCMSError* will be raised.

This function builds and returns an ICC transform from the `inputProfile` to the `outputProfile` using the `renderingIntent` to determine what to do with out-of-gamut colors. It will ONLY work for converting images that are in `inMode` to images that are in `outMode` color format (PIL mode, i.e. "RGB", "RGBA", "CMYK", etc.).

Building the transform is a fair part of the overhead in ImageCms.profileToProfile(), so if you're planning on converting multiple images using the same input/output settings, this can save you time. Once you have a transform object, it can be used with ImageCms.applyProfile() to convert images without the need to re-compute the lookup table for the transform.

The reason pyCMS returns a class object rather than a handle directly to the transform is that it needs to keep track of the PIL input/output modes that the transform is meant for. These attributes are stored in the `inMode`

and outMode attributes of the object (which can be manually overridden if you really want to, but I don't know of any time that would be of use, or would even work).

Parameters

- **inputProfile** – String, as a valid filename path to the ICC input profile you wish to use for this transform, or a profile object
- **outputProfile** – String, as a valid filename path to the ICC output profile you wish to use for this transform, or a profile object
- **inMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. "RGB", "RGBA", "CMYK", etc.)
- **outMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. "RGB", "RGBA", "CMYK", etc.)
- **renderingIntent** – Integer (0-3) specifying the rendering intent you wish to use for the transform

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what they do.
- **flags** – Integer (0-...) specifying additional flags

Returns
A CmsTransform class object.

Raises
PyCMSError –

PIL.ImageCms.**buildTransformFromOpenProfiles**(*inputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *outputProfile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *inMode: str*, *outMode: str*, *renderingIntent:* Intent = *Intent.PERCEPTUAL*, *flags:* Flags = <*Flags.NONE: 0*>) → *ImageCmsTransform*

(pyCMS) Builds an ICC transform mapping from the inputProfile to the outputProfile. Use applyTransform to apply the transform to a given image.

If the input or output profiles specified are not valid filenames, a *PyCMSError* will be raised. If an error occurs during creation of the transform, a *PyCMSError* will be raised.

If inMode or outMode are not a mode supported by the outputProfile (or by pyCMS), a *PyCMSError* will be raised.

This function builds and returns an ICC transform from the inputProfile to the outputProfile using the renderingIntent to determine what to do with out-of-gamut colors. It will ONLY work for converting images that are in inMode to images that are in outMode color format (PIL mode, i.e. "RGB", "RGBA", "CMYK", etc.).

Building the transform is a fair part of the overhead in ImageCms.profileToProfile(), so if you're planning on converting multiple images using the same input/output settings, this can save you time. Once you have a transform object, it can be used with ImageCms.applyProfile() to convert images without the need to re-compute the lookup table for the transform.

The reason pyCMS returns a class object rather than a handle directly to the transform is that it needs to keep track of the PIL input/output modes that the transform is meant for. These attributes are stored in the inMode and outMode attributes of the object (which can be manually overridden if you really want to, but I don't know of any time that would be of use, or would even work).

Parameters

- **inputProfile** – String, as a valid filename path to the ICC input profile you wish to use for this transform, or a profile object
- **outputProfile** – String, as a valid filename path to the ICC output profile you wish to use for this transform, or a profile object
- **inMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. “RGB”, “RGBA”, “CMYK”, etc.)
- **outMode** – String, as a valid PIL mode that the appropriate profile also supports (i.e. “RGB”, “RGBA”, “CMYK”, etc.)
- **renderingIntent** – Integer (0-3) specifying the rendering intent you wish to use for the transform

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what they do.
- **flags** – Integer (0-...) specifying additional flags

Returns
A CmsTransform class object.

Raises
PyCMSError –

PIL.ImageCms.**createProfile**(*colorSpace: Literal['LAB', 'XYZ', 'sRGB']*, *colorTemp: SupportsFloat = 0*) → *CmsProfile*

(pyCMS) Creates a profile.

If colorSpace not in `["LAB", "XYZ", "sRGB"]`, a *PyCMSError* is raised.

If using LAB and `colorTemp` is not a positive integer, a *PyCMSError* is raised.

If an error occurs while creating the profile, a *PyCMSError* is raised.

Use this function to create common profiles on-the-fly instead of having to supply a profile on disk and knowing the path to it. It returns a normal CmsProfile object that can be passed to ImageCms.buildTransformFromOpenProfiles() to create a transform to apply to images.

Parameters

- **colorSpace** – String, the color space of the profile you wish to create. Currently only “LAB”, “XYZ”, and “sRGB” are supported.
- **colorTemp** – Positive number for the white point for the profile, in degrees Kelvin (i.e. 5000, 6500, 9600, etc.). The default is for D50 illuminant if omitted (5000k). colorTemp is ONLY applied to LAB profiles, and is ignored for XYZ and sRGB.

Returns
A CmsProfile class object

Raises
PyCMSError –

PIL.ImageCms.**getDefaultIntent**(*profile: str* | SupportsRead[*bytes*] | CmsProfile | ImageCmsProfile) → int

(pyCMS) Gets the default intent name for the given profile.

If `profile` isn't a valid CmsProfile object or filename to a profile, a *PyCMSError* is raised.

If an error occurs while trying to obtain the default intent, a *PyCMSError* is raised.

Use this function to determine the default (and usually best optimized) rendering intent for this profile. Most profiles support multiple rendering intents, but are intended mostly for one type of conversion. If you wish to use a different intent than returned, use ImageCms.isIntentSupported() to verify it will work first.

Parameters
profile – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.

Returns

Integer 0-3 specifying the default rendering intent for this profile.

ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

see the pyCMS documentation for details on rendering intents and what
they do.

Raises
PyCMSError –

PIL.ImageCms.**getOpenProfile**(*profileFilename: str | SupportsRead[bytes] | CmsProfile*) → *ImageCmsProfile*

(pyCMS) Opens an ICC profile file.

The PyCMSProfile object can be passed back into pyCMS for use in creating transforms and such (as in ImageCms.buildTransformFromOpenProfiles()).

If `profileFilename` is not a valid filename for an ICC profile, a *PyCMSError* will be raised.

Parameters
profileFilename – String, as a valid filename path to the ICC profile you wish to open, or a file-like object.

Returns
A CmsProfile class object.

Raises
PyCMSError –

PIL.ImageCms.**getProfileCopyright**(*profile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*) → str

(pyCMS) Gets the copyright for the given profile.

If `profile` isn't a valid CmsProfile object or filename to a profile, a *PyCMSError* is raised.

If an error occurs while trying to obtain the copyright tag, a *PyCMSError* is raised.

Use this function to obtain the information stored in the profile's copyright tag.

Parameters
profile – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.

Returns
A string containing the internal profile information stored in an ICC tag.

Raises
PyCMSError –

PIL.ImageCms.**getProfileDescription**(*profile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*) → str

(pyCMS) Gets the description for the given profile.

If `profile` isn't a valid CmsProfile object or filename to a profile, a *PyCMSError* is raised.

If an error occurs while trying to obtain the description tag, a *PyCMSError* is raised.

Use this function to obtain the information stored in the profile's description tag.

Parameters
profile – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.

Returns
A string containing the internal profile information stored in an ICC tag.

Raises
PyCMSError –

PIL.ImageCms.**getProfileInfo**(*profile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*) → str

(pyCMS) Gets the internal product information for the given profile.

If `profile` isn't a valid CmsProfile object or filename to a profile, a *PyCMSError* is raised.

If an error occurs while trying to obtain the info tag, a *PyCMSError* is raised.

Use this function to obtain the information stored in the profile's info tag. This often contains details about the profile, and how it was created, as supplied by the creator.

Parameters
profile – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.

Returns
A string containing the internal profile information stored in an ICC tag.

Raises
PyCMSError –

PIL.ImageCms.**getProfileManufacturer**(*profile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*) → str

(pyCMS) Gets the manufacturer for the given profile.

If `profile` isn't a valid CmsProfile object or filename to a profile, a *PyCMSError* is raised.

If an error occurs while trying to obtain the manufacturer tag, a *PyCMSError* is raised.

Use this function to obtain the information stored in the profile's manufacturer tag.

Parameters
profile – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.

Returns
A string containing the internal profile information stored in an ICC tag.

Raises
PyCMSError –

PIL.ImageCms.**getProfileModel**(*profile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*) → str

(pyCMS) Gets the model for the given profile.

If `profile` isn't a valid CmsProfile object or filename to a profile, a *PyCMSError* is raised.

If an error occurs while trying to obtain the model tag, a *PyCMSError* is raised.

Use this function to obtain the information stored in the profile's model tag.

Parameters
profile – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.

Returns
A string containing the internal profile information stored in an ICC tag.

Raises
PyCMSError –

PIL.ImageCms.**getProfileName**(*profile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*) → str

(pyCMS) Gets the internal product name for the given profile.

If `profile` isn't a valid CmsProfile object or filename to a profile, a *PyCMSError* is raised If an error occurs while trying to obtain the name tag, a *PyCMSError* is raised.

Use this function to obtain the INTERNAL name of the profile (stored in an ICC tag in the profile itself), usually the one used when the profile was originally created. Sometimes this tag also contains additional information supplied by the creator.

Parameters
profile – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.

Returns
A string containing the internal name of the profile as stored in an ICC tag.

Raises
PyCMSError –

PIL.ImageCms.**get_display_profile**(*handle: SupportsInt | None = None*) → *ImageCmsProfile* | None

(experimental) Fetches the profile for the current display device.

Returns
`None` if the profile is not known.

PIL.ImageCms.**isIntentSupported**(*profile: str | SupportsRead[bytes] | CmsProfile | ImageCmsProfile*, *intent: Intent*, *direction: Direction*) → Literal[-1, 1]

(pyCMS) Checks if a given intent is supported.

Use this function to verify that you can use your desired `intent` with `profile`, and that `profile` can be used for the input/output/proof profile as you desire.

Some profiles are created specifically for one "direction", can cannot be used for others. Some profiles can only be used for certain rendering intents, so it's best to either verify this before trying to create a transform with them (using this function), or catch the potential *PyCMSError* that will occur if they don't support the modes you select.

Parameters

- **profile** – EITHER a valid CmsProfile object, OR a string of the filename of an ICC profile.
- **intent** – Integer (0-3) specifying the rendering intent you wish to use with this profile

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what
  they do.

- **direction** – Integer specifying if the profile is to be used for input, output, or proof

  INPUT = 0 (or use ImageCms.Direction.INPUT) OUTPUT = 1 (or use ImageCms.Direction.OUTPUT) PROOF = 2 (or use ImageCms.Direction.PROOF)

Returns
1 if the intent/direction are supported, -1 if they are not.

Raises
PyCMSError –

PIL.ImageCms.**profileToProfile**(*im:* Image, *inputProfile: str* | SupportsRead[*bytes*] | CmsProfile | ImageCmsProfile, *outputProfile: str* | SupportsRead[*bytes*] | CmsProfile | ImageCmsProfile, *renderingIntent:* Intent = *Intent.PERCEPTUAL*, *outputMode: str* | *None* = *None*, *inPlace: bool* = *False*, *flags:* Flags = <*Flags.NONE: 0*>) → *Image* | None

(pyCMS) Applies an ICC transformation to a given image, mapping from `inputProfile` to `outputProfile`.

If the input or output profiles specified are not valid filenames, a *PyCMSError* will be raised. If `inPlace` is `True` and `outputMode != im.mode`, a *PyCMSError* will be raised. If an error occurs during application of the profiles, a *PyCMSError* will be raised. If `outputMode` is not a mode supported by the `outputProfile` (or by pyCMS), a *PyCMSError* will be raised.

This function applies an ICC transformation to im from `inputProfile`'s color space to `outputProfile`'s color space using the specified rendering intent to decide how to handle out-of-gamut colors.

`outputMode` can be used to specify that a color mode conversion is to be done using these profiles, but the specified profiles must be able to handle that mode. I.e., if converting im from RGB to CMYK using profiles, the input profile must handle RGB data, and the output profile must handle CMYK data.

Parameters

- **im** – An open *Image* object (i.e. Image.new(...) or Image.open(...), etc.)
- **inputProfile** – String, as a valid filename path to the ICC input profile you wish to use for this image, or a profile object
- **outputProfile** – String, as a valid filename path to the ICC output profile you wish to use for this image, or a profile object
- **renderingIntent** – Integer (0-3) specifying the rendering intent you wish to use for the transform

  ImageCms.Intent.PERCEPTUAL = 0 (DEFAULT) ImageCms.Intent.RELATIVE_COLORIMETRIC = 1 ImageCms.Intent.SATURATION = 2 ImageCms.Intent.ABSOLUTE_COLORIMETRIC = 3

  see the pyCMS documentation for details on rendering intents and what they do.
- **outputMode** – A valid PIL mode for the output image (i.e. "RGB", "CMYK", etc.). Note: if rendering the image "inPlace", outputMode MUST be the same mode as the input, or omitted completely. If omitted, the outputMode will be the same as the mode of the input image (im.mode)
- **inPlace** – Boolean. If `True`, the original image is modified in-place, and `None` is returned. If `False` (default), a new *Image* object is returned with the transform applied.
- **flags** – Integer (0-...) specifying additional flags

Returns
Either None or a new *Image* object, depending on the value of `inPlace`

Raises
PyCMSError –

CmsProfile

The ICC color profiles are wrapped in an instance of the class *CmsProfile*. The specification ICC.1:2010 contains more information about the meaning of the values in ICC profiles.

For convenience, all XYZ-values are also given as xyY-values (so they can be easily displayed in a chromaticity diagram, for example).

class PIL.ImageCms.core.**CmsProfile**

creation_date: datetime.datetime | None

Date and time this profile was first created (see 7.2.1 of ICC.1:2010).

version: float

The version number of the ICC standard that this profile follows (e.g. `2.0`).

icc_version: int

Same as `version`, but in encoded format (see 7.2.4 of ICC.1:2010).

device_class: str

4-character string identifying the profile class. One of `scnr`, `mntr`, `prtr`, `link`, `spac`, `abst`, `nmcl` (see 7.2.5 of ICC.1:2010 for details).

xcolor_space: str

4-character string (padded with whitespace) identifying the color space, e.g. `XYZ␣`, `RGB␣` or `CMYK` (see 7.2.6 of ICC.1:2010 for details).

connection_space: str

4-character string (padded with whitespace) identifying the color space on the B-side of the transform (see 7.2.7 of ICC.1:2010 for details).

header_flags: int

The encoded header flags of the profile (see 7.2.11 of ICC.1:2010 for details).

header_manufacturer: str

4-character string (padded with whitespace) identifying the device manufacturer, which shall match the signature contained in the appropriate section of the ICC signature registry found at www.color.org (see 7.2.12 of ICC.1:2010).

header_model: str

4-character string (padded with whitespace) identifying the device model, which shall match the signature contained in the appropriate section of the ICC signature registry found at www.color.org (see 7.2.13 of ICC.1:2010).

attributes: int

Flags used to identify attributes unique to the particular device setup for which the profile is applicable (see 7.2.14 of ICC.1:2010 for details).

rendering_intent: int

The rendering intent to use when combining this profile with another profile (usually overridden at run-time, but provided here for DeviceLink and embedded source profiles, see 7.2.15 of ICC.1:2010).

One of `ImageCms.Intent.ABSOLUTE_COLORIMETRIC`, `ImageCms.Intent.PERCEPTUAL`, `ImageCms.Intent.RELATIVE_COLORIMETRIC` and `ImageCms.Intent.SATURATION`.

profile_id: bytes

A sequence of 16 bytes identifying the profile (via a specially constructed MD5 sum), or 16 binary zeroes if the profile ID has not been calculated (see 7.2.18 of ICC.1:2010).

copyright: str | None

The text copyright information for the profile (see 9.2.21 of ICC.1:2010).

manufacturer: str | None

The (English) display string for the device manufacturer (see 9.2.22 of ICC.1:2010).

model: str | None

The (English) display string for the device model of the device for which this profile is created (see 9.2.23 of ICC.1:2010).

profile_description: str | None

The (English) display string for the profile description (see 9.2.41 of ICC.1:2010).

target: str | None

The name of the registered characterization data set, or the measurement data for a characterization target (see 9.2.14 of ICC.1:2010).

red_colorant: tuple[tuple[float, float, float], tuple[float, float, float]] | None

The first column in the matrix used in matrix/TRC transforms (see 9.2.44 of ICC.1:2010).

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

green_colorant: tuple[tuple[float, float, float], tuple[float, float, float]] | None

The second column in the matrix used in matrix/TRC transforms (see 9.2.30 of ICC.1:2010).

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

blue_colorant: tuple[tuple[float, float, float], tuple[float, float, float]] | None

The third column in the matrix used in matrix/TRC transforms (see 9.2.4 of ICC.1:2010).

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

luminance: tuple[tuple[float, float, float], tuple[float, float, float]] | None

The absolute luminance of emissive devices in candelas per square metre as described by the Y channel (see 9.2.32 of ICC.1:2010).

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

chromaticity: tuple[tuple[float, float, float], tuple[float, float, float], tuple[float, float, float]] | None

The data of the phosphor/colorant chromaticity set used (red, green and blue channels, see 9.2.16 of ICC.1:2010).

The value is in the format `((x, y, Y), (x, y, Y), (x, y, Y))`, if available.

chromatic_adaption: tuple[tuple[tuple[float, float, float], tuple[float, float, float], tuple[float, float, float]], tuple[tuple[float, float, float], tuple[float, float, float], tuple[float, float, float]]] | None

The chromatic adaption matrix converts a color measured using the actual illumination conditions and relative to the actual adopted white, to a color relative to the PCS adopted white, with complete adaptation from the actual adopted white chromaticity to the PCS adopted white chromaticity (see 9.2.15 of ICC.1:2010).

Two 3-tuples of floats are returned in a 2-tuple, one in (X, Y, Z) space and one in (x, y, Y) space.

colorant_table: list[str]

This tag identifies the colorants used in the profile by a unique name and set of PCSXYZ or PCSLAB values (see 9.2.19 of ICC.1:2010).

colorant_table_out: list[str]

This tag identifies the colorants used in the profile by a unique name and set of PCSLAB values (for DeviceLink profiles only, see 9.2.19 of ICC.1:2010).

colorimetric_intent: str | None

4-character string (padded with whitespace) identifying the image state of PCS colorimetry produced using the colorimetric intent transforms (see 9.2.20 of ICC.1:2010 for details).

perceptual_rendering_intent_gamut: str | None

4-character string (padded with whitespace) identifying the (one) standard reference medium gamut (see 9.2.37 of ICC.1:2010 for details).

saturation_rendering_intent_gamut: str | None

4-character string (padded with whitespace) identifying the (one) standard reference medium gamut (see 9.2.37 of ICC.1:2010 for details).

technology: str | None

4-character string (padded with whitespace) identifying the device technology (see 9.2.47 of ICC.1:2010 for details).

media_black_point: tuple[tuple[float, float, float], tuple[float, float, float]] | None

This tag specifies the media black point and is used for generating absolute colorimetry.

This tag was available in ICC 3.2, but it is removed from version 4.

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

media_white_point: tuple[tuple[float, float, float], tuple[float, float, float]] | None

This tag specifies the media white point and is used for generating absolute colorimetry.

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

media_white_point_temperature: float | None

Calculates the white point temperature (see the LCMS documentation for more information).

viewing_condition: str | None

The (English) display string for the viewing conditions (see 9.2.48 of ICC.1:2010).

screening_description: str | None

The (English) display string for the screening conditions.

This tag was available in ICC 3.2, but it is removed from version 4.

red_primary: tuple[tuple[float, float, float], tuple[float, float, float]] | None

The XYZ-transformed of the RGB primary color red (1, 0, 0).

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

green_primary: tuple[tuple[float, float, float], tuple[float, float, float]] | None

The XYZ-transformed of the RGB primary color green (0, 1, 0).

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

blue_primary: tuple[tuple[float, float, float], tuple[float, float, float]] | None

The XYZ-transformed of the RGB primary color blue (0, 0, 1).

The value is in the format `((X, Y, Z), (x, y, Y))`, if available.

is_matrix_shaper: `bool`

True if this profile is implemented as a matrix shaper (see documentation on LCMS).

clut: `dict[int, tuple[bool, bool, bool]] | None`

Returns a dictionary of all supported intents and directions for the CLUT model.

The dictionary is indexed by intents (`ImageCms.Intent.ABSOLUTE_COLORIMETRIC`, `ImageCms.Intent.PERCEPTUAL`, `ImageCms.Intent.RELATIVE_COLORIMETRIC` and `ImageCms.Intent.SATURATION`).

The values are 3-tuples indexed by directions (`ImageCms.Direction.INPUT`, `ImageCms.Direction.OUTPUT`, `ImageCms.Direction.PROOF`).

The elements of the tuple are booleans. If the value is `True`, that intent is supported for that direction.

intent_supported: `dict[int, tuple[bool, bool, bool]] | None`

Returns a dictionary of all supported intents and directions.

The dictionary is indexed by intents (`ImageCms.Intent.ABSOLUTE_COLORIMETRIC`, `ImageCms.Intent.PERCEPTUAL`, `ImageCms.Intent.RELATIVE_COLORIMETRIC` and `ImageCms.Intent.SATURATION`).

The values are 3-tuples indexed by directions (`ImageCms.Direction.INPUT`, `ImageCms.Direction.OUTPUT`, `ImageCms.Direction.PROOF`).

The elements of the tuple are booleans. If the value is `True`, that intent is supported for that direction.

There is one function defined on the class:

is_intent_supported(*intent: int*, *direction: int*, */ (Positional-only parameter separator (PEP 570))*)

Returns if the intent is supported for the given direction.

Note that you can also get this information for all intents and directions with *intent_supported*.

Parameters

- **intent** – One of `ImageCms.Intent.ABSOLUTE_COLORIMETRIC`, `ImageCms.Intent.PERCEPTUAL`, `ImageCms.Intent.RELATIVE_COLORIMETRIC` and `ImageCms.Intent.SATURATION`.
- **direction** – One of `ImageCms.Direction.INPUT`, `ImageCms.Direction.OUTPUT` and `ImageCms.Direction.PROOF`

Returns

Boolean if the intent and direction is supported.

1.3.4 *ImageColor* module

The *ImageColor* module contains color tables and converters from CSS3-style color specifiers to RGB tuples. This module is used by *PIL.Image.new()* and the *ImageDraw* module, among others.

Color names

The ImageColor module supports the following string formats:

- Hexadecimal color specifiers, given as `#rgb`, `#rgba`, `#rrggbb` or `#rrggbbaa`, where `r` is red, `g` is green, `b` is blue and `a` is alpha (also called 'opacity'). For example, `#ff0000` specifies pure red, and `#ff0000cc` specifies red with 80% opacity (`cc` is 204 in decimal form, and 204 / 255 = 0.8).
- RGB functions, given as `rgb(red, green, blue)` where the color values are integers in the range 0 to 255. Alternatively, the color values can be given as three percentages (0% to 100%). For example, `rgb(255,0,0)` and `rgb(100%,0%,0%)` both specify pure red.

- RGBA functions, given as `rgba(red, green, blue, alpha)` where the color values and the alpha value are integers in the range 0 to 255. For example, `rgba(255,0,0,128)` specifies pure red with 50% opacity.
- Hue-Saturation-Lightness (HSL) functions, given as `hsl(hue, saturation%, lightness%)` where hue is the color given as an angle between 0 and 360 (red=0, green=120, blue=240), saturation is a value between 0% and 100% (gray=0%, full color=100%), and lightness is a value between 0% and 100% (black=0%, normal=50%, white=100%). For example, `hsl(0,100%,50%)` is pure red.
- Hue-Saturation-Value (HSV) functions, given as `hsv(hue, saturation%, value%)` where hue and saturation are the same as HSL, and value is between 0% and 100% (black=0%, normal=100%). For example, `hsv(0, 100%,100%)` is pure red. This format is also known as Hue-Saturation-Brightness (HSB), and can be given as `hsb(hue, saturation%, brightness%)`, where each of the values are used as they are in HSV.
- Common HTML color names. The *ImageColor* module provides some 140 standard color names, based on the colors supported by the X Window system and most web browsers. color names are case insensitive. For example, `red` and `Red` both specify pure red.

Functions

`PIL.ImageColor.`**`getrgb`**(*color*)

Convert a color string to an RGB tuple. If the string cannot be parsed, this function raises a `ValueError` exception.

Added in version 1.1.4.

`PIL.ImageColor.`**`getcolor`**(*color*, *mode*)

Same as *getrgb()*, but converts the RGB value to a grayscale value if the mode is not color or a palette image. If the string cannot be parsed, this function raises a `ValueError` exception.

Added in version 1.1.4.

1.3.5 *ImageDraw* module

The *ImageDraw* module provides simple 2D graphics for *Image* objects. You can use this module to create new images, annotate or retouch existing images, and to generate graphics on the fly for web use.

For a more advanced drawing library for PIL, see the aggdraw module.

Example: Draw a gray cross over an image

```
import sys
from PIL import Image, ImageDraw

with Image.open("hopper.jpg") as im:

    draw = ImageDraw.Draw(im)
    draw.line((0, 0) + im.size, fill=128)
    draw.line((0, im.size[1], im.size[0], 0), fill=128)

    # write to stdout
    im.save(sys.stdout, "PNG")
```

Concepts

Coordinates

The graphics interface uses the same coordinate system as PIL itself, with (0, 0) in the upper left corner. Any pixels drawn outside of the image bounds will be discarded.

Colors

To specify colors, you can use numbers or tuples just as you would use with *PIL.Image.new()*. See *Colors* for more information.

For palette images (mode "P"), use integers as color indexes. In 1.1.4 and later, you can also use RGB 3-tuples or color names (see below). The drawing layer will automatically assign color indexes, as long as you don't draw with more than 256 colors.

Color names

See *Color names* for the color names supported by Pillow.

Alpha channel

By default, when drawing onto an existing image, the image's pixel values are simply replaced by the new color:

```
im = Image.new("RGBA", (1, 1), (255, 0, 0))
d = ImageDraw.Draw(im)
d.rectangle((0, 0, 1, 1), (0, 255, 0, 127))
assert im.getpixel((0, 0)) == (0, 255, 0, 127)

# Alpha channel values have no effect when drawing with RGB mode
im = Image.new("RGB", (1, 1), (255, 0, 0))
d = ImageDraw.Draw(im)
d.rectangle((0, 0, 1, 1), (0, 255, 0, 127))
assert im.getpixel((0, 0)) == (0, 255, 0)
```

If you would like to combine translucent color with an RGB image, then initialize the ImageDraw instance with the RGBA mode:

```
from PIL import Image, ImageDraw
im = Image.new("RGB", (1, 1), (255, 0, 0))
d = ImageDraw.Draw(im, "RGBA")
d.rectangle((0, 0, 1, 1), (0, 255, 0, 127))
assert im.getpixel((0, 0)) == (128, 127, 0)
```

If you would like to combine translucent color with an RGBA image underneath, you will need to combine multiple images:

```
from PIL import Image, ImageDraw
im = Image.new("RGBA", (1, 1), (255, 0, 0, 255))
im2 = Image.new("RGBA", (1, 1))
d = ImageDraw.Draw(im2)
d.rectangle((0, 0, 1, 1), (0, 255, 0, 127))
im.paste(im2.convert("RGB"), mask=im2)
assert im.getpixel((0, 0)) == (128, 127, 0, 255)
```

Fonts

PIL can use bitmap fonts or OpenType/TrueType fonts.

Bitmap fonts are stored in PIL's own format, where each font typically consists of two files, one named .pil and the other usually named .pbm. The former contains font metrics, the latter raster data.

To load a bitmap font, use the load functions in the *ImageFont* module.

To load an OpenType/TrueType font, use the truetype function in the *ImageFont* module. Note that this function depends on third-party libraries, and may not be available in all PIL builds.

Example: Draw partial opacity text

```
from PIL import Image, ImageDraw, ImageFont

# get an image
with Image.open("Pillow/Tests/images/hopper.png").convert("RGBA") as base:

    # make a blank image for the text, initialized to transparent text color
    txt = Image.new("RGBA", base.size, (255, 255, 255, 0))

    # get a font
    fnt = ImageFont.truetype("Pillow/Tests/fonts/FreeMono.ttf", 40)
    # get a drawing context
    d = ImageDraw.Draw(txt)

    # draw text, half opacity
    d.text((10, 10), "Hello", font=fnt, fill=(255, 255, 255, 128))
    # draw text, full opacity
    d.text((10, 60), "World", font=fnt, fill=(255, 255, 255, 255))

    out = Image.alpha_composite(base, txt)

    out.show()
```

Example: Draw multiline text

```
from PIL import Image, ImageDraw, ImageFont

# create an image
out = Image.new("RGB", (150, 100), (255, 255, 255))

# get a font
fnt = ImageFont.truetype("Pillow/Tests/fonts/FreeMono.ttf", 40)
# get a drawing context
d = ImageDraw.Draw(out)

# draw multiline text
d.multiline_text((10, 10), "Hello\nWorld", font=fnt, fill=(0, 0, 0))

out.show()
```

Functions

PIL.ImageDraw.**Draw**(*im*, *mode=None*)

Creates an object that can be used to draw in the given image.

Note that the image will be modified in place.

Parameters

- **im** – The image to draw in.

- **mode** – Optional mode to use for color values. For RGB images, this argument can be RGB or RGBA (to blend the drawing into the image). For all other modes, this argument must be the same as the image mode. If omitted, the mode defaults to the mode of the image.

Attributes

ImageDraw.**fill**: bool = False

Selects whether *ImageDraw.ink* should be used as a fill or outline color.

ImageDraw.**font**

The current default font.

Can be set per instance:

```
from PIL import ImageDraw, ImageFont
draw = ImageDraw.Draw(image)
draw.font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
```

Or globally for all future ImageDraw instances:

```
from PIL import ImageDraw, ImageFont
ImageDraw.ImageDraw.font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
```

ImageDraw.**fontmode**

The current font drawing mode.

Set to "1" to disable antialiasing or "L" to enable it.

ImageDraw.**ink**: int

The internal representation of the current default color.

Methods

ImageDraw.**getfont**()

Get the current default font, *ImageDraw.font*.

If the current default font is None, it is initialized with *ImageFont.load_default()*.

Returns
An image font.

ImageDraw.**arc**(*xy*, *start*, *end*, *fill=None*, *width=0*)

Draws an arc (a portion of a circle outline) between the start and end angles, inside the given bounding box.

Parameters

- **xy** – Two points to define the bounding box. Sequence of `[(x0, y0), (x1, y1)]` or `[x0, y0, x1, y1]`, where `x1 >= x0` and `y1 >= y0`.
- **start** – Starting angle, in degrees. Angles are measured from 3 o'clock, increasing clockwise.
- **end** – Ending angle, in degrees.
- **fill** – Color to use for the arc.
- **width** – The line width, in pixels.

  Added in version 5.3.0.

ImageDraw.**bitmap**(*xy*, *bitmap*, *fill=None*)

Draws a bitmap (mask) at the given position, using the current fill color for the non-zero portions. The bitmap should be a valid transparency mask (mode "1") or matte (mode "L" or "RGBA").

This is equivalent to doing `image.paste(xy, color, bitmap)`.

To paste pixel data into an image, use the *paste()* method on the image itself.

ImageDraw.**chord**(*xy*, *start*, *end*, *fill=None*, *outline=None*, *width=1*)

Same as *arc()*, but connects the end points with a straight line.

Parameters

- **xy** – Two points to define the bounding box. Sequence of `[(x0, y0), (x1, y1)]` or `[x0, y0, x1, y1]`, where `x1 >= x0` and `y1 >= y0`.
- **outline** – Color to use for the outline.
- **fill** – Color to use for the fill.
- **width** – The line width, in pixels.

  Added in version 5.3.0.

ImageDraw.**circle**(*xy*, *radius*, *fill=None*, *outline=None*, *width=1*)

Draws a circle with a given radius centering on a point.

Added in version 10.4.0.

Parameters

- **xy** – The point for the center of the circle, e.g. `(x, y)`.
- **radius** – Radius of the circle.
- **outline** – Color to use for the outline.
- **fill** – Color to use for the fill.
- **width** – The line width, in pixels.

ImageDraw.**ellipse**(*xy*, *fill=None*, *outline=None*, *width=1*)

Draws an ellipse inside the given bounding box.

Parameters

- **xy** – Two points to define the bounding box. Sequence of either `[(x0, y0), (x1, y1)]` or `[x0, y0, x1, y1]`, where `x1 >= x0` and `y1 >= y0`.
- **outline** – Color to use for the outline.
- **fill** – Color to use for the fill.
- **width** – The line width, in pixels.

  Added in version 5.3.0.

ImageDraw.**line**(*xy*, *fill=None*, *width=0*, *joint=None*)

Draws a line between the coordinates in the `xy` list. The coordinate pixels are included in the drawn line.

Parameters

- **xy** – Sequence of either 2-tuples like `[(x, y), (x, y), ...]` or numeric values like `[x, y, x, y, ...]`.
- **fill** – Color to use for the line.

- **width** – The line width, in pixels.

  Added in version 1.1.5.

  > **Note**
  >
  > This option was broken until version 1.1.6.

- **joint** – Joint type between a sequence of lines. It can be `"curve"`, for rounded edges, or `None`.

  Added in version 5.3.0.

`ImageDraw.pieslice`(*xy*, *start*, *end*, *fill=None*, *outline=None*, *width=1*)

Same as arc, but also draws straight lines between the end points and the center of the bounding box.

Parameters

- **xy** – Two points to define the bounding box. Sequence of `[(x0, y0), (x1, y1)]` or `[x0, y0, x1, y1]`, where `x1 >= x0` and `y1 >= y0`.
- **start** – Starting angle, in degrees. Angles are measured from 3 o'clock, increasing clockwise.
- **end** – Ending angle, in degrees.
- **fill** – Color to use for the fill.
- **outline** – Color to use for the outline.
- **width** – The line width, in pixels.

  Added in version 5.3.0.

`ImageDraw.point`(*xy*, *fill=None*)

Draws points (individual pixels) at the given coordinates.

Parameters

- **xy** – Sequence of either 2-tuples like `[(x, y), (x, y), ...]` or numeric values like `[x, y, x, y, ...]`.
- **fill** – Color to use for the point.

`ImageDraw.polygon`(*xy*, *fill=None*, *outline=None*, *width=1*)

Draws a polygon.

The polygon outline consists of straight lines between the given coordinates, plus a straight line between the last and the first coordinate. The coordinate pixels are included in the drawn polygon.

Parameters

- **xy** – Sequence of either 2-tuples like `[(x, y), (x, y), ...]` or numeric values like `[x, y, x, y, ...]`.
- **fill** – Color to use for the fill.
- **outline** – Color to use for the outline.
- **width** – The line width, in pixels.

ImageDraw.**regular_polygon**(*bounding_circle*, *n_sides*, *rotation=0*, *fill=None*, *outline=None*, *width=1*)

Draws a regular polygon inscribed in `bounding_circle`, with `n_sides`, and rotation of `rotation` degrees.

Parameters

- **bounding_circle** – The bounding circle is a tuple defined by a point and radius. (e.g. `bounding_circle=(x, y, r)` or `((x, y), r)`). The polygon is inscribed in this circle.
- **n_sides** – Number of sides (e.g. `n_sides=3` for a triangle, `6` for a hexagon).
- **rotation** – Apply an arbitrary rotation to the polygon (e.g. `rotation=90`, applies a 90 degree rotation).
- **fill** – Color to use for the fill.
- **outline** – Color to use for the outline.
- **width** – The line width, in pixels.

ImageDraw.**rectangle**(*xy*, *fill=None*, *outline=None*, *width=1*)

Draws a rectangle.

Parameters

- **xy** – Two points to define the bounding box. Sequence of either `[(x0, y0), (x1, y1)]` or `[x0, y0, x1, y1]`, where `x1 >= x0` and `y1 >= y0`. The bounding box is inclusive of both endpoints.
- **fill** – Color to use for the fill.
- **outline** – Color to use for the outline.
- **width** – The line width, in pixels.

  Added in version 5.3.0.

ImageDraw.**rounded_rectangle**(*xy*, *radius=0*, *fill=None*, *outline=None*, *width=1*, *corners=None*)

Draws a rounded rectangle.

Parameters

- **xy** – Two points to define the bounding box. Sequence of either `[(x0, y0), (x1, y1)]` or `[x0, y0, x1, y1]`, where `x1 >= x0` and `y1 >= y0`. The bounding box is inclusive of both endpoints.
- **radius** – Radius of the corners, limited to half of the smallest dimension of the bounding box.
- **fill** – Color to use for the fill.
- **outline** – Color to use for the outline.
- **width** – The line width, in pixels.
- **corners** – A tuple of whether to round each corner, `(top_left, top_right, bottom_right, bottom_left)`. Keyword-only argument.

Added in version 8.2.0.

ImageDraw.**shape**(*shape*, *fill=None*, *outline=None*)

> **Warning**
>
> This method is experimental.

Draw a shape.

ImageDraw.**text**(*xy*, *text*, *fill=None*, *font=None*, *anchor=None*, *spacing=4*, *align='left'*, *direction=None*, *features=None*, *language=None*, *stroke_width=0*, *stroke_fill=None*, *embedded_color=False*, *font_size=None*)

Draws the string at the given position.

Parameters

- **xy** – The anchor coordinates of the text.
- **text** – String to be drawn. If it contains any newline characters, the text is passed on to *multiline_text()*.
- **fill** – Color to use for the text.
- **font** – An *ImageFont* instance.
- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details. This parameter is ignored for non-TrueType fonts.

  > **Note**
  >
  > This parameter was present in earlier versions of Pillow, but implemented only in version 8.0.0.

- **spacing** – If the text is passed on to *multiline_text()*, the number of pixels between lines.
- **align** – If the text is passed on to *multiline_text()*, `"left"`, `"center"`, `"right"` or `"justify"`. Determines the relative alignment of lines. Use the `anchor` parameter to specify the alignment to `xy`.

  Added in version 11.2.1: `"justify"`

- **direction** – Direction of the text. It can be `"rtl"` (right to left), `"ltr"` (left to right) or `"ttb"` (top to bottom). Requires libraqm.

  Added in version 4.2.0.

- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example `"dlig"` or `"ss01"`, but can be also used to turn off default font features, for example `"-liga"` to disable ligatures or `"-kern"` to disable kerning. To get all supported features, see OpenType docs. Requires libraqm.

  Added in version 4.2.0.

- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code. Requires libraqm.

  Added in version 6.0.0.

- **stroke_width** – The width of the text stroke.

  Added in version 6.2.0.

- **stroke_fill** – Color to use for the text stroke. If not given, will default to the `fill` parameter.

  Added in version 6.2.0.

- **embedded_color** – Whether to use font embedded color glyphs (COLR, CBDT, SBIX).

  Added in version 8.0.0.

- **font_size** –

  If `font` is not provided, then the size to use for the default font. Keyword-only argument.

  Added in version 10.1.0.

`ImageDraw.`**`multiline_text`**(*xy*, *text*, *fill=None*, *font=None*, *anchor=None*, *spacing=4*, *align='left'*, *direction=None*, *features=None*, *language=None*, *stroke_width=0*, *stroke_fill=None*, *embedded_color=False*, *font_size=None*)

Draws the string at the given position.

Parameters

- **xy** – The anchor coordinates of the text.
- **text** – String to be drawn.
- **fill** – Color to use for the text.
- **font** – An *ImageFont* instance.
- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details. This parameter is ignored for non-TrueType fonts.

  Note

  This parameter was present in earlier versions of Pillow, but implemented only in version 8.0.0.

- **spacing** – The number of pixels between lines.
- **align** – `"left"`, `"center"`, `"right"` or `"justify"`. Determines the relative alignment of lines. Use the `anchor` parameter to specify the alignment to `xy`.

  Added in version 11.2.1: `"justify"`

- **direction** – Direction of the text. It can be `"rtl"` (right to left), `"ltr"` (left to right) or `"ttb"` (top to bottom). Requires libraqm.

  Added in version 4.2.0.

- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example `"dlig"` or `"ss01"`, but can be also used to turn off default font features, for example `"-liga"` to disable ligatures or `"-kern"` to disable kerning. To get all supported features, see OpenType docs. Requires libraqm.

  Added in version 4.2.0.

- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code. Requires libraqm.

  Added in version 6.0.0.

- **stroke_width** – The width of the text stroke.

  Added in version 6.2.0.

- **stroke_fill** –

  Color to use for the text stroke. If not given, will default to the `fill` parameter.

  Added in version 6.2.0.

- **embedded_color** – Whether to use font embedded color glyphs (COLR, CBDT, SBIX).

  Added in version 8.0.0.

- **font_size** –

  If `font` is not provided, then the size to use for the default font. Keyword-only argument.

  Added in version 10.1.0.

ImageDraw.**textlength**(*text*, *font=None*, *direction=None*, *features=None*, *language=None*, *embedded_color=False*, *font_size=None*)

Returns length (in pixels with 1/64 precision) of given text when rendered in font with provided direction, features, and language.

This is the amount by which following text should be offset. Text bounding box may extend past the length in some fonts, e.g. when using italics or accents.

The result is returned as a float; it is a whole number if using basic layout.

Note that the sum of two lengths may not equal the length of a concatenated string due to kerning. If you need to adjust for kerning, include the following character and subtract its length.

For example, instead of

```
hello = draw.textlength("Hello", font)
world = draw.textlength("World", font)
hello_world = hello + world  # not adjusted for kerning
assert hello_world == draw.textlength("HelloWorld", font)  # may fail
```

use

```
hello = draw.textlength("HelloW", font) - draw.textlength(
    "W", font
)  # adjusted for kerning
world = draw.textlength("World", font)
hello_world = hello + world  # adjusted for kerning
assert hello_world == draw.textlength("HelloWorld", font)  # True
```

or disable kerning with (requires libraqm)

```
hello = draw.textlength("Hello", font, features=["-kern"])
world = draw.textlength("World", font, features=["-kern"])
hello_world = hello + world  # kerning is disabled, no need to adjust
assert hello_world == draw.textlength("HelloWorld", font, features=["-kern"])  #␣
↪True
```

See also

PIL.ImageText.Text.get_length()

Added in version 8.0.0.

Parameters

- **text** – Text to be measured. May not contain any newline characters.
- **font** – An *ImageFont* instance.
- **direction** – Direction of the text. It can be `"rtl"` (right to left), `"ltr"` (left to right) or `"ttb"` (top to bottom). Requires libraqm.
- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example `"dlig"` or `"ss01"`, but can be also used to turn off default font features, for example `"-liga"` to disable ligatures or `"-kern"` to disable kerning. To get all supported features, see OpenType docs. Requires libraqm.
- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code. Requires libraqm.
- **embedded_color** – Whether to use font embedded color glyphs (COLR, CBDT, SBIX).
- **font_size** –

  If `font` is not provided, then the size to use for the default font. Keyword-only argument.

  Added in version 10.1.0.

Returns

Either width for horizontal text, or height for vertical text.

ImageDraw.**textbbox**(*xy*, *text*, *font=None*, *anchor=None*, *spacing=4*, *align='left'*, *direction=None*, *features=None*, *language=None*, *stroke_width=0*, *embedded_color=False*, *font_size=None*)

Returns bounding box (in pixels) of given text relative to given anchor when rendered in font with provided direction, features, and language. Only supported for TrueType fonts.

Use *textlength()* to get the offset of following text with 1/64 pixel precision. The bounding box includes extra margins for some fonts, e.g. italics or accents.

Added in version 8.0.0.

Parameters

- **xy** – The anchor coordinates of the text.
- **text** – Text to be measured. If it contains any newline characters, the text is passed on to *multiline_textbbox()*.

- **font** – A *FreeTypeFont* instance.
- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details. This parameter is ignored for non-TrueType fonts.
- **spacing** – If the text is passed on to *multiline_textbbox()*, the number of pixels between lines.
- **align** – If the text is passed on to *multiline_textbbox()*, `"left"`, `"center"`, `"right"` or `"justify"`. Determines the relative alignment of lines. Use the `anchor` parameter to specify the alignment to `xy`.

  Added in version 11.2.1: `"justify"`
- **direction** – Direction of the text. It can be `"rtl"` (right to left), `"ltr"` (left to right) or `"ttb"` (top to bottom). Requires libraqm.
- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example `"dlig"` or `"ss01"`, but can be also used to turn off default font features, for example `"-liga"` to disable ligatures or `"-kern"` to disable kerning. To get all supported features, see OpenType docs. Requires libraqm.
- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code. Requires libraqm.
- **stroke_width** – The width of the text stroke.
- **embedded_color** – Whether to use font embedded color glyphs (COLR, CBDT, SBIX).
- **font_size** –

  If `font` is not provided, then the size to use for the default font. Keyword-only argument.

  Added in version 10.1.0.

Returns

`(left, top, right, bottom)` bounding box

ImageDraw.**multiline_textbbox**(*xy*, *text*, *font=None*, *anchor=None*, *spacing=4*, *align='left'*, *direction=None*, *features=None*, *language=None*, *stroke_width=0*, *embedded_color=False*, *font_size=None*)

Returns bounding box (in pixels) of given text relative to given anchor when rendered in font with provided direction, features, and language. Only supported for TrueType fonts.

Use *textlength()* to get the offset of following text with 1/64 pixel precision. The bounding box includes extra margins for some fonts, e.g. italics or accents.

See also

PIL.ImageText.Text.get_bbox()

Added in version 8.0.0.

Parameters

- **xy** – The anchor coordinates of the text.

- **text** – Text to be measured.
- **font** – A *FreeTypeFont* instance.
- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details. This parameter is ignored for non-TrueType fonts.
- **spacing** – The number of pixels between lines.
- **align** – `"left"`, `"center"`, `"right"` or `"justify"`. Determines the relative alignment of lines. Use the `anchor` parameter to specify the alignment to `xy`.

  Added in version 11.2.1: `"justify"`
- **direction** – Direction of the text. It can be `"rtl"` (right to left), `"ltr"` (left to right) or `"ttb"` (top to bottom). Requires libraqm.
- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example `"dlig"` or `"ss01"`, but can be also used to turn off default font features, for example `"-liga"` to disable ligatures or `"-kern"` to disable kerning. To get all supported features, see OpenType docs. Requires libraqm.
- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code. Requires libraqm.
- **stroke_width** – The width of the text stroke.
- **embedded_color** – Whether to use font embedded color glyphs (COLR, CBDT, SBIX).
- **font_size** –

  If `font` is not provided, then the size to use for the default
  font. Keyword-only argument.

  Added in version 10.1.0.

Returns
`(left, top, right, bottom)` bounding box

`PIL.ImageDraw.`**`getdraw`**(*im=None*, *hints=None*)

> **Warning**
>
> This method is experimental.

A more advanced 2D drawing interface for PIL images, based on the WCK interface.

Parameters

- **im** – The image to draw in.
- **hints** – An optional list of hints.

Returns
A (drawing context, drawing resource factory) tuple.

`PIL.ImageDraw.`**`floodfill`**(*image:* Image, *xy: tuple[int, int]*, *value: float | tuple[int, ...]*, *border: float | tuple[int, ...] | None = None*, *thresh: float = 0*) → None

> **Warning**
>
> This method is experimental.

Fills a bounded region with a given color.

Parameters

- **image** – Target image.
- **xy** – Seed position (a 2-item coordinate tuple). See *Coordinate system*.
- **value** – Fill color.
- **border** – Optional border value. If given, the region consists of pixels with a color different from the border color. If not given, the region consists of pixels having the same color as the seed pixel.
- **thresh** – Optional threshold value which specifies a maximum tolerable difference of a pixel value from the 'background' in order for it to be replaced. Useful for filling regions of non-homogeneous, but similar, colors.

1.3.6 *ImageEnhance* module

The *ImageEnhance* module contains a number of classes that can be used for image enhancement.

Example: Vary the sharpness of an image

```
from PIL import ImageEnhance

enhancer = ImageEnhance.Sharpness(image)

for i in range(8):
    factor = i / 4.0
    enhancer.enhance(factor).show(f"Sharpness {factor:f}")
```

Also see the `enhancer.py` demo program in the `Scripts/` directory.

Classes

All enhancement classes implement a common interface, containing a single method:

class PIL.ImageEnhance.**_Enhance**

enhance(*factor*)

Returns an enhanced image.

Parameters

factor – A floating point value controlling the enhancement. Factor 1.0 always returns a copy of the original image, lower factors mean less color (brightness, contrast, etc), and higher values more. There are no restrictions on this value.

class PIL.ImageEnhance.**Color**(*image*)

Adjust image color balance.

This class can be used to adjust the colour balance of an image, in a manner similar to the controls on a colour TV set. An *enhancement factor* of 0.0 gives a black and white image. A factor of 1.0 gives the original image.

class PIL.ImageEnhance.**Contrast**(*image*)

Adjust image contrast.

This class can be used to control the contrast of an image, similar to the contrast control on a TV set. An *enhancement factor* of 0.0 gives a solid gray image, a factor of 1.0 gives the original image, and greater values increase the contrast of the image.

class PIL.ImageEnhance.**Brightness**(*image*)

Adjust image brightness.

This class can be used to control the brightness of an image. An *enhancement factor* of 0.0 gives a black image, a factor of 1.0 gives the original image, and greater values increase the brightness of the image.

class PIL.ImageEnhance.**Sharpness**(*image*)

Adjust image sharpness.

This class can be used to adjust the sharpness of an image. An *enhancement factor* of 0.0 gives a blurred image, a factor of 1.0 gives the original image, and a factor of 2.0 gives a sharpened image.

1.3.7 *ImageFile* module

The *ImageFile* module provides support functions for the image open and save functions.

In addition, it provides a *Parser* class which can be used to decode an image piece by piece (e.g. while receiving it over a network connection). This class implements the same consumer interface as the standard **sgmllib** and **xmllib** modules.

Example: Parse an image

```
from PIL import ImageFile

fp = open("hopper.ppm", "rb")

p = ImageFile.Parser()

while 1:
    s = fp.read(1024)
    if not s:
        break
    p.feed(s)

im = p.close()

im.save("copy.jpg")
```

Classes

class PIL.ImageFile.**_Tile**

Bases: NamedTuple

_Tile(codec_name, extents, offset, args)

codec_name: *str*

Alias for field number 0

extents: *tuple[int, int, int, int] | None*

Alias for field number 1

offset: int

Alias for field number 2

args: tuple[Any, ...] | str | None

Alias for field number 3

class PIL.ImageFile.**Parser**

Incremental image parser. This class implements the standard feed/close consumer interface.

close() → *Image*

(Consumer) Close the stream.

Returns
An image object.

Raises
OSError – If the parser failed to parse the image file either because it cannot be identified or cannot be decoded.

feed(*data: bytes*) → None

(Consumer) Feed data to the parser.

Parameters
data – A string buffer.

Raises
OSError – If the parser failed to parse the image file.

reset() → None

(Consumer) Reset the parser. Note that you can only call this method immediately after you've created a parser; parser instances cannot be reused.

class PIL.ImageFile.**PyCodec**

cleanup() → None

Override to perform codec specific cleanup

Returns
None

init(*args: tuple[Any, ...]*) → None

Override to perform codec specific initialization

Parameters
args – Tuple of arg items from the tile entry

Returns
None

setfd(*fd: IO[bytes]*) → None

Called from ImageFile to set the Python file-like object

Parameters
fd – A Python file-like object

Returns
None

setimage(*im:* Image.core.ImagingCore, *extents: tuple[int, int, int, int] | None = None*) → None

Called from ImageFile to set the core output image for the codec

Parameters

- **im** – A core image object
- **extents** – a 4 tuple of (x0, y0, x1, y1) defining the rectangle for this tile

Returns
None

class PIL.ImageFile.**PyDecoder**

Bases: *PyCodec*

Python implementation of a format decoder. Override this class and add the decoding logic in the *decode()* method.

See *Writing Your Own File Codec in Python*

decode(*buffer: bytes | bytearray | memoryview | SupportsArrayInterface*) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

set_as_raw(*data: bytes | bytearray*, *rawmode: str | None = None*, *extra: tuple[Any, ...] = ()*) → None

Convenience method to set the internal image from a stream of raw data

Parameters

- **data** – Bytes to be set
- **rawmode** – The rawmode to be used for the decoder. If not specified, it will default to the mode of the image
- **extra** – Extra arguments for the decoder.

Returns
None

class PIL.ImageFile.**PyEncoder**

Bases: *PyCodec*

Python implementation of a format encoder. Override this class and add the decoding logic in the *encode()* method.

See *Writing Your Own File Codec in Python*

encode(*bufsize: int*) → tuple[int, int, bytes]

Override to perform the encoding process.

Parameters
bufsize – Buffer size.

Returns
A tuple of `(bytes encoded, errcode, bytes)`. If finished with encoding return 1 for the error code. Err codes are from *ImageFile.ERRORS*.

encode_to_file(*fh: int*, *bufsize: int*) → int

Parameters

- **fh** – File handle.

- **bufsize** – Buffer size.

Returns
If finished successfully, return 0. Otherwise, return an error code. Err codes are from *ImageFile.ERRORS*.

encode_to_pyfd() → tuple[int, int]

If `pushes_fd` is `True`, then this method will be used, and `encode()` will only be called once.

Returns
A tuple of `(bytes consumed, errcode)`. Err codes are from *ImageFile.ERRORS*.

class PIL.ImageFile.**ImageFile**

Bases: *Image*

Base class for image file format handlers.

custom_mimetype: str | None

tile: list[*_Tile*]

A list of tile descriptors

decoderconfig: tuple[Any, ...]

fp: IO[bytes] | None

close() → None

Closes the file pointer, if possible.

This operation will destroy the image core and release its memory. The image data will be unusable afterward.

This function is required to close images that have multiple frames or have not had their file read and closed by the *load()* method. See *File handling in Pillow* for more information.

get_child_images() → list[*ImageFile*]

get_format_mimetype() → str | None

verify() → None

Check file integrity

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

load_prepare() → None

load_end() → None

class PIL.ImageFile.**StubHandler**

Bases: ABC

class PIL.ImageFile.**StubImageFile**

Bases: *ImageFile*

Base class for stub image loaders.

A stub loader is an image loader that can identify files of a certain format, but relies on external code to load the file.

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

Constants

PIL.ImageFile.**LOAD_TRUNCATED_IMAGES = False**

Whether or not to load truncated image files. User code may change this.

PIL.ImageFile.**MAXBLOCK = 65536**

By default, Pillow processes image data in blocks. This helps to prevent excessive use of resources. Codecs may disable this behaviour with `_pulls_fd` or `_pushes_fd`.

When reading an image, this is the number of bytes to read at once.

When writing an image, this is the number of bytes to write at once. If the image width times 4 is greater, then that will be used instead. Plugins may also set a greater number.

User code may set this to another number.

PIL.ImageFile.**ERRORS**

Dict of known error codes returned from *PyDecoder.decode()*, *PyEncoder.encode()* *PyEncoder.encode_to_pyfd()* and *PyEncoder.encode_to_file()*.

1.3.8 *ImageFilter* module

The *ImageFilter* module contains definitions for a pre-defined set of filters, which can be used with the *Image.filter()* method.

Example: Filter an image

```
from PIL import ImageFilter

im1 = im.filter(ImageFilter.BLUR)

im2 = im.filter(ImageFilter.MinFilter(3))
im3 = im.filter(ImageFilter.MinFilter)  # same as MinFilter(3)
```

Filters

Pillow provides the following set of predefined image enhancement filters:

- **BLUR**
- **CONTOUR**
- **DETAIL**
- **EDGE_ENHANCE**
- **EDGE_ENHANCE_MORE**
- **EMBOSS**
- **FIND_EDGES**
- **SHARPEN**
- **SMOOTH**
- **SMOOTH_MORE**

class PIL.ImageFilter.**Color3DLUT**(*size: int | tuple[int, int, int]*, *table: Sequence[float] | Sequence[Sequence[int]] | NumpyArray*, *channels: int = 3*, *target_mode: str | None = None*, ***kwargs: bool*)

Three-dimensional color lookup table.

Transforms 3-channel pixels using the values of the channels as coordinates in the 3D lookup table and interpolating the nearest elements.

This method allows you to apply almost any color transformation in constant time by using pre-calculated decimated tables.

Added in version 5.2.0.

Parameters

- **size** – Size of the table. One int or tuple of (int, int, int). Minimal size in any dimension is 2, maximum is 65.
- **table** – Flat lookup table. A list of `channels * size**3` float elements or a list of `size**3` channels-sized tuples with floats. Channels are changed first, then first dimension, then second, then third. Value 0.0 corresponds lowest value of output, 1.0 highest.
- **channels** – Number of channels in the table. Could be 3 or 4. Default is 3.
- **target_mode** – A mode for the result image. Should have not less than `channels` channels. Default is `None`, which means that mode wouldn't be changed.

classmethod generate(*size: int | tuple[int, int, int]*, *callback: Callable[[float, float, float], tuple[float, ...]]*, *channels: int = 3*, *target_mode: str | None = None*) → *Color3DLUT*

Generates new LUT using provided callback.

Parameters

- **size** – Size of the table. Passed to the constructor.
- **callback** – Function with three parameters which correspond three color channels. Will be called `size**3` times with values from 0.0 to 1.0 and should return a tuple with `channels` elements.
- **channels** – The number of channels which should return callback.
- **target_mode** – Passed to the constructor of the resulting lookup table.

transform(*callback: Callable[..., tuple[float, ...]]*, *with_normals: bool = False*, *channels: int | None = None*, *target_mode: str | None = None*) → *Color3DLUT*

Transforms the table values using provided callback and returns a new LUT with altered values.

Parameters

- **callback** – A function which takes old lookup table values and returns a new set of values. The number of arguments which function should take is `self.channels` or `3 + self.channels` if `with_normals` flag is set. Should return a tuple of `self.channels` or `channels` elements if it is set.
- **with_normals** – If true, `callback` will be called with coordinates in the color cube as the first three arguments. Otherwise, `callback` will be called only with actual color values.
- **channels** – The number of channels in the resulting lookup table.
- **target_mode** – Passed to the constructor of the resulting lookup table.

class PIL.ImageFilter.**BoxBlur**(*radius: float | Sequence[float]*)

Blurs the image by setting each pixel to the average value of the pixels in a square box extending radius pixels in each direction. Supports float radius of arbitrary size. Uses an optimized implementation which runs in linear time relative to the size of the image for any radius value.

Parameters
radius – Size of the box in a direction. Either a sequence of two numbers for x and y, or a single number for both.

Radius 0 does not blur, returns an identical image. Radius 1 takes 1 pixel in each direction, i.e. 9 pixels in total.

class PIL.ImageFilter.**GaussianBlur**(*radius: float | Sequence[float] = 2*)

Blurs the image with a sequence of extended box filters, which approximates a Gaussian kernel. For details on accuracy see <https://www.mia.uni-saarland.de/Publications/gwosdek-ssvm11.pdf>

Parameters
radius – Standard deviation of the Gaussian kernel. Either a sequence of two numbers for x and y, or a single number for both.

class PIL.ImageFilter.**UnsharpMask**(*radius: float = 2*, *percent: int = 150*, *threshold: int = 3*)

Unsharp mask filter.

See Wikipedia's entry on digital unsharp masking for an explanation of the parameters.

Parameters

- **radius** – Blur Radius
- **percent** – Unsharp strength, in percent
- **threshold** – Threshold controls the minimum brightness change that will be sharpened

class PIL.ImageFilter.**Kernel**(*size: tuple[int, int]*, *kernel: Sequence[float]*, *scale: float | None = None*, *offset: float = 0*)

Create a convolution kernel. This only supports 3x3 and 5x5 integer and floating point kernels.

Kernels can only be applied to "L" and "RGB" images.

Parameters

- **size** – Kernel size, given as (width, height). This must be (3,3) or (5,5).
- **kernel** – A sequence containing kernel weights. The kernel will be flipped vertically before being applied to the image.
- **scale** – Scale factor. If given, the result for each pixel is divided by this value. The default is the sum of the kernel weights.
- **offset** – Offset. If given, this value is added to the result, after it has been divided by the scale factor.

class PIL.ImageFilter.**RankFilter**(*size: int*, *rank: int*)

Create a rank filter. The rank filter sorts all pixels in a window of the given size, and returns the `rank`'th value.

Parameters

- **size** – The kernel size, in pixels.
- **rank** – What pixel value to pick. Use 0 for a min filter, `size * size / 2` for a median filter, `size * size - 1` for a max filter, etc.

class PIL.ImageFilter.**MedianFilter**(*size: int = 3*)

Create a median filter. Picks the median pixel value in a window with the given size.

Parameters
size – The kernel size, in pixels.

class PIL.ImageFilter.**MinFilter**(*size: int = 3*)

Create a min filter. Picks the lowest pixel value in a window with the given size.

Parameters
size – The kernel size, in pixels.

class PIL.ImageFilter.**MaxFilter**(*size: int = 3*)

Create a max filter. Picks the largest pixel value in a window with the given size.

Parameters
size – The kernel size, in pixels.

class PIL.ImageFilter.**ModeFilter**(*size: int = 3*)

Create a mode filter. Picks the most frequent pixel value in a box with the given size. Pixel values that occur only once or twice are ignored; if no pixel value occurs more than twice, the original pixel value is preserved.

Parameters
size – The kernel size, in pixels.

class PIL.ImageFilter.**Filter**

An abstract mixin used for filtering images (for use with *filter()*).

Implementors must provide the following method:

filter(*self*, *image*)

Applies a filter to a single-band image, or a single band of an image.

Returns
A filtered copy of the image.

class PIL.ImageFilter.**MultibandFilter**

An abstract mixin used for filtering multi-band images (for use with *filter()*).

Implementors must provide the following method:

filter(*self*, *image*)

Applies a filter to a multi-band image.

Returns
A filtered copy of the image.

1.3.9 *ImageFont* module

The *ImageFont* module defines a class with the same name. Instances of this class store bitmap fonts, and are used with the *PIL.ImageDraw.ImageDraw.text()* method or the *PIL.ImageText.Text* class.

Pillow uses its own font file format to store bitmap fonts, limited to 256 characters. You can use *to_imagefont()* to convert BDF and PCF font descriptors (X Window font formats) to this format:

```
from PIL import PcfFontFile
with open("Tests/fonts/10x20-ISO8859-1.pcf", "rb") as fp:
    font = PcfFontFile.PcfFontFile(fp)
    imagefont = font.to_imagefont()
```

Starting with version 1.1.4, PIL can be configured to support TrueType and OpenType fonts (as well as other font formats supported by the FreeType library). For earlier versions, TrueType support is only available as part of the imToolkit package.

When measuring text sizes, this module will not break at newline characters. For multiline text, see the *ImageDraw* module.

Warning

To protect against potential DOS attacks when using arbitrary strings as text input, Pillow will raise a ValueError if the number of characters is over a certain limit, *MAX_STRING_LENGTH*.

This threshold can be changed by setting *MAX_STRING_LENGTH*. It can be disabled by setting ImageFont.MAX_STRING_LENGTH = None.

Example

```
from PIL import ImageFont, ImageDraw

draw = ImageDraw.Draw(image)

# use a bitmap font
font = ImageFont.load("arial.pil")

draw.text((10, 10), "hello", font=font)

# use a truetype font
font = ImageFont.truetype("arial.ttf", 15)

draw.text((10, 25), "world", font=font)
```

Functions

PIL.ImageFont.**load**(*filename: str*) → *ImageFont*

Load a font file. This function loads a font object from the given bitmap font file, and returns the corresponding font object. For loading TrueType or OpenType fonts instead, see *truetype()*.

Parameters
filename – Name of font file.

Returns
A font object.

Raises
OSError – If the file could not be read.

PIL.ImageFont.**load_path**(*filename: str | bytes*) → *ImageFont*

Load font file. Same as *load()*, but searches for a bitmap font along the Python path.

Parameters
filename – Name of font file.

Returns
A font object.

Raises
OSError – If the file could not be read.

PIL.ImageFont.**truetype**(*font: str | bytes | PathLike[str] | PathLike[bytes] | BinaryIO*, *size: float = 10*, *index: int = 0*, *encoding: str = ''*, *layout_engine: Layout | None = None*) → *FreeTypeFont*

Load a TrueType or OpenType font from a file or file-like object, and create a font object. This function loads a font object from the given file or file-like object, and creates a font object for a font of the given size. For loading bitmap fonts instead, see *load()* and *load_path()*.

Pillow uses FreeType to open font files. On Windows, be aware that FreeType will keep the file open as long as the FreeTypeFont object exists. Windows limits the number of files that can be open in C at once to 512, so if many fonts are opened simultaneously and that limit is approached, an `OSError` may be thrown, reporting that FreeType "cannot open resource". A workaround would be to copy the file(s) into memory, and open that instead.

This function requires the _imagingft service.

Parameters

- **font** – A filename or file-like object containing a TrueType font. If the file is not found in this filename, the loader may also search in other directories, such as:
  - The `fonts/` directory on Windows,
  - `/Library/Fonts/`, `/System/Library/Fonts/` and `~/Library/Fonts/` on macOS.
  - `~/.local/share/fonts`, `/usr/local/share/fonts`, and `/usr/share/fonts` on Linux; or those specified by the `XDG_DATA_HOME` and `XDG_DATA_DIRS` environment variables for user-installed and system-wide fonts, respectively.
- **size** – The requested size, in pixels.
- **index** – Which font face to load (default is first available face).
- **encoding** – Which font encoding to use (default is Unicode). Possible encodings include (see the FreeType documentation for more information):
  - "unic" (Unicode)
  - "symb" (Microsoft Symbol)
  - "ADOB" (Adobe Standard)
  - "ADBE" (Adobe Expert)
  - "ADBC" (Adobe Custom)
  - "armn" (Apple Roman)
  - "sjis" (Shift JIS)
  - "gb " (PRC)
  - "big5"
  - "wans" (Extended Wansung)
  - "joha" (Johab)
  - "lat1" (Latin-1)

  This specifies the character set to use. It does not alter the encoding of any text provided in subsequent operations.
- **layout_engine** – Which layout engine to use, if available: *ImageFont.Layout.BASIC* or *ImageFont.Layout.RAQM*. If it is available, Raqm layout will be used by default. Otherwise, basic layout will be used.

  Raqm layout is recommended for all non-English text. If Raqm layout is not required, basic layout will have better performance.

  You can check support for Raqm layout using *PIL.features.check_feature()* with `feature="raqm"`.

  Added in version 4.2.0.

Returns
A font object.

Raises

- **OSError** – If the file could not be read.
- **ValueError** – If the font size is not greater than zero.

PIL.ImageFont.**load_default**(*size: float | None = None*) → *FreeTypeFont | ImageFont*

If FreeType support is available, load a version of Aileron Regular, https://dotcolon.net/fonts/aileron, with a more limited character set.

Otherwise, load a "better than nothing" font.

Added in version 1.1.4.

Parameters
size – The font size of Aileron Regular.

Added in version 10.1.0.

Returns
A font object.

PIL.ImageFont.**load_default_imagefont**() → *ImageFont*

Methods

class PIL.ImageFont.**BaseImageFont**

Used by ImageDraw and ImageText

class PIL.ImageFont.**ImageFont**

Bases: *BaseImageFont*

PIL font wrapper

getbbox(*text: str | bytes | bytearray, *args: Any, **kwargs: Any*) → tuple[int, int, int, int]

Returns bounding box (in pixels) of given text.

Added in version 9.2.0.

Parameters
text – Text to render.

Returns
`(left, top, right, bottom)` bounding box

getlength(*text: str | bytes | bytearray, *args: Any, **kwargs: Any*) → int

Returns length (in pixels) of given text. This is the amount by which following text should be offset.

Added in version 9.2.0.

getmask(*text: str | bytes, mode: str = '', *args: Any, **kwargs: Any*) → *Image.core.ImagingCore*

Create a bitmap for the text.

If the font uses antialiasing, the bitmap should have mode `L` and use a maximum value of 255. Otherwise, it should have mode `1`.

Parameters

- **text** – Text to render.

- **mode** – Used by some graphics drivers to indicate what mode the driver prefers; if empty, the renderer may return either mode. Note that the mode is always a string, to simplify C-level implementations.

  Added in version 1.1.5.

Returns
An internal PIL storage memory instance as defined by the *PIL.Image.core* interface module.

class PIL.ImageFont.**FreeTypeFont**(*font: str | bytes | PathLike[str] | PathLike[bytes] | BinaryIO*, *size: float = 10*, *index: int = 0*, *encoding: str = ''*, *layout_engine:* Layout | *None = None*)

Bases: *BaseImageFont*

FreeType font wrapper (requires _imagingft service)

font_variant(*font: str | bytes | PathLike[str] | PathLike[bytes] | BinaryIO | None = None*, *size: float | None = None*, *index: int | None = None*, *encoding: str | None = None*, *layout_engine:* Layout | *None = None*) → *FreeTypeFont*

Create a copy of this FreeTypeFont object, using any specified arguments to override the settings.

Parameters are identical to the parameters used to initialize this object.

Returns
A FreeTypeFont object.

get_variation_axes() → list[*Axis*]

Returns
A list of the axes in a variation font.

Raises
OSError – If the font is not a variation font.

get_variation_names() → list[bytes]

Returns
A list of the named styles in a variation font.

Raises
OSError – If the font is not a variation font.

getbbox(*text: str | bytes | bytearray*, *mode: str = ''*, *direction: str | None = None*, *features: list[str] | None = None*, *language: str | None = None*, *stroke_width: float = 0*, *anchor: str | None = None*) → tuple[float, float, float, float]

Returns bounding box (in pixels) of given text relative to given anchor when rendered in font with provided direction, features, and language.

Use *getlength()* to get the offset of following text with 1/64 pixel precision. The bounding box includes extra margins for some fonts, e.g. italics or accents.

Added in version 8.0.0.

Parameters

- **text** – Text to render.
- **mode** – Used by some graphics drivers to indicate what mode the driver prefers; if empty, the renderer may return either mode. Note that the mode is always a string, to simplify C-level implementations.

- **direction** – Direction of the text. It can be 'rtl' (right to left), 'ltr' (left to right) or 'ttb' (top to bottom). Requires libraqm.
- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example 'dlig' or 'ss01', but can be also used to turn off default font features for example '-liga' to disable ligatures or '-kern' to disable kerning. To get all supported features, see https://learn.microsoft.com/en-us/typography/opentype/spec/featurelist Requires libraqm.
- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code Requires libraqm.
- **stroke_width** – The width of the text stroke.
- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details.

Returns

`(left, top, right, bottom)` bounding box

getlength(*text: str | bytes*, *mode: str = ''*, *direction: str | None = None*, *features: list[str] | None = None*, *language: str | None = None*) → float

Returns length (in pixels with 1/64 precision) of given text when rendered in font with provided direction, features, and language.

This is the amount by which following text should be offset. Text bounding box may extend past the length in some fonts, e.g. when using italics or accents.

The result is returned as a float; it is a whole number if using basic layout.

Note that the sum of two lengths may not equal the length of a concatenated string due to kerning. If you need to adjust for kerning, include the following character and subtract its length.

For example, instead of

```
hello = font.getlength("Hello")
world = font.getlength("World")
hello_world = hello + world  # not adjusted for kerning
assert hello_world == font.getlength("HelloWorld")  # may fail
```

use

```
hello = font.getlength("HelloW") - font.getlength("W")  # adjusted for kerning
world = font.getlength("World")
hello_world = hello + world  # adjusted for kerning
assert hello_world == font.getlength("HelloWorld")  # True
```

or disable kerning with (requires libraqm)

```
hello = draw.textlength("Hello", font, features=["-kern"])
world = draw.textlength("World", font, features=["-kern"])
hello_world = hello + world  # kerning is disabled, no need to adjust
assert hello_world == draw.textlength("HelloWorld", font, features=["-kern"])
```

Added in version 8.0.0.

Parameters

- **text** – Text to measure.
- **mode** – Used by some graphics drivers to indicate what mode the driver prefers; if empty, the renderer may return either mode. Note that the mode is always a string, to simplify C-level implementations.
- **direction** – Direction of the text. It can be 'rtl' (right to left), 'ltr' (left to right) or 'ttb' (top to bottom). Requires libraqm.
- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example 'dlig' or 'ss01', but can be also used to turn off default font features for example '-liga' to disable ligatures or '-kern' to disable kerning. To get all supported features, see https://learn.microsoft.com/en-us/typography/opentype/spec/featurelist Requires libraqm.
- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code Requires libraqm.

Returns

Either width for horizontal text, or height for vertical text.

getmask(*text: str | bytes*, *mode: str = ''*, *direction: str | None = None*, *features: list[str] | None = None*, *language: str | None = None*, *stroke_width: float = 0*, *anchor: str | None = None*, *ink: int = 0*, *start: tuple[float, float] | None = None*) → *Image.core.ImagingCore*

Create a bitmap for the text.

If the font uses antialiasing, the bitmap should have mode `L` and use a maximum value of 255. If the font has embedded color data, the bitmap should have mode `RGBA`. Otherwise, it should have mode `1`.

Parameters

- **text** – Text to render.
- **mode** – Used by some graphics drivers to indicate what mode the driver prefers; if empty, the renderer may return either mode. Note that the mode is always a string, to simplify C-level implementations.

  Added in version 1.1.5.
- **direction** – Direction of the text. It can be 'rtl' (right to left), 'ltr' (left to right) or 'ttb' (top to bottom). Requires libraqm.

  Added in version 4.2.0.
- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example 'dlig' or 'ss01', but can be also used to turn off default font features for example '-liga' to disable ligatures or '-kern' to disable kerning. To get all supported features, see https://learn.microsoft.com/en-us/typography/opentype/spec/featurelist Requires libraqm.

  Added in version 4.2.0.
- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code Requires libraqm.

  Added in version 6.0.0.
- **stroke_width** – The width of the text stroke.

Added in version 6.2.0.

- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details.

  Added in version 8.0.0.

- **ink** – Foreground ink for rendering in RGBA mode.

  Added in version 8.0.0.

- **start** – Tuple of horizontal and vertical offset, as text may render differently when starting at fractional coordinates.

  Added in version 9.4.0.

Returns

An internal PIL storage memory instance as defined by the `PIL.Image.core` interface module.

getmask2(*text: str | bytes*, *mode: str = ''*, *direction: str | None = None*, *features: list[str] | None = None*, *language: str | None = None*, *stroke_width: float = 0*, *anchor: str | None = None*, *ink: int = 0*, *start: tuple[float, float] | None = None*, **args: Any*, ***kwargs: Any*) → tuple[*Image.core.ImagingCore*, tuple[int, int]]

Create a bitmap for the text.

If the font uses antialiasing, the bitmap should have mode `L` and use a maximum value of 255. If the font has embedded color data, the bitmap should have mode `RGBA`. Otherwise, it should have mode `1`.

Parameters

- **text** – Text to render.
- **mode** – Used by some graphics drivers to indicate what mode the driver prefers; if empty, the renderer may return either mode. Note that the mode is always a string, to simplify C-level implementations.

  Added in version 1.1.5.

- **direction** – Direction of the text. It can be 'rtl' (right to left), 'ltr' (left to right) or 'ttb' (top to bottom). Requires libraqm.

  Added in version 4.2.0.

- **features** – A list of OpenType font features to be used during text layout. This is usually used to turn on optional font features that are not enabled by default, for example 'dlig' or 'ss01', but can be also used to turn off default font features for example '-liga' to disable ligatures or '-kern' to disable kerning. To get all supported features, see https://learn.microsoft.com/en-us/typography/opentype/spec/featurelist Requires libraqm.

  Added in version 4.2.0.

- **language** – Language of the text. Different languages may use different glyph shapes or ligatures. This parameter tells the font which language the text is in, and to apply the correct substitutions as appropriate, if available. It should be a BCP 47 language code Requires libraqm.

  Added in version 6.0.0.

- **stroke_width** – The width of the text stroke.

  Added in version 6.2.0.

- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details.

  Added in version 8.0.0.

- **ink** – Foreground ink for rendering in RGBA mode.

  Added in version 8.0.0.

- **start** – Tuple of horizontal and vertical offset, as text may render differently when starting at fractional coordinates.

  Added in version 9.4.0.

Returns
A tuple of an internal PIL storage memory instance as defined by the *PIL.Image.core* interface module, and the text offset, the gap between the starting coordinate and the first marking

getmetrics() → tuple[int, int]

Returns
A tuple of the font ascent (the distance from the baseline to the highest outline point) and descent (the distance from the baseline to the lowest outline point, a negative value)

getname() → tuple[str | None, str | None]

Returns
A tuple of the font family (e.g. Helvetica) and the font style (e.g. Bold)

set_variation_by_axes(*axes: list[float]*) → None

Parameters
axes – A list of values for each axis.

Raises
OSError – If the font is not a variation font.

set_variation_by_name(*name: str | bytes*) → None

Parameters
name – The name of the style.

Raises
OSError – If the font is not a variation font.

class PIL.ImageFont.**TransposedFont**(*font:* ImageFont | FreeTypeFont, *orientation:* Transpose | *None* = *None*)

Bases: *BaseImageFont*

Wrapper for writing rotated or mirrored text

getbbox(*text: str | bytes | bytearray*, **args: Any*, ***kwargs: Any*) → tuple[int, int, float, float]

getlength(*text: str | bytes*, **args: Any*, ***kwargs: Any*) → float

getmask(*text: str | bytes*, *mode: str* = '', **args: Any*, ***kwargs: Any*) → *Image.core.ImagingCore*

Constants

class PIL.ImageFont.**Layout**

BASIC

Use basic text layout for TrueType font. Advanced features such as text direction are not supported.

RAQM

Use Raqm text layout for TrueType font. Advanced features are supported.

Requires Raqm, you can check support using *PIL.features.check_feature()* with `feature="raqm"`.

PIL.ImageFont.**MAX_STRING_LENGTH**

Set to 1,000,000, to protect against potential DOS attacks. Pillow will raise a `ValueError` if the number of characters is over this limit. The check can be disabled by setting `ImageFont.MAX_STRING_LENGTH = None`.

Dictionaries

class PIL.ImageFont.**Axis**

Bases: `TypedDict`

default: `int | None`

maximum: `int | None`

minimum: `int | None`

name: `bytes | None`

1.3.10 *ImageGrab* module

The *ImageGrab* module can be used to copy the contents of the screen or the clipboard to a PIL image memory.

Added in version 1.1.3.

PIL.ImageGrab.**grab**(*bbox=None*, *include_layered_windows=False*, *all_screens=False*, *xdisplay=None*, *window=None*, *scale_down=False*)

Take a snapshot of the screen. The pixels inside the bounding box are returned as "RGBA" on macOS, or "RGB" image otherwise. If the bounding box is omitted, the entire screen is copied.

On macOS, it will be at 2x if on a Retina screen. If this is not desired, pass `scale_down=True`.

On Linux, if `xdisplay` is `None` and the default X11 display does not return a snapshot of the screen, `gnome-screenshot`, `grim` or `spectacle` will be used as a fallback if they are installed. To disable this behaviour, pass `xdisplay=""` instead.

Added in version 1.1.3: Windows support

Added in version 3.0.0: macOS support

Added in version 7.1.0: Linux support

Parameters

- **bbox** – What region to copy. Default is the entire screen. On macOS, this is increased to 2x for Retina screens, so the full width of a Retina screen would be 2880, not 1440. On Windows, the top-left point may be negative if `all_screens=True` is used.
- **include_layered_windows** – Includes layered windows. Windows OS only.

  Added in version 6.1.0.

- **all_screens** – Capture all monitors. Windows OS only.

  Added in version 6.2.0.

- **xdisplay** – X11 Display address. Pass `None` to grab the default system screen. Pass `""` to grab the default X11 screen on Windows or macOS.

  You can check X11 support using *PIL.features.check_feature()* with `feature="xcb"`.

  Added in version 7.1.0.

- **window** – Capture a single window. On Windows, this is a HWND. On macOS, this is a CGWindowID.

  Added in version 11.2.1: Windows support

  Added in version 12.1.0: macOS support

- **scale_down** –

  On macOS, Retina screens will provide bbox images at 2x size by default. This will prevent that, and scale down to 1x.
  Keyword-only argument.

  Added in version 12.3.0.

Returns
An image

PIL.ImageGrab.**grabclipboard**()

Take a snapshot of the clipboard image, if any.

On Linux, `wl-paste` or `xclip` is required.

Added in version 1.1.4: Windows support

Added in version 3.3.0: macOS support

Added in version 9.4.0: Linux support

Returns

On Windows, an image, a list of filenames, or None if the clipboard does not contain image data or filenames. Note that if a list is returned, the filenames may not represent image files.

On Mac, an image, or None if the clipboard does not contain image data.

On Linux, an image.

1.3.11 *ImageMath* module

The *ImageMath* module can be used to evaluate "image expressions", that can take a number of images and generate a result.

ImageMath only supports single-layer images. To process multi-band images, use the *split()* method or *merge()* function.

Example: Using the *ImageMath* module

```
from PIL import Image, ImageMath

with Image.open("image1.jpg") as im1:
    with Image.open("image2.jpg") as im2:
```

(continues on next page)

(continued from previous page)

```
        out = ImageMath.lambda_eval(
          lambda args: args["convert"](args["min"](args["a"], args["b"]), 'L'),
          a=im1,
          b=im2
        )
        out = ImageMath.unsafe_eval(
          "convert(min(a, b), 'L')",
          a=im1,
          b=im2
        )
```

PIL.ImageMath.**lambda_eval**(*expression*, *options*, ***kw*)

Returns the result of an image function.

Parameters

- **expression** – A function that receives a dictionary.
- **options** – Values to add to the function's dictionary. Note that the names must be valid Python identifiers. Deprecated. You can instead use one or more keyword arguments, as shown in the above example.
- ****kw** – Values to add to the function's dictionary, mapping image names to Image instances.

Returns

An image, an integer value, a floating point value, or a pixel tuple, depending on the expression.

PIL.ImageMath.**unsafe_eval**(*expression*, *options*, ***kw*)

Evaluates an image expression.

Danger

This uses Python's `eval()` function to process the expression string, and carries the security risks of doing so. It is not recommended to process expressions without considering this. *lambda_eval()* is a more secure alternative.

ImageMath only supports single-layer images. To process multi-band images, use the *split()* method or *merge()* function.

Parameters

- **expression** – A string which uses the standard Python expression syntax. In addition to the standard operators, you can also use the functions described below.
- **options** – Values to add to the evaluation context. Note that the names must be valid Python identifiers. Deprecated. You can instead use one or more keyword arguments, as shown in the above example.
- ****kw** – Values to add to the evaluation context, mapping image names to Image instances.

Returns

An image, an integer value, a floating point value, or a pixel tuple, depending on the expression.

Expression syntax

- *lambda_eval()* expressions are functions that receive a dictionary containing images and operators.
- *unsafe_eval()* expressions are standard Python expressions, but they're evaluated in a non-standard environment.

Danger

unsafe_eval() uses Python's `eval()` function to process the expression string, and carries the security risks of doing so. It is not recommended to process expressions without considering this. *lambda_eval()* is a more secure alternative.

Standard operators

You can use standard arithmetical operators for addition (+), subtraction (-), multiplication (*), and division (/).

The module also supports unary minus (-), modulo (%), and power (**) operators.

Note that all operations are done with 32-bit integers or 32-bit floating point values, as necessary. For example, if you add two 8-bit images, the result will be a 32-bit integer image. If you add a floating point constant to an 8-bit image, the result will be a 32-bit floating point image.

You can force conversion using the `convert()`, `float()`, and `int()` functions described below.

Bitwise operators

The module also provides operations that operate on individual bits. This includes and (&), or (|), and exclusive or (^). You can also invert (~) all pixel bits.

Note that the operands are converted to 32-bit signed integers before the bitwise operation is applied. This means that you'll get negative values if you invert an ordinary grayscale image. You can use the and (&) operator to mask off unwanted bits.

Bitwise operators don't work on floating point images.

Logical operators

Logical operators like `and`, `or`, and `not` work on entire images, rather than individual pixels.

An empty image (all pixels zero) is treated as false. All other images are treated as true.

Note that `and` and `or` return the last evaluated operand, while not always returns a boolean value.

Built-in functions

These functions are applied to each individual pixel.

abs(*image*)

Absolute value.

convert(*image*, *mode*)

Convert image to the given mode. The mode must be given as a string constant.

float(*image*)

Convert image to 32-bit floating point. This is equivalent to convert(image, "F").

int(*image*)

Convert image to 32-bit integer. This is equivalent to convert(image, "I").

Note that 1-bit and 8-bit images are automatically converted to 32-bit integers if necessary to get a correct result.

max(*image1*, *image2*)

Maximum value.

min(*image1*, *image2*)

Minimum value.

1.3.12 *ImageMorph* module

The *ImageMorph* module allows morphology operators ("MorphOp") to be applied to 1 or L mode images:

```
from PIL import Image, ImageMorph
img = Image.open("Tests/images/hopper.bw")
mop = ImageMorph.MorphOp(op_name="erosion4")
count, imgOut = mop.apply(img)
imgOut.show()
```

In addition to applying operators, you can also analyse images.

You can inspect an image in isolation to determine which pixels are non-empty:

```
print(mop.get_on_pixels(img))  # [(0, 0), (1, 0), (2, 0), ...]
```

Or you can retrieve a list of pixels that match the operator. This is the number of pixels that will be non-empty after the operator is applied:

```
coords = mop.match(img)
print(coords)  # [(17, 1), (18, 1), (34, 1), ...]
print(len(coords))  # 550

imgOut = mop.apply(img)[1]
print(len(mop.get_on_pixels(imgOut)))  # 550
```

If you would like more customized operators, you can pass patterns to the MorphOp class:

```
mop = ImageMorph.MorphOp(patterns=["1:(... ... ...)->0", "4:(00. 01. ...)->1"])
```

Or you can pass lookup table ("LUT") data directly. This LUT data can be constructed with the *LutBuilder*:

```
builder = ImageMorph.LutBuilder()
mop = ImageMorph.MorphOp(lut=builder.build_lut())
```

class PIL.ImageMorph.**LutBuilder**(*patterns: list[str] | None = None*, *op_name: str | None = None*)

Bases: `object`

A class for building a MorphLut from a descriptive language

The input patterns is a list of a strings sequences like these:

```
4:(...
   .1.
   111)->1
```

(whitespaces including linebreaks are ignored). The option 4 describes a series of symmetry operations (in this case a 4-rotation), the pattern is described by:

- . or X - Ignore
- 1 - Pixel is on
- 0 - Pixel is off

The result of the operation is described after "->" string.

The default is to return the current pixel value, which is returned if no other match is found.

Operations:

- 4 - 4 way rotation
- N - Negate
- 1 - Dummy op for no other operation (an op must always be given)
- M - Mirroring

Example:

```
lb = LutBuilder(patterns = ["4:(... .1. 111)->1"])
lut = lb.build_lut()
```

add_patterns(*patterns: list[str]*) → None

Append to list of patterns.

Parameters
patterns – Additional patterns.

build_default_lut() → bytearray

Set the current LUT, and return it.

This is the default LUT that patterns will be applied against when building.

build_lut() → bytearray

Compile all patterns into a morphology LUT, and return it.

This is the data to be passed into MorphOp.

get_lut() → bytearray | None

Returns the current LUT

class PIL.ImageMorph.**MorphOp**(*lut: bytearray | None = None*, *op_name: str | None = None*, *patterns: list[str] | None = None*)

Bases: object

A class for binary morphological operators

apply(*image:* Image) → tuple[int, *Image*]

Run a single morphological operation on an image.

Returns a tuple of the number of changed pixels and the morphed image.

Parameters
image – A 1-mode or L-mode image.

Raises

- **Exception** – If the current operator is None.

- **ValueError** – If the image is not 1 or L mode.

get_on_pixels(*image:* Image) → list[tuple[int, int]]

Get a list of all turned on pixels in a 1 or L mode image.

Returns a list of tuples of (x,y) coordinates of all non-empty pixels. See *Coordinate system*.

Parameters
image – A 1-mode or L-mode image.

Raises
ValueError – If the image is not 1 or L mode.

load_lut(*filename: str*) → None

Load an operator from an mrl file

Parameters
filename – The file to read from.

Raises
Exception – If the length of the file data is not 512.

match(*image:* Image) → list[tuple[int, int]]

Get a list of coordinates matching the morphological operation on an image.

Returns a list of tuples of (x,y) coordinates of all matching pixels. See *Coordinate system*.

Parameters
image – A 1-mode or L-mode image.

Raises

- **Exception** – If the current operator is None.
- **ValueError** – If the image is not 1 or L mode.

save_lut(*filename: str*) → None

Save an operator to an mrl file.

Parameters
filename – The destination file.

Raises
Exception – If the current operator is None.

set_lut(*lut: bytearray | None*) → None

Set the LUT from an external source

Parameters
lut – A new LUT.

1.3.13 *ImageOps* module

The *ImageOps* module contains a number of 'ready-made' image processing operations. This module is somewhat experimental, and most operators only work on L and RGB images.

Added in version 1.1.3.

PIL.ImageOps.**autocontrast**(*image:* Image, *cutoff: float | tuple[float, float] = 0*, *ignore: int | Sequence[int] | None = None*, *mask:* Image | *None = None*, *preserve_tone: bool = False*) → *Image*

Maximize (normalize) image contrast. This function calculates a histogram of the input image (or mask region), removes `cutoff` percent of the lightest and darkest pixels from the histogram, and remaps the image so that the darkest pixel becomes black (0), and the lightest becomes white (255).

Parameters

- **image** – The image to process.
- **cutoff** – The percent to cut off from the histogram on the low and high ends. Either a tuple of (low, high), or a single number for both.
- **ignore** – The background pixel value (use None for no background).
- **mask** – Histogram used in contrast operation is computed using pixels within the mask. If no mask is given the entire image is used for histogram computation.
- **preserve_tone** – Preserve image tone in Photoshop-like style autocontrast.

  Added in version 8.2.0.

Returns

An image.

PIL.ImageOps.**colorize**(*image:* Image, *black: str | tuple[int, ...]*, *white: str | tuple[int, ...]*, *mid: str | int | tuple[int, ...] | None = None*, *blackpoint: int = 0*, *whitepoint: int = 255*, *midpoint: int = 127*) → *Image*

Colorize grayscale image. This function calculates a color wedge which maps all black pixels in the source image to the first color and all white pixels to the second color. If `mid` is specified, it uses three-color mapping. The `black` and `white` arguments should be RGB tuples or color names; optionally you can use three-color mapping by also specifying `mid`. Mapping positions for any of the colors can be specified (e.g. `blackpoint`), where these parameters are the integer value corresponding to where the corresponding color should be mapped. These parameters must have logical order, such that `blackpoint <= midpoint <= whitepoint` (if `mid` is specified).

Parameters

- **image** – The image to colorize.
- **black** – The color to use for black input pixels.
- **white** – The color to use for white input pixels.
- **mid** – The color to use for midtone input pixels.
- **blackpoint** – an int value [0, 255] for the black mapping.
- **whitepoint** – an int value [0, 255] for the white mapping.
- **midpoint** – an int value [0, 255] for the midtone mapping.

Returns

An image.

PIL.ImageOps.**crop**(*image:* Image, *border: int = 0*) → *Image*

Remove border from image. The same amount of pixels are removed from all four sides. This function works on all image modes.

See also

crop()

Parameters

- **image** – The image to crop.
- **border** – The number of pixels to remove.

Returns
An image.

PIL.ImageOps.**scale**(*image:* Image, *factor:* *float*, *resample:* *int* = *Resampling.BICUBIC*) → *Image*

Returns a rescaled image by a specific factor given in parameter. A factor greater than 1 expands the image, between 0 and 1 contracts the image.

Parameters

- **image** – The image to rescale.
- **factor** – The expansion factor, as a float.
- **resample** – Resampling method to use. Default is *BICUBIC*. See *Filters*.

Returns
An *Image* object.

class PIL.ImageOps.**SupportsGetMesh**(**args*, ***kwargs*)

Bases: Protocol

An object that supports the `getmesh` method, taking an image as an argument, and returning a list of tuples. Each tuple contains two tuples, the source box as a tuple of 4 integers, and a tuple of 8 integers for the final quadrilateral, in order of top left, bottom left, bottom right, top right.

PIL.ImageOps.**deform**(*image:* Image, *deformer:* SupportsGetMesh, *resample:* *int* = *Resampling.BILINEAR*) → *Image*

Deform the image.

Parameters

- **image** – The image to deform.
- **deformer** – A deformer object. Any object that implements a `getmesh` method can be used.
- **resample** – An optional resampling filter. Same values possible as in the PIL.Image.transform function.

Returns
An image.

PIL.ImageOps.**equalize**(*image:* Image, *mask:* Image | *None* = *None*) → *Image*

Equalize the image histogram. This function applies a non-linear mapping to the input image, in order to create a uniform distribution of grayscale values in the output image.

Parameters

- **image** – The image to equalize.
- **mask** – An optional mask. If given, only the pixels selected by the mask are included in the analysis.

Returns
An image.

PIL.ImageOps.**expand**(*image:* Image, *border:* *int* | *tuple[int, ...]* = *0*, *fill:* *str* | *int* | *tuple[int, ...]* = *0*) → *Image*

Add border to the image

Parameters

- **image** – The image to expand.
- **border** – Border width, in pixels.

- **fill** – Pixel fill value (a color value). Default is 0 (black).

Returns
An image.

PIL.ImageOps.**flip**(*image:* Image) → *Image*

Flip the image vertically (top to bottom).

Parameters
image – The image to flip.

Returns
An image.

PIL.ImageOps.**grayscale**(*image:* Image) → *Image*

Convert the image to grayscale.

Parameters
image – The image to convert.

Returns
An image.

PIL.ImageOps.**invert**(*image:* Image) → *Image*

Invert (negate) the image.

Parameters
image – The image to invert.

Returns
An image.

PIL.ImageOps.**mirror**(*image:* Image) → *Image*

Flip image horizontally (left to right).

Parameters
image – The image to mirror.

Returns
An image.

PIL.ImageOps.**posterize**(*image:* Image, *bits:* *int*) → *Image*

Reduce the number of bits for each color channel.

Parameters

- **image** – The image to posterize.
- **bits** – The number of bits to keep for each channel (1-8).

Returns
An image.

PIL.ImageOps.**solarize**(*image:* Image, *threshold:* *int* = *128*) → *Image*

Invert all pixel values above a threshold.

Parameters

- **image** – The image to solarize.
- **threshold** – All pixels above this grayscale level are inverted.

Returns
An image.

PIL.ImageOps.**exif_transpose**(*image:* Image, *, *in_place: Literal[True]*) → None

PIL.ImageOps.**exif_transpose**(*image:* Image, *, *in_place: Literal[False] = False*) → *Image*

If an image has an EXIF Orientation tag, other than 1, transpose the image accordingly, and remove the orientation data.

Parameters

- **image** – The image to transpose.
- **in_place** – Boolean. Keyword-only argument. If `True`, the original image is modified in-place, and `None` is returned. If `False` (default), a new *Image* object is returned with the transposition applied. If there is no transposition, a copy of the image will be returned.

Resize relative to a given size

```
from PIL import Image, ImageOps
size = (100, 150)
with Image.open("Tests/images/hopper.webp") as im:
    ImageOps.contain(im, size).save("imageops_contain.webp")
    ImageOps.cover(im, size).save("imageops_cover.webp")
    ImageOps.fit(im, size).save("imageops_fit.webp")
    ImageOps.pad(im, size, color="#f00").save("imageops_pad.webp")

    # thumbnail() can also be used,
    # but will modify the image object in place
    im.thumbnail(size)
    im.save("image_thumbnail.webp")
```

	thumbnail()	*contain()*	*cover()*	*fit()*	*pad()*
Given size	`(100, 150)`	`(100, 150)`	`(100, 150)`	`(100, 150)`	`(100, 150)`
Resulting image					
Resulting size	`100×100`	`100×100`	`150×150`	`100×150`	`100×150`

PIL.ImageOps.**contain**(*image:* Image, *size: tuple[int, int]*, *method: int = Resampling.BICUBIC*) → *Image*

Returns a resized version of the image, set to the maximum width and height within the requested size, while maintaining the original aspect ratio.

Parameters

- **image** – The image to resize.
- **size** – The requested output size in pixels, given as a (width, height) tuple.
- **method** – Resampling method to use. Default is *BICUBIC*. See *Filters*.

Returns

An image.

PIL.ImageOps.**cover**(*image:* Image, *size: tuple[int, int]*, *method: int = Resampling.BICUBIC*) → *Image*

Returns a resized version of the image, so that the requested size is covered, while maintaining the original aspect ratio.

Parameters

- **image** – The image to resize.
- **size** – The requested output size in pixels, given as a (width, height) tuple.

- **method** – Resampling method to use. Default is *BICUBIC*. See *Filters*.

Returns
An image.

PIL.ImageOps.**fit**(*image: Image*, *size: tuple[int, int]*, *method: int = Resampling.BICUBIC*, *bleed: float = 0.0*, *centering: tuple[float, float] = (0.5, 0.5)*) → *Image*

Returns a resized and cropped version of the image, cropped to the requested aspect ratio and size.

This function was contributed by Kevin Cazabon.

Parameters

- **image** – The image to resize and crop.
- **size** – The requested output size in pixels, given as a (width, height) tuple.
- **method** – Resampling method to use. Default is *BICUBIC*. See *Filters*.
- **bleed** – Remove a border around the outside of the image from all four edges. The value is a decimal percentage (use 0.01 for one percent). The default value is 0 (no border). Cannot be greater than or equal to 0.5.
- **centering** – Control the cropping position. Use (0.5, 0.5) for center cropping (e.g. if cropping the width, take 50% off of the left side, and therefore 50% off the right side). (0.0, 0.0) will crop from the top left corner (i.e. if cropping the width, take all of the crop off of the right side, and if cropping the height, take all of it off the bottom). (1.0, 0.0) will crop from the bottom left corner, etc. (i.e. if cropping the width, take all of the crop off the left side, and if cropping the height take none from the top, and therefore all off the bottom).

Returns
An image.

PIL.ImageOps.**pad**(*image: Image*, *size: tuple[int, int]*, *method: int = Resampling.BICUBIC*, *color: str | int | tuple[int, ...] | None = None*, *centering: tuple[float, float] = (0.5, 0.5)*) → *Image*

Returns a resized and padded version of the image, expanded to fill the requested aspect ratio and size.

Parameters

- **image** – The image to resize and crop.
- **size** – The requested output size in pixels, given as a (width, height) tuple.
- **method** – Resampling method to use. Default is *BICUBIC*. See *Filters*.
- **color** – The background color of the padded image.
- **centering** – Control the position of the original image within the padded version.

  (0.5, 0.5) will keep the image centered (0, 0) will keep the image aligned to the top left
  (1, 1) will keep the image aligned to the bottom right

Returns
An image.

1.3.14 *ImagePalette* module

The *ImagePalette* module contains a class of the same name to represent the color palette of palette mapped images.

> **Note**
>
> The *ImagePalette* class has several methods, but they are all marked as "experimental." Read that as you will. The [source] link is there for a reason.

class PIL.ImagePalette.**ImagePalette**(*mode: str = 'RGB'*, *palette: Sequence[int] | bytes | bytearray | None = None*)

Color palette for palette mapped images

Parameters

- **mode** – The mode to use for the palette. See: *Modes*. Defaults to "RGB"
- **palette** – An optional palette. If given, it must be a bytearray, an array or a list of ints between 0-255. The list must consist of all channels for one color followed by the next color (e.g. RGBRGBRGB). Defaults to an empty palette.

getcolor(*color: tuple[int, ...]*, *image:* Image.Image | *None = None*) → int

Given an rgb tuple, allocate palette entry.

> **Warning**
>
> This method is experimental.

getdata() → tuple[str, Sequence[int] | bytes | bytearray]

Get palette contents in format suitable for the low-level `im.putpalette` primitive.

> **Warning**
>
> This method is experimental.

save(*fp: str | IO[str]*) → None

Save palette to text file.

> **Warning**
>
> This method is experimental.

tobytes() → bytes

Convert palette to bytes.

> **Warning**
>
> This method is experimental.

tostring() → bytes

Convert palette to bytes.

> **Warning**
>
> This method is experimental.

1.3.15 *ImagePath* module

The *ImagePath* module is used to store and manipulate 2-dimensional vector data. Path objects can be passed to the methods on the *ImageDraw* module.

class PIL.ImagePath.**Path**

A path object. The coordinate list can be any sequence object containing either 2-tuples [(x, y), ...] or numeric values [x, y, ...].

You can also create a path object from another path object.

In 1.1.6 and later, you can also pass in any object that implements Python's buffer API. The buffer should provide read access, and contain C floats in machine byte order.

The path object implements most parts of the Python sequence interface, and behaves like a list of (x, y) pairs. You can use len(), item access, and slicing as usual. However, this does not support slice assignment, or item and slice deletion.

Parameters

xy – A sequence. The sequence can contain 2-tuples [(x, y), ...] or a flat list of numbers [x, y, ...].

PIL.ImagePath.Path.**compact**(*distance=2*)

Compacts the path, by removing points that are close to each other. This method modifies the path in place, and returns the number of points left in the path.

`distance` is measured as Manhattan distance and defaults to two pixels.

PIL.ImagePath.Path.**getbbox**()

Gets the bounding box of the path.

Returns

`(x0, y0, x1, y1)`

PIL.ImagePath.Path.**map**(*function*)

Maps the path through a function.

PIL.ImagePath.Path.**tolist**(*flat=False*)

Converts the path to a Python list [(x, y), ...].

Parameters

flat – By default, this function returns a list of 2-tuples [(x, y), ...]. If this argument is `True`, it returns a flat list [x, y, ...] instead.

Returns

A list of coordinates. See `flat`.

PIL.ImagePath.Path.**transform**(*matrix*)

Transforms the path in place, using an affine transform. The matrix is a 6-tuple (a, b, c, d, e, f), and each point is mapped as follows:

```
xOut = xIn * a + yIn * b + c
yOut = xIn * d + yIn * e + f
```

1.3.16 *ImageQt* module

The *ImageQt* module contains support for creating PyQt6 or PySide6 QImage objects from PIL images.

Added in version 1.1.6.

class PIL.ImageQt.**ImageQt**(*image*)

Creates an *ImageQt* object from a PIL *Image* object. This class is a subclass of QtGui.QImage, which means that you can pass the resulting objects directly to PyQt6/PySide6 API functions and methods.

This operation is currently supported for mode 1, L, P, RGB, and RGBA images. To handle other modes, you need to convert the image first.

1.3.17 *ImageSequence* module

The *ImageSequence* module contains a wrapper class that lets you iterate over the frames of an image sequence.

Extracting frames from an animation

```
from PIL import Image, ImageSequence

with Image.open("animation.fli") as im:
    index = 1
    for frame in ImageSequence.Iterator(im):
        frame.save(f"frame{index}.png")
        index += 1
```

The *Iterator* class

class PIL.ImageSequence.**Iterator**(*im:* Image)

This class implements an iterator object that can be used to loop over an image sequence.

You can use the `[]` operator to access elements by index. This operator will raise an `IndexError` if you try to access a nonexistent frame.

Parameters
im – An image object.

Functions

PIL.ImageSequence.**all_frames**(*im:* Image.Image | *list*[Image.Image], *func: Callable*[[Image.Image], Image.Image] | *None* = *None*) → list[*Image.Image*]

Applies a given function to all frames in an image or a list of images. The frames are returned as a list of separate images.

Parameters

- **im** – An image, or a list of images.
- **func** – The function to apply to all of the image frames.

Returns
A list of images.

1.3.18 *ImageShow* module

The *ImageShow* module is used to display images. All default viewers convert the image to be shown to PNG format.

PIL.ImageShow.**show**(*image:* Image, *title:* *str* | *None* = *None*, ***options:* *Any*) → bool

Display a given image.

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.

class PIL.ImageShow.**IPythonViewer**

The viewer for IPython frontends.

class PIL.ImageShow.**WindowsViewer**

The default viewer on Windows is the default system application for PNG files.

class PIL.ImageShow.**MacViewer**

The default viewer on macOS using `Preview.app`.

class PIL.ImageShow.**UnixViewer**

The following viewers may be registered on Unix-based systems, if the given command is found:

class XDGViewer

The freedesktop.org `xdg-open` command.

class DisplayViewer

The ImageMagick `display` command. This viewer supports the `title` parameter.

class GmDisplayViewer

The GraphicsMagick `gm display` command.

class EogViewer

The GNOME Image Viewer `eog` command.

class XVViewer

The X Viewer `xv` command. This viewer supports the `title` parameter.

To provide maximum functionality on Unix-based systems, temporary files created from images will not be automatically removed by Pillow.

PIL.ImageShow.**register**(*viewer:* *type*[Viewer] | Viewer, *order:* *int* = *1*) → None

The *register()* function is used to register additional viewers:

```
from PIL import ImageShow
ImageShow.register(MyViewer())  # MyViewer will be used as a last resort
ImageShow.register(MySecondViewer(), 0)  # MySecondViewer will be prioritised
ImageShow.register(ImageShow.XVViewer(), 0)  # XVViewer will be prioritised
```

Parameters

- **viewer** – The viewer to be registered.

- **order** – Zero or a negative integer to prepend this viewer to the list, a positive integer to append it.

class PIL.ImageShow.**Viewer**

Base class for viewers.

show(*image:* Image, ***options:* *Any*) → int

The main function for displaying an image. Converts the given image to the target format and displays it.

format: `str | None = None`

The format to convert the image into.

options: `dict[str, Any] = {}`

Additional options used to convert the image.

get_format(*image:* Image) → str | None

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

get_command(*file: str*, ***options:* *Any*) → str

Returns the command used to display the file. Not implemented in the base class.

save_image(*image:* Image) → str

Save to temporary file and return filename.

show_image(*image:* Image, ***options:* *Any*) → int

Display the given image.

show_file(*path: str*, ***options:* *Any*) → int

Display given file.

1.3.19 *ImageStat* module

The *ImageStat* module calculates global statistics for an image, or for a region of an image.

class PIL.ImageStat.**Stat**(*image_or_list:* Image | *list[int]*, *mask:* Image | *None* = *None*)

__init__(*image_or_list:* Image | *list[int]*, *mask:* Image | *None* = *None*) → None

Calculate statistics for the given image. If a mask is included, only the regions covered by that mask are included in the statistics. You can also pass in a previously calculated histogram.

Parameters

- **image** – A PIL image, or a precalculated histogram.

> **Note**
>
> For a PIL image, calculations rely on the *histogram()* method. The pixel counts are grouped into 256 bins, even if the image has more than 8 bits per channel. So `I` and `F` mode images have a maximum `mean`, `median` and `rms` of 255, and cannot have an `extrema` maximum of more than 255.

- **mask** – An optional mask.

property count: `list[int]`

Total number of pixels for each band in the image.

property extrema: list[tuple[int, int]]

Min/max values for each band in the image.

> **Note**
>
> This relies on the *histogram()* method, and simply returns the low and high bins used. This is correct for images with 8 bits per channel, but fails for other modes such as `I` or `F`. Instead, use *getextrema()* to return per-band extrema for the image. This is more correct and efficient because, for non-8-bit modes, the histogram method uses *getextrema()* to determine the bins used.

property mean: list[float]

Average (arithmetic mean) pixel level for each band in the image.

property median: list[int]

Median pixel level for each band in the image.

property rms: list[float]

RMS (root-mean-square) for each band in the image.

property stddev: list[float]

Standard deviation for each band in the image.

property sum: list[float]

Sum of all pixels for each band in the image.

property sum2: list[float]

Squared sum of all pixels for each band in the image.

property var: list[float]

Variance for each band in the image.

1.3.20 *ImageText* module

The *ImageText* module defines a *Text* class. Instances of this class provide a way to use fonts with text strings or bytes. The result is a simple API to apply styling to pieces of text and measure or draw them.

Example

```
from PIL import Image, ImageDraw, ImageFont, ImageText
font = ImageFont.truetype("Tests/fonts/FreeMono.ttf", 24)

text = ImageText.Text("Hello world", font)
text.embed_color()
text.stroke(2, "#0f0")

print(text.get_length())  # 154.0
print(text.get_bbox())  # (-2, 3, 156, 22)

im = Image.new("RGB", text.get_bbox()[2:])
d = ImageDraw.Draw(im)
d.text((0, 0), text, "#f00")
```

Comparison

Without `ImageText.Text`:

```
from PIL import Image, ImageDraw
im = Image.new(mode, size)
d = ImageDraw.Draw(im)

d.textlength(text, font, direction, features, language, embedded_color)
d.multiline_textbbox(xy, text, font, anchor, spacing, align, direction, features,␣
↪language, stroke_width, embedded_color)
d.text(xy, text, fill, font, anchor, spacing, align, direction, features, language,␣
↪stroke_width, stroke_fill, embedded_color)
```

With `ImageText.Text`:

```
from PIL import ImageText
text = ImageText.Text(text, font, mode, spacing, direction, features, language)
text.embed_color()
text.stroke(stroke_width, stroke_fill)

text.get_length()
text.get_bbox(xy, anchor, align)

im = Image.new(mode, size)
d = ImageDraw.Draw(im)
d.text(xy, text, fill, anchor=anchor, align=align)
```

Methods

class PIL.ImageText.**Text**(*text: AnyStr*, *font:* BaseImageFont | *None* = *None*, *mode: str* = *'RGB'*, *spacing: float* = *4*, *direction: str* | *None* = *None*, *features: list[str]* | *None* = *None*, *language: str* | *None* = *None*)

embed_color() → None

Use embedded color glyphs (COLR, CBDT, SBIX).

get_bbox(*xy: tuple[float, float]* = *(0, 0)*, *anchor: str* | *None* = *None*, *align: str* = *'left'*) → tuple[float, float, float, float]

Returns bounding box (in pixels) of text.

Use *get_length()* to get the offset of following text with 1/64 pixel precision. The bounding box includes extra margins for some fonts, e.g. italics or accents.

Parameters

- **xy** – The anchor coordinates of the text.
- **anchor** – The text anchor alignment. Determines the relative location of the anchor to the text. The default alignment is top left, specifically `la` for horizontal text and `lt` for vertical text. See *Text anchors* for details.
- **align** – For multiline text, `"left"`, `"center"`, `"right"` or `"justify"` determines the relative alignment of lines. Use the `anchor` parameter to specify the alignment to `xy`.

Returns

`(left, top, right, bottom)` bounding box

get_length() → float

Returns length (in pixels with 1/64 precision) of text.

This is the amount by which following text should be offset. Text bounding box may extend past the length in some fonts, e.g. when using italics or accents.

The result is returned as a float; it is a whole number if using basic layout.

Note that the sum of two lengths may not equal the length of a concatenated string due to kerning. If you need to adjust for kerning, include the following character and subtract its length.

For example, instead of:

```
hello = ImageText.Text("Hello", font).get_length()
world = ImageText.Text("World", font).get_length()
helloworld = ImageText.Text("HelloWorld", font).get_length()
assert hello + world == helloworld
```

use:

```
hello = (
    ImageText.Text("HelloW", font).get_length() -
    ImageText.Text("W", font).get_length()
)  # adjusted for kerning
world = ImageText.Text("World", font).get_length()
helloworld = ImageText.Text("HelloWorld", font).get_length()
assert hello + world == helloworld
```

or disable kerning with (requires libraqm):

```
hello = ImageText.Text("Hello", font, features=["-kern"]).get_length()
world = ImageText.Text("World", font, features=["-kern"]).get_length()
helloworld = ImageText.Text(
    "HelloWorld", font, features=["-kern"]
).get_length()
assert hello + world == helloworld
```

Returns

Either width for horizontal text, or height for vertical text.

stroke(*width: float = 0*, *fill: float | tuple[int, ...] | str | None = None*) → None

Parameters

- **width** – The width of the text stroke.
- **fill** – Color to use for the text stroke when drawing. If not given, will default to the `fill` parameter from *ImageDraw.ImageDraw.text()*.

wrap(*width: int*, *height: int | None = None*, *scaling: str | tuple[str, int] | None = None*) → *Text* | None

Wrap text to fit within a given width.

Parameters

- **width** – The width to fit within.
- **height** – An optional height limit. Any text that does not fit within this will be returned as a new *Text* object.

- **scaling** – An optional directive to scale the text, either "grow" as much as possible within the given dimensions, or "shrink" until it fits. It can also be a tuple of (direction, limit), with an integer limit to stop scaling at.

Returns
An *Text* object, or None.

1.3.21 *ImageTk* module

The *ImageTk* module contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

For examples, see the demo programs in the Scripts directory.

class PIL.ImageTk.**BitmapImage**(*image:* Image | *None* = *None*, ***kw: Any*)

A Tkinter-compatible bitmap image. This can be used everywhere Tkinter expects an image object.

The given image must have mode "1". Pixels having value 0 are treated as transparent. Options, if any, are passed on to Tkinter. The most commonly used option is `foreground`, which is used to specify the color for the non-transparent parts. See the Tkinter documentation for information on how to specify colours.

Parameters
image – A PIL image.

height() → int

Get the height of the image.

Returns
The height, in pixels.

width() → int

Get the width of the image.

Returns
The width, in pixels.

class PIL.ImageTk.**PhotoImage**(*image:* Image | *str* | *None* = *None*, *size: tuple[int, int]* | *None* = *None*, ***kw: Any*)

A Tkinter-compatible photo image. This can be used everywhere Tkinter expects an image object. If the image is an RGBA image, pixels having alpha 0 are treated as transparent.

The constructor takes either a PIL image, or a mode and a size. Alternatively, you can use the `file` or `data` options to initialize the photo image object.

Parameters

- **image** – Either a PIL image, or a mode string. If a mode string is used, a size must also be given.
- **size** – If the first argument is a mode string, this defines the size of the image.
- **file** – A filename to load the image from (using `Image.open(file)`).
- **data** – An 8-bit string containing image data (as loaded from an image file).

height() → int

Get the height of the image.

Returns
The height, in pixels.

paste(*im:* Image) → None

Paste a PIL image into the photo image. Note that this can be very slow if the photo image is displayed.

Parameters
im – A PIL image. The size must match the target region. If the mode does not match, the image is converted to the mode of the bitmap image.

width() → int

Get the width of the image.

Returns
The width, in pixels.

1.3.22 *ImageTransform* module

The *ImageTransform* module contains implementations of *ImageTransformHandler* for some of the builtin *Image.Transform* methods.

class PIL.ImageTransform.**Transform**(*data: Sequence[Any]*)

Bases: *ImageTransformHandler*

Base class for other transforms defined in *ImageTransform*.

getdata() → tuple[*Transform*, Sequence[int]]

method: *Transform*

transform(*size: tuple[int, int]*, *image:* Image, ***options: Any*) → *Image*

Perform the transform. Called from *Image.transform()*.

class PIL.ImageTransform.**AffineTransform**(*data: Sequence[Any]*)

Bases: *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 the inverse of 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 *Image.transform()*

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

method: *Transform* = **0**

class PIL.ImageTransform.**PerspectiveTransform**(*data: Sequence[Any]*)

Bases: *Transform*

Define a perspective image transform.

This function takes an 8-tuple (a, b, c, d, e, f, g, h). For each pixel (x, y) in the output image, the new value is taken from a position ((a x + b y + c) / (g x + h y + 1), (d x + e y + f) / (g x + h y + 1)) in the input image, rounded to nearest pixel.

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

See *Image.transform()*

Parameters
matrix – An 8-tuple (a, b, c, d, e, f, g, h).

method: *Transform* = 2

class PIL.ImageTransform.**ExtentTransform**(*data: Sequence[Any]*)

Bases: *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 output image, and (x1, y1) at size.

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 *Image.transform()*

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

method: *Transform* = 1

class PIL.ImageTransform.**QuadTransform**(*data: Sequence[Any]*)

Bases: *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 *Image.transform()*

Parameters
xy – An 8-tuple (x0, y0, x1, y1, x2, y2, x3, y3) which contain the upper left, lower left, lower right, and upper right corner of the source quadrilateral.

method: *Transform* = 3

class PIL.ImageTransform.**MeshTransform**(*data: Sequence[Any]*)

Bases: *Transform*

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

See *Image.transform()*

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

method: *Transform* = 4

1.3.23 *ImageWin* module (Windows-only)

The *ImageWin* module contains support to create and display images on Windows.

ImageWin can be used with PythonWin and other user interface toolkits that provide access to Windows device contexts or window handles. For example, Tkinter makes the window handle available via the winfo_id method:

```
from PIL import ImageWin

dib = ImageWin.Dib(...)

hwnd = ImageWin.HWND(widget.winfo_id())
dib.draw(hwnd, xy)
```

class PIL.ImageWin.**Dib**(*image:* Image | *str*, *size: tuple[int, int]* | *None* = *None*)

A Windows bitmap with the given mode and size. The mode can be one of "1", "L", "P", or "RGB".

If the display requires a palette, this constructor creates a suitable palette and associates it with the image. For an "L" image, 128 graylevels are allocated. For an "RGB" image, a 6x6x6 colour cube is used, together with 20 graylevels.

To make sure that palettes work properly under Windows, you must call the `palette` method upon certain events from Windows.

Parameters

- **image** – Either a PIL image, or a mode string. If a mode string is used, a size must also be given. The mode can be one of "1", "L", "P", or "RGB".
- **size** – If the first argument is a mode string, this defines the size of the image.

draw(*handle: int* | HDC | HWND, *dst: tuple[int, int, int, int]*, *src: tuple[int, int, int, int]* | *None* = *None*) → None

Same as expose, but allows you to specify where to draw the image, and what part of it to draw.

The destination and source areas are given as 4-tuple rectangles. If the source is omitted, the entire image is copied. If the source and the destination have different sizes, the image is resized as necessary.

expose(*handle: int* | HDC | HWND) → None

Copy the bitmap contents to a device context.

Parameters

handle – Device context (HDC), cast to a Python integer, or an HDC or HWND instance. In PythonWin, you can use `CDC.GetHandleAttrib()` to get a suitable handle.

frombytes(*buffer: bytes* | *memoryview*) → None

Load display memory contents from byte data.

Parameters

buffer – A buffer containing display data (usually data returned from *tobytes()*)

paste(*im:* Image, *box: tuple[int, int, int, int]* | *None* = *None*) → None

Paste a PIL image into the bitmap image.

Parameters

- **im** – A PIL image. The size must match the target region. If the mode does not match, the image is converted to the mode of the bitmap image.
- **box** – A 4-tuple defining the left, upper, right, and lower pixel coordinate. See *Coordinate system*. If None is given instead of a tuple, all of the image is assumed.

query_palette(*handle: int* | HDC | HWND) → int

Installs the palette associated with the image in the given device context.

This method should be called upon **QUERYNEWPALETTE** and **PALETTECHANGED** events from Windows. If this method returns a non-zero value, one or more display palette entries were changed, and the image should be redrawn.

Parameters
handle – Device context (HDC), cast to a Python integer, or an HDC or HWND instance.

Returns
The number of entries that were changed (if one or more entries, this indicates that the image should be redrawn).

tobytes() → bytes

Copy display memory contents to bytes object.

Returns
A bytes object containing display data.

class PIL.ImageWin.**HDC**(*dc: int*)

Wraps an HDC integer. The resulting object can be passed to the *draw()* and *expose()* methods.

class PIL.ImageWin.**HWND**(*wnd: int*)

Wraps an HWND integer. The resulting object can be passed to the *draw()* and *expose()* methods, instead of a DC.

1.3.24 *ExifTags* module

The *ExifTags* module exposes several enum.IntEnum classes which provide constants and clear-text names for various well-known EXIF tags.

PIL.ExifTags.**Base**

```
>>> from PIL.ExifTags import Base
>>> Base.ImageDescription.value
270
>>> Base(270).name
'ImageDescription'
```

PIL.ExifTags.**GPS**

```
>>> from PIL.ExifTags import GPS
>>> GPS.GPSDestLatitude.value
20
>>> GPS(20).name
'GPSDestLatitude'
```

PIL.ExifTags.**Interop**

```
>>> from PIL.ExifTags import Interop
>>> Interop.RelatedImageFileFormat.value
4096
>>> Interop(4096).name
'RelatedImageFileFormat'
```

PIL.ExifTags.**IFD**

```
>>> from PIL.ExifTags import IFD
>>> IFD.Exif.value
34665
>>> IFD(34665).name
'Exif'
```

PIL.ExifTags.**LightSource**

```
>>> from PIL.ExifTags import LightSource
>>> LightSource.Unknown.value
0
>>> LightSource(0).name
'Unknown'
```

Two of these values are also exposed as dictionaries.

PIL.ExifTags.**TAGS: dict**

The TAGS dictionary maps 16-bit integer EXIF tag enumerations to descriptive string names. For instance:

```
>>> from PIL.ExifTags import TAGS
>>> TAGS[0x010e]
'ImageDescription'
```

PIL.ExifTags.**GPSTAGS: dict**

The GPSTAGS dictionary maps 8-bit integer EXIF GPS enumerations to descriptive string names. For instance:

```
>>> from PIL.ExifTags import GPSTAGS
>>> GPSTAGS[20]
'GPSDestLatitude'
```

1.3.25 *TiffTags* module

The *TiffTags* module exposes many of the standard TIFF metadata tag numbers, names, and type information.

PIL.TiffTags.**lookup**(*tag*)

Parameters

- **tag** – Integer tag number
- **group** – Which *TAGS_V2_GROUPS* to look in

Added in version 8.3.0.

Returns

Taginfo namedtuple, from the *TAGS_V2* info if possible, otherwise just populating the value and name from *TAGS*. If the tag is not recognized, "unknown" is returned for the name

Added in version 3.1.0.

class PIL.TiffTags.**TagInfo**

__init__(*self*, *value=None*, *name='unknown'*, *type=None*, *length=0*, *enum=None*)

Parameters

- **value** – Integer Tag Number
- **name** – Tag Name
- **type** – Integer type from *PIL.TiffTags.TYPES*
- **length** – Array length: 0 == variable, 1 == single value, n = fixed
- **enum** – Dict of name:integer value options for an enumeration

cvt_enum(*self*, *value*)

Parameters
value – The enumerated value name

Returns
The integer corresponding to the name.

Added in version 3.0.0.

PIL.TiffTags.**TAGS_V2**: dict

The TAGS_V2 dictionary maps 16-bit integer tag numbers to *PIL.TiffTags.TagInfo* tuples for metadata fields defined in the TIFF spec.

Added in version 3.0.0.

PIL.TiffTags.**TAGS_V2_GROUPS**: dict

TAGS_V2 is one dimensional and doesn't account for the fact that tags actually exist in different groups. This dictionary is used when the tag in question is part of a group.

Added in version 8.3.0.

PIL.TiffTags.**TAGS**: dict

The TAGS dictionary maps 16-bit integer TIFF tag numbers to descriptive string names. For instance:

```
>>> from PIL.TiffTags import TAGS
>>> TAGS[0x010e]
'ImageDescription'
```

This dictionary contains a superset of the tags in *TAGS_V2*, common EXIF tags, and other well known metadata tags.

PIL.TiffTags.**TYPES**: dict

The TYPES dictionary maps the TIFF type short integer to a human readable type name.

PIL.TiffTags.**LIBTIFF_CORE**: list

A list of supported tag IDs when writing using LibTIFF.

1.3.26 *JpegPresets* module

JPEG quality settings equivalent to the Photoshop settings. Can be used when saving JPEG files.

The following presets are available by default: `web_low`, `web_medium`, `web_high`, `web_very_high`, `web_maximum`, `low`, `medium`, `high`, `maximum`. More presets can be added to the *presets* dict if needed.

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

Subsampling is the practice of encoding images by implementing less resolution 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 4:2:0.

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

In JPEG compressed data a JPEG marker is used instead of an EXIF tag. (ref.: https://exiv2.org/tags.html)

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 lists. You can pass this dict directly as the qtables argument when saving a JPEG.

The quantization table format in presets is a list with sublists. These formats are interchangeable.

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

PIL.JpegPresets.**presets**: dict

A dictionary of all supported presets.

1.3.27 *PSDraw* module

The *PSDraw* module provides simple print support for PostScript printers. You can print text, graphics and images through this module.

class PIL.PSDraw.**PSDraw**(*fp: IO[bytes] | None = None*)

Sets up printing to the given file. If `fp` is omitted, `sys.stdout.buffer` is assumed.

begin_document(*id: str | None = None*) → None

Set up printing of a document. (Write PostScript DSC header.)

end_document() → None

Ends printing. (Write PostScript DSC footer.)

image(*box: tuple[int, int, int, int]*, *im:* Image.Image, *dpi: int | None = None*) → None

Draw a PIL image, centered in the given box.

line(*xy0: tuple[int, int]*, *xy1: tuple[int, int]*) → None

Draws a line between the two points. Coordinates are given in PostScript point coordinates (72 points per inch, (0, 0) is the lower left corner of the page).

rectangle(*box: tuple[int, int, int, int]*) → None

Draws a rectangle.

Parameters

box – A tuple of four integers, specifying left, bottom, width and height.

setfont(*font: str*, *size: int*) → None

Selects which font to use.

Parameters

- **font** – A PostScript font name
- **size** – Size in points.

text(*xy: tuple[int, int]*, *text: str*) → None

Draws text at the given position. You must use *setfont()* before calling this method.

1.3.28 *PixelAccess* class

The PixelAccess class provides read and write access to *PIL.Image* data at a pixel level.

> **Note**
>
> Accessing individual pixels is fairly slow. If you are looping over all of the pixels in an image, there is likely a faster way using other parts of the Pillow API.
>
> *Image*, *ImageChops* and *ImageOps* have methods for many standard operations. If you wish to perform a custom mapping, check out *point()*.

Example

The following script loads an image, accesses one pixel from it, then changes it.

```
from PIL import Image

with Image.open("hopper.jpg") as im:
    px = im.load()
print(px[4, 4])
px[4, 4] = (0, 0, 0)
print(px[4, 4])
```

Results in the following:

```
(23, 24, 68)
(0, 0, 0)
```

Access using negative indexes is also possible.

```
px[-1, -1] = (0, 0, 0)
print(px[-1, -1])
```

PixelAccess class

class PixelAccess

__getitem__(*self*, *xy: tuple[int, int]*) → float | tuple[int, ...]

Returns the pixel at x,y. The pixel is returned as a single value for single band images or a tuple for multi-band images.

Parameters

xy – The pixel coordinate, given as (x, y).

Returns
a pixel value for single band images, a tuple of pixel values for multiband images.

`__setitem__`(*self*, *xy: tuple[int, int]*, *color: float | tuple[int, ...]*) → None

Modifies the pixel at x,y. The color is given as a single numerical value for single band images, and a tuple for multi-band images. See *Colors* for more information.

Parameters

- **xy** – The pixel coordinate, given as (x, y).
- **color** – The pixel value according to its mode, e.g. tuple (r, g, b) for RGB mode.

1.3.29 *features* module

The *PIL.features* module can be used to detect which Pillow features are available on your system.

PIL.features.**pilinfo**(*out: IO[str] | None = None*, *supported_formats: bool = True*) → None

Prints information about this installation of Pillow. This function can be called with `python3 -m PIL`. It can also be called with `python3 -m PIL.report` or `python3 -m PIL --report` to have "supported_formats" set to `False`, omitting the list of all supported image file formats.

Parameters

- **out** – The output stream to print to. Defaults to `sys.stdout` if `None`.
- **supported_formats** – If `True`, a list of all supported image file formats will be printed.

PIL.features.**check**(*feature: str*) → bool | None

Parameters
feature – A module, codec, or feature name.

Returns
`True` if the module, codec, or feature is available, `False` or `None` otherwise.

PIL.features.**version**(*feature: str*) → str | None

Parameters
feature – The module, codec, or feature to check for.

Returns
The version number as a string, or `None` if unknown or not available.

PIL.features.**get_supported**() → list[str]

Returns
A list of all supported modules, features, and codecs.

Modules

Support for the following modules can be checked:

- `pil`: The Pillow core module, required for all functionality.
- `tkinter`: Tkinter support.
- `freetype2`: FreeType font support via *PIL.ImageFont.truetype()*.
- `littlecms2`: LittleCMS 2 support via *PIL.ImageCms*.
- `webp`: WebP image support.
- `avif`: AVIF image support.

PIL.features.**check_module**(*feature: str*) → bool

Checks if a module is available.

Parameters
feature – The module to check for.

Returns
True if available, False otherwise.

Raises
ValueError – If the module is not defined in this version of Pillow.

PIL.features.**version_module**(*feature: str*) → str | None

Parameters
feature – The module to check for.

Returns
The loaded version number as a string, or None if unknown or not available.

Raises
ValueError – If the module is not defined in this version of Pillow.

PIL.features.**get_supported_modules**() → list[str]

Returns
A list of all supported modules.

Codecs

Support for these is only checked during Pillow compilation. If the required library was uninstalled from the system, the pil core module may fail to load instead. Except for jpg, the version number is checked at run-time.

Support for the following codecs can be checked:

- jpg: (compile time) Libjpeg support, required for JPEG based image formats. Only compile time version number is available.
- jpg_2000: (compile time) OpenJPEG support, required for JPEG 2000 image formats.
- zlib: (compile time) Zlib support, required for zlib compressed formats, such as PNG.
- libtiff: (compile time) LibTIFF support, required for TIFF based image formats.

PIL.features.**check_codec**(*feature: str*) → bool

Checks if a codec is available.

Parameters
feature – The codec to check for.

Returns
True if available, False otherwise.

Raises
ValueError – If the codec is not defined in this version of Pillow.

PIL.features.**version_codec**(*feature: str*) → str | None

Parameters
feature – The codec to check for.

Returns
The version number as a string, or None if not available. Checked at compile time for jpg, run-time otherwise.

Raises
ValueError – If the codec is not defined in this version of Pillow.

PIL.features.**get_supported_codecs**() → list[str]

Returns
A list of all supported codecs.

Features

Some of these are only checked during Pillow compilation. If the required library was uninstalled from the system, the relevant module may fail to load instead. Feature version numbers are available only where stated.

Support for the following features can be checked:

- `libjpeg_turbo`: (compile time) Whether Pillow was compiled against the libjpeg-turbo version of libjpeg. Compile-time version number is available.
- `mozjpeg`: (compile time) Whether Pillow was compiled against the MozJPEG version of libjpeg. Compile-time version number is available.
- `zlib_ng`: (compile time) Whether Pillow was compiled against the zlib-ng version of zlib. Compile-time version number is available.
- `raqm`: Raqm library, required for `ImageFont.Layout.RAQM` in *PIL.ImageFont.truetype()*. Run-time version number is available for Raqm 0.7.0 or newer.
- `libimagequant`: (compile time) ImageQuant quantization support in *PIL.Image.Image.quantize()*. Run-time version number is available.
- `xcb`: (compile time) Support for X11 in *PIL.ImageGrab.grab()* via the XCB library.

PIL.features.**check_feature**(*feature: str*) → bool | None

Checks if a feature is available.

Parameters
feature – The feature to check for.

Returns
`True` if available, `False` if unavailable, `None` if unknown.

Raises
ValueError – If the feature is not defined in this version of Pillow.

PIL.features.**version_feature**(*feature: str*) → str | None

Parameters
feature – The feature to check for.

Returns
The version number as a string, or `None` if not available.

Raises
ValueError – If the feature is not defined in this version of Pillow.

PIL.features.**get_supported_features**() → list[str]

Returns
A list of all supported features.

1.3.30 PIL package (autodoc of remaining modules)

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

PIL module

exception PIL.**UnidentifiedImageError**

Bases: OSError

Raised in *PIL.Image.open()* if an image cannot be opened and identified.

If a PNG image raises this error, setting *ImageFile.LOAD_TRUNCATED_IMAGES* to true may allow the image to be opened after all. The setting will ignore missing data and checksum failures.

BdfFontFile module

Parse X Bitmap Distribution Format (BDF)

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

Bases: *FontFile*

Font file plugin for the X11 BDF format.

PIL.BdfFontFile.**bdf_char**(*f: BinaryIO*) → tuple[str, int, tuple[tuple[int, int], tuple[int, int, int, int], tuple[int, int, int, int]], *Image*] | None

ContainerIO module

class PIL.ContainerIO.**ContainerIO**(*file: IO*, *offset: int*, *length: int*)

Bases: IO

A file object that provides read access to a part of an existing file (for example a TAR file).

close() → None

fileno() → int

flush() → None

isatty() → bool

read(*n: int = -1*) → AnyStr

Read data.

Parameters
n – Number of bytes to read. If omitted, zero or negative, read until end of region.

Returns
An 8-bit string.

readable() → bool

readline(*n: int = -1*) → AnyStr

Read a line of text.

Parameters
n – Number of bytes to read. If omitted, zero or negative, read until end of line.

Returns
An 8-bit string.

readlines(*n: int | None = -1*) → list[AnyStr]

Read multiple lines of text.

Parameters
n – Number of lines to read. If omitted, zero, negative or None, read until end of region.

Returns
A list of 8-bit strings.

seek(*offset: int*, *mode: int = 0*) → int

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.

Returns
Offset from start of region, in bytes.

seekable() → bool

tell() → int

Get current file pointer.

Returns
Offset from start of region, in bytes.

truncate(*size: int | None = None*) → int

writable() → bool

write(*b: AnyStr*) → NoReturn

writelines(*lines: Iterable*) → NoReturn

FontFile module

class PIL.FontFile.**FontFile**

Bases: object

Base class for raster font file handlers.

bitmap: *Image* | None = None

compile() → None

Create metrics and bitmap

save(*filename: str*) → None

Save font

to_imagefont() → *ImageFont*

Convert to ImageFont

PIL.FontFile.**puti16**(*fp: BinaryIO*, *values: tuple[int, int, int, int, int, int, int, int, int, int]*) → None

Write network order (big-endian) 16-bit sequence

GdImageFile module

> **Note**
>
> This format cannot be automatically recognized, so the class is not registered for use with *PIL.Image.open()*. To open a gd file, use the *PIL.GdImageFile.open()* function instead.

> **Warning**
>
> THE GD FORMAT IS NOT DESIGNED FOR DATA INTERCHANGE. This implementation is provided for convenience and demonstrational purposes only.

class PIL.GdImageFile.**GdImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

Image plugin for the GD uncompressed format. Note that this format is not supported by the standard *PIL.Image.open()* function. To use this plugin, you have to import the *PIL.GdImageFile* module and use the *PIL.GdImageFile.open()* function.

format = 'GD'

format_description = 'GD uncompressed images'

PIL.GdImageFile.**open**(*fp: str | bytes | PathLike[str] | PathLike[bytes] | IO[bytes]*, *mode: str = 'r'*) → *GdImageFile*

Load texture from a GD image file.

Parameters

- **fp** – 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

OSError – If the image could not be read.

GimpGradientFile module

Stuff to translate curve segments to palette values (derived from the corresponding code in GIMP, written by Federico Mena Quintero. See the GIMP distribution for more information.)

PIL.GimpGradientFile.**EPSILON = 1e-10**

class PIL.GimpGradientFile.**GimpGradientFile**(*fp: IO[bytes]*)

Bases: *GradientFile*

File handler for GIMP's gradient format.

class PIL.GimpGradientFile.**GradientFile**

Bases: object

getpalette(*entries: int = 256*) → tuple[bytes, str]

gradient: list[tuple[float, float, float, list[float], list[float], Callable[[float, float], float]]] | None = None

PIL.GimpGradientFile.**SEGMENTS** = [<function linear>, <function curved>, <function sine>, <function sphere_increasing>, <function sphere_decreasing>]

PIL.GimpGradientFile.**curved**(*middle: float*, *pos: float*) → float

PIL.GimpGradientFile.**linear**(*middle: float*, *pos: float*) → float

PIL.GimpGradientFile.**sine**(*middle: float*, *pos: float*) → float

PIL.GimpGradientFile.**sphere_decreasing**(*middle: float*, *pos: float*) → float

PIL.GimpGradientFile.**sphere_increasing**(*middle: float*, *pos: float*) → float

GimpPaletteFile module

class PIL.GimpPaletteFile.**GimpPaletteFile**(*fp: IO[bytes]*)

Bases: object

File handler for GIMP's palette format.

classmethod frombytes(*data: bytes*) → *GimpPaletteFile*

getpalette() → tuple[bytes, str]

rawmode = 'RGB'

ImageDraw2 module

(Experimental) WCK-style drawing interface operations

> **See also**
>
> *PIL.ImageDraw*

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

Bases: object

Stores an outline color and width.

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

Bases: object

Stores a fill color

class PIL.ImageDraw2.**Font**(*color: str*, *file: str | bytes | PathLike[str] | PathLike[bytes] | BinaryIO*, *size: float = 12*)

Bases: object

Stores a TrueType font and color

class PIL.ImageDraw2.**Draw**(*image: Image | str*, *size: tuple[int, int] | list[int] | None = None*, *color: float | tuple[float, ...] | str | None = None*)

Bases: object

(Experimental) WCK-style drawing interface

flush() → *Image*

render(*op: str*, *xy: Sequence[float] | Sequence[Sequence[float]]*, *pen:* Pen | Brush | *None*, *brush:* Brush | Pen | *None* = *None*, ***kwargs: Any*) → None

settransform(*offset: tuple[float, float]*) → None

Sets a transformation offset.

arc(*xy: Sequence[float] | Sequence[Sequence[float]]*, *pen:* Pen | Brush | *None*, *start: float*, *end: float*, **options: Any*) → None

Draws an arc (a portion of a circle outline) between the start and end angles, inside the given bounding box.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.arc()*

chord(*xy: Sequence[float] | Sequence[Sequence[float]]*, *pen:* Pen | Brush | *None*, *start: float*, *end: float*, **options: Any*) → None

Same as *arc()*, but connects the end points with a straight line.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.chord()*

ellipse(*xy: Sequence[float] | Sequence[Sequence[float]]*, *pen:* Pen | Brush | *None*, **options: Any*) → None

Draws an ellipse inside the given bounding box.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.ellipse()*

line(*xy: Sequence[float] | Sequence[Sequence[float]]*, *pen:* Pen | Brush | *None*, **options: Any*) → None

Draws a line between the coordinates in the `xy` list.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.line()*

pieslice(*xy: Sequence[float] | Sequence[Sequence[float]]*, *pen:* Pen | Brush | *None*, *start: float*, *end: float*, **options: Any*) → None

Same as arc, but also draws straight lines between the end points and the center of the bounding box.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.pieslice()*

polygon(*xy: Sequence[float] | Sequence[Sequence[float]]*, *pen: Pen | Brush | None*, **options: Any*) → None

Draws a polygon.

The polygon outline consists of straight lines between the given coordinates, plus a straight line between the last and the first coordinate.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.polygon()*

rectangle(*xy: Sequence[float] | Sequence[Sequence[float]]*, *pen: Pen | Brush | None*, **options: Any*) → None

Draws a rectangle.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.rectangle()*

text(*xy: tuple[float, float]*, *text: AnyStr*, *font: Font*) → None

Draws the string at the given position.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.text()*

textbbox(*xy: tuple[float, float]*, *text: AnyStr*, *font: Font*) → tuple[float, float, float, float]

Returns bounding box (in pixels) of given text.

Returns
`(left, top, right, bottom)` bounding box

> **See also**
>
> *PIL.ImageDraw.ImageDraw.textbbox()*

textlength(*text: AnyStr*, *font: Font*) → float

Returns length (in pixels) of given text. This is the amount by which following text should be offset.

> **See also**
>
> *PIL.ImageDraw.ImageDraw.textlength()*

ImageMode module

class PIL.ImageMode.**ModeDescriptor**(*mode: str*, *bands: tuple[str, ...]*, *basemode: str*, *basetype: str*, *typestr: str*)

Bases: NamedTuple

Wrapper for mode strings.

bands: tuple[str, ...]
Alias for field number 1

basemode: str
Alias for field number 2

basetype: str
Alias for field number 3

mode: str
Alias for field number 0

typestr: str
Alias for field number 4

PIL.ImageMode.**getmode**(*mode: str*) → *ModeDescriptor*
Gets a mode descriptor for the given mode.

PaletteFile module

class PIL.PaletteFile.**PaletteFile**(*fp: IO[bytes]*)
Bases: object

File handler for Teragon-style palette files.

getpalette() → tuple[bytes, str]

rawmode = 'RGB'

PcfFontFile module

class PIL.PcfFontFile.**PcfFontFile**(*fp: BinaryIO*, *charset_encoding: str = 'iso8859-1'*)
Bases: *FontFile*

Font file plugin for the X11 PCF format.

name = 'name'

PIL.PcfFontFile.**sz**(*s: bytes*, *o: int*) → bytes

PngImagePlugin.iTXt class

class PIL.PngImagePlugin.**iTXt**(*text: str*, *lang: str | None = None*, *tkey: str | None = None*)
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

lang: str | bytes | None

tkey: str | bytes | None

PngImagePlugin.PngInfo class

class PIL.PngImagePlugin.**PngInfo**

Bases: object

PNG chunk container (for use with save(pnginfo=))

add(*cid: bytes*, *data: bytes*, *after_idat: bool = False*) → None

Appends an arbitrary chunk. Use with caution.

Parameters

- **cid** – a byte string, 4 bytes long.
- **data** – a byte string of the encoded data
- **after_idat** – for use with private chunks. Whether the chunk should be written after IDAT

add_itxt(*key: str | bytes*, *value: str | bytes*, *lang: str | bytes = ''*, *tkey: str | bytes = ''*, *zip: bool = False*) → None

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: str | bytes*, *value: str | bytes | iTXt*, *zip: bool = False*) → None

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

chunks: `list[tuple[bytes, bytes, bool]]`

TarIO module

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

Bases: *ContainerIO*[bytes]

A file object that provides read access to a given member of a TAR file.

WalImageFile module

This reader is based on the specification available from: https://www.flipcode.com/archives/Quake_2_BSP_File_Format.shtml and has been tested with a few sample files found using google.

Note

This format cannot be automatically recognized, so the reader is not registered for use with *PIL.Image.open()*. To open a WAL file, use the *PIL.WalImageFile.open()* function instead.

class PIL.WalImageFile.**WalImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'WAL'

format_description = 'Quake2 Texture'

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

PIL.WalImageFile.**open**(*filename: str | bytes | PathLike[str] | PathLike[bytes] | IO[bytes]*) → *WalImageFile*

Load texture from a Quake2 WAL texture file.

By default, a Quake2 standard palette is attached to the texture. To override the palette, use the *PIL.Image.Image.putpalette()* method.

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

Returns
An image instance.

1.3.31 Plugin reference

AvifImagePlugin module

class PIL.AvifImagePlugin.**AvifImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'AVIF'

format_description = 'AVIF image'

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

load_seek(*pos: int*) → None

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

PIL.AvifImagePlugin.**get_codec_version**(*codec_name: str*) → str | None

BmpImagePlugin module

class PIL.BmpImagePlugin.**BmpImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

Image plugin for the Windows Bitmap format (BMP)

BITFIELDS = 3

COMPRESSIONS = {'BITFIELDS': 3, 'JPEG': 4, 'PNG': 5, 'RAW': 0, 'RLE4': 2, 'RLE8': 1}

JPEG = 4

PNG = 5

RAW = 0

RLE4 = 2

RLE8 = 1

format = 'BMP'

format_description = 'Windows Bitmap'

k = 'PNG'

v = 5

class PIL.BmpImagePlugin.**BmpRleDecoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes* | *bytearray* | *memoryview* | SupportsArrayInterface) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

class PIL.BmpImagePlugin.**DibImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *BmpImageFile*

format = 'DIB'

format_description = 'Windows Bitmap'

BufrStubImagePlugin module

class PIL.BufrStubImagePlugin.**BufrStubImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *StubImageFile*

format = 'BUFR'

format_description = 'BUFR'

PIL.BufrStubImagePlugin.**register_handler**(*handler:* StubHandler | *None*) → None

Install application-specific BUFR image handler.

Parameters
handler – Handler object.

CurImagePlugin module

class PIL.CurImagePlugin.**CurImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *BmpImageFile*

format = 'CUR'

format_description = 'Windows Cursor'

load_end() → None

load_prepare() → None

DcxImagePlugin module

class PIL.DcxImagePlugin.**DcxImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *PcxImageFile*

format = 'DCX'

format_description = 'Intel DCX'

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

DdsImagePlugin module

A Pillow plugin for .dds files (S3TC-compressed aka DXTC) Jerome Leclanche <jerome@leclan.ch>

Documentation: https://web.archive.org/web/20170802060935/http://oss.sgi.com/projects/ogl-sample/registry/EXT/texture_compression_s3tc.txt

The contents of this file are hereby released in the public domain (CC0) Full text of the CC0 license: https://creativecommons.org/publicdomain/zero/1.0/

class PIL.DdsImagePlugin.**D3DFMT**(**values*)

Bases: IntEnum

A1 = 118

A16B16G16R16 = 36

A16B16G16R16F = 113

A1R5G5B5 = 25

A2B10G10R10 = 31

A2B10G10R10_XR_BIAS = 119

A2R10G10B10 = 35

A2W10V10U10 = 67

A32B32G32R32F = 116

A4L4 = 52

A4R4G4B4 = 26

A8 = 28

A8B8G8R8 = 32

A8L8 = 51

A8P8 = 40

A8R3G3B2 = 29

A8R8G8B8 = 21

ATI1 = 826889281

ATI2 = 843666497

BC4S = 1395934018

BC4U = 1429488450

BC5S = 1395999554

BC5U = 1429553986

BINARYBUFFER = 199

CxV8U8 = 117

D15S1 = 73

D16 = 80

D16_LOCKABLE = 70

D24FS8 = 83

D24S8 = 75

D24X4S4 = 79

D24X8 = 77

D32 = 71

D32F_LOCKABLE = 82

D32_LOCKABLE = 84

DX10 = 808540228

DXT1 = 827611204

DXT2 = 844388420

DXT3 = 861165636

DXT4 = 877942852

DXT5 = 894720068

G16R16 = 34

G16R16F = 112

G32R32F = 115

G8R8_G8B8 = 1111970375

INDEX16 = 101

INDEX32 = 102

L16 = 81

L6V5U5 = 61

L8 = 50

MULTI2_ARGB8 = 827606349

P8 = 41

Q16W16V16U16 = 110

Q8W8V8U8 = 63

R16F = 111

R32F = 114

R3G3B2 = 27

R5G6B5 = 23

R8G8B8 = 20

R8G8_B8G8 = 1195525970

S8_LOCKABLE = 85

UNKNOWN = 0

UYVY = 1498831189

V16U16 = 64

V8U8 = 60

VERTEXDATA = 100

X1R5G5B5 = 24

X4R4G4B4 = 30

X8B8G8R8 = 33

X8L8V8U8 = 62

X8R8G8B8 = 22

YUY2 = 844715353

class PIL.DdsImagePlugin.**DDPF**(**values*)

Bases: IntFlag

ALPHA = 2

ALPHAPIXELS = 1

FOURCC = 4

LUMINANCE = 131072

PALETTEINDEXED8 = 32

RGB = 64

class PIL.DdsImagePlugin.**DDSCAPS**(**values*)

Bases: IntFlag

COMPLEX = 8

MIPMAP = 4194304

TEXTURE = 4096

class PIL.DdsImagePlugin.**DDSCAPS2**(**values*)

Bases: IntFlag

CUBEMAP = 512

CUBEMAP_NEGATIVEX = 2048

CUBEMAP_NEGATIVEY = 8192

CUBEMAP_NEGATIVEZ = 32768

CUBEMAP_POSITIVEX = 1024

CUBEMAP_POSITIVEY = 4096

CUBEMAP_POSITIVEZ = 16384

VOLUME = 2097152

class PIL.DdsImagePlugin.**DDSD**(**values*)

Bases: IntFlag

CAPS = 1

DEPTH = 8388608

HEIGHT = 2

LINEARSIZE = 524288

MIPMAPCOUNT = 131072

PITCH = 8

PIXELFORMAT = 4096

WIDTH = 4

class PIL.DdsImagePlugin.**DXGI_FORMAT**(**values*)

Bases: IntEnum

A8P8 = 114

A8_UNORM = 65

AI44 = 111

AYUV = 100

B4G4R4A4_UNORM = 115

B5G5R5A1_UNORM = 86

B5G6R5_UNORM = 85

B8G8R8A8_TYPELESS = 90

B8G8R8A8_UNORM = 87

B8G8R8A8_UNORM_SRGB = 91

B8G8R8X8_TYPELESS = 92

B8G8R8X8_UNORM = 88

B8G8R8X8_UNORM_SRGB = 93

BC1_TYPELESS = 70

BC1_UNORM = 71

BC1_UNORM_SRGB = 72

BC2_TYPELESS = 73

BC2_UNORM = 74

BC2_UNORM_SRGB = 75

BC3_TYPELESS = 76

BC3_UNORM = 77

BC3_UNORM_SRGB = 78

BC4_SNORM = 81

BC4_TYPELESS = 79

BC4_UNORM = 80

BC5_SNORM = 84

BC5_TYPELESS = 82

BC5_UNORM = 83

BC6H_SF16 = 96

BC6H_TYPELESS = 94

BC6H_UF16 = 95

BC7_TYPELESS = 97

BC7_UNORM = 98

BC7_UNORM_SRGB = 99

D16_UNORM = 55

D24_UNORM_S8_UINT = 45

D32_FLOAT = 40

D32_FLOAT_S8X24_UINT = 20

G8R8_G8B8_UNORM = 69

IA44 = 112

NV11 = 110

NV12 = 103

OPAQUE_420 = 106

P010 = 104

P016 = 105

P208 = 130

P8 = 113

R10G10B10A2_TYPELESS = 23

R10G10B10A2_UINT = 25

R10G10B10A2_UNORM = 24

R10G10B10_XR_BIAS_A2_UNORM = 89

R11G11B10_FLOAT = 26

R16G16B16A16_FLOAT = 10

R16G16B16A16_SINT = 14

R16G16B16A16_SNORM = 13

R16G16B16A16_TYPELESS = 9

R16G16B16A16_UINT = 12

R16G16B16A16_UNORM = 11

R16G16_FLOAT = 34

R16G16_SINT = 38

R16G16_SNORM = 37

R16G16_TYPELESS = 33

R16G16_UINT = 36

R16G16_UNORM = 35

R16_FLOAT = 54

R16_SINT = 59

R16_SNORM = 58

R16_TYPELESS = 53

R16_UINT = 57

R16_UNORM = 56

R1_UNORM = 66

R24G8_TYPELESS = 44

R24_UNORM_X8_TYPELESS = 46

R32G32B32A32_FLOAT = 2

R32G32B32A32_SINT = 4

R32G32B32A32_TYPELESS = 1

R32G32B32A32_UINT = 3

R32G32B32_FLOAT = 6

R32G32B32_SINT = 8

R32G32B32_TYPELESS = 5

R32G32B32_UINT = 7

R32G32_FLOAT = 16

R32G32_SINT = 18

R32G32_TYPELESS = 15

R32G32_UINT = 17

R32G8X24_TYPELESS = 19

R32_FLOAT = 41

R32_FLOAT_X8X24_TYPELESS = 21

R32_SINT = 43

R32_TYPELESS = 39

R32_UINT = 42

R8G8B8A8_SINT = 32

R8G8B8A8_SNORM = 31

R8G8B8A8_TYPELESS = 27

R8G8B8A8_UINT = 30

R8G8B8A8_UNORM = 28

R8G8B8A8_UNORM_SRGB = 29

R8G8_B8G8_UNORM = 68

R8G8_SINT = 52

R8G8_SNORM = 51

R8G8_TYPELESS = 48

R8G8_UINT = 50

R8G8_UNORM = 49

R8_SINT = 64

R8_SNORM = 63

R8_TYPELESS = 60

R8_UINT = 62

R8_UNORM = 61

R9G9B9E5_SHAREDEXP = 67

SAMPLER_FEEDBACK_MIN_MIP_OPAQUE = 189

SAMPLER_FEEDBACK_MIP_REGION_USED_OPAQUE = 190

UNKNOWN = 0

V208 = 131

V408 = 132

X24_TYPELESS_G8_UINT = 47

X32_TYPELESS_G8X24_UINT = 22

Y210 = 108

Y216 = 109

Y410 = 101

Y416 = 102

YUY2 = 107

class PIL.DdsImagePlugin.**DdsImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'DDS'

format_description = 'DirectDraw Surface'

load_seek(*pos: int*) → None

class PIL.DdsImagePlugin.**DdsRgbDecoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes* | *bytearray* | *memoryview* | SupportsArrayInterface) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

EpsImagePlugin module

class PIL.EpsImagePlugin.**EpsImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

EPS File Parser for the Python Imaging Library

format = 'EPS'

format_description = 'Encapsulated Postscript'

load(*scale: int = 1*, *transparency: bool = False*) → *Image.core.PixelAccess* | None

Load image data based on tile list

load_seek(*pos: int*) → None

mode_map = {1: 'L', 2: 'LAB', 3: 'RGB', 4: 'CMYK'}

PIL.EpsImagePlugin.**Ghostscript**(*tile: list[*ImageFile._Tile*]*, *size: tuple[int, int]*, *fp: IO[bytes]*, *scale: int = 1*, *transparency: bool = False*) → *Image.core.ImagingCore*

Render an image using Ghostscript

PIL.EpsImagePlugin.**has_ghostscript**() → bool

FitsImagePlugin module

class PIL.FitsImagePlugin.**FitsGzipDecoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes | bytearray | memoryview | SupportsArrayInterface*) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

class PIL.FitsImagePlugin.**FitsImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'FITS'

format_description = 'FITS'

FliImagePlugin module

class PIL.FliImagePlugin.**FliImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'FLI'

format_description = 'Autodesk FLI/FLC Animation'

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

FpxImagePlugin module

class PIL.FpxImagePlugin.**FpxImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

close() → None

Closes the file pointer, if possible.

This operation will destroy the image core and release its memory. The image data will be unusable afterward.

This function is required to close images that have multiple frames or have not had their file read and closed by the *load()* method. See *File handling in Pillow* for more information.

format = 'FPX'

format_description = 'FlashPix'

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

GbrImagePlugin module

class PIL.GbrImagePlugin.**GbrImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'GBR'

format_description = 'GIMP brush file'

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

GifImagePlugin module

class PIL.GifImagePlugin.**GifImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

data() → bytes | None

format = 'GIF'

format_description = 'Compuserve GIF'

global_palette = None

property is_animated: bool

load_end() → None

load_prepare() → None

property n_frames: int

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

PIL.GifImagePlugin.**LOADING_STRATEGY = LoadingStrategy.RGB_AFTER_FIRST**

Added in version 9.1.0.

class PIL.GifImagePlugin.**LoadingStrategy**(**values*)

Bases: IntEnum

Added in version 9.1.0.

RGB_AFTER_DIFFERENT_PALETTE_ONLY = 1

RGB_AFTER_FIRST = 0

RGB_ALWAYS = 2

PIL.GifImagePlugin.**get_interlace**(*im:* Image) → int

PIL.GifImagePlugin.**getdata**(*im:* Image, *offset: tuple[int, int] = (0, 0)*, ***params: Any*) → list[bytes]

Legacy Method

Return a list of strings representing this image. The first string is a local image header, the rest contains encoded image data.

To specify duration, add the time in milliseconds, e.g. `getdata(im_frame, duration=1000)`

Parameters

- **im** – Image object
- **offset** – Tuple of (x, y) pixels. Defaults to (0, 0)
- ****params** – e.g. duration or other encoder info parameters

Returns
List of bytes containing GIF encoded frame data

PIL.GifImagePlugin.**getheader**(*im:* Image, *palette:* *bytes* | *bytearray* | *list[int]* | ImagePalette | *None* = *None*, *info:* *dict[str, Any]* | *None* = *None*) → tuple[list[bytes], list[int] | None]

Legacy Method to get Gif data from image.

Warning:: May modify image data.

Parameters

- **im** – Image object
- **palette** – bytes object containing the source palette, or ....
- **info** – encoderinfo

Returns
tuple of(list of header items, optimized palette)

GribStubImagePlugin module

class PIL.GribStubImagePlugin.**GribStubImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename:* *str* | *bytes* | *None* = *None*)

Bases: *StubImageFile*

format = 'GRIB'

format_description = 'GRIB'

PIL.GribStubImagePlugin.**register_handler**(*handler:* StubHandler | *None*) → None

Install application-specific GRIB image handler.

Parameters
handler – Handler object.

Hdf5StubImagePlugin module

class PIL.Hdf5StubImagePlugin.**HDF5StubImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename:* *str* | *bytes* | *None* = *None*)

Bases: *StubImageFile*

format = 'HDF5'

format_description = 'HDF5'

PIL.Hdf5StubImagePlugin.**register_handler**(*handler:* StubHandler | *None*) → None

Install application-specific HDF5 image handler.

Parameters
handler – Handler object.

IcnsImagePlugin module

class PIL.IcnsImagePlugin.**IcnsFile**(*fobj:* *IO[bytes]*)

Bases: object

SIZES = {(16, 16, 1): [(b'icp4', <function read_png_or_jpeg2000>), (b'is32', <function read_32>), (b's8mk', <function read_mk>)], (16, 16, 2): [(b'ic11', <function read_png_or_jpeg2000>)], (32, 32, 1): [(b'icp5', <function read_png_or_jpeg2000>), (b'il32', <function read_32>), (b'l8mk', <function read_mk>)], (32, 32, 2): [(b'ic12', <function read_png_or_jpeg2000>)], (48, 48, 1): [(b'ih32', <function read_32>), (b'h8mk', <function read_mk>)], (64, 64, 1): [(b'icp6', <function read_png_or_jpeg2000>)], (128, 128, 1): [(b'ic07', <function read_png_or_jpeg2000>), (b'it32', <function read_32t>), (b't8mk', <function read_mk>)], (128, 128, 2): [(b'ic13', <function read_png_or_jpeg2000>)], (256, 256, 1): [(b'ic08', <function read_png_or_jpeg2000>)], (256, 256, 2): [(b'ic14', <function read_png_or_jpeg2000>)], (512, 512, 1): [(b'ic09', <function read_png_or_jpeg2000>)], (512, 512, 2): [(b'ic10', <function read_png_or_jpeg2000>)]}

bestsize() → tuple[int, int, int]

dataforsize(*size: tuple[int, int, int]*) → dict[str, *Image*]

Get an icon resource as {channel: array}. Note that the arrays are bottom-up like windows bitmaps and will likely need to be flipped or transposed in some way.

getimage(*size: tuple[int, int] | tuple[int, int, int] | None = None*) → *Image*

itersizes() → list[tuple[int, int, int]]

class PIL.IcnsImagePlugin.**IcnsImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

PIL image support for Mac OS .icns files. Chooses the best resolution, but will possibly load a different size image if you mutate the size attribute before calling 'load'.

The info dictionary has a key 'sizes' that is a list of sizes that the icns file has.

format = 'ICNS'

format_description = 'Mac OS icns resource'

load(*scale: int | None = None*) → *Image.core.PixelAccess* | None

Load image data based on tile list

property size: tuple[int, int]

PIL.IcnsImagePlugin.**nextheader**(*fobj: IO[bytes]*) → tuple[bytes, int]

PIL.IcnsImagePlugin.**read_32**(*fobj: IO[bytes]*, *start_length: tuple[int, int]*, *size: tuple[int, int, int]*) → dict[str, *Image*]

Read a 32bit RGB icon resource. Seems to be either uncompressed or an RLE packbits-like scheme.

PIL.IcnsImagePlugin.**read_32t**(*fobj: IO[bytes]*, *start_length: tuple[int, int]*, *size: tuple[int, int, int]*) → dict[str, *Image*]

PIL.IcnsImagePlugin.**read_mk**(*fobj: IO[bytes]*, *start_length: tuple[int, int]*, *size: tuple[int, int, int]*) → dict[str, *Image*]

PIL.IcnsImagePlugin.**read_png_or_jpeg2000**(*fobj: IO[bytes]*, *start_length: tuple[int, int]*, *size: tuple[int, int, int]*) → dict[str, *Image*]

IcoImagePlugin module

class PIL.IcoImagePlugin.**IcoFile**(*buf: IO[bytes]*)

Bases: object

frame(*idx: int*) → *Image*

Get an image from frame idx

getentryindex(*size: tuple[int, int]*, *bpp: int | bool = False*) → int

getimage(*size: tuple[int, int]*, *bpp: int | bool = False*) → *Image*

Get an image from the icon

sizes() → set[tuple[int, int]]

Get a set of all available icon sizes and color depths.

class PIL.IcoImagePlugin.**IcoImageFile**(*fp: StrOrBytesPath | IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

PIL read-only image support for Microsoft Windows .ico files.

By default the largest resolution image in the file will be loaded. This can be changed by altering the 'size' attribute before calling 'load'.

The info dictionary has a key 'sizes' that is a list of the sizes available in the icon file.

Handles classic, XP and Vista icon formats.

When saving, PNG compression is used. Support for this was only added in Windows Vista. If you are unable to view the icon in Windows, convert the image to "RGBA" mode before saving.

This plugin is a refactored version of Win32IconImagePlugin by Bryan Davis <casadebender@gmail.com>. https://code.google.com/archive/p/casadebender/wikis/Win32IconImagePlugin.wiki

format = 'ICO'

format_description = 'Windows Icon'

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

load_seek(*pos: int*) → None

property size: tuple[int, int]

class PIL.IcoImagePlugin.**IconHeader**(*width*, *height*, *nb_color*, *reserved*, *planes*, *bpp*, *size*, *offset*, *dim*, *square*, *color_depth*)

Bases: NamedTuple

bpp: int

Alias for field number 5

color_depth: int

Alias for field number 10

dim: tuple[int, int]

Alias for field number 8

height: int

Alias for field number 1

nb_color: *int*

Alias for field number 2

offset: *int*

Alias for field number 7

planes: *int*

Alias for field number 4

reserved: *int*

Alias for field number 3

size: *int*

Alias for field number 6

square: *int*

Alias for field number 9

width: *int*

Alias for field number 0

ImImagePlugin module

class PIL.ImImagePlugin.**ImImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'IM'

format_description = 'IFUNC Image Memory'

property is_animated: *bool*

property n_frames: *int*

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

PIL.ImImagePlugin.**number**(*s: Any*) → float

ImtImagePlugin module

class PIL.ImtImagePlugin.**ImtImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'IMT'

format_description = 'IM Tools'

IptcImagePlugin module

class PIL.IptcImagePlugin.**IptcImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

field() → tuple[tuple[int, int] | None, int]

format = 'IPTC'

format_description = 'IPTC/NAA'

getint(*key: tuple[int, int]*) → int

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

PIL.IptcImagePlugin.**getiptcinfo**(*im:* ImageFile) → dict[tuple[int, int], bytes | list[bytes]] | None

Get IPTC information from TIFF, JPEG, or IPTC file.

Parameters

im – An image containing IPTC data.

Returns

A dictionary containing IPTC information, or None if no IPTC information block was found.

JpegImagePlugin module

PIL.JpegImagePlugin.**APP**(*self:* JpegImageFile, *marker: int*) → None

PIL.JpegImagePlugin.**COM**(*self:* JpegImageFile, *marker: int*) → None

PIL.JpegImagePlugin.**DQT**(*self:* JpegImageFile, *marker: int*) → None

class PIL.JpegImagePlugin.**JpegImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

draft(*mode: str* | *None*, *size: tuple[int, int]* | *None*) → tuple[str, tuple[int, int, float, float]] | None

Configures the image file loader so it returns a version of the image that as closely as possible matches the given mode and size. For example, you can use this method to convert a color JPEG to grayscale while loading it.

If any changes are made, returns a tuple with the chosen `mode` and `box` with coordinates of the original image within the altered one.

Note that this method modifies the *Image* object in place. If the image has already been loaded, this method has no effect.

Note: This method is not implemented for most images. It is currently implemented only for JPEG and MPO images.

Parameters

- **mode** – The requested mode.
- **size** – The requested size in pixels, as a 2-tuple: (width, height).

format = 'JPEG'

format_description = 'JPEG (ISO 10918)'

load_djpeg() → None

load_read(*read_bytes: int*) → bytes

internal: read more image data For premature EOF and LOAD_TRUNCATED_IMAGES adds EOI marker so libjpeg can finish decoding

PIL.JpegImagePlugin.**SOF**(*self:* JpegImageFile, *marker: int*) → None

PIL.JpegImagePlugin.**Skip**(*self:* JpegImageFile, *marker: int*) → None

PIL.JpegImagePlugin.**get_sampling**(*im:* Image) → int

PIL.JpegImagePlugin.**jpeg_factory**(*fp: IO[bytes]*, *filename: str | bytes | None = None*) → *JpegImageFile* | *MpoImageFile*

Jpeg2KImagePlugin module

class PIL.Jpeg2KImagePlugin.**BoxReader**(*fp: IO[bytes]*, *length: int = -1*)

Bases: object

A small helper class to read fields stored in JPEG2000 header boxes and to easily step into and read sub-boxes.

has_next_box() → bool

next_box_type() → bytes

read_boxes() → *BoxReader*

read_fields(*field_format: str*) → tuple[int | bytes, ...]

class PIL.Jpeg2KImagePlugin.**Jpeg2KImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'JPEG2000'

format_description = 'JPEG 2000 (ISO 15444)'

load() → *Image.core.PixelAccess* | None

Load image data based on tile list

property reduce: Callable[[int | tuple[int, int], tuple[int, int, int, int] | None], *Image.Image*] | int

Returns a copy of the image reduced `factor` times. If the size of the image is not dividable by `factor`, the resulting size will be rounded up.

Parameters

- **factor** – A greater than 0 integer or tuple of two integers for width and height separately.

- **box** – An optional 4-tuple of ints providing the source image region to be reduced. The values must be within `(0, 0, width, height)` rectangle. If omitted or `None`, the entire source is used.

McIdasImagePlugin module

class PIL.McIdasImagePlugin.**McIdasImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'MCIDAS'

format_description = 'McIdas area file'

MicImagePlugin module

class PIL.MicImagePlugin.**MicImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *TiffImageFile*

close() → None

Closes the file pointer, if possible.

This operation will destroy the image core and release its memory. The image data will be unusable afterward.

This function is required to close images that have multiple frames or have not had their file read and closed by the *load()* method. See *File handling in Pillow* for more information.

format = 'MIC'

format_description = 'Microsoft Image Composer'

seek(*frame: int*) → None

Select a given frame as current image

tell() → int

Return the current frame number

MpegImagePlugin module

class PIL.MpegImagePlugin.**BitStream**(*fp:* SupportsRead*[bytes]*)

Bases: `object`

next() → int

peek(*bits: int*) → int

read(*bits: int*) → int

skip(*bits: int*) → None

class PIL.MpegImagePlugin.**MpegImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'MPEG'

format_description = 'MPEG'

MpoImagePlugin module

class PIL.MpoImagePlugin.**MpoImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *JpegImageFile*

static adopt(*jpeg_instance:* JpegImageFile, *mpheader: dict[int, Any]* | *None* = *None*) → *MpoImageFile*

Transform the instance of JpegImageFile into an instance of MpoImageFile. After the call, the JpegImageFile is extended to be an MpoImageFile.

This is essentially useful when opening a JPEG file that reveals itself as an MPO, to avoid double call to _open.

format = 'MPO'

format_description = 'MPO (CIPA DC-007)'

load_seek(*pos: int*) → None

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

MspImagePlugin module

class PIL.MspImagePlugin.**MspDecoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes* | *bytearray* | *memoryview* | SupportsArrayInterface) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

class PIL.MspImagePlugin.**MspImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'MSP'

format_description = 'Windows Paint'

PalmImagePlugin module

PIL.PalmImagePlugin.**build_prototype_image**() → *Image*

PcdImagePlugin module

class PIL.PcdImagePlugin.**PcdImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'PCD'

format_description = 'Kodak PhotoCD'

load_end() → None

load_prepare() → None

PcxImagePlugin module

class PIL.PcxImagePlugin.**PcxImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'PCX'

format_description = 'Paintbrush'

PdfImagePlugin module

PixarImagePlugin module

class PIL.PixarImagePlugin.**PixarImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'PIXAR'

format_description = 'PIXAR raster image'

PngImagePlugin module

class PIL.PngImagePlugin.**Blend**(**values*)

Bases: IntEnum

OP_OVER = 1

This frame should be alpha composited with the previous output image contents. See *Saving APNG sequences*.

OP_SOURCE = 0

All color components of this frame, including alpha, overwrite the previous output image contents. See *Saving APNG sequences*.

class PIL.PngImagePlugin.**ChunkStream**(*fp: IO[bytes]*)

Bases: object

call(*cid: bytes*, *pos: int*, *length: int*) → bytes

Call the appropriate chunk handler

close() → None

crc(*cid: bytes*, *data: bytes*) → None

Read and verify checksum

crc_skip(*cid: bytes*, *data: bytes*) → None

Read checksum

push(*cid: bytes*, *pos: int*, *length: int*) → None

read() → tuple[bytes, int, int]

Fetch a new chunk. Returns header information.

verify(*endchunk: bytes = b'IEND'*) → list[bytes]

fp: IO[bytes] | None

queue: list[tuple[bytes, int, int]] | None

class PIL.PngImagePlugin.**Disposal**(**values*)

Bases: IntEnum

OP_BACKGROUND = 1

This frame's modified region is cleared to fully transparent black before rendering the next frame. See *Saving APNG sequences*.

OP_NONE = 0

No disposal is done on this frame before rendering the next frame. See *Saving APNG sequences*.

OP_PREVIOUS = 2

This frame's modified region is reverted to the previous frame's contents before rendering the next frame. See *Saving APNG sequences*.

class PIL.PngImagePlugin.**PngImageFile**(*fp: StrOrBytesPath | IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

getexif() → *Exif*

Gets EXIF data from the image.

Returns
an *Exif* object.

load_end() → None

internal: finished reading image data

load_prepare() → None

internal: prepare to read PNG file

load_read(*read_bytes: int*) → bytes

internal: read more image data

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

verify() → None

Verify PNG file

format = 'PNG'

format_description = 'Portable network graphics'

property text: dict[str, str | *iTXt*]

class PIL.PngImagePlugin.**PngStream**(*fp: IO[bytes]*)

Bases: *ChunkStream*

check_text_memory(*chunklen: int*) → None

chunk_IDAT(*pos: int*, *length: int*) → NoReturn

chunk_IEND(*pos: int*, *length: int*) → NoReturn

chunk_IHDR(*pos: int*, *length: int*) → bytes

chunk_PLTE(*pos: int*, *length: int*) → bytes

chunk_acTL(*pos: int*, *length: int*) → bytes

chunk_cHRM(*pos: int*, *length: int*) → bytes

chunk_eXIf(*pos: int*, *length: int*) → bytes

chunk_fcTL(*pos: int*, *length: int*) → bytes

chunk_fdAT(*pos: int*, *length: int*) → bytes

chunk_gAMA(*pos: int*, *length: int*) → bytes

chunk_iCCP(*pos: int*, *length: int*) → bytes

chunk_iTXt(*pos: int*, *length: int*) → bytes

chunk_pHYs(*pos: int*, *length: int*) → bytes

chunk_sRGB(*pos: int*, *length: int*) → bytes

chunk_tEXt(*pos: int*, *length: int*) → bytes

chunk_tRNS(*pos: int*, *length: int*) → bytes

chunk_zTXt(*pos: int*, *length: int*) → bytes

rewind() → None

save_rewind() → None

im_custom_mimetype: str | None

im_info: dict[str | tuple[int, int], Any]

im_n_frames: int | None

im_palette: tuple[str, bytes] | None

im_text: dict[str, str | *iTXt*]

im_tile: list[*ImageFile._Tile*]

PIL.PngImagePlugin.**getchunks**(*im:* Image.Image, ***params: Any*) → list[tuple[bytes, bytes, bytes]]

Return a list of PNG chunks representing this image.

PIL.PngImagePlugin.**is_cid**(*string*, *pos=0*, *endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

PIL.PngImagePlugin.**putchunk**(*fp: IO[bytes]*, *cid: bytes*, **data: bytes*) → None

Write a PNG chunk (including CRC field)

PIL.PngImagePlugin.**MAX_TEXT_CHUNK = 1048576**

Maximum decompressed size for a iTXt or zTXt chunk. Eliminates decompression bombs where compressed chunks can expand 1000x. See *Text in PNG File Format*.

PIL.PngImagePlugin.**MAX_TEXT_MEMORY = 67108864**

Set the maximum total text chunk size. See *Text in PNG File Format*.

PpmImagePlugin module

class PIL.PpmImagePlugin.**PpmDecoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes | bytearray | memoryview | SupportsArrayInterface*) → tuple[int, int]

Override to perform the decoding process.

Parameters

buffer – A bytes object with the data to be decoded.

Returns

A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

class PIL.PpmImagePlugin.**PpmImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'PPM'

format_description = 'Pbmplus image'

class PIL.PpmImagePlugin.**PpmPlainDecoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes | bytearray | memoryview | SupportsArrayInterface*) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

PsdImagePlugin module

class PIL.PsdImagePlugin.**PsdImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'PSD'

format_description = 'Adobe Photoshop'

property is_animated: bool

property layers: list[tuple[str, str, tuple[int, int, int, int], list[*_Tile*]]]

property n_frames: int

seek(*layer: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

SgiImagePlugin module

class PIL.SgiImagePlugin.**SGI16Decoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes | bytearray | memoryview | SupportsArrayInterface*) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

class PIL.SgiImagePlugin.**SgiImageFile**(*fp: StrOrBytesPath | IO[bytes], filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'SGI'

format_description = 'SGI Image File Format'

SpiderImagePlugin module

class PIL.SpiderImagePlugin.**SpiderImageFile**(*fp: StrOrBytesPath | IO[bytes], filename: str | bytes | None = None*)

Bases: *ImageFile*

convert2byte(*depth: int = 255*) → *Image*

format = 'SPIDER'

format_description = 'Spider 2D image'

property is_animated: bool

property n_frames: int

seek(*frame: int*) → None

Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int

Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

tkPhotoImage() → *ImageTk.PhotoImage*

PIL.SpiderImagePlugin.**isInt**(*f: Any*) → int

PIL.SpiderImagePlugin.**isSpiderHeader**(*t: tuple[float, ...]*) → int

PIL.SpiderImagePlugin.**isSpiderImage**(*filename: str*) → int

PIL.SpiderImagePlugin.**loadImageSeries**(*filelist: list[str] | None = None*) → list[*Image*] | None

create a list of *Image* objects for use in a montage

PIL.SpiderImagePlugin.**makeSpiderHeader**(*im:* Image) → list[bytes]

SunImagePlugin module

class PIL.SunImagePlugin.**SunImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'SUN'

format_description = 'Sun Raster File'

TgaImagePlugin module

class PIL.TgaImagePlugin.**TgaImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'TGA'

format_description = 'Targa'

load_end() → None

TiffImagePlugin module

class PIL.TiffImagePlugin.**AppendingTiffWriter**(*fn:* StrOrBytesPath | *IO[bytes]*, *new: bool = False*)

Bases: BytesIO

Tags = {273, 288, 324, 519, 520, 521}

close() → None

Disable all I/O operations.

f: IO[bytes]

fieldSizes = [0, 1, 1, 2, 4, 8, 1, 1, 2, 4, 8, 4, 8, 4, 2, 4, 8]

finalize() → None

fixIFD() → None

fixOffsets(*count: int*, *isShort: bool = False*, *isLong: bool = False*) → None

goToEnd() → None

newFrame() → None

readLong() → int

readShort() → int

rewriteLastLong(*value: int*) → None

rewriteLastShort(*value: int*) → None

rewriteLastShortToLong(*value: int*) → None

seek(*offset: int*, *whence: int = 0*) → int

Parameters

- **offset** – Distance to seek.
- **whence** – Whether the distance is relative to the start, end or current position.

Returns
The resulting position, relative to the start.

setEndian(*endian: str*) → None

setup() → None

skipIFDs() → None

tell() → int
Current file position, an integer.

write(*data:* Buffer, */*) → int
Write bytes to file.

Return the number of bytes written.

writeLong(*value: int*) → None

writeShort(*value: int*) → None

class PIL.TiffImagePlugin.**IFDRational**(*value: float | Fraction | IFDRational*, *denominator: int = 1*)
Bases: Rational

Implements a rational class where 0/0 is a legal value to match the in the wild use of exif rationals.

e.g., DigitalZoomRatio - 0.00/0.00 indicates that no digital zoom was used

property denominator: int
The denominator of a rational number in lowest terms.

This denominator should be positive.

limit_rational(*max_denominator: int*) → tuple[*IntegralLike*, int]

Parameters
max_denominator – Integer, the maximum denominator value

Returns
Tuple of (numerator, denominator)

property numerator: *IntegralLike*
The numerator of a rational number in lowest terms.

PIL.TiffImagePlugin.**ImageFileDirectory**
alias of *ImageFileDirectory_v1*

class PIL.TiffImagePlugin.**ImageFileDirectory_v1**(**args: Any*, ***kwargs: Any*)

Bases: *ImageFileDirectory_v2*

This class represents the **legacy** interface to a TIFF tag directory.

Exposes a dictionary interface of the tags in the directory:

```
ifd = ImageFileDirectory_v1()
ifd[key] = 'Some Data'
ifd.tagtype[key] = TiffTags.ASCII
print(ifd[key])
('Some Data',)
```

Also contains a dictionary of tag types as read from the tiff image file, *tagtype*.

Values are returned as a tuple.

Deprecated since version 3.0.0.

classmethod from_v2(*original:* ImageFileDirectory_v2) → *ImageFileDirectory_v1*

Returns an *ImageFileDirectory_v1* instance with the same data as is contained in the original *ImageFileDirectory_v2* instance.

Returns
ImageFileDirectory_v1

property tagdata

property tags

tagtype: dict[int, int]

Dictionary of tag types

to_v2() → *ImageFileDirectory_v2*

Returns an *ImageFileDirectory_v2* instance with the same data as is contained in the original *ImageFileDirectory_v1* instance.

Returns
ImageFileDirectory_v2

class PIL.TiffImagePlugin.**ImageFileDirectory_v2**(*ifh: bytes = b'II*\x00\x00\x00\x00\x00'*, *prefix: bytes | None = None*, *group: int | None = None*)

Bases: MutableMapping

This class represents a TIFF tag directory. To speed things up, we don't decode tags unless they're asked for.

Exposes a dictionary interface of the tags in the directory:

```
ifd = ImageFileDirectory_v2()
ifd[key] = 'Some Data'
ifd.tagtype[key] = TiffTags.ASCII
print(ifd[key])
'Some Data'
```

Individual values are returned as the strings or numbers, sequences are returned as tuples of the values.

The tiff metadata type of each item is stored in a dictionary of tag types in *tagtype*. The types are read from a tiff file, guessed from the type added, or added manually.

Data Structures:

- `self.tagtype = {}`
  - Key: numerical TIFF tag number
  - Value: integer corresponding to the data type from *TiffTags.TYPES*

  Added in version 3.0.0.

'Internal' data structures:

- `self._tags_v2 = {}`
  - Key: numerical TIFF tag number
  - Value: decoded data, as tuple for multiple values
- `self._tagdata = {}`
  - Key: numerical TIFF tag number
  - Value: undecoded byte string from file
- `self._tags_v1 = {}`
  - Key: numerical TIFF tag number
  - Value: decoded data in the v1 format

Tags will be found in the private attributes `self._tagdata`, and in `self._tags_v2` once decoded.

`self.legacy_api` is a value for internal use, and shouldn't be changed from outside code. In cooperation with *ImageFileDirectory_v1*, if `legacy_api` is true, then decoded tags will be populated into both `_tags_v1` and `_tags_v2`. `_tags_v2` will be used if this IFD is used in the TIFF save routine. Tags should be read from `_tags_v1` if `legacy_api == true`.

property legacy_api: bool

load(*fp: IO[bytes]*) → None

load_byte(*data: bytes*, *legacy_api: bool = True*) → bytes

load_double(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_float(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_long(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_long8(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_rational(*data: bytes*, *legacy_api: bool = True*) → tuple[tuple[int, int] | *IFDRational*, ...]

load_short(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_signed_byte(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_signed_long(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_signed_rational(*data: bytes*, *legacy_api: bool = True*) → tuple[tuple[int, int] | *IFDRational*, ...]

load_signed_short(*data: bytes*, *legacy_api: bool = True*) → tuple[Any, ...]

load_string(*data: bytes*, *legacy_api: bool = True*) → str

load_undefined(*data: bytes*, *legacy_api: bool = True*) → bytes

named() → dict[str, Any]

Returns
dict of name|key: value

Returns the complete tag dictionary, with named tags where possible.

property offset

property prefix

reset() → None

save(*fp: IO[bytes]*) → int

tagtype: dict[int, int]
Dictionary of tag types

tobytes(*offset: int = 0*) → bytes

write_byte(*data: bytes | int | IFDRational*) → bytes

write_double(**values*)

write_float(**values*)

write_long(**values*)

write_long8(**values*)

write_rational(**values: IFDRational*) → bytes

write_short(**values*)

write_signed_byte(**values*)

write_signed_long(**values*)

write_signed_rational(**values: IFDRational*) → bytes

write_signed_short(**values*)

write_string(*value: str | bytes | int*) → bytes

write_undefined(*value: bytes | int | IFDRational*) → bytes

class PIL.TiffImagePlugin.**TiffImageFile**(*fp: StrOrBytesPath | IO[bytes], filename: str | bytes | None = None*)

Bases: *ImageFile*

format = 'TIFF'

format_description = 'Adobe TIFF'

get_photoshop_blocks() → dict[int, dict[str, bytes]]

Returns a dictionary of Photoshop "Image Resource Blocks". The keys are the image resource ID. For more information, see https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/#50577409_pgfId-1037727

Returns
Photoshop "Image Resource Blocks" in a dictionary.

load() → *Image.core.PixelAccess* | None
Load image data based on tile list

load_end() → None

load_prepare() → None

property n_frames: int

seek(*frame: int*) → None
Select a given frame as current image

tag: *ImageFileDirectory_v1*
Legacy tag entries

tag_v2: *ImageFileDirectory_v2*
Image file directory (tag dictionary)

tell() → int
Return the current frame number

WebPImagePlugin module

class PIL.WebPImagePlugin.**WebPImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'WEBP'

format_description = 'WebP image'

load() → *Image.core.PixelAccess* | None
Load image data based on tile list

load_seek(*pos: int*) → None

seek(*frame: int*) → None
Seeks to the given frame in this sequence file. If you seek beyond the end of the sequence, the method raises an `EOFError` exception. When a sequence file is opened, the library automatically seeks to frame 0.

See *tell()*.

If defined, *n_frames* refers to the number of available frames.

Parameters
frame – Frame number, starting at 0.

Raises
EOFError – If the call attempts to seek beyond the end of the sequence.

tell() → int
Returns the current frame number. See *seek()*.

If defined, *n_frames* refers to the number of available frames.

Returns
Frame number, starting with 0.

WmfImagePlugin module

class PIL.WmfImagePlugin.**WmfStubImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *StubImageFile*

format = 'WMF'

format_description = 'Windows Metafile'

load(*dpi: float* | *tuple[float, float]* | *None* = *None*) → *Image.core.PixelAccess* | None

Load image data based on tile list

PIL.WmfImagePlugin.**register_handler**(*handler:* StubHandler | *None*) → None

Install application-specific WMF image handler.

Parameters
handler – Handler object.

XVThumbImagePlugin module

class PIL.XVThumbImagePlugin.**XVThumbImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'XVThumb'

format_description = 'XV thumbnail image'

XbmImagePlugin module

class PIL.XbmImagePlugin.**XbmImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'XBM'

format_description = 'X11 Bitmap'

XpmImagePlugin module

class PIL.XpmImagePlugin.**XpmDecoder**(*mode: str*, **args: Any*)

Bases: *PyDecoder*

decode(*buffer: bytes* | *bytearray* | *memoryview* | SupportsArrayInterface) → tuple[int, int]

Override to perform the decoding process.

Parameters
buffer – A bytes object with the data to be decoded.

Returns
A tuple of `(bytes consumed, errcode)`. If finished with decoding return -1 for the bytes consumed. Err codes are from *ImageFile.ERRORS*.

class PIL.XpmImagePlugin.**XpmImageFile**(*fp:* StrOrBytesPath | *IO[bytes]*, *filename: str* | *bytes* | *None* = *None*)

Bases: *ImageFile*

format = 'XPM'

format_description = 'X11 Pixel Map'

load_read(*read_bytes: int*) → bytes

1.3.32 Internal reference

File handling in Pillow

When opening a file as an image, Pillow requires a filename, `os.PathLike` object, or a file-like object. Pillow uses the filename or `Path` to open a file, so for the rest of this article, they will all be treated as a file-like object.

The following are all equivalent:

```
from PIL import Image
import io
import pathlib

with Image.open("test.jpg") as im:
    ...

with Image.open(pathlib.Path("test.jpg")) as im2:
    ...

with open("test.jpg", "rb") as f:
    im3 = Image.open(f)
    ...

with open("test.jpg", "rb") as f:
    im4 = Image.open(io.BytesIO(f.read()))
    ...
```

If a filename or a path-like object is passed to Pillow, then the resulting file object opened by Pillow may also be closed by Pillow after the `Image.Image.load()` method is called, provided the associated image does not have multiple frames.

Pillow cannot in general close and reopen a file, so any access to that file needs to be prior to the close.

Image lifecycle

- `Image.open()` Filenames and `Path` objects are opened as a file. Metadata is read from the open file. The file is left open for further usage.
- `Image.Image.load()` When the pixel data from the image is required, `load()` is called. The current frame is read into memory. The image can now be used independently of the underlying image file.

  Any Pillow method that creates a new image instance based on another will internally call `load()` on the original image and then read the data. The new image instance will not be associated with the original image file.

  If a filename or a `Path` object was passed to `Image.open()`, then the file object was opened by Pillow and is considered to be used exclusively by Pillow. So if the image is a single-frame image, the file will be closed in this method after the frame is read. If the image is a multi-frame image, (e.g. multipage TIFF and animated GIF) the image file is left open so that `Image.Image.seek()` can load the appropriate frame.
- `Image.Image.close()` Closes the file and destroys the core image object.

  The Pillow context manager will also close the file, but will not destroy the core image object. e.g.:

```
with Image.open("test.jpg") as img:
    img.load()
assert img.fp is None
img.save("test.png")
```

The lifecycle of a single-frame image is relatively simple. The file must remain open until the `load()` or `close()` function is called or the context manager exits.

Multi-frame images are more complicated. The `load()` method is not a terminal method, so it should not close the underlying file. In general, Pillow does not know if there are going to be any requests for additional data until the caller has explicitly closed the image.

Complications

- `TiffImagePlugin` has some code to pass the underlying file descriptor into libtiff (if working on an actual file). Since libtiff closes the file descriptor internally, it is duplicated prior to passing it into libtiff.
- After a file has been closed, operations that require file access will fail:

```
with open("test.jpg", "rb") as f:
    im5 = Image.open(f)
im5.load()  # FAILS, closed file

with Image.open("test.jpg") as im6:
    pass
im6.load()  # FAILS, closed file
```

Proposed file handling

- `Image.Image.load()` should close the image file, unless there are multiple frames.
- `Image.Image.seek()` should never close the image file.
- Users of the library should use a context manager or call `Image.Image.close()` on any image opened with a filename or `Path` object to ensure that the underlying file is closed.

Limits

This page is documentation to the various fundamental size limits in the Pillow implementation.

Internal limits

- Image sizes cannot be negative. These are checked both in `Storage.c` and `Image.py`
- Image sizes may be 0. (Although not in 3.4)
- Maximum pixel dimensions are limited to INT32, or 2^31 by the sizes in the image header.
- Individual allocations are limited to 2GB in `Storage.c`
- The 2GB allocation puts an upper limit to the xsize of the image of either 2^31 for 'L' or 2^29 for 'RGB'
- Individual memory mapped segments are limited to 2GB in map.c based on the overflow checks. This requires that any memory mapped image is smaller than 2GB, as calculated by `y*stride` (so 2Gpx for 'L' images, and .5Gpx for 'RGB')

Format size limits

- ICO: Max size is 256x256
- WebP: 16383x16383 (underlying library size limit: https://developers.google.com/speed/webp/docs/api)

Block allocator

Previous design

Historically there have been two image allocators in Pillow: `ImagingAllocateBlock` and `ImagingAllocateArray`. The first works for images smaller than 16MB of data and allocates one large chunk of memory of `im->linesize * im->ysize` bytes. The second works for large images and makes one allocation for each scan line of size `im->linesize` bytes. This makes for a very sharp transition between one allocation and potentially thousands of small allocations, leading to unpredictable performance penalties around the transition.

New design

`ImagingAllocateArray` now allocates space for images as a chain of blocks with a maximum size of 16MB. If there is a memory allocation error, it falls back to allocating a 4KB block, or at least one scan line. This is now the default for all internal allocations.

`ImagingAllocateBlock` is now only used for those cases when we are specifically requesting a single segment of memory for sharing with other code.

Memory pools

There is now a memory pool to contain a supply of recently freed blocks, which can then be reused without going back to the OS for a fresh allocation. This caching of free blocks is currently disabled by default, but can be enabled and tweaked using three environment variables:

- `PILLOW_ALIGNMENT`, in bytes. Specifies the alignment of memory allocations. Valid values are powers of 2 between 1 and 128, inclusive. Defaults to 1.
- `PILLOW_BLOCK_SIZE`, in bytes, K, or M. Specifies the maximum block size for `ImagingAllocateArray`. Valid values are integers, with an optional `k` or `m` suffix. Defaults to 16M.
- `PILLOW_BLOCKS_MAX` Specifies the number of freed blocks to retain to fill future memory requests. Any freed blocks over this threshold will be returned to the OS immediately. Defaults to 0.

Internal modules

_binary module

Binary input/output support routines.

PIL._binary.**i16be**(*c: bytes*, *o: int = 0*) → int

PIL._binary.**i16le**(*c: bytes*, *o: int = 0*) → int

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

Parameters

- **c** – string containing bytes to convert
- **o** – offset of bytes to convert in string

PIL._binary.**i32be**(*c: bytes*, *o: int = 0*) → int

PIL._binary.**i32le**(*c: bytes*, *o: int = 0*) → int

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

Parameters

- **c** – string containing bytes to convert
- **o** – offset of bytes to convert in string

PIL._binary.**i8**(*c: bytes*) → int

PIL._binary.**o16be**(*i: int*) → bytes

PIL._binary.**o16le**(*i: int*) → bytes

PIL._binary.**o32be**(*i: int*) → bytes

PIL._binary.**o32le**(*i: int*) → bytes

PIL._binary.**o8**(*i: int*) → bytes

PIL._binary.**si16be**(*c: bytes*, *o: int = 0*) → int

Converts a 2-bytes (16 bits) string to a signed integer, big endian.

Parameters

- **c** – string containing bytes to convert
- **o** – offset of bytes to convert in string

PIL._binary.**si16le**(*c: bytes*, *o: int = 0*) → int

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

Parameters

- **c** – string containing bytes to convert
- **o** – offset of bytes to convert in string

PIL._binary.**si32be**(*c: bytes*, *o: int = 0*) → int

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

Parameters

- **c** – string containing bytes to convert
- **o** – offset of bytes to convert in string

PIL._binary.**si32le**(*c: bytes*, *o: int = 0*) → int

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

Parameters

- **c** – string containing bytes to convert
- **o** – offset of bytes to convert in string

_deprecate module

PIL._deprecate.**deprecate**(*deprecated: str*, *when: int | None*, *replacement: str | None = None*, ***, *action: str | None = None*, *plural: bool = False*, *stacklevel: int = 3*) → None

Deprecations helper.

Parameters

- **deprecated** – Name of thing to be deprecated.
- **when** – Pillow major version to be removed in.
- **replacement** – Name of replacement.
- **action** – Instead of "replacement", give a custom call to action e.g. "Upgrade to new thing".
- **plural** – if the deprecated thing is plural, needing "are" instead of "is".

Usually of the form:

"[deprecated] is deprecated and will be removed in Pillow [when] (yyyy-mm-dd). Use [replacement] instead."

You can leave out the replacement sentence:

"[deprecated] is deprecated and will be removed in Pillow [when] (yyyy-mm-dd)"

Or with another call to action:

"[deprecated] is deprecated and will be removed in Pillow [when] (yyyy-mm-dd). [action]."

_tkinter_finder module

Find compiled module linking to Tcl / Tk libraries

_typing module

Provides a convenient way to import type hints that are not available on some Python versions.

class PIL._typing.**Buffer**

Typing alias.

class PIL._typing.**IntegralLike**

Typing alias.

class PIL._typing.**NumpyArray**

Typing alias.

class PIL._typing.**StrOrBytesPath**

Typing alias.

class PIL._typing.**SupportsRead**

An object that supports the read method.

_util module

class PIL._util.**DeferredError**(*ex: BaseException*)

Bases: object

static new(*ex: BaseException*) → Any

Creates an object that raises the wrapped exception `ex` when used, and casts it to Any type.

PIL._util.**is_path**(*f: Any*) → TypeGuard[*StrOrBytesPath*]

_version module

PIL._version.**__version__**: str

This is the master version number for Pillow, all other uses reference this module.

PIL.Image.core module

An internal interface module previously known as *_imaging*, implemented in _imaging.c.

class PIL.Image.core.**ImagingCore**

A representation of the image data.

C extension debugging on Linux, with GBD/Valgrind

Install the tools

You need some basics in addition to the basic tools to build pillow. These are what's required on Ubuntu, YMMV for other distributions.

- python3-dbg package for the gdb extensions and python symbols
- gdb and valgrind
- Potentially debug symbols for libraries. On Ubuntu you can follow those instructions to install the corresponding packages: Debug Symbol Packages

Then sudo apt-get install libtiff5-dbgsym

- There's a bug with the python3-dbg package for at least Python 3.8 on Ubuntu 20.04, and you need to add a new link or two to make it autoload when running Python:

```
cd /usr/share/gdb/auto-load/usr/bin
ln -s python3.8m-gdb.py python3.8d-gdb.py
```

- In Ubuntu 18.04, it's actually including the path to the virtualenv in the search for the python3.*-gdb.py file, but you can helpfully put in the same directory as the binary.
- I also find that history is really useful for gdb, so I added this to my ~/.gdbinit file:

```
set history filename ~/.gdb_history
set history save on
```

- If the python stack isn't working in gdb, then set debug auto-load can also be helpful in .gdbinit.
- Make a virtualenv with the debug python and activate it, then install whatever dependencies are required and build. You want to build with the debug python so you get symbols for your extension.

```
virtualenv -p python3.8-dbg ~/vpy38-dbg
source ~/vpy38-dbg/bin/activate
cd ~/Pillow && make install
```

Test case

Take your test image, and make a really simple harness.

```
from PIL import Image

```

(continues on next page)

(continued from previous page)

```
with Image.open(path) as im:
    im.load()
```

- Run this through valgrind, but note that python triggers some issues on its own, so you're looking for items within the Pillow hierarchy that don't look like they're solely in the python call chain. In this example, the ones we're interested in are after the warnings, and have `decode.c` and `TiffDecode.c` in the call stack:

```
(vpy38-dbg) ubuntu@primary:~/Home/tests$ valgrind python test_tiff.py
==51890== Memcheck, a memory error detector
==51890== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==51890== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==51890== Command: python test_tiff.py
==51890==
==51890== Invalid read of size 4
==51890==    at 0x472E3D: address_in_range (obmalloc.c:1401)
==51890==    by 0x472EEA: pymalloc_free (obmalloc.c:1677)
==51890==    by 0x474960: _PyObject_Free (obmalloc.c:1896)
==51890==    by 0x473BAC: _PyMem_DebugRawFree (obmalloc.c:2187)
==51890==    by 0x473BD4: _PyMem_DebugFree (obmalloc.c:2318)
==51890==    by 0x474C08: PyObject_Free (obmalloc.c:709)
==51890==    by 0x45DD60: dictresize (dictobject.c:1259)
==51890==    by 0x45DD76: insertion_resize (dictobject.c:1019)
==51890==    by 0x464F30: PyDict_SetDefault (dictobject.c:2924)
==51890==    by 0x4D03BE: PyUnicode_InternInPlace (unicodeobject.c:15289)
==51890==    by 0x4D0700: PyUnicode_InternFromString (unicodeobject.c:15322)
==51890==    by 0x64D2FC: descr_new (descrobject.c:857)
==51890==  Address 0x4c1b020 is 384 bytes inside a block of size 1,160 free'd
==51890==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_
↪memcheck-amd64-linux.so)
==51890==    by 0x4735D3: _PyMem_RawFree (obmalloc.c:127)
==51890==    by 0x473BAC: _PyMem_DebugRawFree (obmalloc.c:2187)
==51890==    by 0x474941: PyMem_RawFree (obmalloc.c:595)
==51890==    by 0x47496E: _PyObject_Free (obmalloc.c:1898)
==51890==    by 0x473BAC: _PyMem_DebugRawFree (obmalloc.c:2187)
==51890==    by 0x473BD4: _PyMem_DebugFree (obmalloc.c:2318)
==51890==    by 0x474C08: PyObject_Free (obmalloc.c:709)
==51890==    by 0x45DD60: dictresize (dictobject.c:1259)
==51890==    by 0x45DD76: insertion_resize (dictobject.c:1019)
==51890==    by 0x464F30: PyDict_SetDefault (dictobject.c:2924)
==51890==    by 0x4D03BE: PyUnicode_InternInPlace (unicodeobject.c:15289)
==51890==  Block was alloc'd at
==51890==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_
↪memcheck-amd64-linux.so)
==51890==    by 0x473646: _PyMem_RawMalloc (obmalloc.c:99)
==51890==    by 0x473529: _PyMem_DebugRawAlloc (obmalloc.c:2120)
==51890==    by 0x473565: _PyMem_DebugRawMalloc (obmalloc.c:2153)
==51890==    by 0x4748B1: PyMem_RawMalloc (obmalloc.c:572)
==51890==    by 0x475909: _PyObject_Malloc (obmalloc.c:1628)
==51890==    by 0x473529: _PyMem_DebugRawAlloc (obmalloc.c:2120)
==51890==    by 0x473565: _PyMem_DebugRawMalloc (obmalloc.c:2153)
==51890==    by 0x4736B0: _PyMem_DebugMalloc (obmalloc.c:2303)
==51890==    by 0x474B78: PyObject_Malloc (obmalloc.c:685)
```

(continues on next page)

(continued from previous page)

```
==51890==    by 0x45C435: new_keys_object (dictobject.c:558)
==51890==    by 0x45DA95: dictresize (dictobject.c:1202)
==51890==
==51890== Invalid read of size 4
==51890==    at 0x472E3D: address_in_range (obmalloc.c:1401)
==51890==    by 0x47594A: pymalloc_realloc (obmalloc.c:1929)
==51890==    by 0x475A02: _PyObject_Realloc (obmalloc.c:1982)
==51890==    by 0x473DCA: _PyMem_DebugRawRealloc (obmalloc.c:2240)
==51890==    by 0x473FF8: _PyMem_DebugRealloc (obmalloc.c:2326)
==51890==    by 0x4749FB: PyMem_Realloc (obmalloc.c:623)
==51890==    by 0x44A6FC: list_resize (listobject.c:70)
==51890==    by 0x44A872: app1 (listobject.c:340)
==51890==    by 0x44FD65: PyList_Append (listobject.c:352)
==51890==    by 0x514315: r_ref (marshal.c:945)
==51890==    by 0x516034: r_object (marshal.c:1139)
==51890==    by 0x516C70: r_object (marshal.c:1389)
==51890==  Address 0x4c41020 is 32 bytes before a block of size 1,600 in arena "client"
==51890==
==51890== Conditional jump or move depends on uninitialised value(s)
==51890==    at 0x472E46: address_in_range (obmalloc.c:1403)
==51890==    by 0x47594A: pymalloc_realloc (obmalloc.c:1929)
==51890==    by 0x475A02: _PyObject_Realloc (obmalloc.c:1982)
==51890==    by 0x473DCA: _PyMem_DebugRawRealloc (obmalloc.c:2240)
==51890==    by 0x473FF8: _PyMem_DebugRealloc (obmalloc.c:2326)
==51890==    by 0x4749FB: PyMem_Realloc (obmalloc.c:623)
==51890==    by 0x44A6FC: list_resize (listobject.c:70)
==51890==    by 0x44A872: app1 (listobject.c:340)
==51890==    by 0x44FD65: PyList_Append (listobject.c:352)
==51890==    by 0x5E3321: _posix_listdir (posixmodule.c:3823)
==51890==    by 0x5E33A8: os_listdir_impl (posixmodule.c:3879)
==51890==    by 0x5E4D77: os_listdir (posixmodule.c.h:1197)
==51890==
==51890== Use of uninitialised value of size 8
==51890==    at 0x472E59: address_in_range (obmalloc.c:1403)
==51890==    by 0x47594A: pymalloc_realloc (obmalloc.c:1929)
==51890==    by 0x475A02: _PyObject_Realloc (obmalloc.c:1982)
==51890==    by 0x473DCA: _PyMem_DebugRawRealloc (obmalloc.c:2240)
==51890==    by 0x473FF8: _PyMem_DebugRealloc (obmalloc.c:2326)
==51890==    by 0x4749FB: PyMem_Realloc (obmalloc.c:623)
==51890==    by 0x44A6FC: list_resize (listobject.c:70)
==51890==    by 0x44A872: app1 (listobject.c:340)
==51890==    by 0x44FD65: PyList_Append (listobject.c:352)
==51890==    by 0x5E3321: _posix_listdir (posixmodule.c:3823)
==51890==    by 0x5E33A8: os_listdir_impl (posixmodule.c:3879)
==51890==    by 0x5E4D77: os_listdir (posixmodule.c.h:1197)
==51890==
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 16908288 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
```

(continues on next page)

(continued from previous page)

```
↪read 67895296 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 1572864 bytes but only got 0. Skipping tag 42
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 116647 bytes but only got 4867. Skipping tag 42738
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 3468830728 bytes but only got 4851. Skipping tag 279
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 2198732800 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 67239937 bytes but only got 4125. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33947764 bytes but only got 0. Skipping tag 139
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 17170432 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 80478208 bytes but only got 0. Skipping tag 1
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 787460 bytes but only got 4882. Skipping tag 20
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 1075 bytes but only got 0. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 120586240 bytes but only got 0. Skipping tag 194
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 65536 bytes but only got 0. Skipping tag 3
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
```

(continues on next page)

(continued from previous page)

```
↪read 198656 bytes but only got 0. Skipping tag 279
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 206848 bytes but only got 0. Skipping tag 64512
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 130968 bytes but only got 4882. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 77848 bytes but only got 4689. Skipping tag 64270
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 262156 bytes but only got 0. Skipping tag 257
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33624064 bytes but only got 0. Skipping tag 49152
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 67178752 bytes but only got 4627. Skipping tag 50688
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33632768 bytes but only got 0. Skipping tag 56320
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 134386688 bytes but only got 4115. Skipping tag 2048
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33912832 bytes but only got 0. Skipping tag 7168
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 151966208 bytes but only got 4627. Skipping tag 10240
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 119032832 bytes but only got 3859. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 46535680 bytes but only got 0. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
```

(continues on next page)

(continued from previous page)

```
↪read 35651584 bytes but only got 0. Skipping tag 42
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 524288 bytes but only got 0. Skipping tag 0
  warnings.warn(
_TIFFVSetField: tempfile.tif: Null count for "Tag 769" (type 1, writecount -3, passcount␣
↪1).
_TIFFVSetField: tempfile.tif: Null count for "Tag 42754" (type 1, writecount -3,␣
↪passcount 1).
_TIFFVSetField: tempfile.tif: Null count for "Tag 769" (type 1, writecount -3, passcount␣
↪1).
_TIFFVSetField: tempfile.tif: Null count for "Tag 42754" (type 1, writecount -3,␣
↪passcount 1).
ZIPDecode: Decoding error at scanline 0, incorrect header check.
==51890== Invalid write of size 4
==51890==    at 0x61C39E6: putcontig8bitYCbCr22tile (tif_getimage.c:2146)
==51890==    by 0x61C5865: gtStripContig (tif_getimage.c:977)
==51890==    by 0x6094317: ReadStrip (TiffDecode.c:269)
==51890==    by 0x6094749: ImagingLibTiffDecode (TiffDecode.c:479)
==51890==    by 0x60615D1: _decode (decode.c:136)
==51890==    by 0x64BF47: method_vectorcall_VARARGS (descrobject.c:300)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==    by 0x43627B: function_code_fastcall (call.c:283)
==51890==    by 0x436D21: _PyFunction_Vectorcall (call.c:410)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==  Address 0x6f456d4 is 0 bytes after a block of size 68 alloc'd
==51890==    at 0x483DFAF: realloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_
↪memcheck-amd64-linux.so)
==51890==    by 0x60946D0: ImagingLibTiffDecode (TiffDecode.c:469)
==51890==    by 0x60615D1: _decode (decode.c:136)
==51890==    by 0x64BF47: method_vectorcall_VARARGS (descrobject.c:300)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==    by 0x43627B: function_code_fastcall (call.c:283)
==51890==    by 0x436D21: _PyFunction_Vectorcall (call.c:410)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==    by 0x4DFDFB: _PyEval_EvalCodeWithName (ceval.c:4298)
==51890==    by 0x436C40: _PyFunction_Vectorcall (call.c:435)
==51890==
==51890== Invalid write of size 4
```

(continues on next page)

(continued from previous page)

```
==51890==    at 0x61C39B5: putcontig8bitYCbCr22tile (tif_getimage.c:2145)
==51890==    by 0x61C5865: gtStripContig (tif_getimage.c:977)
==51890==    by 0x6094317: ReadStrip (TiffDecode.c:269)
==51890==    by 0x6094749: ImagingLibTiffDecode (TiffDecode.c:479)
==51890==    by 0x60615D1: _decode (decode.c:136)
==51890==    by 0x64BF47: method_vectorcall_VARARGS (descrobject.c:300)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==    by 0x43627B: function_code_fastcall (call.c:283)
==51890==    by 0x436D21: _PyFunction_Vectorcall (call.c:410)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==  Address 0x6f456d8 is 4 bytes after a block of size 68 alloc'd
==51890==    at 0x483DFAF: realloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_
↪memcheck-amd64-linux.so)
==51890==    by 0x60946D0: ImagingLibTiffDecode (TiffDecode.c:469)
==51890==    by 0x60615D1: _decode (decode.c:136)
==51890==    by 0x64BF47: method_vectorcall_VARARGS (descrobject.c:300)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==    by 0x43627B: function_code_fastcall (call.c:283)
==51890==    by 0x436D21: _PyFunction_Vectorcall (call.c:410)
==51890==    by 0x4EB73C: _PyObject_Vectorcall (abstract.h:127)
==51890==    by 0x4EB73C: call_function (ceval.c:4963)
==51890==    by 0x4EB73C: _PyEval_EvalFrameDefault (ceval.c:3486)
==51890==    by 0x4DF2EE: PyEval_EvalFrameEx (ceval.c:741)
==51890==    by 0x4DFDFB: _PyEval_EvalCodeWithName (ceval.c:4298)
==51890==    by 0x436C40: _PyFunction_Vectorcall (call.c:435)
==51890==
TIFFFillStrip: Invalid strip byte count 0, strip 1.
Traceback (most recent call last):
  File "test_tiff.py", line 8, in <module>
    im.load()
  File "/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_
↪64.egg/PIL/TiffImagePlugin.py", line 1087, in load
    return self._load_libtiff()
  File "/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_
↪64.egg/PIL/TiffImagePlugin.py", line 1191, in _load_libtiff
    raise OSError(err)
OSError: -2
sys:1: ResourceWarning: unclosed file <_io.BufferedReader name='crash-2020-10-test.tiff'>
==51890==
==51890== HEAP SUMMARY:
==51890==     in use at exit: 748,734 bytes in 444 blocks
==51890==   total heap usage: 6,320 allocs, 5,876 frees, 69,142,969 bytes allocated
==51890==
```

(continues on next page)

(continued from previous page)

```
==51890== LEAK SUMMARY:
==51890==    definitely lost: 0 bytes in 0 blocks
==51890==    indirectly lost: 0 bytes in 0 blocks
==51890==      possibly lost: 721,538 bytes in 372 blocks
==51890==    still reachable: 27,196 bytes in 72 blocks
==51890==         suppressed: 0 bytes in 0 blocks
==51890== Rerun with --leak-check=full to see details of leaked memory
==51890==
==51890== Use --track-origins=yes to see where uninitialised values come from
==51890== For lists of detected and suppressed errors, rerun with: -s
==51890== ERROR SUMMARY: 2556 errors from 6 contexts (suppressed: 0 from 0)
(vpy38-dbg) ubuntu@primary:~/Home/tests$
```

- Now that we've confirmed that there's something odd/bad going on, it's time to gdb.
- Start with `gdb python`
- Set a break point starting with the valgrind stack trace. `b TiffDecode.c:269`
- Run the script with `r test_tiff.py`
- When the break point is hit, explore the state with `info locals`, `bt`, `py-bt`, or `p [variable]`. For pointers, `p *[variable]` is useful.

```
(vpy38-dbg) ubuntu@primary:~/Home/tests$ gdb python
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <https://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from python...
(gdb) b TiffDecode.c:269
No source file named TiffDecode.c.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (TiffDecode.c:269) pending.
(gdb) r test_tiff.py
Starting program: /home/ubuntu/vpy38-dbg/bin/python test_tiff.py
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 16908288 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
```

(continues on next page)

(continued from previous page)

```
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 67895296 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 1572864 bytes but only got 0. Skipping tag 42
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 116647 bytes but only got 4867. Skipping tag 42738
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 3468830728 bytes but only got 4851. Skipping tag 279
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 2198732800 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 67239937 bytes but only got 4125. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33947764 bytes but only got 0. Skipping tag 139
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 17170432 bytes but only got 0. Skipping tag 0
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 80478208 bytes but only got 0. Skipping tag 1
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 787460 bytes but only got 4882. Skipping tag 20
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 1075 bytes but only got 0. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 120586240 bytes but only got 0. Skipping tag 194
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 65536 bytes but only got 0. Skipping tag 3
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
```

(continues on next page)

(continued from previous page)

```
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 198656 bytes but only got 0. Skipping tag 279
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 206848 bytes but only got 0. Skipping tag 64512
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 130968 bytes but only got 4882. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 77848 bytes but only got 4689. Skipping tag 64270
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 262156 bytes but only got 0. Skipping tag 257
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33624064 bytes but only got 0. Skipping tag 49152
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 67178752 bytes but only got 4627. Skipping tag 50688
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33632768 bytes but only got 0. Skipping tag 56320
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 134386688 bytes but only got 4115. Skipping tag 2048
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 33912832 bytes but only got 0. Skipping tag 7168
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 151966208 bytes but only got 4627. Skipping tag 10240
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 119032832 bytes but only got 3859. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 46535680 bytes but only got 0. Skipping tag 256
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
```

(continues on next page)

(continued from previous page)

```
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 35651584 bytes but only got 0. Skipping tag 42
  warnings.warn(
/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_64.egg/
↪PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to␣
↪read 524288 bytes but only got 0. Skipping tag 0
  warnings.warn(
_TIFFVSetField: tempfile.tif: Null count for "Tag 769" (type 1, writecount -3, passcount␣
↪1).
_TIFFVSetField: tempfile.tif: Null count for "Tag 42754" (type 1, writecount -3,␣
↪passcount 1).
_TIFFVSetField: tempfile.tif: Null count for "Tag 769" (type 1, writecount -3, passcount␣
↪1).
_TIFFVSetField: tempfile.tif: Null count for "Tag 42754" (type 1, writecount -3,␣
↪passcount 1).

Breakpoint 1, ReadStrip (tiff=tiff@entry=0xae9b90, row=0, buffer=0xac2eb0) at src/
↪libImaging/TiffDecode.c:269
269                 ok = TIFFRGBAImageGet(&img, buffer, img.width, rows_to_read);
(gdb) p img
$1 = {tif = 0xae9b90, stoponerr = 0, isContig = 1, alpha = 0, width = 20, height = 1536,␣
↪bitspersample = 8, samplesperpixel = 3,
  orientation = 1, req_orientation = 1, photometric = 6, redcmap = 0x0, greencmap = 0x0,␣
↪bluecmap = 0x0, get =
    0x7ffff71d0710 <gtStripContig>, put = {any = 0x7ffff71ce550
↪<putcontig8bitYCbCr22tile>,
    contig = 0x7ffff71ce550 <putcontig8bitYCbCr22tile>, separate = 0x7ffff71ce550
↪<putcontig8bitYCbCr22tile>}, Map = 0x0,
  BWmap = 0x0, PALmap = 0x0, ycbcr = 0xaf24b0, cielab = 0x0, UaToAa = 0x0, Bitdepth16To8␣
↪= 0x0, row_offset = 0, col_offset = 0}
(gdb) up
#1  0x00007ffff736174a in ImagingLibTiffDecode (im=0xac1f90, state=0x7ffff76767e0,␣
↪buffer=<optimized out>, bytes=<optimized out>)
    at src/libImaging/TiffDecode.c:479
479                 if (ReadStrip(tiff, state->y, (UINT32 *)state->buffer) == -1) {
(gdb) p *state
$2 = {count = 0, state = 0, errcode = 0, x = 0, y = 0, ystep = 0, xsize = 17, ysize =␣
↪108, xoff = 0, yoff = 0,
  shuffle = 0x7ffff735f411 <copy4>, bits = 32, bytes = 68, buffer = 0xac2eb0 "P\354\336\
↪367\377\177", context = 0xa75440, fd = 0x0}
(gdb) py-bt
Traceback (most recent call first):
  File "/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_
↪64.egg/PIL/TiffImagePlugin.py", line 1428, in _load_libtiff

  File "/home/ubuntu/vpy38-dbg/lib/python3.8/site-packages/Pillow-8.0.1-py3.8-linux-x86_
↪64.egg/PIL/TiffImagePlugin.py", line 1087, in load
    return self._load_libtiff()
  File "test_tiff.py", line 8, in <module>
    im.load()
```

- Poke around till you understand what's going on. In this case, state->xsize and img.width are different, which led to an out of bounds write, as the receiving buffer was sized for the smaller of the two.

Caveats

- If your program is running/hung in a docker container and your host has the appropriate tools, you can run gdb as the superuser in the host and you may be able to get a trace of where the process is hung. You probably won't have the capability to do that from within the docker container, as the trace capacity isn't allowed by default.
- Variations of this are possible on macOS/Windows, but the details are going to be different.
- IIRC, Fedora has the gdb bits working by default. Ubuntu has always been a bit of a battle to make it work.

Arrow support

Arrow is an in-memory data exchange format that is the spiritual successor to the NumPy array interface. It provides for zero-copy access to columnar data, which in our case is `Image` data.

The goal with Arrow is to provide native zero-copy interoperability with any Arrow provider or consumer in the Python ecosystem.

> **Warning**
>
> Zero-copy does not mean zero allocation – the internal memory layout of Pillow images contains an allocation for row pointers, so there is a non-zero, but significantly smaller than a full-copy memory cost to reading an Arrow image.

Data formats

Pillow currently supports exporting Arrow images in all modes.

For single-band images, the exported array is width*height elements, with each pixel corresponding to the appropriate Arrow type.

For multiband images, the exported array is width*height fixed-length four-element arrays of uint8. This is memory compatible with the raw image storage of four bytes per pixel.

Mode 1 images are exported as one uint8 byte/pixel, as this is consistent with the internal storage.

Pillow will accept, but not produce, one other format. For any multichannel image with 32-bit storage per pixel, Pillow will accept an array of width*height int32 elements, which will then be interpreted using the mode-specific interpretation of the bytes.

The image mode must match the Arrow band format when reading single channel images.

Memory allocator

Pillow's default memory allocator, the *Block allocator*, allocates up to a 16 MB block for images by default. Larger images overflow into additional blocks. Arrow requires a single continuous memory allocation, so images allocated in multiple blocks cannot be exported in the Arrow format.

To enable the single block allocator:

```
from PIL import Image
Image.core.set_use_block_allocator(1)
```

Note that this is a global setting, not a per-image setting.

Unsupported features

- Table/dataframe protocol. We support a single array.
- Null markers, producing or consuming. Null values are inferred from the mode, e.g. RGB images are stored in the first three bytes of each 32-bit pixel, and the last byte is an implied null.
- Schema negotiation. There is an optional schema for the requested datatype in the Arrow source interface. We ignore that parameter.
- Array metadata.

Internal details

Python Arrow C interface: https://arrow.apache.org/docs/format/CDataInterface/PyCapsuleInterface.html

The memory that is exported from the Arrow interface is shared – not copied, so the lifetime of the memory allocation is no longer strictly tied to the life of the Python object.

The core imaging struct now has a refcount associated with it, and the lifetime of the core image struct is now divorced from the Python image object. Creating an arrow reference to the image increments the refcount, and the imaging struct is only released when the refcount reaches zero.

1.4 Porting

Porting existing PIL-based code to Pillow

Pillow is a functional drop-in replacement for the Python Imaging Library.

PIL is Python 2 only. Pillow dropped support for Python 2 in Pillow 7.0. So if you would like to run the latest version of Pillow, you will first and foremost need to port your code from Python 2 to 3.

To run your existing PIL-compatible code with Pillow, it needs to be modified to import the `Image` module from the `PIL` namespace *instead* of the global namespace. Change this:

```
import Image
```

to this:

```
from PIL import Image
```

The *PIL._imaging* module has been moved to *PIL.Image.core*. You can now import it like this:

```
from PIL.Image import core as _imaging
```

The image plugin loading mechanism has changed. Pillow no longer automatically imports any file in the Python path with a name ending in `ImagePlugin.py`. You will need to import your image plugin manually.

Pillow will raise an exception if the core extension can't be loaded for any reason, including a version mismatch between the Python and extension code. Previously PIL allowed Python only code to run if the core extension was not available.

1.5 About

1.5.1 Goals

The fork author's goal is to foster and support active development of PIL through:

- Continuous integration testing via GitHub Actions

- Publicized development activity on GitHub
- Regular releases to the Python Package Index

1.5.2 License

Like PIL, Pillow is licensed under the open source MIT-CMU License

1.5.3 Why a fork?

PIL is not setuptools compatible. Please see this Image-SIG post for a more detailed explanation. Also, PIL's bi-yearly (or greater) release schedule was too infrequent to accommodate the large number and frequency of issues reported.

1.5.4 What about PIL?

> **Note**
>
> Prior to Pillow 2.0.0, very few image code changes were made. Pillow 2.0.0 added Python 3 support and includes many bug fixes from many contributors.

The last PIL release was in 2009 (1.1.7) and no future releases are expected. In January 2020, the PyPI moderators exhausted the PEP 541 process for contacting the PIL project owner and the PIL project on PyPI was transferred to the Pillow team. The Pillow team has no plans to update the PIL project on PyPI.

1.6 Release notes

Pillow is released quarterly on January 2nd, April 1st, July 1st and October 15th. Patch releases are created if the latest release contains severe bugs, or if security fixes are put together before a scheduled release. See *Versioning* for more information.

Please use the latest version of Pillow. Functionality and security fixes should not be expected to be backported to earlier versions.

> **Note**
>
> Contributors please include release notes as needed or appropriate with your bug fixes, feature additions and tests.

1.6.1 Versioning

Pillow follows Semantic Versioning:

> Given a version number MAJOR.MINOR.PATCH, increment the:
>
> 1. MAJOR version when you make incompatible API changes,
> 2. MINOR version when you add functionality in a backwards compatible manner, and
> 3. PATCH version when you make backwards compatible bug fixes.

Quarterly releases ("Main Release") bump at least the MINOR version, as new functionality has likely been added in the prior three months.

A quarterly release bumps the MAJOR version when incompatible API changes are made, such as removing deprecated APIs or dropping an EOL Python version. In practice, these occur every October, guided by Python's EOL schedule, and any APIs that have been deprecated for at least a year are removed at the same time.

PATCH versions ("Point Release" or "Embargoed Release") are for security, installation or critical bug fixes. These are less common as it is preferred to stick to quarterly releases.

Between quarterly releases, `.dev0` is appended to the `main` branch, indicating that this is not a formally released copy.

1.6.2 12.3.0 (2026-07-01)

Security

Prevent decompression bomb when parsing PDF

When parsing a PDF, `PdfStream.decode()` attempts to decompress data without any limit. A default maximum of `ImageFile.SAFEBLOCK` has been added. `PdfParser` was added in Pillow 5.1.0. Reported by redyank.

CVE 2026-55798: WindowsViewer.get_command injection

If an attacker has control over the path passed to `ImageShow.WindowsViewer.get_command()`, and the result is executed by the user, the attacker may be able to execute arbitrary shell commands. Reported by Bin Luo, University of Electronic Science and Technology of China (luob87709@gmail.com).

EPS image infinite loop

If a negative byte count is specified for the BeginBinary byte count, an infinite loop is possible as Pillow seeks repeatedly backwards. This value will now be validated. BeginBinary parsing was only added in Pillow 12.0.0. Reported by jiagongzheng-stack.

JPEG2000 image memory usage

The total component width was incorrectly accumulated across tiles within a JPEG2000 image, potentially leading to excessive memory use. This was introduced as part of an earlier security fix in Pillow 8.2.0, but has now been fixed. Reported by Fr3v1.

Mcidas out-of-bounds (OOB) read

McIdas images can specify the stride, and if incorrectly set, may cause an OOB read. This has been fixed. Reported by Devansh Shah, RUDRA Cybersecurity Pvt. Ltd.

Out-of-bounds (OOB) read when saving 1 mode TGA images

Saving 1 mode images as TGA with run-length encoding can trigger an OOB read. This was added in Pillow 5.2.0, but the functionality is not possible under the TGA specification, so it has been removed. Reported by Seratov.

Out-of-bounds (OOB) write from large RankFilter sizes

`RankFilter`, and its subclasses, can potentially write out-of-bounds if given a large size. This has been fixed. Reported by Seratov.

Out-of-bounds (OOB) write from `Image.paste()`

Large paste box dimensions can cause an OOB write in `Image.crop()`, `Image.paste()` and `Image.alpha_composite()`. This has been fixed. Reported by Seratov.

Out-of-bounds (OOB) write in `ImageCmsTransform`

Apply a transform mode to a different image mode can trigger an OOB write in `ImageCmsTransform`. Errors are now raised if the mode does not match. Reported by Seratov.

CVE 2026-54059, CVE 2026-54060, CVE 2026-55379: Prevent FontFile decompression bomb

Decompression bomb checks have now been added to *`FontFile`*, and its subclasses, checking the width and height of characters. Reported by Rahul Singh and Dinesh.

CVE 2026-55380: Prevent GD decompression bomb

A decompression bomb check has been added to *`GdImageFile`*. This is format must be explicitly called though, rather than being accessible from `Image.open()`. Reported by Rahul Singh.

API additions

Added `scale_down` argument to `ImageGrab.grab()`

`grab()` now accepts an optional keyword argument of `scale_down`. This affects macOS screenshots with a `bbox` on a Retina screen. By default, images will be captured at 2x. If `scale_down` is `True`, they will be at 1x.

Previously, macOS screenshots with a `bbox` were captured at 1x by default.

Added `max_length` argument to `PdfStream.decode()`

When calling `PdfStream.decode()`, the maximum length of data to decode can now be specified. This will default to `ImageFile.SAFEBLOCK`.

Other changes

Python 3.15 beta

To help other projects prepare for Python 3.15, wheels are now built for the 3.15 beta as a preview. This is not official support for Python 3.15, but rather an opportunity for you to test how Pillow works with the beta and report any problems.

SBOM in wheels

Following **PEP 770**, the Software Bill-of-Materials (SBOM) has now been embedded into Pillow's wheels.

Removed Python 3.13 free-threaded wheels

Python 3.13 added an experimental free-threaded mode, and Pillow 11.0.0 added corresponding wheels. Now that Python 3.14 includes official support for it, Pillow has removed wheels for Python 3.13 free-threaded mode.

Performance improvements

A number of changes have been made to optimize the use of C when manipulating images. This improves performance by up to 5.6 times of `Image`'s `alpha_composite`, `fill`, `filter`, `getchannel`, `linear_gradient`, `matrix`, `merge`, `negative`, `putalpha`, `quantize`, `radial_gradient`, `resample`, `split`, and `ImageChops` operations.

1.6.3 12.2.0 (2026-04-01)

Security

CVE 2026-40192: Prevent FITS decompression bomb

When decompressing GZIP data from a FITS image, Pillow did not limit the amount of data being read, meaning that it was vulnerable to GZIP decompression bombs. This was introduced in Pillow 10.3.0.

The data being read is now limited to only the necessary amount.

CVE 2026-42311: Fix OOB write with invalid tile extents

Pillow 12.1.1 addressed CVE 2026-25990 by improving checks for tile extents to prevent an OOB write from specially crafted PSD images in Pillow >= 10.3.0. However, these checks did not consider integer overflow. This has been corrected.

CVE 2026-42310: Prevent PDF parsing trailer infinite loop

When parsing a PDF, if a trailer refers to itself, or a more complex cyclic loop exists, then an infinite loop occurs. Pillow now keeps a record of which trailers it has already processed. PdfParser was added in Pillow 5.1.0.

CVE 2026-42308: Integer overflow when processing fonts

If a font advances for each glyph by an exceedingly large amount, when Pillow keeps track of the current position, it may lead to an integer overflow. This has been fixed.

CVE 2026-42309: Heap buffer overflow with nested list coordinates

Passing nested lists as coordinates to APIs that accept coordinates such as `ImagePath.Path`, *polygon()* and *line()* could cause a heap buffer overflow, as nested lists were recursively unpacked beyond the allocated buffer. Coordinate lists are now validated to contain exactly two numeric coordinates. This was introduced in Pillow 11.2.1.

API changes

Error when encoding an empty image

Attempting to encode an image with zero width or height would previously raise a `SystemError`. That has now been changed to a `ValueError`.

This does not add any new errors. SGI, ICNS and ICO formats are still able to save (0, 0) images.

API additions

FontFile.to_imagefont()

FontFile instances can now be directly converted to *ImageFont* instances:

```
>>> from PIL import PcfFontFile
>>> with open("Tests/fonts/10x20-ISO8859-1.pcf", "rb") as fp:
...     pcffont = PcfFontFile.PcfFontFile(fp)
...     pcffont.to_imagefont()
...
<PIL.ImageFont.ImageFont object at 0x10457bb80>
```

ImageText.Text.wrap

ImageText.Text.wrap() has been added, to wrap text to fit within a given width:

```
from PIL import ImageText
text = ImageText.Text("Hello World!")
text.wrap(50)
print(text.text)  # "Hello\nWorld!"
```

or within a certain width and height, returning a new *ImageText.Text* instance if the text does not fit:

```
text = ImageText.Text("Text does not fit within height")
print(text.wrap(50, 25).text == " within height")
print(text.text)  # "Text does\nnot fit"
```

or scaling, optionally with a font size limit:

```
text.wrap(50, 15, "shrink")
text.wrap(50, 15, ("shrink", 7))
text.wrap(58, 10, "grow")
text.wrap(50, 50, ("grow", 12))
```

EXIF tag FrameRate

The EXIF tag `FrameRate` has been added.

Other changes

Support reading JPEG2000 images with CMYK palettes

JPEG2000 images with CMYK palettes can now be read. This is the first integration of CMYK palettes into Pillow.

Lazy plugin loading

When opening or saving an image, Pillow now lazily loads only the required plugin based on the file extension, instead of importing all plugins upfront. This makes `open` 2.3-15.6x faster and `save` 2.2-9x faster for common formats.

Thread safety for free-threaded Python

Critical sections are now used to protect FreeType font objects, improving thread safety when using fonts in the free-threaded build of Python.

1.6.4 12.1.1 (2026-02-11)

Security

CVE 2026-25990: Fix OOB write with invalid tile extents

Check that tile extents do not use negative x or y offsets when decoding or encoding, and raise an error if they do, rather than allowing an OOB write.

An out-of-bounds write may be triggered when opening a specially crafted PSD image. This only affects Pillow >= 10.3.0. Reported by Yarden Porat.

Other changes

Patch libavif for svt-av1 4.0 compatibility

A patch has been added to `depends/install_libavif.sh`, to allow libavif 1.3.0 to be compatible with the recently released svt-av1 4.0.0.

1.6.5 12.1.0 (2026-01-02)

Deprecations

Image getdata()

getdata() has been deprecated. *get_flattened_data()* can be used instead. This new method is identical, except that it returns a tuple of pixel values, instead of an internal Pillow data type.

API changes

ImageMorph build_default_lut()

To match the behaviour of *build_lut()*, *build_default_lut()* now returns the new LUT.

API additions

Image get_flattened_data()

get_flattened_data() is identical to the deprecated *getdata()*, except that the new method returns a tuple of pixel values, instead of an internal Pillow data type.

Specify window in ImageGrab on macOS

When using *grab()*, a specific window can now be selected on macOS in addition to Windows. On macOS, this is a CGWindowID:

```
from PIL import ImageGrab
ImageGrab.grab(window=cgwindowid)
```

Other changes

Added MorphOp support for 1 mode images

MorphOp now supports both 1 mode and L mode images.

1.6.6 12.0.0 (2025-10-15)

Backwards incompatible changes

Python 3.9

Pillow has dropped support for Python 3.9, which reached end-of-life in October 2025.

ImageFile.raise_oserror

`ImageFile.raise_oserror()` has been removed. The function was undocumented and was only useful for translating error codes returned by a codec's `decode()` method, which ImageFile already did automatically.

IptcImageFile helper functions

The functions `IptcImageFile.dump` and `IptcImageFile.i`, and the constant `IptcImageFile.PAD` have been removed. These were undocumented helper functions intended for internal use, so there is no replacement. They can each be replaced by a single line of code using builtin functions in Python.

ImageCms constants and versions() function

A number of constants and a function in *ImageCms* have been removed. This includes a table of flags based on LittleCMS version 1 which has been replaced with a new class *ImageCms.Flags* based on LittleCMS 2 flags.

Deprecated	Use instead
`ImageCms.DESCRIPTION`	No replacement
`ImageCms.VERSION`	`PIL.__version__`
`ImageCms.FLAGS["MATRIXINPUT"]`	*ImageCms.Flags.CLUT_POST_LINEARIZATION*
`ImageCms.FLAGS["MATRIXOUTPUT"]`	*ImageCms.Flags.FORCE_CLUT*
`ImageCms.FLAGS["MATRIXONLY"]`	No replacement
`ImageCms.FLAGS["NOWHITEONWHITEFIXU`	*ImageCms.Flags.NOWHITEONWHITEFIXUP*
`ImageCms.FLAGS["NOPRELINEARIZATION`	*ImageCms.Flags.CLUT_PRE_LINEARIZATION*
`ImageCms.FLAGS["GUESSDEVICECLASS"]`	*ImageCms.Flags.GUESSDEVICECLASS*
`ImageCms.FLAGS["NOTCACHE"]`	*ImageCms.Flags.NOCACHE*
`ImageCms.FLAGS["NOTPRECALC"]`	*ImageCms.Flags.NOOPTIMIZE*
`ImageCms.FLAGS["NULLTRANSFORM"]`	*ImageCms.Flags.NULLTRANSFORM*
`ImageCms.FLAGS["HIGHRESPRECALC"]`	*ImageCms.Flags.HIGHRESPRECALC*
`ImageCms.FLAGS["LOWRESPRECALC"]`	*ImageCms.Flags.LOWRESPRECALC*
`ImageCms.FLAGS["GAMUTCHECK"]`	*ImageCms.Flags.GAMUTCHECK*
`ImageCms.FLAGS["WHITEBLACKCOMPENSA`	*ImageCms.Flags.BLACKPOINTCOMPENSATION*
`ImageCms.FLAGS["BLACKPOINTCOMPENSA`	*ImageCms.Flags.BLACKPOINTCOMPENSATION*
`ImageCms.FLAGS["SOFTPROOFING"]`	*ImageCms.Flags.SOFTPROOFING*
`ImageCms.FLAGS["PRESERVEBLACK"]`	*ImageCms.Flags.NONEGATIVES*
`ImageCms.FLAGS["NODEFAULTRESOURCED`	*ImageCms.Flags.NODEFAULTRESOURCEDEF*
`ImageCms.FLAGS["GRIDPOINTS"]`	*ImageCms.Flags.GRIDPOINTS()*
`ImageCms.versions()`	*PIL.features.version_module()* with `feature="littlecms2"`, `sys.version` or `sys.version_info`, and `PIL.__version__`

ImageMath eval()

`ImageMath.eval()` has been removed. Use *lambda_eval()* or *unsafe_eval()* instead.

BGR;15, BGR;16 and BGR;24

The experimental BGR;15, BGR;16 and BGR;24 modes have been removed.

Non-image modes in ImageCms

The use in *ImageCms* of input modes and output modes that are not Pillow image modes has been removed. Defaulting to "L" or "1" if the mode cannot be mapped has also been removed.

Support for LibTIFF earlier than 4

Support for LibTIFF earlier than version 4 has been removed. Upgrade to a newer version of LibTIFF instead.

ImageDraw.getdraw hints parameter

The `hints` parameter in *getdraw()* has been removed.

FreeType 2.9.0

Support for FreeType 2.9.0 has been removed. FreeType 2.9.1 is the minimum version supported.

We recommend upgrading to at least FreeType 2.10.4, which fixed a severe vulnerability introduced in FreeType 2.6 (**CVE 2020-15999**).

Deprecations

Image._show

`Image._show` has been deprecated, and will be removed in Pillow 13 (2026-10-15). Use *show()* instead.

ImageCms.ImageCmsProfile.product_name and .product_info

`ImageCms.ImageCmsProfile.product_name` and the corresponding `.product_info` attributes have been deprecated, and will be removed in Pillow 13 (2026-10-15). They have been set to `None` since Pillow 2.3.0.

API changes

Image.alpha_composite: LA images

alpha_composite() can now use LA images as well as RGBA.

API additions

Added ImageText.Text

PIL.ImageText.Text has been added, as a simpler way to use fonts with text strings or bytes.

Without `ImageText.Text`:

```
from PIL import Image, ImageDraw
im = Image.new(mode, size)
d = ImageDraw.Draw(im)
```

(continues on next page)

(continued from previous page)

```
d.textlength(text, font, direction, features, language, embedded_color)
d.multiline_textbbox(xy, text, font, anchor, spacing, align, direction, features,␣
↪language, stroke_width, embedded_color)
d.text(xy, text, fill, font, anchor, spacing, align, direction, features, language,␣
↪stroke_width, stroke_fill, embedded_color)
```

With `ImageText.Text`:

```
from PIL import ImageText
text = ImageText.Text(text, font, mode, spacing, direction, features, language)
text.embed_color()
text.stroke(stroke_width, stroke_fill)

text.get_length()
text.get_bbox(xy, anchor, align)

im = Image.new(mode, size)
d = ImageDraw.Draw(im)
d.text(xy, text, fill, anchor=anchor, align=align)
```

Other changes

Python 3.14

Pillow 11.3.0 had wheels built against Python 3.14 beta, available as a preview to help others prepare for 3.14, and to ensure Pillow could be used immediately at the release of 3.14.0 final (2025-10-07, **PEP 745**).

Pillow 12.0.0 now officially supports Python 3.14.

Image.fromarray mode parameter

In Pillow 11.3.0, the `mode` parameter in *`fromarray()`* was deprecated. Part of this functionality has been restored in Pillow 12.0.0. Since pixel values do not contain information about palettes or color spaces, the parameter can be used to place grayscale L mode data within a P mode image, or read RGB data as YCbCr for example.

ImageMorph operations must have length 1

Valid ImageMorph operations are 4, N, 1 and M. By limiting the length to 1 character within Pillow, long execution times can be avoided if a user provided long pattern strings. Reported by Jang Choi.

1.6.7 11.3.0 (2025-07-01)

Security

CVE 2025-48379: Write buffer overflow on BCn encoding

There is a heap buffer overflow when writing a sufficiently large (>64k encoded with default settings) image in the DDS format due to writing into a buffer without checking for available space.

This only affects users who save untrusted data as a compressed DDS image.

- Unclear how large the potential write could be. It is likely limited by process segfault, so it's not necessarily deterministic. It may be practically unbounded.

- Unclear if there's a restriction on the bytes that could be emitted. It's likely that the only restriction is that the bytes would be emitted in chunks of 8 or 16.

This was introduced in Pillow 11.2.0 when the feature was added.

Deprecations

Image.fromarray mode parameter

The `mode` parameter in *`fromarray()`* has been deprecated. The mode can be automatically determined from the object's shape and type instead.

> **Note**
>
> Since pixel values do not contain information about palettes or color spaces, part of this functionality was restored in Pillow 12.0.0. The parameter can be used to place grayscale L mode data within a P mode image, or read RGB data as YCbCr for example.

Saving I mode images as PNG

In order to fit the 32 bits of I mode images into PNG, when PNG images can only contain at most 16 bits for a channel, Pillow has been clipping the values. Rather than quietly changing the data, this is now deprecated. Instead, the image can be converted to another mode before saving:

```
from PIL import Image
im = Image.new("I", (1, 1))
im.convert("I;16").save("out.png")
```

Other changes

Added QOI saving

Support has been added for saving QOI images. `colorspace` can be used to specify the colorspace as sRGB with linear alpha, e.g. `im.save("out.qoi", colorspace="sRGB")`. By default, all channels will be linear.

Support using more screenshot utilities with ImageGrab on Linux

`grab()` is now able to use GNOME Screenshot, grim or Spectacle on Linux in order to take a snapshot of the screen.

Do not build against libavif < 1

Pillow only supports libavif 1.0.0 or later. In order to prevent errors when building from source, if a user happens to have an earlier libavif on their system, Pillow will now ignore it.

AVIF support in wheels

Support for reading and writing AVIF images is now included in Pillow's wheels, except for Windows ARM64 and iOS. libaom is available as an encoder and dav1d as a decoder. (Thank you Frankie Dintino and Andrew Murray!)

iOS

Pillow now provides wheels that can be used on iOS ARM64 devices, and on the iOS simulator on ARM64 and x86_64. Currently, only Python 3.13 wheels are available. (Thank you Russell Keith-Magee and Andrew Murray!)

Python 3.14 beta

To help other projects prepare for Python 3.14, wheels are now built for the 3.14 beta as a preview. This is not official support for Python 3.14, but rather an opportunity for you to test how Pillow works with the beta and report any problems.

1.6.8 11.2.1 (2025-04-12)

> **Warning**
>
> The release of Pillow *11.2.0* was halted prematurely, due to hitting PyPI's project size limit and concern over the size of Pillow wheels containing libavif. The PyPI limit has now been increased and Pillow *11.2.1* has been released instead, without libavif included in the wheels. To avoid confusion, the incomplete 11.2.0 release has been removed from PyPI.

Security

Undefined shift when loading compressed DDS images

When loading some compressed DDS formats, an integer was bitshifted by 24 places to generate the 32 bits of the lookup table. This was undefined behaviour, and has been present since Pillow 3.4.0.

Deprecations

Image.Image.get_child_images()

Deprecated since version 11.2.1.

`Image.Image.get_child_images()` has been deprecated, and will be removed in Pillow 13 (2026-10-15). It will be moved to `ImageFile.ImageFile.get_child_images()`. The method uses an image's file pointer, and so child images could only be retrieved from an *PIL.ImageFile.ImageFile* instance.

API changes

`append_images` no longer requires `save_all`

Previously, `save_all` was required in order to use `append_images`. Now, `save_all` will default to `True` if `append_images` is not empty and the format supports saving multiple frames:

```
im.save("out.gif", append_images=ims)
```

API additions

`"justify"` multiline text alignment

In addition to `"left"`, `"center"` and `"right"`, multiline text can also be aligned using `"justify"` in *ImageDraw*:

```
from PIL import Image, ImageDraw
im = Image.new("RGB", (50, 25))
draw = ImageDraw.Draw(im)
draw.multiline_text((0, 0), "Multiline\ntext 1", align="justify")
draw.multiline_textbbox((0, 0), "Multiline\ntext 2", align="justify")
```

Specify window in ImageGrab on Windows

When using *grab()*, a specific window can be selected using the HWND:

```
from PIL import ImageGrab
ImageGrab.grab(window=hwnd)
```

Check for MozJPEG

You can check if Pillow has been built against the MozJPEG version of the libjpeg library, and what version of MozJPEG is being used:

```
from PIL import features
features.check_feature("mozjpeg")  # True or False
features.version_feature("mozjpeg")  # "4.1.1" for example, or None
```

Saving compressed DDS images

Compressed DDS images can now be saved using a `pixel_format` argument. DXT1, DXT3, DXT5, BC2, BC3 and BC5 are supported:

```
im.save("out.dds", pixel_format="DXT1")
```

Other changes

Arrow support

Arrow is an in-memory data exchange format that is the spiritual successor to the NumPy array interface. It provides for zero-copy access to columnar data, which in our case is `Image` data.

To create an image with zero-copy shared memory from an object exporting the arrow_c_array interface protocol:

```
from PIL import Image
import pyarrow as pa
arr = pa.array([0]*(5*5*4), type=pa.uint8())
im = Image.fromarrow(arr, 'RGBA', (5, 5))
```

Pillow images can also be converted to Arrow objects:

```
from PIL import Image
import pyarrow as pa
im = Image.open('hopper.jpg')
arr = pa.array(im)
```

Reading and writing AVIF images

Pillow can now read and write AVIF images when built from source with libavif 1.0.0 or later.

1.6.9 11.1.0 (2025-01-02)

Deprecations

ExifTags.IFD.Makernote

`ExifTags.IFD.Makernote` has been deprecated. Instead, use `ExifTags.IFD.MakerNote`.

API changes

Writing XMP bytes to JPEG and MPO

Pillow 11.0.0 added writing XMP data to JPEG and MPO images:

```
im.info["xmp"] = b"test"
im.save("out.jpg")
```

However, this meant that XMP data was automatically kept from an opened image, which is inconsistent with the rest of Pillow's behaviour. This functionality has been removed. To write XMP data, the `xmp` argument can still be used for JPEG files:

```
im.save("out.jpg", xmp=b"test")
```

To save XMP data to the second frame of an MPO image, `encoderinfo` can now be used:

```
second_im.encoderinfo = {"xmp": b"test"}
im.save("out.mpo", save_all=True, append_images=[second_im])
```

API additions

Check for zlib-ng

You can check if Pillow has been built against the zlib-ng version of the zlib library, and what version of zlib-ng is being used:

```
from PIL import features
features.check_feature("zlib_ng")  # True or False
features.version_feature("zlib_ng")  # "2.2.2" for example, or None
```

Saving TIFF as BigTIFF

TIFF images can now be saved as BigTIFF using a `big_tiff` argument:

```
im.save("out.tiff", big_tiff=True)
```

Other changes

Reading JPEG 2000 comments

When opening a JPEG 2000 image, the comment may now be read into *info* for J2K images, not just JP2 images.

Saving JPEG 2000 CMYK images

With OpenJPEG 2.5.3 or later, Pillow can now save CMYK images as JPEG 2000 files.

Minimum C version

C99 is now the minimum version of C required to compile Pillow from source.

zlib-ng in wheels

Wheels are now built against zlib-ng for improved speed. In tests, saving a PNG was found to be more than twice as fast at higher compression levels.

1.6.10 11.0.0 (2024-10-15)

Backwards incompatible changes

Python 3.8

Pillow has dropped support for Python 3.8, which reached end-of-life in October 2024.

Python 3.12 on macOS <= 10.12

The latest version of Python 3.12 only supports macOS versions 10.13 and later, and so Pillow has also updated the deployment target for its prebuilt Python 3.12 wheels.

PSFile

The `PSFile` class was removed in Pillow 11 (2024-10-15). This class was only made as a helper to be used internally, so there is no replacement. If you need this functionality though, it is a very short class that can easily be recreated in your own code.

PyAccess and Image.USE_CFFI_ACCESS

Since Pillow's C API is now faster than PyAccess on PyPy, `PyAccess` has been removed. Pillow's C API will now be used on PyPy instead.

`Image.USE_CFFI_ACCESS`, for switching from the C API to PyAccess, was similarly removed.

TiffImagePlugin IFD_LEGACY_API

An unused setting, `TiffImagePlugin.IFD_LEGACY_API`, has been removed.

WebP 0.4

Support for WebP 0.4 and earlier has been removed; WebP 0.5 is the minimum supported.

Deprecations

FreeType 2.9.0

Deprecated since version 11.0.0.

Support for FreeType 2.9.0 is deprecated and will be removed in Pillow 12.0.0 (2025-10-15), when FreeType 2.9.1 will be the minimum supported.

We recommend upgrading to at least FreeType 2.10.4, which fixed a severe vulnerability introduced in FreeType 2.6 (**CVE 2020-15999**).

Get internal pointers to objects

Deprecated since version 11.0.0.

`Image.core.ImagingCore.id` and `Image.core.ImagingCore.unsafe_ptrs` have been deprecated and will be removed in Pillow 12 (2025-10-15). They were used for obtaining raw pointers to `ImagingCore` internals. To interact with C code, you can use `Image.Image.getim()`, which returns a `Capsule` object.

ICNS (width, height, scale) sizes

Deprecated since version 11.0.0.

Setting an ICNS image size to `(width, height, scale)` before loading has been deprecated. Instead, `load(scale)` can be used.

Image isImageType()

Deprecated since version 11.0.0.

`Image.isImageType(im)` has been deprecated. Use `isinstance(im, Image.Image)` instead.

ImageMath.lambda_eval and ImageMath.unsafe_eval options parameter

Deprecated since version 11.0.0.

The `options` parameter in *lambda_eval()* and *unsafe_eval()* has been deprecated. One or more keyword arguments can be used instead.

JpegImageFile.huffman_ac and JpegImageFile.huffman_dc

Deprecated since version 11.0.0.

The `huffman_ac` and `huffman_dc` dictionaries on JPEG images were unused. They have been deprecated, and will be removed in Pillow 12 (2025-10-15).

Specific WebP feature checks

Deprecated since version 11.0.0.

`features.check("transp_webp")`, `features.check("webp_mux")` and `features.check("webp_anim")` are now deprecated. They will always return `True` if the WebP module is installed, until they are removed in Pillow 12.0.0 (2025-10-15).

API changes

Default resampling filter for I;16* image modes

The default resampling filter for I;16, I;16L, I;16B and I;16N has been changed from `Image.NEAREST` to `Image.BICUBIC`, to match the majority of modes.

API additions

Writing XMP bytes to JPEG and MPO

XMP data can now be saved to JPEG files using an `xmp` argument:

```
im.save("out.jpg", xmp=b"test")
```

The data can also be set through *info*, for use when saving either JPEG or MPO images:

```
im.info["xmp"] = b"test"
im.save("out.jpg")
```

Other changes

Python 3.13

Pillow 10.4.0 had wheels built against Python 3.13 beta, available as a preview to help others prepare for 3.13, and to ensure Pillow could be used immediately at the release of 3.13.0 final (2024-10-07, **PEP 719**).

Pillow 11.0.0 now officially supports Python 3.13.

Support has also been added for the experimental free-threaded mode of **PEP 703**.

Python 3.13 only supports macOS versions 10.13 and later.

C-level flags

Some compiling flags like `WITH_THREADING`, `WITH_IMAGECHOPS`, and other `WITH_*` were removed. These flags were not available through the build system, but they could be edited in the C source.

1.6.11 10.4.0 (2024-07-01)

Security

ImageShow.WindowsViewer.show_file

If an attacker has control over the `path` passed to `ImageShow.WindowsViewer.show_file()`, they may be able to execute arbitrary shell commands.

To prevent this, a `FileNotFoundError` will be raised if the `path` does not exist as a file. To provide a consistent experience, the error has been added to all *`ImageShow`* viewers.

Deprecations

BGR;15, BGR;16 and BGR;24

The experimental BGR;15, BGR;16 and BGR;24 modes have been deprecated.

Non-image modes in ImageCms

The use in *`ImageCms`* of input modes and output modes that are not Pillow image modes has been deprecated. Defaulting to "L" or "1" if the mode cannot be mapped is also deprecated.

Support for LibTIFF earlier than 4

Support for LibTIFF earlier than version 4 has been deprecated. Upgrade to a newer version of LibTIFF instead.

ImageDraw.getdraw hints parameter

The `hints` parameter in *`getdraw()`* has been deprecated.

API additions

ImageDraw.circle

Added *`circle()`*. It provides the same functionality as *`ellipse()`*, but instead of taking a bounding box, it takes a center point and radius.

Other changes

Python 3.13 beta

To help others prepare for Python 3.13, wheels have been built against the 3.13 beta as a preview. This is not official support for Python 3.13, but simply an opportunity for users to test how Pillow works with the beta and report any problems.

1.6.12 10.3.0 (2024-04-01)

Security

ImageMath eval()

> **Danger**
>
> `ImageMath.eval()` uses Python's `eval()` function to process the expression string, and carries the security risks of doing so. A direct replacement for this is the new *`unsafe_eval()`*, but that carries the same risks. It is not recommended to process expressions without considering this. *`lambda_eval()`* is a more secure alternative.

CVE 2024-28219: Fix buffer overflow in `_imagingcms.c`

In `_imagingcms.c`, two `strcpy` calls were able to copy too much data into fixed length strings. This has been fixed by using `strncpy` instead.

Deprecations

ImageCms constants and versions() function

A number of constants and a function in *`ImageCms`* have been deprecated. This includes a table of flags based on LittleCMS version 1 which has been replaced with a new class *`ImageCms.Flags`* based on LittleCMS 2 flags.

Deprecated	Use instead
ImageCms.DESCRIPTION	No replacement
ImageCms.VERSION	PIL.__version__
ImageCms.FLAGS["MATRIXINPUT"]	*ImageCms.Flags.CLUT_POST_LINEARIZATION*
ImageCms.FLAGS["MATRIXOUTPUT"]	*ImageCms.Flags.FORCE_CLUT*
ImageCms.FLAGS["MATRIXONLY"]	No replacement
ImageCms.FLAGS["NOWHITEONWHITEFIXU	*ImageCms.Flags.NOWHITEONWHITEFIXUP*
ImageCms.FLAGS["NOPRELINEARIZATION	*ImageCms.Flags.CLUT_PRE_LINEARIZATION*
ImageCms.FLAGS["GUESSDEVICECLASS"]	*ImageCms.Flags.GUESSDEVICECLASS*
ImageCms.FLAGS["NOTCACHE"]	*ImageCms.Flags.NOCACHE*
ImageCms.FLAGS["NOTPRECALC"]	*ImageCms.Flags.NOOPTIMIZE*
ImageCms.FLAGS["NULLTRANSFORM"]	*ImageCms.Flags.NULLTRANSFORM*
ImageCms.FLAGS["HIGHRESPRECALC"]	*ImageCms.Flags.HIGHRESPRECALC*
ImageCms.FLAGS["LOWRESPRECALC"]	*ImageCms.Flags.LOWRESPRECALC*
ImageCms.FLAGS["GAMUTCHECK"]	*ImageCms.Flags.GAMUTCHECK*
ImageCms.FLAGS["WHITEBLACKCOMPENSA	*ImageCms.Flags.BLACKPOINTCOMPENSATION*
ImageCms.FLAGS["BLACKPOINTCOMPENSA	*ImageCms.Flags.BLACKPOINTCOMPENSATION*
ImageCms.FLAGS["SOFTPROOFING"]	*ImageCms.Flags.SOFTPROOFING*
ImageCms.FLAGS["PRESERVEBLACK"]	*ImageCms.Flags.NONEGATIVES*
ImageCms.FLAGS["NODEFAULTRESOURCED	*ImageCms.Flags.NODEFAULTRESOURCEDEF*
ImageCms.FLAGS["GRIDPOINTS"]	*ImageCms.Flags.GRIDPOINTS()*
ImageCms.versions()	*PIL.features.version_module()* with feature="littlecms2", sys.version or sys.version_info, and PIL.__version__

ImageMath.eval()

ImageMath.eval() has been deprecated. Use *lambda_eval()* or *unsafe_eval()* instead. See earlier security notes for more information.

API changes

Added alpha_quality argument when saving WebP images

When saving WebP images, an `alpha_quality` argument can be passed to the encoder. It is an integer value between 0 to 100, where values other than 100 will provide lossy compression.

Negative kmeans error

When calling *quantize()*, a negative `kmeans` will now raise a `ValueError`, unless a palette is supplied to make the value redundant.

Negative P1-P3 PPM value error

If a P1-P3 PPM image contains a negative value, a `ValueError` will now be raised.

API additions

Added PerspectiveTransform

PerspectiveTransform has been added, meaning that all of the *Transform* values now have a corresponding subclass of *Transform*.

Other changes

Portable FloatMap (PFM) images

Support has been added for reading and writing grayscale (Pf format) Portable FloatMap (PFM) files containing `F` data.

Release GIL when fetching WebP frames

Python's Global Interpreter Lock is now released when fetching WebP frames from the libwebp decoder.

Type hints

Pillow now has type hints for a large part of its modules, and the package includes a `py.typed` file and the `Typing :: Typed` Trove classifier.

1.6.13 10.2.0 (2024-01-02)

Security

ImageFont.getmask: Applied ImageFont.MAX_STRING_LENGTH

To protect against potential DOS attacks when using arbitrary strings as text input, Pillow will now raise a `ValueError` if the number of characters passed into *PIL.ImageFont.ImageFont.getmask()* is over a certain limit, *PIL.ImageFont.MAX_STRING_LENGTH*.

This threshold can be changed by setting *PIL.ImageFont.MAX_STRING_LENGTH*. It can be disabled by setting `ImageFont.MAX_STRING_LENGTH = None`.

A decompression bomb check has also been added to *PIL.ImageFont.ImageFont.getmask()*.

ImageFont.getmask: Trim glyph size

To protect against potential DOS attacks when using PIL fonts, *PIL.ImageFont.ImageFont* now trims the size of individual glyphs so that they do not extend beyond the bitmap image.

CVE 2023-50447: ImageMath.eval: Restricted environment keys

If an attacker has control over the keys passed to the `environment` argument of `PIL.ImageMath.eval()`, they may be able to execute arbitrary code. To prevent this, keys matching the names of builtins and keys containing double underscores will now raise a `ValueError`.

Deprecations

ImageFile.raise_oserror

`ImageFile.raise_oserror()` has been deprecated and will be removed in Pillow 12.0.0 (2025-10-15). The function is undocumented and is only useful for translating error codes returned by a codec's `decode()` method, which ImageFile already does automatically.

IptcImageFile helper functions

The functions `IptcImageFile.dump` and `IptcImageFile.i`, and the constant `IptcImageFile.PAD` have been deprecated and will be removed in Pillow 12.0.0 (2025-10-15). These are undocumented helper functions intended for internal use, so there is no replacement. They can each be replaced by a single line of code using builtin functions in Python.

API changes

Zero or negative font size error

When creating a *FreeTypeFont* instance, either directly or through *truetype()*, if the font size is zero or less, a `ValueError` will now be raised.

API additions

Added DdsImagePlugin enums

DDSD, *DDSCAPS*, *DDSCAPS2*, *DDPF*, *DXGI_FORMAT* and *D3DFMT* enums have been added to *PIL.DdsImagePlugin*.

JPEG RGB color space

When saving JPEG files, `keep_rgb` can now be set to `True`. This will store RGB images in the RGB color space instead of being converted to YCbCr automatically by libjpeg. When this option is enabled, attempting to chroma-subsample RGB images with the `subsampling` option will raise an `OSError`.

JPEG restart marker interval

When saving JPEG files, `restart_marker_blocks` and `restart_marker_rows` can now be used to emit restart markers whenever the specified number of MCU blocks or rows have been produced.

JPEG tables-only streamtype

When saving JPEG files, `streamtype` can now be set to 1, for tables-only. This will output only the quantization and Huffman tables for the image.

Other changes

Added DDS BC4U and DX10 BC1 and BC4 reading

Support has been added to read the BC4U format of DDS images.

Support has also been added to read DX10 BC1 and BC4, whether UNORM or TYPELESS.

Support arbitrary masks for uncompressed RGB DDS images

All masks are now supported when reading DDS images with uncompressed RGB data, allowing for bit counts other than 24 and 32.

Saving TIFF tag RowsPerStrip

When saving TIFF images, the TIFF tag RowsPerStrip can now be one of the tags set by the user, rather than always being calculated by Pillow.

Optimized ImageColor.getrgb and getcolor

The color calculations of *getrgb* and *getcolor* are now cached using `functools.lru_cache()`. Cached calls of `getrgb` are 3.1 - 91.4 times as fast and `getcolor` are 5.1 - 19.6 times as fast.

Optimized ImageMode.getmode

The lookups made by *getmode* are now cached using `functools.lru_cache()` instead of a custom cache. Cached calls are 1.2 times as fast.

Optimized ImageStat.Stat count and extrema

Calculating the *count* and *extrema* statistics is now faster. After the histogram is created in `st = ImageStat.Stat(im)`, `st.count` is 3 times as fast on average and `st.extrema` is 12 times as fast on average.

Encoder errors now report error detail as string

`OSError` exceptions from image encoders now include a textual description of the error instead of a numeric error code.

Type hints

Work has begun to add type annotations to Pillow, including:

- *ContainerIO*
- *FontFile*, *BdfFontFile* and *PcfFontFile*
- *ImageChops*
- *ImageMode*
- *ImageSequence*
- *ImageTransform*
- *TarIO*

1.6.14 10.1.0 (2023-10-15)

API changes

Setting image mode

If you attempt to set the mode of an image directly, e.g. `im.mode = "RGBA"`, you will now receive an `AttributeError`. This is not about removing existing functionality, but instead about raising an explicit error to prevent later consequences. The `convert` method is the correct way to change an image's mode.

Accept a list in getpixel()

`getpixel()` now accepts a list of coordinates, as well as a tuple.

```
from PIL import Image
im = Image.new("RGB", (1, 1))
im.getpixel((0, 0))
im.getpixel([0, 0])
```

BoxBlur and GaussianBlur allow for different x and y radii

`BoxBlur` and *`GaussianBlur`* now allow a sequence of x and y radii to be specified, rather than a single number for both dimensions.

```
from PIL import ImageFilter
ImageFilter.BoxBlur((2, 5))
ImageFilter.GaussianBlur((2, 5))
```

API additions

EpsImagePlugin.gs_binary

`EpsImagePlugin.gs_windows_binary` stores the name of the Ghostscript executable on Windows. `EpsImagePlugin.gs_binary` has now been added for all platforms, and can be used to customise the name of the executable, or disable use entirely through `EpsImagePlugin.gs_binary = False`.

has_transparency_data

Images now have *`has_transparency_data`* to indicate whether the image has transparency data, whether in the form of an alpha channel, a palette with an alpha channel, or a "transparency" key in the *`info`* dictionary.

Even if this attribute is true, the image might still appear solid, if all of the values shown within are opaque.

ImageOps.cover

Returns a resized version of the image, so that the requested size is covered, while maintaining the original aspect ratio.

See *Resize relative to a given size* for a comparison between this and similar `ImageOps` methods.

size and font_size arguments when using default font

Pillow has had a "better than nothing" default font, which can only be drawn at one font size. Now, if FreeType support is available, a version of Aileron Regular is loaded, which can be drawn at chosen font sizes.

The following `size` and `font_size` arguments can now be used to specify a font size for this new builtin font:

```
ImageFont.load_default(size=24)
draw.text((0, 0), "test", font_size=24)
draw.textlength((0, 0), "test", font_size=24)
draw.textbbox((0, 0), "test", font_size=24)
draw.multiline_text((0, 0), "test", font_size=24)
draw.multiline_textbbox((0, 0), "test", font_size=24)
```

Other changes

Python 3.12

Pillow 10.0.0 had wheels built against Python 3.12 beta, available as a preview to help others prepare for 3.12, and to ensure Pillow could be used immediately at the release of 3.12.0 final (2023-10-02, **PEP 693**).

Pillow 10.1.0 now officially supports Python 3.12.

Added support for DDS BC5U and 8-bit color indexed images

Support has been added to read BC5U DDS files as RGB images, and PALETTEINDEXED8 DDS files as P mode images.

Support reading signed 8-bit YCbCr TIFF images

TIFF images with unsigned integer data, 8 bits per sample and a photometric interpretation of YCbCr can now be read.

1.6.15 10.0.1 (2023-09-15)

Security

CVE 2023-4863: Updated install script and updated wheels

This release provides an updated install script and updated wheels to include libwebp 1.3.2, preventing a potential heap buffer overflow in WebP.

Other changes

Updated tests to pass with latest zlib version

The release of zlib 1.3 caused one of the tests in the Pillow test suite to fail.

1.6.16 10.0.0 (2023-07-01)

Security

Limit size even if one dimension is zero

When performing decompression bomb checks, Pillow did not reject images with excessive width and zero height, or zero width and excessive height. That has now been fixed.

This effectively dates to the PIL fork, since problem images would still have been processed before Pillow started checking for decompression bombs.

CVE 2023-44271: Added ImageFont.MAX_STRING_LENGTH

To protect against potential DOS attacks when using arbitrary strings as text input, Pillow will now raise a `ValueError` if the number of characters passed into ImageFont methods is over a certain limit, *PIL.ImageFont.MAX_STRING_LENGTH*.

This threshold can be changed by setting *PIL.ImageFont.MAX_STRING_LENGTH*. It can be disabled by setting `ImageFont.MAX_STRING_LENGTH = None`.

Backwards incompatible changes

Categories

`im.category` has been removed, along with the related `Image.NORMAL`, `Image.SEQUENCE` and `Image.CONTAINER` attributes.

To determine if an image has multiple frames or not, `getattr(im, "is_animated", False)` can be used instead.

Tk/Tcl 8.4

Support for Tk/Tcl 8.4 has been removed.

JpegImagePlugin.convert_dict_qtables

Since deprecation in Pillow 8.3.0, the `convert_dict_qtables` method no longer performed any operations on the data given to it, and has been removed.

ImagePalette size parameter

Before Pillow 8.3.0, `ImagePalette` required palette data of particular lengths by default, and the `size` parameter could be used to override that. Pillow 8.3.0 removed the default required length, also removing the need for the `size` parameter.

ImageShow.Viewer.show_file file argument

The `file` argument in *show_file()* has been removed and replaced by `path`.

In effect, `viewer.show_file("test.jpg")` will continue to work unchanged.

Constants

A number of constants have been removed. Instead, `enum.IntEnum` classes have been added.

Removed	Use instead
Image.LINEAR	Image.BILINEAR or Image.Resampling.BILINEAR
Image.CUBIC	Image.BICUBIC or Image.Resampling.BICUBIC
Image.ANTIALIAS	Image.LANCZOS or Image.Resampling.LANCZOS
ImageCms.INTENT_PERCEPTUAL	ImageCms.Intent.PERCEPTUAL
ImageCms.INTENT_RELATIVE_COLORMETRIC	ImageCms.Intent.RELATIVE_COLORMETRIC
ImageCms.INTENT_SATURATION	ImageCms.Intent.SATURATION
ImageCms.INTENT_ABSOLUTE_COLORIMETRIC	ImageCms.Intent.ABSOLUTE_COLORIMETRIC
ImageCms.DIRECTION_INPUT	ImageCms.Direction.INPUT
ImageCms.DIRECTION_OUTPUT	ImageCms.Direction.OUTPUT
ImageCms.DIRECTION_PROOF	ImageCms.Direction.PROOF
ImageFont.LAYOUT_BASIC	ImageFont.Layout.BASIC
ImageFont.LAYOUT_RAQM	ImageFont.Layout.RAQM
BlpImagePlugin.BLP_FORMAT_JPEG	BlpImagePlugin.Format.JPEG
BlpImagePlugin.BLP_ENCODING_UNCOMPRESSED	BlpImagePlugin.Encoding.UNCOMPRESSED
BlpImagePlugin.BLP_ENCODING_DXT	BlpImagePlugin.Encoding.DXT
BlpImagePlugin.BLP_ENCODING_UNCOMPRESSED_RAW_I	BlpImagePlugin.Encoding.UNCOMPRESSED_RAW_RGBA
BlpImagePlugin.BLP_ALPHA_ENCODING_DXT1	BlpImagePlugin.AlphaEncoding.DXT1
BlpImagePlugin.BLP_ALPHA_ENCODING_DXT3	BlpImagePlugin.AlphaEncoding.DXT3
BlpImagePlugin.BLP_ALPHA_ENCODING_DXT5	BlpImagePlugin.AlphaEncoding.DXT5
FtexImagePlugin.FORMAT_DXT1	FtexImagePlugin.Format.DXT1
FtexImagePlugin.FORMAT_UNCOMPRESSED	FtexImagePlugin.Format.UNCOMPRESSED
PngImagePlugin.APNG_DISPOSE_OP_NONE	PngImagePlugin.Disposal.OP_NONE
PngImagePlugin.APNG_DISPOSE_OP_BACKGROUND	PngImagePlugin.Disposal.OP_BACKGROUND
PngImagePlugin.APNG_DISPOSE_OP_PREVIOUS	PngImagePlugin.Disposal.OP_PREVIOUS
PngImagePlugin.APNG_BLEND_OP_SOURCE	PngImagePlugin.Blend.OP_SOURCE
PngImagePlugin.APNG_BLEND_OP_OVER	PngImagePlugin.Blend.OP_OVER

FitsStubImagePlugin

The stub image plugin `FitsStubImagePlugin` has been removed. FITS images can be read without a handler through *FitsImagePlugin* instead.

Font size and offset methods

Several functions for computing the size and offset of rendered text have been removed:

Removed	Use instead
FreeTypeFont.getsize() and FreeTypeFont.getoffset()	*FreeTypeFont.getbbox()* and *FreeTypeFont.getlength()*
FreeTypeFont.getsize_multiline()	*ImageDraw.multiline_textbbox()*
ImageFont.getsize()	*ImageFont.getbbox()* and *ImageFont.getlength()*
TransposedFont.getsize()	*TransposedFont.getbbox()* and *TransposedFont.getlength()*
ImageDraw.textsize() and ImageDraw.multiline_textsize()	*ImageDraw.textbbox()*, *ImageDraw.textlength()* and *ImageDraw.multiline_textbbox()*
ImageDraw2.Draw.textsize()	*ImageDraw2.Draw.textbbox()* and *ImageDraw2.Draw.textlength()*

FreeTypeFont.getmask2 fill parameter

The undocumented `fill` parameter of *`FreeTypeFont.getmask2()`* has been removed.

PhotoImage.paste box parameter

The `box` parameter was unused and has been removed.

PyQt5 and PySide2

Qt 5 reached end-of-life on 2020-12-08 for open-source users (and will reach EOL on 2023-12-08 for commercial licence holders).

Support for PyQt5 and PySide2 has been removed from `ImageQt`. Upgrade to PyQt6 or PySide6 instead.

Image.coerce_e

This undocumented method has been removed.

Deprecations

PyAccess and Image.USE_CFFI_ACCESS

Since Pillow's C API is now faster than PyAccess on PyPy, `PyAccess` has been deprecated and will be removed in Pillow 11.0.0 (2024-10-15). Pillow's C API will now be used by default on PyPy instead.

`Image.USE_CFFI_ACCESS`, for switching from the C API to PyAccess, is similarly deprecated.

API changes

Added line width parameter to ImageDraw regular_polygon

An optional line `width` parameter has been added to `ImageDraw.Draw.regular_polygon`.

API additions

Added `alpha_only` argument to `getbbox()`

`getbbox()` now accepts a keyword argument of `alpha_only`. This is an optional flag, defaulting to `True`. If `True` and the image has an alpha channel, trim transparent pixels. Otherwise, trim pixels when all channels are zero.

Other changes

32-bit wheels

32-bit wheels are no longer provided.

Support display_jpeg() in IPython

In addition to `display()` and `display_png`, `display_jpeg()` can now also be used to display images in IPython:

```
from PIL import Image
from IPython.display import display_jpeg

im = Image.new("RGB", (100, 100), (255, 0, 0))
display_jpeg(im)
```

Support reading signed 8-bit TIFF images

TIFF images with signed integer data, 8 bits per sample and a photometric interpretation of BlackIsZero can now be read.

1.6.17 9.5.0 (2023-04-01)

Security

Clear PPM half token after use

Image files that are small on disk are often prevented from expanding to be big images consuming a large amount of resources simply because they lack the data to populate those resources.

PpmImagePlugin might hold onto the last data read for a pixel value in case the pixel value has not been finished yet. However, that data was not being cleared afterwards, meaning that infinite data could be available to fill any image size. This has been present since Pillow 9.2.0.

That data is now cleared after use.

Saving TIFF tag ImageSourceData

If Pillow incorrectly saved the TIFF tag ImageSourceData as ASCII instead of UNDEFINED, a segmentation fault was triggered.

The correct tag type will now be used by default instead.

Deprecations

PSFile

The `PSFile` class has been deprecated and will be removed in Pillow 11 (2024-10-15). This class was only made as a helper to be used internally, so there is no replacement. If you need this functionality though, it is a very short class that can easily be recreated in your own code.

API additions

QOI file format

Pillow can now read images in Quite OK Image format.

Added `dpi` argument when saving PDFs

When saving a PDF, resolution could already be specified using the `resolution` argument. Now, a tuple of `(x_resolution, y_resolution)` can be provided as `dpi`. If both are provided, `dpi` will override `resolution`.

Added `corners` argument to `ImageDraw.rounded_rectangle()`

`ImageDraw.rounded_rectangle()` now accepts a keyword argument of `corners`. This a tuple of Booleans, specifying whether to round each corner, `(top_left, top_right, bottom_right, bottom_left)`.

JPEG2000 comments and PLT marker

When opening a JPEG2000 image, the comment may now be read into *`info`*. The `comment` keyword argument can be used to save it back again.

If OpenJPEG 2.4.0 or later is available and the `plt` keyword argument is present and true when saving JPEG2000 images, tell the encoder to generate PLT markers.

Other changes

Added support for saving PDFs in RGBA mode

Using the JPXDecode filter, PDFs can now be saved in RGBA mode.

Improved I;16N support

Support has been added for I;16N access, packing and unpacking. Conversion to and from L mode has also been added.

BGR;* modes

It is now possible to create new BGR;15, BGR;16 and BGR;24 images. Conversely, BGR;32 has been removed from ImageMode and its associated methods, dropping the little support Pillow had for the mode.

With that, all modes listed under *Modes* can now be used to create a new image.

1.6.18 9.4.0 (2023-01-02)

Security

Fix memory DOS in ImageFont

A corrupt or specially crafted TTF font could have font metrics that lead to unreasonably large sizes when rendering text in font. `ImageFont.py` did not check the image size before allocating memory for it. This dates to the PIL fork. Pillow 8.2.0 added a check for large sizes, but did not consider the case where one dimension is zero.

Null pointer dereference crash in ImageFont

Pillow attempted to dereference a null pointer in `ImageFont`, leading to a crash. An error is now raised instead. This has been present since Pillow 8.0.0.

API additions

Added start position for getmask and getmask2

Text may render differently when starting at fractional coordinates, so *FreeTypeFont.getmask()* and *FreeTypeFont.getmask2()* now support a `start` argument. This tuple of horizontal and vertical offset will be used internally by *ImageDraw.text()* to more accurately place text at the `xy` coordinates.

Added the `exact` encoding option for WebP

The `exact` encoding option for WebP is now supported. The WebP encoder removes the hidden RGB values for better compression by default in libwebp 0.5 or later. By setting this option to `True`, the encoder will keep the hidden RGB values.

Added `signed` option when saving JPEG2000

If the `signed` keyword argument is present and true when saving JPEG2000 images, then tell the encoder to save the image as signed.

Added IFD, Interop and LightSource ExifTags enums

IFD has been added, allowing enums to be used with *get_ifd()*:

```
from PIL import Image, ExifTags
im = Image.open("Tests/images/flower.jpg")
print(im.getexif().get_ifd(ExifTags.IFD.Exif))
```

IFD1 can also be used with *get_ifd()*, but it should not be used in other contexts, as the enum value is only internally meaningful.

Interop has been added for tags within the Interop IFD:

```
from PIL import Image, ExifTags
im = Image.open("Tests/images/flower.jpg")
interop_ifd = im.getexif().get_ifd(ExifTags.IFD.Interop)
print(interop_ifd.get(ExifTags.Interop.InteropIndex))  # R98
```

LightSource has been added for values within the LightSource tag:

```
from PIL import Image, ExifTags
im = Image.open("Tests/images/iptc.jpg")
exif_ifd = im.getexif().get_ifd(ExifTags.IFD.Exif)
print(ExifTags.LightSource(exif_ifd[0x9208]))  # LightSource.Unknown
```

getxmp()

XMP data can now be decoded for WEBP images through `getxmp()`.

Writing JPEG comments

When saving a JPEG image, a comment can now be written from *info*, or by using an argument when saving:

```
im.save(out, comment="Test comment")
```

Other changes

Added support for DDS L and LA images

Support has been added to read and write L and LA DDS images in the uncompressed format, known as "luminance" textures.

Constants

In Pillow 9.1.0, the following constants were deprecated. That has been reversed and these constants will now remain available.

- `Image.NONE`
- `Image.NEAREST`
- `Image.ORDERED`
- `Image.RASTERIZE`
- `Image.FLOYDSTEINBERG`
- `Image.WEB`
- `Image.ADAPTIVE`
- `Image.AFFINE`

- Image.EXTENT
- Image.PERSPECTIVE
- Image.QUAD
- Image.MESH
- Image.FLIP_LEFT_RIGHT
- Image.FLIP_TOP_BOTTOM
- Image.ROTATE_90
- Image.ROTATE_180
- Image.ROTATE_270
- Image.TRANSPOSE
- Image.TRANSVERSE
- Image.BOX
- Image.BILINEAR
- Image.HAMMING
- Image.BICUBIC
- Image.LANCZOS
- Image.MEDIANCUT
- Image.MAXCOVERAGE
- Image.FASTOCTREE
- Image.LIBIMAGEQUANT

1.6.19 9.3.0 (2022-10-29)

Security

Initialize libtiff buffer when saving

When saving a TIFF image to a file object using libtiff, the buffer was not initialized. This behaviour introduced in Pillow 2.0.0, and has now been fixed.

Decode JPEG compressed BLP1 data in original mode

Within the BLP image format, BLP1 data may use JPEG compression. Instead of telling the JPEG library that this data is in BGRX mode, Pillow will now decode the data in its natural CMYK mode, then convert it to RGB and rearrange the channels afterwards. Trying to load the data in an incorrect mode could result in a segmentation fault. This issue was introduced in Pillow 9.1.0.

Limit SAMPLESPERPIXEL to avoid runtime DOS

A large value in the SAMPLESPERPIXEL tag could lead to a memory and runtime DOS in TiffImagePlugin.py when setting up the context for image decoding. This was introduced in Pillow 9.2.0, found with OSS-Fuzz and fixed by limiting SAMPLESPERPIXEL to the number of planes that we can decode.

API additions

Allow default ImageDraw font to be set

Rather than specifying a font when calling text-related ImageDraw methods, or setting a font on each ImageDraw instance, the default font can now be set for all future ImageDraw operations:

```
from PIL import ImageDraw, ImageFont
ImageDraw.ImageDraw.font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
```

Saving multiple MPO frames

Multiple MPO frames can now be saved. Using the `save_all` argument, all of an image's frames will be saved to file:

```
from PIL import Image
im = Image.open("frozenpond.mpo")
im.save(out, save_all=True)
```

Additional images can also be appended when saving, by combining the `save_all` argument with the `append_images` argument:

```
im.save(out, save_all=True, append_images=[im1, im2, ...])
```

Added ExifTags enums

The data from *TAGS* and *GPSTAGS* is now also exposed as `enum.IntEnum` classes: *Base* and *GPS*.

Other changes

Python 3.11 wheels

Pillow 9.2.0 had wheels built against Python 3.11 beta, available as a preview to help others prepare for 3.11, and ensure Pillow can be used immediately on release day of 3.11.0 final (2022-10-24, **PEP 664**).

Pillow 9.3.0 now officially includes binary wheels for Python 3.11 final.

Windows wheels

This release contains wheels for Windows built using GitHub Actions.

Previously they were built by Christoph Gohlke.

A huge thanks to Christoph for building Windows binaries for us for around a decade, plus testing, and fixing over a hundred bug fixes along the way, in addition to building and hosting unofficial Windows binaries for hundreds of Python projects!

Added DDS ATI1, ATI2 and BC6H reading

Support has been added to read the ATI1, ATI2 and BC6H formats of DDS images.

Release GIL when converting images using matrix operations

Python's Global Interpreter Lock is now released when converting images using matrix operations.

Show all frames with ImageShow

When calling *show()* or using *ImageShow*, all frames will now be shown.

1.6.20 9.2.0 (2022-07-01)

Security

An additional decompression bomb check has been added for the GIF format.

Deprecations

PyQt5 and PySide2

Deprecated since version 9.2.0.

Qt 5 reached end-of-life on 2020-12-08 for open-source users (and will reach EOL on 2023-12-08 for commercial licence holders).

Support for PyQt5 and PySide2 has been deprecated from `ImageQt` and will be removed in Pillow 10 (2023-07-01). Upgrade to PyQt6 or PySide6 instead.

FreeTypeFont.getmask2 fill parameter

Deprecated since version 9.2.0.

The undocumented `fill` parameter of *FreeTypeFont.getmask2()* has been deprecated and will be removed in Pillow 10 (2023-07-01).

PhotoImage.paste box parameter

Deprecated since version 9.2.0.

The `box` parameter is unused. It will be removed in Pillow 10.0.0 (2023-07-01).

Image.coerce_e

Deprecated since version 9.2.0.

This undocumented method has been deprecated and will be removed in Pillow 10 (2023-07-01).

Font size and offset methods

Deprecated since version 9.2.0.

Several functions for computing the size and offset of rendered text have been deprecated and will be removed in Pillow 10 (2023-07-01):

Deprecated	Use instead
FreeTypeFont.getsize() and FreeTypeFont.getoffset()	*FreeTypeFont.getbbox()* and *FreeTypeFont.getlength()*
FreeTypeFont.getsize_multiline()	*ImageDraw.multiline_textbbox()*
ImageFont.getsize()	*ImageFont.getbbox()* and *ImageFont.getlength()*
TransposedFont.getsize()	*TransposedFont.getbbox()* and *TransposedFont.getlength()*
ImageDraw.textsize() and ImageDraw.multiline_textsize()	*ImageDraw.textbbox()*, *ImageDraw.textlength()* and *ImageDraw.multiline_textbbox()*
ImageDraw2.Draw.textsize()	*ImageDraw2.Draw.textbbox()* and *ImageDraw2.Draw.textlength()*

Previous code:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
width, height = font.getsize("Hello world")
left, top = font.getoffset("Hello world")

im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
width, height = draw.textsize("Hello world", font)

width, height = font.getsize_multiline("Hello\nworld")
width, height = draw.multiline_textsize("Hello\nworld", font)
```

Use instead:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
left, top, right, bottom = font.getbbox("Hello world")
width, height = right - left, bottom - top

im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
width = draw.textlength("Hello world", font)

left, top, right, bottom = draw.multiline_textbbox((0, 0), "Hello\nworld", font)
width, height = right - left, bottom - top
```

Previously, the `size` methods returned a `height` that included the vertical offset of the text, while the new `bbox` methods distinguish this as a `top` offset.

If you are using these methods for aligning text, consider using *Text anchors* instead which avoid issues that can occur with non-English text or unusual fonts. For example, instead of the following code:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
```

(continues on next page)

(continued from previous page)

```
im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
width, height = draw.textsize("Hello world", font)
x, y = (100 - width) / 2, (100 - height) / 2
draw.text((x, y), "Hello world", font=font)
```

Use instead:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")

im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
draw.text((100 / 2, 100 / 2), "Hello world", font=font, anchor="mm")
```

API additions

Image.apply_transparency

Added *apply_transparency()*, a method to take a P mode image with "transparency" in `im.info`, and apply the transparency to the palette instead. The image's palette mode will become "RGBA", and "transparency" will be removed from `im.info`.

Other changes

Using gnome-screenshot on Linux

In *grab()* on Linux, if `xdisplay` is `None` then `gnome-screenshot` will be used to capture the display if it is installed. To capture the default X11 display instead, pass `xdisplay=""`.

1.6.21 9.1.1 (2022-05-17)

Security

This release addresses several security issues.

CVE 2022-30595: Heap buffer overflow

When reading a TGA file with RLE packets that cross scan lines, Pillow reads the information past the end of the first line without deducting that from the length of the remaining file data. This vulnerability was introduced in Pillow 9.1.0, and can cause a heap buffer overflow.

Decompression bomb check fix

Opening an image with a zero or negative height has been found to bypass a decompression bomb check. This will now raise a `SyntaxError` instead, in turn raising a `PIL.UnidentifiedImageError`.

1.6.22 9.1.0 (2022-04-01)

Deprecations

Constants

A number of constants have been deprecated and will be removed in Pillow 10.0.0 (2023-07-01). Instead, `enum.IntEnum` classes have been added.

> **Note**
>
> Some of these deprecations were restored in Pillow 9.4.0. See *Constants*

Deprecated	Use instead
`Image.NONE`	Either `Image.Dither.NONE` or `Image.Resampling.NEAREST`
`Image.NEAREST`	Either `Image.Dither.NONE` or `Image.Resampling.NEAREST`
`Image.ORDERED`	`Image.Dither.ORDERED`
`Image.RASTERIZE`	`Image.Dither.RASTERIZE`
`Image.FLOYDSTEINBERG`	`Image.Dither.FLOYDSTEINBERG`
`Image.WEB`	`Image.Palette.WEB`
`Image.ADAPTIVE`	`Image.Palette.ADAPTIVE`
`Image.AFFINE`	`Image.Transform.AFFINE`
`Image.EXTENT`	`Image.Transform.EXTENT`
`Image.PERSPECTIVE`	`Image.Transform.PERSPECTIVE`
`Image.QUAD`	`Image.Transform.QUAD`
`Image.MESH`	`Image.Transform.MESH`
`Image.FLIP_LEFT_RIGHT`	`Image.Transpose.FLIP_LEFT_RIGHT`
`Image.FLIP_TOP_BOTTOM`	`Image.Transpose.FLIP_TOP_BOTTOM`
`Image.ROTATE_90`	`Image.Transpose.ROTATE_90`
`Image.ROTATE_180`	`Image.Transpose.ROTATE_180`
`Image.ROTATE_270`	`Image.Transpose.ROTATE_270`
`Image.TRANSPOSE`	`Image.Transpose.TRANSPOSE`
`Image.TRANSVERSE`	`Image.Transpose.TRANSVERSE`
`Image.BOX`	`Image.Resampling.BOX`
`Image.BILINEAR`	`Image.Resampling.BILINEAR`
`Image.LINEAR`	`Image.Resampling.BILINEAR`
`Image.HAMMING`	`Image.Resampling.HAMMING`
`Image.BICUBIC`	`Image.Resampling.BICUBIC`
`Image.CUBIC`	`Image.Resampling.BICUBIC`
`Image.LANCZOS`	`Image.Resampling.LANCZOS`
`Image.ANTIALIAS`	`Image.Resampling.LANCZOS`
`Image.MEDIANCUT`	`Image.Quantize.MEDIANCUT`
`Image.MAXCOVERAGE`	`Image.Quantize.MAXCOVERAGE`
`Image.FASTOCTREE`	`Image.Quantize.FASTOCTREE`
`Image.LIBIMAGEQUANT`	`Image.Quantize.LIBIMAGEQUANT`
`ImageCms.INTENT_PERCEPTUAL`	`ImageCms.Intent.PERCEPTUAL`
`ImageCms.INTENT_RELATIVE_COLORMETRIC`	`ImageCms.Intent.RELATIVE_COLORMETRIC`
`ImageCms.INTENT_SATURATION`	`ImageCms.Intent.SATURATION`
`ImageCms.INTENT_ABSOLUTE_COLORIMETRIC`	`ImageCms.Intent.ABSOLUTE_COLORIMETRIC`

continues on next page

Table 4 – continued from previous page

Deprecated	Use instead
ImageCms.DIRECTION_INPUT	ImageCms.Direction.INPUT
ImageCms.DIRECTION_OUTPUT	ImageCms.Direction.OUTPUT
ImageCms.DIRECTION_PROOF	ImageCms.Direction.PROOF
ImageFont.LAYOUT_BASIC	ImageFont.Layout.BASIC
ImageFont.LAYOUT_RAQM	ImageFont.Layout.RAQM
BlpImagePlugin.BLP_FORMAT_JPEG	BlpImagePlugin.Format.JPEG
BlpImagePlugin.BLP_ENCODING_UNCOMPRESSED	BlpImagePlugin.Encoding.UNCOMPRESSED
BlpImagePlugin.BLP_ENCODING_DXT	BlpImagePlugin.Encoding.DXT
BlpImagePlugin.BLP_ENCODING_UNCOMPRESSED_RAW	BlpImagePlugin.Encoding. UNCOMPRESSED_RAW_RGBA
BlpImagePlugin.BLP_ALPHA_ENCODING_DXT1	BlpImagePlugin.AlphaEncoding.DXT1
BlpImagePlugin.BLP_ALPHA_ENCODING_DXT3	BlpImagePlugin.AlphaEncoding.DXT3
BlpImagePlugin.BLP_ALPHA_ENCODING_DXT5	BlpImagePlugin.AlphaEncoding.DXT5
FtexImagePlugin.FORMAT_DXT1	FtexImagePlugin.Format.DXT1
FtexImagePlugin.FORMAT_UNCOMPRESSED	FtexImagePlugin.Format.UNCOMPRESSED
PngImagePlugin.APNG_DISPOSE_OP_NONE	PngImagePlugin.Disposal.OP_NONE
PngImagePlugin.APNG_DISPOSE_OP_BACKGROUND	PngImagePlugin.Disposal.OP_BACKGROUND
PngImagePlugin.APNG_DISPOSE_OP_PREVIOUS	PngImagePlugin.Disposal.OP_PREVIOUS
PngImagePlugin.APNG_BLEND_OP_SOURCE	PngImagePlugin.Blend.OP_SOURCE
PngImagePlugin.APNG_BLEND_OP_OVER	PngImagePlugin.Blend.OP_OVER

ImageShow.Viewer.show_file file argument

The `file` argument in *show_file()* has been deprecated and will be removed in Pillow 10.0.0 (2023-07-01). It has been replaced by `path`.

In effect, `viewer.show_file("test.jpg")` will continue to work unchanged. `viewer.show_file(file="test.jpg")` will raise a deprecation warning, and suggest `viewer.show_file(path="test.jpg")` instead.

FitsStubImagePlugin

Deprecated since version 9.1.0.

The stub image plugin `FitsStubImagePlugin` has been deprecated and will be removed in Pillow 10.0.0 (2023-07-01). FITS images can be read without a handler through *FitsImagePlugin* instead.

API changes

Raise an error when performing a negative crop

Performing a negative crop on an image previously just returned a `(0, 0)` image. Now it will raise a `ValueError`, to help reduce confusion if a user has unintentionally provided the wrong arguments.

Added specific error if path coordinate type is incorrect

Rather than returning a `SystemError`, passing the incorrect types of coordinates into a path will now raise a more specific `ValueError`, with the message "incorrect coordinate type".

Replace requirements.txt with extras

Rather than installing all dependencies for docs and tests via `requirements.txt`, `extras_require` is used instead. This installs only those needed and at the same time as installing Pillow.

For example:

```
# Install with dependencies for tests:
python3 -m pip install .[tests]

# Or for building docs:
python3 -m pip install .[docs]

# Or for all:
python3 -m pip install .[docs,tests]
```

On macOS, the last argument may need to be wrapped in quotes, e.g. `python3 -m pip install ".[tests]"`

Therefore `requirements.txt` has been removed along with the `make install-req` command for installing its contents.

API additions

Added get_photoshop_blocks() to parse Photoshop TIFF tag

get_photoshop_blocks() has been added, to allow users to determine what Photoshop "Image Resource Blocks" are contained within an image. The keys of the returned dictionary are the image resource IDs.

At present, the information within each block is merely returned as a dictionary with a "data" entry. This will allow more useful information to be added in the future without breaking backwards compatibility.

Added mct and no_jp2 options for saving JPEG 2000

The *PIL.Image.Image.save()* method now supports the following options for JPEG 2000:

mct
: If `1` then enable multiple component transformation when encoding, otherwise use `0` for no component transformation (default). If MCT is enabled and `irreversible` is `True` then the Irreversible Color Transformation will be applied, otherwise encoding will use the Reversible Color Transformation. MCT works best with a `mode` of `RGB` and is only applicable when the image data has 3 components.

no_jp2
: If `True` then don't wrap the raw codestream in the JP2 file format when saving, otherwise the extension of the filename will be used to determine the format (default).

Added PyEncoder

PyEncoder has been added, allowing for file encoders to be written in Python. See *Writing Your Own File Codec in Python* for more information.

GifImagePlugin loading strategy

Pillow 9.0.0 introduced the conversion of subsequent GIF frames to `RGB` or `RGBA`. This behaviour can now be changed so that the first `P` frame is converted to `RGB` as well.

```
from PIL import GifImagePlugin
GifImagePlugin.LOADING_STRATEGY = GifImagePlugin.LoadingStrategy.RGB_ALWAYS
```

Or subsequent frames can be kept in P mode as long as there is only a single palette.

```
from PIL import GifImagePlugin
GifImagePlugin.LOADING_STRATEGY = GifImagePlugin.LoadingStrategy.RGB_AFTER_DIFFERENT_
↪PALETTE_ONLY
```

Other changes

musllinux wheels

Pillow now builds binary wheels for musllinux, suitable for Linux distributions based on the musl C standard library such as Alpine (rather than the glibc library used by manylinux wheels). See **PEP 656**.

ImageShow temporary files on Unix

When calling *show()* or using *ImageShow*, a temporary file is created from the image. On Unix, Pillow will no longer delete these files, and instead leave it to the operating system to do so.

Image._repr_pretty_

`im._repr_pretty_` has been added to provide a representation of an image without the identity of the object. This allows Jupyter to describe an image and have that description stay the same on subsequent executions of the same code.

Added BigTIFF reading

Support has been added for reading BigTIFF images.

Added BLP saving

Support has been added for saving BLP images. `blp_version` can be used to specify whether the image should be saved as BLP1 or BLP2, e.g. `im.save("out.blp", blp_version="BLP1")`. By default, BLP2 will be used.

1.6.23 9.0.1 (2022-02-03)

Security

This release addresses several security problems.

CVE 2022-24303: Temp image removal

If the path to the temporary directory on Linux or macOS contained a space, this would break removal of the temporary image file after `im.show()` (and related actions), and potentially remove an unrelated file. This has been present since PIL.

CVE 2022-22817: Restrict lambda expressions

While Pillow 9.0 restricted top-level builtins available to `PIL.ImageMath.eval()`, it did not prevent builtins available to lambda expressions. These are now also restricted.

Other changes

Pillow 9.0 added support for `xdg-open` as an image viewer, but there have been reports that the temporary image file was removed too quickly to be loaded into the final application. A delay has been added.

1.6.24 9.0.0 (2022-01-02)

Fredrik Lundh

This release is dedicated to the memory of Fredrik Lundh, aka Effbot, who died in November 2021. Fredrik created PIL in 1995 and he was instrumental in the early success of Python.

Guido wrote:

> Fredrik was an early Python contributor (e.g. Elementtree and the 're' module) and his enthusiasm for the language and community were inspiring for all who encountered him or his work. He spent countless hours on comp.lang.python answering questions from newbies and advanced users alike.
>
> He also co-founded an early Python startup, Secret Labs AB, which among other software released an IDE named PythonWorks. Fredrik also created the Python Imaging Library (PIL) which is still THE way to interact with images in Python, now most often through its Pillow fork. His effbot.org site was a valuable resource for generations of Python users, especially its Tkinter documentation.

Thank you, Fredrik.

Security

Ensure JpegImagePlugin stops at the end of a truncated file

`JpegImagePlugin` may append an EOF marker to the end of a truncated file, so that the last segment of the data will still be processed by the decoder.

If the EOF marker is not detected as such however, this could lead to an infinite loop where `JpegImagePlugin` keeps trying to end the file.

Remove consecutive duplicate tiles that only differ by their offset

To prevent attempts to slow down loading times for images, if an image has consecutive duplicate tiles that only differ by their offset, only load the last tile. Credit to Google's OSS-Fuzz project for finding this issue.

CVE 2022-22817: Restrict builtins available to ImageMath.eval

To limit *PIL.ImageMath* to working with images, Pillow will now restrict the builtins available to `PIL.ImageMath.eval()`. This will help prevent problems arising if users evaluate arbitrary expressions, such as `ImageMath.eval("exec(exit())")`.

CVE 2022-22815, CVE 2022-22816: ImagePath.Path array handling

CVE 2022-22815 (**CWE 126**) and **CVE 2022-22816** (**CWE 665**) were found when initializing `ImagePath.Path`.

Backwards incompatible changes

Python 3.6

Pillow has dropped support for Python 3.6, which reached end-of-life on 2021-12-23.

PILLOW_VERSION constant

`PILLOW_VERSION` has been removed. Use `__version__` instead.

FreeType 2.7

Support for FreeType 2.7 has been removed; FreeType 2.8 is the minimum supported.

We recommend upgrading to at least FreeType 2.10.4, which fixed a severe vulnerability introduced in FreeType 2.6 (**CVE 2020-15999**).

Image.show command parameter

The `command` parameter has been removed. Use a subclass of *PIL.ImageShow.Viewer* instead.

Image._showxv

`Image._showxv` has been removed. Use *show()* instead. If custom behaviour is required, use *register()* to add a custom *Viewer* class.

ImageFile.raise_ioerror

`IOError` was merged into `OSError` in Python 3.3. So, `ImageFile.raise_ioerror` has been removed. Use `ImageFile.raise_oserror` instead.

API changes

Added line width parameter to ImageDraw polygon

An optional line `width` parameter has been added to `ImageDraw.Draw.polygon`.

API additions

ImageShow.XDGViewer

If `xdg-open` is present on Linux, this new *PIL.ImageShow.Viewer* subclass will be registered. It displays images using the application selected by the system.

It is higher in priority than the other default *PIL.ImageShow.Viewer* instances, so it will be preferred by `im.show()` or *ImageShow.show()*.

Added support for "title" argument to DisplayViewer

Support has been added for the "title" argument in *DisplayViewer*, so that when `im.show()` or *ImageShow.show()* use the `display` command line tool, the "title" argument will also now be supported, e.g. `im.show(title="My Image")` and `ImageShow.show(im, title="My Image")`.

Other changes

Convert subsequent GIF frames to RGB or RGBA

Since each frame of a GIF can have up to 256 colors, after the first frame it is possible for there to be too many colors to fit in a P mode image. To allow for this, seeking to any subsequent GIF frame will now convert the image to RGB or RGBA, depending on whether or not the first frame had transparency.

Switched to libjpeg-turbo in macOS and Linux wheels

The Pillow wheels from PyPI for macOS and Linux have switched from libjpeg to libjpeg-turbo. It is a fork of libjpeg, popular for its speed.

Because different JPEG decoders load images differently, JPEG pixels may be altered slightly with this change.

Added support for pickling TrueType fonts

TrueType fonts may now be pickled and unpickled. For example:

```
import pickle
from PIL import ImageFont

font = ImageFont.truetype("arial.ttf", size=30)
pickled_font = pickle.dumps(font, protocol=pickle.HIGHEST_PROTOCOL)

# Later...
unpickled_font = pickle.loads(pickled_font)
```

Added support for additional TGA orientations

TGA images with top right or bottom right orientations are now supported.

1.6.25 8.4.0 (2021-10-15)

Deprecations

ImagePalette size parameter

The `size` parameter will be removed in Pillow 10.0.0 (2023-07-01).

Before Pillow 8.3.0, `ImagePalette` required palette data of particular lengths by default, and the size parameter could be used to override that. Pillow 8.3.0 removed the default required length, also removing the need for the size parameter.

API additions

Added "transparency" argument for loading EPS images

This new argument switches the Ghostscript device from "ppmraw" to "pngalpha", generating an RGBA image with a transparent background instead of an RGB image with a white background.

```
with Image.open("sample.eps") as im:
    im.load(transparency=True)
```

Added WalImageFile class

PIL.WalImageFile.open() previously returned a generic *PIL.Image.Image* instance. It now returns a dedicated *PIL.WalImageFile.WalImageFile* class.

Other changes

Speed improvement when rotating square images

Starting with Pillow 3.3.0, the speed of rotating images by 90 or 270 degrees was improved by quickly returning *transpose()* instead, if the rotate operation allowed for expansion and did not specify a center or post-rotate translation.

Since the `expand` flag makes no difference for square images though, Pillow now uses this faster method for square images without the `expand` flag as well.

1.6.26 8.3.2 (2021-09-02)

Security

CVE 2021-23437: Avoid potential ReDoS (regular expression denial of service)

Avoid a potential ReDoS (regular expression denial of service) in *ImageColor*'s *getrgb()* by raising ValueError if the color specifier is too long. Present since Pillow 5.2.0.

Fix 6-byte out-of-bounds (OOB) read

Fix 6-byte out-of-bounds (OOB) read. The previous bounds check in `FliDecode.c` incorrectly calculated the required read buffer size when copying a chunk, potentially reading six extra bytes off the end of the allocated buffer from the heap. Present since Pillow 7.1.0.

This bug was found by Google's OSS-Fuzz CIFuzz runs.

Other changes

Python 3.10 wheels

Pillow now includes binary wheels for Python 3.10.

The Python 3.10 release candidate was released on 2021-08-03 with the final release due 2021-10-04 (**PEP 619**). The CPython core team strongly encourages maintainers of third-party Python projects to prepare for 3.10 compatibility. And as there are no ABI changes planned we are releasing wheels to help others prepare for 3.10, and ensure Pillow can be used immediately on release day of 3.10.0 final.

Fixed regressions

- Ensure TIFF `RowsPerStrip` is multiple of 8 for JPEG compression (#5588).
- Updates for *ImagePalette* channel order (#5599).
- Hide FriBiDi shim symbols to avoid conflict with real FriBiDi library (#5651).

1.6.27 8.3.1 (2021-07-06)

Fixed regression converting to NumPy arrays

This fixes a regression introduced in 8.3.0 when converting an image to a NumPy array with a `dtype` argument.

```
>>> from PIL import Image
>>> import numpy
>>> im = Image.new("RGB", (100, 100))
>>> numpy.array(im, dtype=numpy.float64)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __array__() takes 1 positional argument but 2 were given
>>>
```

Catch OSError when checking if destination is sys.stdout

In 8.3.0, a check to see if the destination was `sys.stdout` when saving an image was updated. This lead to an OSError being raised if the environment restricted access.

The OSError is now silently caught.

Fixed removing orientation in ImageOps.exif_transpose

In 8.3.0, *exif_transpose()* was changed to ensure that the original image EXIF data was not modified, and the orientation was only removed from the modified copy.

However, for certain images the orientation was already missing from the modified image, leading to a KeyError.

This error has been resolved, and the copying of metadata to the modified image improved.

1.6.28 8.3.0 (2021-07-01)

Security

CVE 2021-34552: Fix buffer overflow

PIL since 1.1.4 and Pillow since 1.0 allowed parameters passed into a convert function to trigger buffer overflow in Convert.c.

Parsing XML

Pillow previously parsed XMP data using Python's xml module. However, this module is not secure.

- *getexif()* has used xml to potentially retrieve orientation data since Pillow 7.2.0. It has been refactored to use re instead.
- getxmp() was added to *JpegImageFile* in Pillow 8.2.0. It will now use defusedxml instead. If the dependency is not present, an empty dictionary will be returned and a warning raised.

Deprecations

JpegImagePlugin.convert_dict_qtables

JPEG quantization is now automatically converted, but still returned as a dictionary. The convert_dict_qtables method no longer performs any operations on the data given to it, has been deprecated and will be removed in Pillow 10.0.0 (2023-07-01).

API changes

Changed WebP default "method" value when saving

Previously, it was 0, for the best speed. The default has now been changed to 4, to match WebP's default, for higher quality with still some speed optimisation.

Default resampling filter for special image modes

Pillow 7.0 changed the default resampling filter to Image.BICUBIC. However, as this is not supported yet for images with a custom number of bits, the default filter for those modes has been reverted to Image.NEAREST.

ImageMorph incorrect mode errors

For apply(), match() and get_on_pixels(), if the image mode is not L, an Exception was thrown. This has now been changed to a ValueError.

getxmp()

XMP data can now be returned for PNG and TIFF images, through getxmp() for each format.

The returned dictionary will start from the base of the XML, meaning that the top level should contain an "xmpmeta" key. JPEG's getxmp() method has also been updated to this structure.

TIFF getexif()

TIFF *tag_v2* data can now be accessed through *getexif()*. This also provides access to the GPS and EXIF IFDs, through `im.getexif().get_ifd(0x8825)` and `im.getexif().get_ifd(0x8769)` respectively.

API additions

ImageOps.contain

Returns a resized version of the image, set to the maximum width and height within `size`, while maintaining the original aspect ratio.

To compare it to other ImageOps methods:

- *fit()* expands an image until is fills `size`, cropping the parts of the image that do not fit.
- *pad()* expands an image to fill `size`, without cropping, but instead filling the extra space with `color`.
- *contain()* is similar to *pad()*, but it does not fill the extra space. Instead, the original aspect ratio is maintained. So unlike the other two methods, it is not guaranteed to return an image of `size`.

ICO saving: bitmap_format argument

By default, Pillow saves ICO files in the PNG format. They can now also be saved in BMP format, through the new `bitmap_format` argument:

```
im.save("out.ico", bitmap_format="bmp")
```

Other changes

Added DDS BC5 reading and uncompressed saving

Support has been added to read the BC5 format of DDS images, whether UNORM, SNORM or TYPELESS.

Support has also been added to write the uncompressed format of DDS images.

1.6.29 8.2.0 (2021-04-01)

Security

These issues were all found with OSS-Fuzz.

CVE 2021-25287, CVE 2021-25288: OOB read in Jpeg2KDecode

- For J2k images with multiple bands, it's legal to have different widths for each band, e.g. 1 byte for `L`, 4 bytes for `A`.
- This dates to Pillow 2.4.0.

CVE 2021-28675: DOS attack in PsdImagePlugin

- *PsdImagePlugin.PsdImageFile* did not sanity check the number of input layers with regard to the size of the data block, this could lead to a denial-of-service on *open()* prior to *load()*.
- This dates to the PIL fork.

CVE 2021-28676: FLI image DOS attack

- `FliDecode.c` did not properly check that the block advance was non-zero, potentially leading to an infinite loop on load.
- This dates to the PIL fork.

CVE 2021-28677: EPS DOS on _open

- The readline used in EPS has to deal with any combination of `\r` and `\n` as line endings. It accidentally used a quadratic method of accumulating lines while looking for a line ending.
- A malicious EPS file could use this to perform a denial-of-service of Pillow in the open phase, before an image was accepted for opening.
- This dates to the PIL fork.

CVE 2021-28678: BLP DOS attack

- `BlpImagePlugin` did not properly check that reads after jumping to file offsets returned data. This could lead to a denial-of-service where the decoder could be run a large number of times on empty data.
- This dates to Pillow 5.1.0.

Fix memory DOS in ImageFont

- A corrupt or specially crafted TTF font could have font metrics that lead to unreasonably large sizes when rendering text in font. `ImageFont.py` did not check the image size before allocating memory for it.
- This dates to the PIL fork.

Deprecations

Categories

`im.category` is deprecated and will be removed in Pillow 10.0.0 (2023-07-01), along with the related `Image.NORMAL`, `Image.SEQUENCE` and `Image.CONTAINER` attributes.

To determine if an image has multiple frames or not, `getattr(im, "is_animated", False)` can be used instead.

Tk/Tcl 8.4

Support for Tk/Tcl 8.4 is deprecated and will be removed in Pillow 10.0.0 (2023-07-01), when Tk/Tcl 8.5 will be the minimum supported.

API changes

Image.alpha_composite: dest

When calling *alpha_composite()*, the `dest` argument now accepts negative coordinates, like the upper left corner of the `box` argument of *paste()* can be negative. Naturally, this has effect of cropping the overlaid image.

Image.getexif: EXIF and GPS IFD

Previously, *getexif()* flattened the EXIF IFD into the rest of the data, losing information. This information is now kept separate, moved under `im.getexif().get_ifd(0x8769)`.

Direct access to the GPS IFD dictionary was possible through `im.getexif()[0x8825]`. This is now consistent with other IFDs, and must be accessed through `im.getexif().get_ifd(0x8825)`.

These changes only affect *getexif()*, introduced in Pillow 6.0. The older `_getexif()` methods are unaffected.

Image._MODEINFO

This internal dictionary had been deprecated by a comment since PIL, and is now removed. Instead, `Image.getmodebase()`, `Image.getmodetype()`, `Image.getmodebandnames()`, `Image.getmodebands()` or `ImageMode.getmode()` can be used.

API additions

getxmp() for JPEG images

A new method has been added to return XMP data for JPEG images. It reads the XML data into a dictionary of names and values.

For example:

```
>>> from PIL import Image
>>> with Image.open("Tests/images/xmp_test.jpg") as im:
>>>     print(im.getxmp())
{'RDF': {}, 'Description': {'Version': '10.4', 'ProcessVersion': '10.0', ...}, ...}
```

ImageDraw.rounded_rectangle

Added *rounded_rectangle()*. It works the same as *rectangle()*, except with an additional `radius` argument. `radius` is limited to half of the width or the height, so that users can create a circle, but not any other ellipse.

```
from PIL import Image, ImageDraw
im = Image.new("RGB", (200, 200))
draw = ImageDraw.Draw(im)
draw.rounded_rectangle(xy=(10, 20, 190, 180), radius=30, fill="red")
```

ImageOps.autocontrast: preserve_tone

The default behaviour of *autocontrast()* is to normalize separate histograms for each color channel, changing the tone of the image. The new `preserve_tone` argument keeps the tone unchanged by using one luminance histogram for all channels.

ImageShow.GmDisplayViewer

If GraphicsMagick is present, this new *PIL.ImageShow.Viewer* subclass will be registered. It uses GraphicsMagick, an ImageMagick fork, to display images.

The GraphicsMagick based viewer has a lower priority than its ImageMagick counterpart. Thus, if both ImageMagick and GraphicsMagick are installed, `im.show()` and *ImageShow.show()* prefer the viewer based on ImageMagick, i.e the behaviour stays the same for Pillow users having ImageMagick installed.

ImageShow.IPythonViewer

If IPython is present, this new *PIL.ImageShow.Viewer* subclass will be registered. It displays images on all IPython frontends. This will be helpful to users of Google Colab, allowing `im.show()` to display images.

It is lower in priority than the other default *PIL.ImageShow.Viewer* instances, so it will only be used by `im.show()` or *ImageShow.show()* if none of the other viewers are available. This means that the behaviour of *PIL.ImageShow* will stay the same for most Pillow users.

Saving TIFF with ICC profile

As is already possible for JPEG, PNG and WebP, the ICC profile for TIFF files can now be specified through a keyword argument:

```
im.save("out.tif", icc_profile=...)
```

Other changes

GIF writer uses LZW encoding

GIF files are now written using LZW encoding, which will generate smaller files, typically about 70% of the size generated by the older encoder.

The pixel data is encoded using the format specified in the CompuServe GIF standard.

The older encoder used a variant of run-length encoding that was compatible but less efficient.

GraphicsMagick

The test suite can now be run on systems which have GraphicsMagick but not ImageMagick installed. If both are installed, the tests prefer ImageMagick.

Libraqm and FriBiDi linking

The way the libraqm dependency for complex text scripts is linked has been changed:

Source builds will now link against the system version of libraqm at build time rather than at runtime by default.

Binary wheels now include a statically linked modified version of libraqm that links against FriBiDi at runtime instead. This change is intended to address issues with the previous implementation on some platforms. These are created by building Pillow with the new build flags `--vendor-raqm --vendor-fribidi`.

Windows users will now need to install `fribidi.dll` (or `fribidi-0.dll`) only, `libraqm.dll` is no longer used.

See installation documentation for more information.

PyQt6

Support has been added for PyQt6. If it is installed, it will be used instead of PySide6, PyQt5 or PySide2.

1.6.30 8.1.2 (2021-03-06)

Security

CVE 2021-27921, CVE 2021-27922, CVE 2021-27923: Fix DOS attacks

There is an exhaustion of memory DOS attack in BLP, ICNS, ICO images where Pillow did not properly check the reported size of the contained image. These images could cause arbitrarily large memory allocations.

These issues were reported by Jiayi Lin, Luke Shaffer, Xinran Xie and Akshay Ajayan of Arizona State University.

1.6.31 8.1.1 (2021-03-01)

Security

CVE 2021-25289: Correct the fix for CVE 2020-35654

The previous fix for CVE 2020-35654 was insufficient due to incorrect error checking in `TiffDecode.c`.

CVE 2021-25290: Fix buffer overflow in `TiffDecode.c`

In `TiffDecode.c`, there is a negative-offset `memcpy` with an invalid size.

CVE 2021-25291: Fix buffer overflow in `TIFFReadRGBATile`

In `TiffDecode.c`, invalid tile boundaries could lead to an out-of-bounds read in `TIFFReadRGBATile`.

CVE 2021-25292: Fix DOS attack

The PDF parser has a catastrophic backtracking regex that could be used as a DOS attack.

CVE 2021-25293: Fix buffer overflow in `SgiRleDecode.c`

There is an out-of-bounds read in `SgiRleDecode.c` since Pillow 4.3.0.

Other changes

A crash with the feature flags for libimagequant, libjpeg-turbo, WebP and XCB on unreleased Python 3.10 has been fixed (#5193).

1.6.32 8.1.0 (2021-01-02)

Security

This release includes security fixes.

- An out-of-bounds read when saving TIFFs with custom metadata through LibTIFF
- An out-of-bounds read when saving a GIF of 1px width

CVE 2020-35653: Buffer read overrun in PCX decoding

The PCX image decoder used the reported image stride to calculate the row buffer, rather than calculating it from the image size. This issue dates back to the PIL fork. Thanks to Google's OSS-Fuzz project for finding this.

CVE 2020-35654: TIFF out-of-bounds write error

Out-of-bounds write in `TiffDecode.c` when reading corrupt YCbCr files in some LibTIFF versions (4.1.0/Ubuntu 20.04, but not 4.0.9/Ubuntu 18.04). In some cases LibTIFF's interpretation of the file is different when reading in RGBA mode, leading to an out-of-bounds write in `TiffDecode.c`. This potentially affects Pillow versions from 6.0.0 to 8.0.1, depending on the version of LibTIFF. This was reported through Tidelift.

CVE 2020-35655: SGI decode buffer overrun

4 byte read overflow in `SgiRleDecode.c`, where the code was not correctly checking the offsets and length tables. Independently reported through Tidelift and Google's OSS-Fuzz. This vulnerability covers Pillow versions 4.3.0->8.0.1.

Dependencies

OpenJPEG in the macOS and Linux wheels has been updated from 2.3.1 to 2.4.0, including security fixes.

LibTIFF in the macOS and Linux wheels has been updated from 4.1.0 to 4.2.0, including security fixes discovered by fuzzers.

Deprecations

FreeType 2.7

Support for FreeType 2.7 is deprecated and will be removed in Pillow 9.0.0 (2022-01-02), when FreeType 2.8 will be the minimum supported.

We recommend upgrading to at least FreeType 2.10.4, which fixed a severe vulnerability introduced in FreeType 2.6 (**CVE 2020-15999**).

Makefile

The `install-venv` target has been deprecated.

API additions

Append images to ICO

When saving an ICO image, the file may contain versions of the image at different sizes. By default, Pillow will scale down the main image to create these copies.

With this release, a list of images can be provided to the `append_images` parameter when saving, to replace the scaled down versions. This is the same functionality that already exists for the ICNS format.

Other changes

Makefile

The `co` target has been removed.

PyPy wheels

Wheels have been added for PyPy 3.7.

PySide6

Support has been added for PySide6. If it is installed, it will be used instead of PyQt5 or PySide2, since it is based on a newer Qt.

1.6.33 8.0.1 (2020-10-22)

Security

CVE 2020-15999: Update FreeType in wheels to 2.10.4

- A heap buffer overflow has been found in the handling of embedded PNG bitmaps, introduced in FreeType version 2.6.
- If you use option `FT_CONFIG_OPTION_USE_PNG` you should upgrade immediately.

We strongly recommend updating to Pillow 8.0.1 if you are using Pillow 8.0.0, which improved support for bitmap fonts.

In Pillow 7.2.0 and earlier bitmap fonts were disabled with `FT_LOAD_NO_BITMAP`, but it is not clear if this prevents the exploit and we recommend updating to Pillow 8.0.1.

Pillow 8.0.0 and earlier are potentially vulnerable releases, including the last release to support Python 2.7, namely Pillow 6.2.2.

1.6.34 8.0.0 (2020-10-14)

Backwards incompatible changes

Python 3.5

Pillow has dropped support for Python 3.5, which reached end-of-life on 2020-09-13.

PyPy 7.1.x

Pillow has dropped support for PyPy3 7.1.1. PyPy3 7.2.0, released on 2019-10-14, is now the minimum compatible version.

im.offset

`im.offset()` has been removed, call *ImageChops.offset()* instead.

Image.fromstring, im.fromstring and im.tostring

- `Image.fromstring()` has been removed, call *Image.frombytes()* instead.
- `im.fromstring()` has been removed, call *frombytes()* instead.
- `im.tostring()` has been removed, call *tobytes()* instead.

ImageCms.CmsProfile attributes

Some attributes in *PIL.ImageCms.core.CmsProfile* have been removed:

Removed	Use instead
`color_space`	Padded *xcolor_space*
`pcs`	Padded *connection_space*
`product_copyright`	Unicode *copyright*
`product_desc`	Unicode *profile_description*
`product_description`	Unicode *profile_description*
`product_manufacturer`	Unicode *manufacturer*
`product_model`	Unicode *model*

API changes

ImageDraw.text: stroke_width

Fixed issue where passing `stroke_width` with a non-zero value to *ImageDraw.text()* would cause the text to be offset by that amount.

ImageDraw.text: anchor

The `anchor` parameter of *ImageDraw.text()* has been implemented.

Use this parameter to change the position of text relative to the specified `xy` point. See *Text anchors* for details.

Add MIME type to PsdImagePlugin

"image/vnd.adobe.photoshop" is now registered as the *PsdImagePlugin.PsdImageFile* MIME type.

API additions

Image.open: add formats parameter

Added a new `formats` parameter to *Image.open()*:

- A list or tuple of formats to attempt to load the file in. This can be used to restrict the set of formats checked. Pass `None` to try all supported formats. You can print the set of available formats by running `python3 -m PIL` or using the *PIL.features.pilinfo()* function.

ImageOps.autocontrast: add mask parameter

ImageOps.autocontrast() can now take a `mask` parameter:

- Histogram used in contrast operation is computed using pixels within the mask. If no mask is given the entire image is used for histogram computation.

ImageOps.autocontrast cutoffs

Previously, the `cutoff` parameter of *ImageOps.autocontrast()* could only be a single number, used as the percent to cut off from the histogram on the low and high ends.

Now, it can also be a tuple `(low, high)`.

ImageDraw.regular_polygon

A new method *ImageDraw.regular_polygon()*, draws a regular polygon of `n_sides`, inscribed in a `bounding_circle`.

For example `draw.regular_polygon(((100, 100), 50), 5)` draws a pentagon centered at the point `(100, 100)` with a polygon radius of `50`.

ImageDraw.text: embedded_color

The methods *ImageDraw.text()* and *ImageDraw.multiline_text()* now support fonts with embedded color data.

To render text with embedded color data, use the parameter `embedded_color=True`.

Support for CBDT fonts requires FreeType 2.5 compiled with libpng. Support for SBIX fonts requires FreeType 2.5.1 compiled with libpng. Support for COLR fonts requires FreeType 2.10. SVG fonts are not yet supported.

ImageDraw.textlength

Two new methods *ImageDraw.textlength()* and *FreeTypeFont.getlength()* were added, returning the exact advance length of text with 1/64 pixel precision.

These can be used for word-wrapping or rendering text in parts.

ImageDraw.textbbox

Three new methods *ImageDraw.textbbox()*, *ImageDraw.multiline_textbbox()*, and *FreeTypeFont.getbbox()* return the bounding box of rendered text.

These functions accept an `anchor` parameter, see *Text anchors* for details.

Other changes

Improved ellipse-drawing algorithm

The ellipse-drawing algorithm has been changed from drawing a 360-sided polygon to one which resembles Bresenham's algorithm for circles. It should be faster and produce smoother curves, especially for smaller ellipses.

ImageDraw.text and ImageDraw.multiline_text

Fixed multiple issues in methods *ImageDraw.text()* and *ImageDraw.multiline_text()* sometimes causing unexpected text alignment issues.

The `align` parameter of *ImageDraw.multiline_text()* now gives better results in some cases.

TrueType fonts with embedded bitmaps are now supported.

Added writing of subIFDs

When saving EXIF data, Pillow is now able to write subIFDs, such as the GPS IFD. This should happen automatically when saving an image using the EXIF data that it was opened with, such as in *exif_transpose()*.

Previously, the code of the first tag of the subIFD was incorrectly written as the offset.

Error for large BMP files

Previously, if a BMP file was too large, an `OSError` would be raised. Now, `DecompressionBombError` is used instead, as Pillow already uses for other formats.

Dark theme for docs

The https://pillow.readthedocs.io documentation will use a dark theme if the user has requested the system use one. Uses the `prefers-color-scheme` CSS media query.

1.6.35 7.2.0 (2020-06-30)

API changes

Replaced TiffImagePlugin DEBUG with logging

`TiffImagePlugin.DEBUG = True` has been a way to print various debugging information when interacting with TIFF images. This has now been removed in favour of Python's `logging` module, already used in other places in the Pillow source code.

Corrected default offset when writing EXIF data

Previously, the default `offset` argument for *tobytes()* was 0, which did not include the magic header. It is now 8.

Moved to ImageFileDirectory_v2 in Image.Exif

Moved from the legacy *PIL.TiffImagePlugin.ImageFileDirectory_v1* to *PIL.TiffImagePlugin.ImageFileDirectory_v2* in *PIL.Image.Exif*. This means that Exif RATIONALs and SIGNED_RATIONALs are now read as *PIL.TiffImagePlugin.IFDRational*, instead of as a tuple with a numerator and a denominator.

TIFF BYTE tags format

TIFF BYTE tags were previously read as a tuple containing a bytestring. They are now read as just a single bytestring.

Deprecations

Image.show command parameter

The `command` parameter was deprecated and will be removed in a future release. Use a subclass of *PIL.ImageShow.Viewer* instead.

Image._showxv

`Image._showxv` has been deprecated. Use *show()* instead. If custom behaviour is required, use *register()* to add a custom *Viewer* class.

ImageFile.raise_ioerror

`IOError` was merged into `OSError` in Python 3.3. So, `ImageFile.raise_ioerror` is now deprecated and will be removed in a future release. Use `ImageFile.raise_oserror` instead.

1.6.36 7.1.2 (2020-04-25)

Fix another regression seeking PNG files

This fixes a regression introduced in 7.1.0 when adding support for APNG files.

When calling `seek(n)` on a regular PNG where `n > 0`, it failed to raise an `EOFError` as it should have done, resulting in:

```
AttributeError: 'NoneType' object has no attribute 'read'
```

Pillow 7.1.2 now raises the correct exception.

1.6.37 7.1.1 (2020-04-02)

Fix regression seeking PNG files

This fixes a regression introduced in 7.1.0 when adding support for APNG files when calling `seek` and `tell`:

```
>>> from PIL import Image
>>> with Image.open("Tests/images/hopper.png") as im:
...     im.seek(0)
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
↪PIL/PngImagePlugin.py", line 739, in seek
    if not self._seek_check(frame):
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
↪PIL/ImageFile.py", line 306, in _seek_check
    return self.tell() != frame
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
↪PIL/PngImagePlugin.py", line 827, in tell
    return self.__frame
```

(continues on next page)

(continued from previous page)

```
AttributeError: 'PngImageFile' object has no attribute '_PngImageFile__frame'
>>>
```

1.6.38 7.1.0 (2020-04-01)

Security

This release includes many security fixes.

CVE 2020-10177: Multiple out-of-bounds reads in FLI decoding

Pillow before 7.1.0 has multiple out-of-bounds reads in `libImaging/FliDecode.c`.

CVE 2020-10378: Bounds overflow in PCX decoding

In `libImaging/PcxDecode.c` in Pillow before 7.1.0, an out-of-bounds read can occur when reading PCX files where `state->shuffle` is instructed to read beyond `state->buffer`.

CVE 2020-10379: Two buffer overflows in TIFF decoding

In Pillow before 7.1.0, there are two buffer overflows in `libImaging/TiffDecode.c`.

CVE 2020-10994: Bounds overflow in JPEG 2000 decoding

In `libImaging/Jpeg2KDecode.c` in Pillow before 7.1.0, there are multiple out-of-bounds reads via a crafted JP2 file.

CVE 2020-11538: Buffer overflow in SGI-RLE decoding

In `libImaging/SgiRleDecode.c` in Pillow through 7.0.0, a number of out-of-bounds reads exist in the parsing of SGI image files, a different issue than **CVE 2020-5311**.

API changes

Allow saving of zero quality JPEG images

If no quality was specified when saving a JPEG, Pillow internally used a value of zero to indicate that the default quality should be used. However, this removed the ability to actually save a JPEG with zero quality. This has now been resolved.

```
from PIL import Image
im = Image.open("hopper.jpg")
im.save("out.jpg", quality=0)
```

API additions

New channel operations

Three new channel operations have been added: *soft_light()*, *hard_light()* and *overlay()*.

PILLOW_VERSION constant

`PILLOW_VERSION` has been re-added but is deprecated and will be removed in a future release. Use `__version__` instead.

It was initially removed in Pillow 7.0.0, but brought back in 7.1.0 to give projects more time to upgrade.

Reading JPEG comments

When opening a JPEG image, the comment may now be read into *info*.

Support for different charset encodings in PcfFontFile

Previously `PcfFontFile` output only bitmap PIL fonts with ISO 8859-1 encoding, even though the PCF format supports Unicode, making it hard to work with Pillow with bitmap fonts in languages which use different character sets.

Now it's possible to set a different charset encoding in `PcfFontFile`'s class constructor. By default, it generates a PIL font file with ISO 8859-1 as before. The generated PIL font file still contains up to 256 characters, but the character set is different depending on the selected encoding.

To use such a font with `ImageDraw.text`, call it with a bytes object with the same encoding as the font file.

X11 ImageGrab.grab()

Support has been added for `ImageGrab.grab()` on Linux using the X server with the XCB library.

An optional `xdisplay` parameter has been added to select the X server, with the default value of `None` using the default X server.

Passing a different value on Windows or macOS will force taking a snapshot using the selected X server; pass an empty string to use the default X server. XCB support is not included in pre-compiled wheels for Windows and macOS.

Other changes

If present, only use alpha channel for bounding box

When the *getbbox()* method calculates the bounding box, for an RGB image it trims black pixels. Similarly, for an RGBA image it would trim black transparent pixels. This is now changed so that if an image has an alpha channel (RGBA, RGBa, PA, LA, La), any transparent pixels are trimmed.

Improved APNG support

Added support for reading and writing Animated Portable Network Graphics (APNG) images. The PNG plugin now supports using the *seek()* method and the *Iterator* class to read APNG frame sequences. The PNG plugin also now supports using the `append_images` argument to write APNG frame sequences. See *APNG sequences* for further details.

1.6.39 7.0.0 (2020-01-02)

Backwards incompatible changes

Python 2.7

Pillow has dropped support for Python 2.7, which reached end-of-life on 2020-01-01.

PILLOW_VERSION constant

`PILLOW_VERSION` has been removed. Use `__version__` instead.

PIL.*ImagePlugin.__version__ attributes

The version constants of individual plugins have been removed. Use `PIL.__version__` instead.

Removed	Removed	Removed
`BmpImagePlugin.__version__`	`Jpeg2KImagePlugin.__version__`	`PngImagePlugin.__version__`
`CurImagePlugin.__version__`	`JpegImagePlugin.__version__`	`PpmImagePlugin.__version__`
`DcxImagePlugin.__version__`	`McIdasImagePlugin.__version__`	`PsdImagePlugin.__version__`
`EpsImagePlugin.__version__`	`MicImagePlugin.__version__`	`SgiImagePlugin.__version__`
`FliImagePlugin.__version__`	`MpegImagePlugin.__version__`	`SunImagePlugin.__version__`
`FpxImagePlugin.__version__`	`MpoImagePlugin.__version__`	`TgaImagePlugin.__version__`
`GdImageFile.__version__`	`MspImagePlugin.__version__`	`TiffImagePlugin.__version__`
`GifImagePlugin.__version__`	`PalmImagePlugin.__version__`	`WmfImagePlugin.__version__`
`IcoImagePlugin.__version__`	`PcdImagePlugin.__version__`	`XbmImagePlugin.__version__`
`ImImagePlugin.__version__`	`PcxImagePlugin.__version__`	`XpmImagePlugin.__version__`
`ImtImagePlugin.__version__`	`PdfImagePlugin.__version__`	`XVThumbImagePlugin.__version__`
`IptcImagePlugin.__version__`	`PixarImagePlugin.__version__`	

PyQt4 and PySide

Qt 4 reached end-of-life on 2015-12-19. Its Python bindings are also EOL: PyQt4 since 2018-08-31 and PySide since 2015-10-14.

Support for PyQt4 and PySide has been removed from `ImageQt`. Please upgrade to PyQt5 or PySide2.

Setting the size of TIFF images

Setting the size of a TIFF image directly (eg. `im.size = (256, 256)`) throws an error. Use `Image.resize` instead.

Default resampling filter

The default resampling filter has been changed to the high-quality convolution `Image.BICUBIC` instead of `Image.NEAREST`, for the *`resize()`* method and the *`pad()`*, *`scale()`* and *`fit()`* functions. `Image.NEAREST` is still always used for images in "P" and "1" modes. See *Filters* to learn the difference. In short, `Image.NEAREST` is a very fast filter, but simple and low-quality.

Image.draft() return value

If the *draft()* method has no effect, it returns None. If it does have an effect, then it previously returned the image itself. However, unlike other chain methods, *draft()* does not return a modified version of the image, but modifies it in-place. So instead, if *draft()* has an effect, Pillow will now return a tuple of the image mode and a co-ordinate box. The box is the original coordinates in the bounds of resulting image. This may be useful in a subsequent *resize()* call.

API additions

Custom unidentified image error

Pillow will now throw a custom `UnidentifiedImageError` when an image cannot be identified. For backwards compatibility, this will inherit from OSError.

New argument `reducing_gap` for Image.resize() and Image.thumbnail() methods

Speeds up resizing by resizing the image in two steps. The bigger `reducing_gap`, the closer the result to the fair resampling. The smaller `reducing_gap`, the faster resizing. With `reducing_gap` greater or equal to 3.0, the result is indistinguishable from fair resampling.

The default value for *resize()* is None, which means that the optimization is turned off by default.

The default value for *thumbnail()* is 2.0, which is very close to fair resampling while still being faster in many cases. In addition, the same gap is applied when *thumbnail()* calls *draft()*, which may greatly improve the quality of JPEG thumbnails. As a result, *thumbnail()* in the new version provides equally high speed and high quality from any source (JPEG or arbitrary images).

New Image.reduce() method

reduce() is a highly efficient operation to reduce an image by integer times. Normally, it shouldn't be used directly. Used internally by *resize()* and *thumbnail()* methods to speed up resize when a new argument `reducing_gap` is set.

Loading WMF images at a given DPI

On Windows, Pillow can read WMF files, with a default DPI of 72. An image can now also be loaded at another resolution:

```
from PIL import Image
with Image.open("drawing.wmf") as im:
    im.load(dpi=144)
```

Other changes

Image.__del__

Implicitly closing the image's underlying file in `Image.__del__` has been removed. Use a context manager or call *close()* instead to close the file in a deterministic way.

Previous method:

```
im = Image.open("hopper.png")
im.save("out.jpg")
```

Use instead:

```
with Image.open("hopper.png") as im:
    im.save("out.jpg")
```

Better thumbnail geometry

When calculating the new dimensions in *thumbnail()*, round to the nearest integer, instead of always rounding down. This better preserves the original aspect ratio.

When the image width or height is not divisible by 8 the last row and column in the image get the correct weight after JPEG DCT scaling.

1.6.40 6.2.2 (2020-01-02)

Security

This release fixes several buffer overflow issues and a DOS attack vulnerability.

CVE 2020-5310, CVE 2020-5311, CVE 2020-5312, CVE 2020-5313: Overflow checks added

Overflow checks have been added when calculating the size of a memory block to be reallocated in the processing of TIFF, SGI, PCX and FLI images.

CVE 2019-19911: DOS attack vulnerability

If an FPX image reports that it has a large number of bands, a large amount of resources will be used when trying to process the image. This is fixed by limiting the number of bands to those usable by Pillow.

1.6.41 6.2.1 (2019-10-20)

API changes

Deprecations

Python 2.7

Python 2.7 reaches end-of-life on 2020-01-01.

Pillow 7.0.0 will be released on 2020-01-01 and will drop support for Python 2.7, making Pillow 6.2.x the last release series to support Python 2.

Other changes

Support added for Python 3.8

Pillow 6.2.1 supports Python 3.8.

1.6.42 6.2.0 (2019-10-01)

Security

This release catches several buffer overruns and fixes CVE 2019-16865.

Buffer overruns

In `RawDecode.c`, an error is now thrown if skip is calculated to be less than zero. It is intended to skip padding between lines, not to go backwards.

In `PsdImagePlugin`, if the combined sizes of the individual parts is larger than the declared size of the extra data field, then it looked for the next layer by seeking backwards. This is now corrected by seeking to (the start of the layer + the size of the extra data field) instead of (the read parts of the layer + the rest of the layer).

Decompression bomb checks have been added to GIF and ICO formats.

An error is now raised if a TIFF dimension is a string, rather than trying to perform operations on it.

CVE 2019-16865: Fix DOS attack

The CVE is regarding DOS problems, such as consuming large amounts of memory, or taking a large amount of time to process an image.

API changes

Image.getexif

To allow for lazy loading of Exif data, `Image.getexif()` now returns a shared instance of `Image.Exif`.

Deprecations

Image.frombuffer

There has been a longstanding warning that the defaults of `Image.frombuffer` may change in the future for the "raw" decoder. The change will now take place in Pillow 7.0.

API additions

Text stroking

`stroke_width` and `stroke_fill` arguments have been added to text drawing operations. They allow text to be outlined, setting the width of the stroke and and the color respectively. If not provided, `stroke_fill` will default to the `fill` parameter.

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf", 40)
font.getsize_multiline("A", stroke_width=2)
font.getsize("ABC\nAaaa", stroke_width=2)

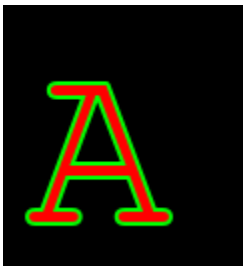
im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
draw.textsize("A", font, stroke_width=2)
draw.multiline_textsize("ABC\nAaaa", font, stroke_width=2)
draw.text((10, 10), "A", "#f00", font, stroke_width=2, stroke_fill="#0f0")
draw.multiline_text((10, 10), "A\nB", "#f00", font,
                    stroke_width=2, stroke_fill="#0f0")
```

For example,

```
from PIL import Image, ImageDraw, ImageFont

im = Image.new("RGB", (120, 130))
draw = ImageDraw.Draw(im)
font = ImageFont.truetype("Tests/fonts/FreeMono.ttf", 120)
draw.text((10, 10), "A", "#f00", font, stroke_width=2, stroke_fill="#0f0")
```

creates the following image:



ImageGrab on multi-monitor Windows

An `all_screens` argument has been added to `ImageGrab.grab`. If `True`, all monitors will be included in the created image.

Other changes

Removed bdist_wininst .exe installers

.exe installers fell out of favour with PEP 527, and will be deprecated in Python 3.8. Pillow will no longer be distributing them. Wheels should be used instead.

Flags for libwebp in wheels

When building libwebp for inclusion in wheels, Pillow now adds the `-O3` and `-DNDEBUG` CFLAGS. These flags would be used by default if building libwebp without debugging, and using them fixes a significant decrease in speed when a wheel-installed copy of Pillow performs libwebp operations.

1.6.43 6.1.0 (2019-07-02)

Deprecations

Image.__del__

Deprecated since version 6.1.0.

Implicitly closing the image's underlying file in `Image.__del__` has been deprecated. Use a context manager or call `Image.close()` instead to close the file in a deterministic way.

Deprecated:

```
im = Image.open("hopper.png")
im.save("out.jpg")
```

Use instead:

```
with Image.open("hopper.png") as im:
    im.save("out.jpg")
```

API additions

Image.entropy

Calculates and returns the entropy for the image. A bilevel image (mode "1") is treated as a grayscale ("L") image by this method. If a mask is provided, the method employs the histogram for those parts of the image where the mask image is non-zero. The mask image must have the same size as the image, and be either a bi-level image (mode "1") or a grayscale image ("L").

ImageGrab.grab

An optional `include_layered_windows` parameter has been added to `ImageGrab.grab`, defaulting to `False`. If true, layered windows will be included in the resulting image on Windows.

ImageSequence.all_frames

A new method to facilitate applying a given function to all frames in an image, or to all frames in a list of images. The frames are returned as a list of separate images. For example, `ImageSequence.all_frames(im, lambda im_frame: im_frame.rotate(90))` could be used to return all frames from an image, each rotated 90 degrees.

Variation fonts

Variation fonts are now supported, allowing for different styles from the same font file. `ImageFont.FreeTypeFont` has four new methods, *PIL.ImageFont.FreeTypeFont.get_variation_names()* and *PIL.ImageFont.FreeTypeFont.set_variation_by_name()* for using named styles, and *PIL.ImageFont.FreeTypeFont.get_variation_axes()* and *PIL.ImageFont.FreeTypeFont.set_variation_by_axes()* for using font axes instead. An `IOError` will be raised if the font is not a variation font. FreeType 2.9.1 or greater is required.

Other changes

ImageTk.getimage

This function is now supported. It returns the contents of an `ImageTk.PhotoImage` as an RGBA `Image.Image` instance.

Image quality for JPEG compressed TIFF

The TIFF encoder accepts a `quality` parameter for `jpeg` compressed TIFF files. A value from 0 (worst) to 100 (best) controls the image quality, similar to the JPEG encoder. The default is 75. For example:

```
im.save("out.tif", compression="jpeg", quality=85)
```

Improve encoding of TIFF tags

The TIFF encoder supports more types, especially arrays. This is required for the GeoTIFF format which encodes geospatial information.

- Pass `tagtype` from v2 directory to libtiff encoder, instead of autodetecting type.
- Use explicit types eg. `uint32_t` for `TIFF_LONG` to fix issues on platforms with 64-bit longs.
- Add support for multiple values (arrays). Requires type in v2 directory and values must be passed as a tuple.
- Add support for signed types eg. `TIFFTypes.TIFF_SIGNED_SHORT`.

Respect PKG_CONFIG environment variable when building

This variable is commonly used by other build systems and using it can help with cross-compiling. Falls back to `pkg-config` as before.

Top-to-bottom complex text rendering

Drawing text in the 'ttb' direction with `ImageFont` has been significantly improved and requires Raqm 0.7 or greater.

1.6.44 6.0.0 (2019-04-02)

Backwards incompatible changes

Python 3.4 dropped

Python 3.4 is EOL since 2019-03-16 and no longer supported. We will not be creating binaries, testing, or retaining compatibility with this version. The final version of Pillow for Python 3.4 is 5.4.1.

Removed deprecated PIL.OleFileIO

`PIL.OleFileIO` was removed as a vendored file and in Pillow 4.0.0 (2017-01) in favour of the upstream olefile Python package, and replaced with an `ImportError`. The deprecated file has now been removed from Pillow. If needed, install from PyPI (eg. `python3 -m pip install olefile`).

Removed deprecated ImageOps functions

Several undocumented functions in `ImageOps` were deprecated in Pillow 4.3.0 (2017-10) and have now been removed: `gaussian_blur`, `gblur`, `unsharp_mask`, `usm` and `box_blur`. Use the equivalent operations in `ImageFilter` instead.

Removed deprecated VERSION

`VERSION` (the old PIL version, always 1.1.7) has been removed. Use `__version__` instead.

API changes

Deprecations

Python 2.7

Python 2.7 reaches end-of-life on 2020-01-01.

Pillow 7.0.0 will be released on 2020-01-01 and will drop support for Python 2.7, making Pillow 6.x the last series to support Python 2.

PyQt4 and PySide

Qt 4 reached end-of-life on 2015-12-19. Its Python bindings are also EOL: PyQt4 since 2018-08-31 and PySide since 2015-10-14.

Support for PyQt4 and PySide has been deprecated from `ImageQt` and will be removed in a future version. Please upgrade to PyQt5 or PySide2.

PIL.*ImagePlugin.__version__ attributes

These version constants have been deprecated and will be removed in a future version.

- `BmpImagePlugin.__version__`
- `CurImagePlugin.__version__`
- `DcxImagePlugin.__version__`
- `EpsImagePlugin.__version__`
- `FliImagePlugin.__version__`
- `FpxImagePlugin.__version__`
- `GdImageFile.__version__`
- `GifImagePlugin.__version__`
- `IcoImagePlugin.__version__`
- `ImImagePlugin.__version__`
- `ImtImagePlugin.__version__`
- `IptcImagePlugin.__version__`
- `Jpeg2KImagePlugin.__version__`
- `JpegImagePlugin.__version__`
- `McIdasImagePlugin.__version__`
- `MicImagePlugin.__version__`
- `MpegImagePlugin.__version__`
- `MpoImagePlugin.__version__`
- `MspImagePlugin.__version__`
- `PalmImagePlugin.__version__`
- `PcdImagePlugin.__version__`
- `PcxImagePlugin.__version__`
- `PdfImagePlugin.__version__`
- `PixarImagePlugin.__version__`
- `PngImagePlugin.__version__`
- `PpmImagePlugin.__version__`
- `PsdImagePlugin.__version__`
- `SgiImagePlugin.__version__`
- `SunImagePlugin.__version__`
- `TgaImagePlugin.__version__`
- `TiffImagePlugin.__version__`
- `WmfImagePlugin.__version__`
- `XbmImagePlugin.__version__`
- `XpmImagePlugin.__version__`

- `XVThumbImagePlugin.__version__`

Use `PIL.__version__` instead.

ImageCms.CmsProfile attributes

Some attributes in `ImageCms.CmsProfile` have been deprecated since Pillow 3.2.0. From 6.0.0, they issue a `DeprecationWarning`:

Deprecated	Use instead
`color_space`	Padded `xcolor_space`
`pcs`	Padded `connection_space`
`product_copyright`	Unicode `copyright`
`product_desc`	Unicode `profile_description`
`product_description`	Unicode `profile_description`
`product_manufacturer`	Unicode `manufacturer`
`product_model`	Unicode `model`

MIME type improvements

Previously, all JPEG2000 images had the MIME type "image/jpx". This has now been corrected. After the file format drivers have been loaded, `Image.MIME["JPEG2000"]` will return "image/jp2". `ImageFile.get_format_mimetype` will return "image/jpx" if a JPX profile is present, or "image/jp2" otherwise.

Previously, all SGI images had the MIME type "image/rgb". This has now been corrected. After the file format drivers have been loaded, `Image.MIME["SGI"]` will return "image/sgi". `ImageFile.get_format_mimetype` will return "image/rgb" if RGB image data is present, or "image/sgi" otherwise.

MIME types have been added to the PPM format. After the file format drivers have been loaded, `Image.MIME["PPM"]` will now return the generic "image/x-portable-anymap". `ImageFile.get_format_mimetype` will return a MIME type specific to the color type.

The TGA, PCX and ICO formats also now have MIME types: "image/x-tga", "image/x-pcx" and "image/x-icon" respectively.

API additions

DIB file format

Pillow now supports reading and writing the Device Independent Bitmap file format.

Image.quantize

The `dither` option is now a customisable parameter (was previously hardcoded to 1). This parameter takes the same values used in *convert()*.

New language parameter

These text-rendering functions now accept a `language` parameter to request language-specific glyphs and ligatures from the font:

- `ImageDraw.ImageDraw.multiline_text()`
- `ImageDraw.ImageDraw.multiline_textsize()`
- `ImageDraw.ImageDraw.text()`

- `ImageDraw.ImageDraw.textsize()`
- `ImageFont.ImageFont.getmask()`
- `ImageFont.ImageFont.getsize_multiline()`
- `ImageFont.ImageFont.getsize()`

Added EXIF class

getexif() has been added, which returns an *Exif* instance. Values can be retrieved and set like a dictionary. When saving JPEG, PNG or WEBP, the instance can be passed as an `exif` argument to include any changes in the output image.

Added ImageOps.exif_transpose

exif_transpose() returns a copy of an image, transposed according to its EXIF Orientation tag.

PNG EXIF data

EXIF data can now be read from and saved to PNG images. However, unlike other image formats, EXIF data is not guaranteed to be present in *info* until *load()* has been called.

Other changes

Reading new DDS image format

Pillow can now read uncompressed RGB data from DDS images.

Reading TIFF with old-style JPEG compression

Added support reading TIFF files with old-style JPEG compression through LibTIFF. All YCbCr TIFF images are now always read as RGB.

TIFF compression codecs

Support has been added for the LZMA, Zstd and WebP TIFF compression codecs.

Improved support for transposing I;16 images

I;16, I;16L and I;16B are now supported image modes for all *transpose()* operations.

1.6.45 5.4.1 (2019-01-06)

This release fixes regressions in 5.4.0.

Installation on Termux

A change to the way Pillow detects libraries during installed prevented installation on Termux, which does not have `/sbin/ldconfig`. This is now fixed.

PNG: Handle IDAT chunks after image end

Some PNG images have multiple IDAT chunks. In some cases, Pillow will stop reading image data before the IDAT chunks finish. A regression caused an `EOFError` exception when previously there was none. This is now fixed, and file reading continues in case there are subsequent text chunks.

PNG: MIME type

The addition of limited APNG support to the PNG plugin also overwrote the MIME type for PNG files, causing "image/apng" to be returned as the MIME type of both APNG and PNG files. This has been fixed so the MIME type of PNG files is "image/png".

File closing

A regression caused an unsupported image file to report a `ValueError: seek of closed file` exception instead of an `OSError`. This has been fixed by ensuring that image plugins only close their internal `__fp` if they are not the same as `ImageFile`'s `fp`, allowing each to manage their own file pointers.

1.6.46 5.4.0 (2019-01-01)

API changes

APNG extension to PNG plugin

Animated Portable Network Graphics (APNG) images are not fully supported but can be opened via the PNG plugin to get some basic info:

```
im = Image.open("image.apng")
print(im.mode)  # "RGBA"
print(im.size)  # (245, 245)
im.show()  # Shows a single frame
```

Check for libjpeg-turbo

You can check if Pillow has been built against the libjpeg-turbo version of the libjpeg library:

```
from PIL import features
features.check_feature("libjpeg_turbo")  # True or False
```

Negative indexes in pixel access

When accessing individual image pixels, negative indexes are now also accepted. For example, to get or set the farthest pixel in the lower right of an image:

```
px = im.load()
print(px[-1, -1])
px[-1, -1] = (0, 0, 0)
```

New custom TIFF tags

TIFF images can now be saved with custom integer, float and string TIFF tags:

```
im = Image.new("RGB", (200, 100))
custom = {
    37000: 4,
    37001: 4.2,
    37002: "custom tag value",
    37003: u"custom tag value",
    37004: b"custom tag value",
}
```

(continues on next page)

(continued from previous page)

```
im.save("output.tif", tiffinfo=custom)

im2 = Image.open("output.tif")
print(im2.tag_v2[37000])  # 4
print(im2.tag_v2[37002])  # "custom tag value"
print(im2.tag_v2[37004])  # b"custom tag value"
```

Other changes

ImageOps.fit

Now uses one resize operation with `box` parameter internally instead of a crop and scale operations sequence. This improves the performance and accuracy of cropping since the `box` parameter accepts float values.

1.6.47 5.3.0 (2018-10-01)

API changes

Image size

If you attempt to set the size of an image directly, e.g. `im.size = (100, 100)`, you will now receive an `AttributeError`. This is not about removing existing functionality, but instead about raising an explicit error to prevent later consequences. The `resize` method is the correct way to change an image's size.

The exceptions to this are:

- The ICO and ICNS image formats, which use `im.size = (100, 100)` to select a subimage.
- The TIFF image format, which now has a `DeprecationWarning` for this action, as direct image size setting was previously necessary to work around an issue with tile extents.

API additions

Added line width parameter to rectangle and ellipse-based shapes

An optional line `width` parameter has been added to `ImageDraw.Draw.arc`, `chord`, `ellipse`, `pieslice` and `rectangle`.

Curved joints for line sequences

`ImageDraw.Draw.line` draws a line, or lines, between points. Previously, when multiple points are given, for a larger `width`, the joints between these lines looked unsightly. There is now an additional optional argument, `joint`, defaulting to `None`. When it is set to `curved`, the joints between the lines will become rounded.

ImageOps.colorize

Previously `ImageOps.colorize` only supported two-color mapping with `black` and `white` arguments being mapped to 0 and 255 respectively. Now it supports three-color mapping with the optional `mid` parameter, and the positions for all three color arguments can each be optionally specified (`blackpoint`, `whitepoint` and `midpoint`). For example, with all optional arguments:

```
ImageOps.colorize(im, black=(32, 37, 79), white='white', mid=(59, 101, 175),
                      blackpoint=15, whitepoint=240, midpoint=100)
```

ImageOps.pad

While `ImageOps.fit` allows users to crop images to a requested aspect ratio and size, new method `ImageOps.pad` pads images to fill a requested aspect ratio and size, filling new space with a provided `color` and positioning the image within the new area through a `centering` argument.

Other changes

Added support for reading tiled TIFF images through LibTIFF. Compressed TIFF images are now read through LibTIFF.

RGB WebP images are now read as RGB mode, rather than RGBX.

1.6.48 5.2.0 (2018-07-01)

API changes

Deprecations

These version constants have been deprecated. `VERSION` will be removed in Pillow 6.0.0, and `PILLOW_VERSION` will be removed after that.

- `PIL.VERSION` (old PIL version 1.1.7)
- `PIL.PILLOW_VERSION`
- `PIL.Image.VERSION`
- `PIL.Image.PILLOW_VERSION`

Use `PIL.__version__` instead.

API additions

3D color lookup tables

Support for 3D color lookup table transformations has been added.

- https://en.wikipedia.org/wiki/3D_lookup_table

`Color3DLUT.generate` transforms 3-channel pixels using the values of the channels as coordinates in the 3D lookup table and interpolating the nearest elements.

It allows you to apply almost any color transformation in constant time by using pre-calculated decimated tables.

`Color3DLUT.transform()` allows altering table values with a callback.

If NumPy is installed, the performance of argument conversion is dramatically improved when a source table supports buffer interface (NumPy && arrays in Python >= 3).

ImageColor.getrgb

Previously `Image.rotate` only supported HSL color strings. Now HSB and HSV strings are also supported, as well as float values. For example, `ImageColor.getrgb("hsv(180,100%,99.5%)")`.

ImageFile.get_format_mimetype

`ImageFile.get_format_mimetype` has been added to return the MIME type of an image file, where available. For example, `Image.open("hopper.jpg").get_format_mimetype()` returns `"image/jpeg"`.

ImageFont.getsize_multiline

A new method to return the size of multiline text, for example `font.getsize_multiline("ABC\nAaaa")`

Image.rotate

A new named parameter, `fillcolor`, has been added to `Image.rotate`. This color specifies the background color to use in the area outside the rotated image. This parameter takes the same color specifications as used in `Image.new`.

TGA file format

Pillow can now read and write LA data (in addition to L, P, RGB and RGBA), and write RLE data (in addition to uncompressed).

Other changes

Support added for Python 3.7

Pillow 5.2 supports Python 3.7.

Build macOS wheels with Xcode 6.4, supporting older macOS versions

The macOS wheels for Pillow 5.1.0 were built with Xcode 9.2, meaning 10.12 Sierra was the lowest supported version.

Prior to Pillow 5.1.0, Xcode 8 was used, supporting El Capitan 10.11.

Instead, Pillow 5.2.0 is built with the oldest available Xcode 6.4 to support at least 10.10 Yosemite.

Fix _i2f compilation with some GCC versions

For example, this allows compilation with GCC 4.8 on NetBSD.

Resolve confusion getting PIL / Pillow version string

Re: "version constants deprecated" listed above, as user gnbl notes in #3082:

- it's confusing that PIL.VERSION returns the version string of the former PIL instead of Pillow's
- ReadTheDocs documentation is missing for some version branches (why is this, will it ever change, . . .)
- it's confusing that PIL.version is a module and does not return the version information directly or hints on how to get it
- the package information header is essentially useless (placeholder, does not even mention Pillow, nor the version)
- PIL._version module documentation comment could explain how to access the version information

We have attempted to resolve these issues in #3083, #3090 and #3218.

1.6.49 5.1.0 (2018-04-02)

API changes

Optional channels for TIFF files

Pillow can now open TIFF files with base modes of `RGB`, `YCbCr`, and `CMYK` with up to 6 8-bit channels, discarding any extra channels if the content is tagged as UNSPECIFIED. Pillow still does not store more than 4 8-bit channels of image data.

API additions

Append to PDF files

Images can now be appended to PDF files in place by passing in `append=True` when saving the image.

New BLP file format

Pillow now supports reading the BLP "Blizzard Mipmap" file format used for tiles in Blizzard's engine.

Other changes

WebP memory leak

A memory leak when opening `WebP` files has been fixed.

1.6.50 5.0.0 (2018-01-01)

Backwards incompatible changes

Python 3.3 dropped

Python 3.3 is EOL and no longer supported due to moving testing from nose, which is deprecated, to pytest, which doesn't support Python 3.3. We will not be creating binaries, testing, or retaining compatibility with this version. The final version of Pillow for Python 3.3 is 4.3.0.

Decompression bombs now raise exceptions

Pillow has previously emitted warnings for images that are unexpectedly large and may be a denial of service. These warnings are now upgraded to `DecompressionBombError`s for images that are twice the size of images that trigger the `DecompressionBombWarning`. The default threshold is 128Mpx, or 0.5GB for an `RGB` or `RGBA` image. This can be disabled or changed by setting `Image.MAX_IMAGE_PIXELS = None`.

Scripts

The scripts formerly installed by Pillow have been split into a separate package, pillow-scripts, living at https://github.com/python-pillow/pillow-scripts.

API changes

OleFileIO.py

The olefile module is no longer a required dependency when installing Pillow. Support for plugins requiring olefile will not be loaded if it is not installed. This allows library consumers to avoid installing this dependency if they choose. Some library consumers have little interest in the format support and would like to keep dependencies to a minimum.

Further, the vendored version was removed in Pillow 4.0.0 and replaced with a deprecation warning that PIL.OleFileIO would be removed in a future version. This warning has been upgraded to an import error pending future removal.

Check parameter on _save

Several image plugins supported a named `check` parameter on their nominally private `_save` method to preflight if the image could be saved in that format. That parameter has been removed.

API additions

Image.transform

A new named parameter, `fillcolor`, has been added to `Image.transform`. This color specifies the background color to use in the area outside the transformed area in the output image. This parameter takes the same color specifications as used in `Image.new`.

GIF disposal

Multiframe GIF images now take an optional disposal parameter to specify the disposal option for changed pixels.

Other changes

Compressed TIFF images

Previously, there were some compression modes (JPEG, Packbits, and LZW) that were supported with Pillow's internal TIFF decoder. All compressed TIFFs are now read using the `libtiff` decoder, as it implements the compression schemes more correctly.

Libraqm is now dynamically linked

The libraqm dependency for complex text scripts is now linked dynamically at runtime rather than at packaging time. This allows us to release binaries with support for libraqm if it is installed on the user's machine.

Source layout changes

The Pillow source is now stored within the `src` directory of the distribution. This prevents accidental imports of the PIL directory when running Python from the project directory.

Setup.py changes

Multiarch support on Linux should be more robust, especially on Debian derivatives on ARM platforms. Debian's multiarch platform configuration is run in preference to the sniffing of machine platform and architecture.

1.6.51 4.3.0 (2017-10-02)

API changes

Deprecations

Several undocumented functions in ImageOps have been deprecated: `gaussian_blur`, `gblur`, `unsharp_mask`, `usm` and `box_blur`. Use the equivalent operations in `ImageFilter` instead. These functions will be removed in a future release.

TIFF metadata changes

- TIFF tags with unknown type/quantity now default to being bare values if they are 1 element, where previously they would be a single element tuple. This is only with the new api, not the legacy api. This normalizes the handling of fields, so that the metadata with inferred or image specified counts are handled the same as metadata with count specified in the TIFF spec.
- The `PhotoshopInfo`, `XMP`, and `JPEGTables` tags now have a defined type (bytes) and a count of 1.
- The `ImageJMetaDataByteCounts` tag now has an arbitrary number of items, as there can be multiple items, one for UTF-8, and one for UTF-16.

Core Image API changes

These are internal functions that should not have been used by user code, but they were accessible from the python layer.

Debugging code within `Image.core.grabclipboard` was removed. It had been marked as `will be removed in future versions` since PIL. When enabled, it identified the format of the clipboard data.

The `PIL.Image.core.copy` and `PIL.Image.Image.im.copy2` methods have been removed.

The `PIL.Image.core.getcount` methods have been removed, use `PIL.Image.core.get_stats()['new_count']` property instead.

API additions

Get one channel from image

A new method *PIL.Image.Image.getchannel()* has been added to return a single channel by index or name. For example, `image.getchannel("A")` will return alpha channel as separate image. `getchannel` should work up to 6 times faster than `image.split()[0]` in previous Pillow versions.

Box blur

A new filter, *PIL.ImageFilter.BoxBlur*, has been added. This is a filter with similar results to a Gaussian blur, but is much faster.

Partial resampling

Added new argument `box` for *PIL.Image.Image.resize()*. This argument defines a source rectangle from within the source image to be resized. This is very similar to the `image.crop(box).resize(size)` sequence except that `box` can be specified with subpixel accuracy.

New transpose operation

The `Image.TRANSVERSE` operation has been added to *PIL.Image.Image.transpose()*. This is equivalent to a transpose operation about the opposite diagonal.

Multiband filters

There is a new *PIL.ImageFilter.MultibandFilter* base class for image filters that can run on all channels of an image in one operation. The original *PIL.ImageFilter.Filter* class remains for image filters that can process only single band images, or require splitting of channels prior to filtering.

Other changes

Loading 16-bit TIFF images

Pillow now can read 16-bit multichannel TIFF files including files with alpha transparency. The image data is truncated to 8-bit precision.

Pillow now can read 16-bit signed integer single channel TIFF files. The image data is promoted to 32-bit for storage and processing.

SGI images

Pillow can now read and write uncompressed 16-bit multichannel SGI images to and from RGB and RGBA formats. The image data is truncated to 8-bit precision.

Pillow can now read RLE encoded SGI images in both 8 and 16-bit precision.

Performance

This release contains several performance improvements:

- Many memory bandwidth-bounded operations such as crop, image allocation, conversion, split into bands and merging from bands are up to 2x faster.
- Upscaling of multichannel images (such as RGB) is accelerated by 5-10%
- JPEG loading is accelerated up to 15% and JPEG saving up to 20% when using a recent version of libjpeg-turbo.
- `Image.transpose` has been accelerated 15% or more by using a cache friendly algorithm.
- ImageFilters based on Kernel convolution are significantly faster due to the new *`MultibandFilter`* feature.
- All memory allocation for images is now done in blocks, rather than falling back to an allocation for each scan line for images larger than the block size.

CMYK conversion

The basic CMYK->RGB conversion has been tweaked to match the formula from Google Chrome. This produces an image that is generally lighter than the previous formula, and more in line with what color managed applications produce.

1.6.52 4.2.1 (2017-07-06)

There are no functional changes in this release.

Fixed Windows PyPy build

A change in the 4.2.0 cycle broke the Windows PyPy build. This has been fixed, and PyPy is now part of the Windows CI matrix.

1.6.53 4.2.0 (2017-07-01)

Backwards incompatible changes

Several deprecated items have been removed

- The methods `PIL.ImageWin.Dib.fromstring`, `PIL.ImageWin.Dib.tostring` and `PIL.TiffImagePlugin.ImageFileDirectory_v2.as_dict` have been removed.
- Before Pillow 4.2.0, attempting to save an RGBA image as JPEG would discard the alpha channel. From Pillow 3.4.0, a deprecation warning was shown. From Pillow 4.2.0, the deprecation warning is removed and an `IOError` is raised.

Removed core Image function

The unused function `Image.core.new_array` was removed. This is an internal function that should not have been used by user code, but it was accessible from the python layer.

Other changes

Added complex text rendering

Pillow now supports complex text rendering for scripts requiring glyph composition and bidirectional flow. This optional feature adds three dependencies: harfbuzz, fribidi, and raqm. See the install documentation for further details. This feature is tested and works on Unix and Mac, but has not yet been built on Windows platforms.

New optional parameters

- *PIL.ImageDraw.floodfill()* has a new optional parameter: threshold. This specifies a tolerance for the color to replace with the flood fill.
- The TIFF and PDF image writers now support the `append_images` optional parameter for specifying additional images to create multipage outputs.

New DecompressionBomb warning

PIL.Image.Image.crop() now may raise a DecompressionBomb warning if the crop region enlarges the image over the threshold specified by *PIL.Image.MAX_IMAGE_PIXELS*.

1.6.54 4.1.1 (2017-04-28)

Fix regression with reading DPI from EXIF data

Some JPEG images don't contain DPI information in the image metadata, but do contain it in the EXIF data. A patch was added in 4.1.0 to read from the EXIF data, but it did not accept all possible types that could be included there. This fix adds the ability to read ints as well as rational values.

Incompatibility between 3.6.0 and 3.6.1

CPython 3.6.1 added a new symbol, PySlice_GetIndicesEx, which was not present in 3.6.0. This had the effect of causing binaries compiled on CPython 3.6.1 to not work on installations of C-Python 3.6.0. This fix undefines PySlice_GetIndicesEx if it exists to restore compatibility with both 3.6.0 and 3.6.1. See https://bugs.python.org/issue29943 for more details.

1.6.55 4.1.0 (2017-04-03)

Deprecations

Several deprecated items have been removed.

- Support for spaces in tiff kwargs in the parameters for 'x resolution', 'y resolution', 'resolution unit', and 'date time' has been removed. Underscores should be used instead.
- The methods `PIL.ImageDraw.ImageDraw.setink`, `PIL.ImageDraw.ImageDraw.setfill`, and `PIL.ImageDraw.ImageDraw.setfont` have been removed.

Other changes

Closing files when opening images

The file handling when opening images has been overhauled. Previously, Pillow would attempt to close some, but not all image formats after loading the image data. Now, the following behavior is specified:

- For images where an open file is passed in, it is the responsibility of the calling code to close the file.
- For images where Pillow opens the file and the file is known to have only one frame, the file is closed after loading.

- If the file has more than one frame, or if it can't be determined, then the file is left open to permit seeking to subsequent frames. It will be closed, eventually, in the `close` or `__del__` methods.
- If the image is memory mapped, then we can't close the mapping to the underlying file until we are done with the image. The mapping will be closed in the `close` or `__del__` method.

Changes to GIF handling when saving

The *PIL.GifImagePlugin* code has been refactored to fix the flow when saving images. There are two external changes that arise from this:

- An *PIL.ImagePalette.ImagePalette* object is now accepted as a specified palette argument in *PIL.Image.Image.save()*.
- The image to be saved is no longer modified in place by any of the operations of the save function. Previously it was modified when optimizing the image palette.

This refactor fixed some bugs with palette handling when saving multiple frame GIFs.

New method: Image.remap_palette

The method *PIL.Image.Image.remap_palette()* has been added. This method was hoisted from the GifImagePlugin code used to optimize the palette.

Added decoder registry and support for Python-based decoders

There is now a decoder registry similar to the image plugin registries. Image plugins can register a decoder, and it will be called when the decoding is requested. This allows for the creation of pure Python decoders. While the Python decoders will not be as fast as their C based counterparts, they may be easier and quicker to develop or safer to run.

Tests

Many tests have been added, including correctness tests for image formats that have been previously untested.

We are now running automated tests in Docker containers against more Linux versions than are provided on Travis CI, which is currently Ubuntu 14.04 x64. This Pillow release is tested on 64-bit Alpine, Arch, Ubuntu 12.04 and 16.04, and 32-bit Debian Stretch and Ubuntu 14.04. This also covers a wider range of dependency versions than are provided on Travis natively.

1.6.56 4.0.0 (2017-01-01)

Python 2.6 and 3.2 dropped

Pillow 4.0 no longer supports Python 2.6 and 3.2. We will not be creating binaries, testing, or retaining compatibility with these releases. This release removes some workarounds for those Python releases, so the final working version of Pillow on 2.6 or 3.2 is 3.4.2.

Support added for Python 3.6

Pillow 4.0 supports Python 3.6.

OleFileIO.py

`OleFileIO.py` has been removed as a vendored file and is now installed from the upstream olefile PyPI package. All internal dependencies are redirected to the olefile package. Direct accesses to `PIL.OlefileIO` raises a deprecation warning, then patches the upstream olefile into `sys.modules` in its place.

SGI image save

It is now possible to save images in modes L, RGB, and RGBA to the uncompressed SGI image format.

Zero sized images

Pillow 3.4.0 removed support for creating images with (0,0) size. This has been reenabled, restoring pre 3.4 behavior.

Internal handles_eof flag

The `handles_eof flag` for decoding images has been removed, as there were no internal users of the flag. Anyone maintaining image decoders outside of the Pillow source tree should consider using the cleanup function pointers instead.

Image.core.stretch removed

The stretch function on the core image object has been removed. This used to be for enlarging the image, but has been aliased to resize recently.

1.6.57 3.4.0 (2016-10-03)

Backwards incompatible changes

Image.core.open_ppm removed

The nominally private/debugging function `Image.core.open_ppm` has been removed. If you were using this function, please use `Image.open` instead.

Deprecations

Deprecation warning when saving JPEGs

JPEG images cannot contain an alpha channel. Pillow prior to 3.4.0 silently drops the alpha channel. With this release Pillow will now issue a `DeprecationWarning` when attempting to save a `RGBA` mode image as a JPEG. This will become an error in Pillow 4.2.

API additions

New resizing filters

Two new filters available for `Image.resize()` and `Image.thumbnail()` functions: `BOX` and `HAMMING`. `BOX` is the high-performance filter with two times shorter window than `BILINEAR`. It can be used for image reduction 3 and more times and produces a sharper result than `BILINEAR`.

`HAMMING` filter has the same performance as `BILINEAR` filter while providing the image downscaling quality comparable to `BICUBIC`. Both new filters don't show good quality for the image upscaling.

New DDS decoders

Pillow can now decode DXT3 images, as well as the previously supported DXT1 and DXT5 formats. All three formats are now decoded in C code for better performance.

Append images to GIF

Additional frames can now be appended when saving a GIF file, through the `append_images` argument. The new frames are passed in as a list of images, which may be have multiple frames themselves.

Note that the `append_images` argument is only used if `save_all` is also in effect, e.g.:

```
im.save(out, save_all=True, append_images=[im1, im2, ...])
```

Save multiple frame TIFF

Multiple frames can now be saved in a TIFF file by using the `save_all` option. e.g.:

```
im.save("filename.tiff", format="TIFF", save_all=True)
```

1.6.58 3.3.2 (2016-09-29)

Security

Integer overflow in map.c

Pillow prior to 3.3.2 may experience integer overflow errors in map.c when reading specially crafted image files. This may lead to memory disclosure or corruption.

Specifically, when parameters from the image are passed into `Image.core.map_buffer`, the size of the image was calculated with `xsize` * `ysize` * `bytes_per_pixel`. This will overflow if the result is larger than SIZE_MAX. This is possible on a 32-bit system.

Furthermore this `size` value was added to a potentially attacker provided `offset` value and compared to the size of the buffer without checking for overflow or negative values.

These values were then used for creating pointers, at which point Pillow could read the memory and include it in other images. The image was marked readonly, so Pillow would not ordinarily write to that memory without duplicating the image first.

This issue was found by Cris Neckar at Divergent Security.

Sign extension in Storage.c

Pillow prior to 3.3.2 and PIL 1.1.7 (at least) do not check for negative image sizes in `ImagingNew` in `Storage.c`. A negative image size can lead to a smaller allocation than expected, leading to arbitrary writes.

This issue was found by Cris Neckar at Divergent Security.

1.6.59 3.3.0 (2016-07-01)

Libimagequant support

There is now support for using libimagequant as a higher quality quantization option in `Image.quantize()` on Unix-like platforms. This support requires building Pillow from source against libimagequant. We cannot distribute binaries due to licensing differences.

New setup.py options

There are two new options to control the `build_ext` task in `setup.py`:

- `--debug` dumps all of the directories and files that are checked when searching for libraries or headers when building the extensions.
- `--disable-platform-guessing` removes many of the directories that are checked for libraries and headers for build systems or cross compilers that specify that information in via environment variables.

Resizing

Image resampling for 8-bit per channel images was rewritten using only integer computings. This is faster on most platforms and doesn't introduce precision errors on the wide range of scales. With other performance improvements, this makes resampling 60% faster on average.

Color calculation for images in the `LA` mode on semitransparent pixels was fixed.

Rotation

Rotation for angles divisible by 90 degrees now always uses transposition. This greatly improves both quality and performance in this case. Also, the bug with wrong image size calculation when rotating by 90 degrees was fixed.

Image metadata

The return type for binary data in version 2 Exif and Tiff metadata has been changed from a tuple of integers to bytes. This is a change from the behavior since 3.0.0.

1.6.60 3.2.0 (2016-04-01)

New DDS and FTEX image plugins

The `DdsImagePlugin` reading DXT1 and DXT5 encoded `.dds` images was added. DXT3 images are not currently supported.

The `FtexImagePlugin` reads textures used for 3D objects in Independence War 2: Edge Of Chaos. The plugin reads a single texture per file, in the `.ftc` (compressed) and `.ftu` (uncompressed) formats.

Updates to the GbrImagePlugin

The `GbrImagePlugin` (GIMP brush format) has been updated to fix support for version 1 files and add support for version 2 files.

Passthrough parameters for ImageDraw.text

`ImageDraw.multiline_text` and `ImageDraw.multiline_size` take extra spacing parameters above what are used in `ImageDraw.text` and `ImageDraw.size`. These parameters can now be passed into `ImageDraw.text` and `ImageDraw.size` and they will be passed through to the corresponding multiline functions.

ImageSequence.Iterator changes

`ImageSequence.Iterator` is now an actual iterator implementing the Iterator protocol. It is also now possible to seek to the first image of the file when using direct indexing.

1.6.61 3.1.2 (2016-04-01)

Security

CVE 2016-3076: Buffer overflow in Jpeg2KEncode.c

Pillow between 2.5.0 and 3.1.1 may overflow a buffer when writing large Jpeg2000 files, allowing for code execution or other memory corruption.

This occurs specifically in the function `j2k_encode_entry`, at the line:

```
state->buffer = malloc (tile_width * tile_height * components * prec / 8);
```

This vulnerability requires a particular value for `height * width` such that `height * width * components * precision` overflows, at which point the malloc will be for a smaller value than expected. The buffer that is allocated will be `((height * width * components * precision) mod (2^31) / 8)`, where components is 1-4 and precision is either 8 or 16. Common values would be 4 components at precision 8 for a standard `RGBA` image.

The unpackers then split an image that is laid out:

```
RGBARGBARGBA....
```

into:

```
RRR.
GGG.
BBB.
AAA.
```

If this buffer is smaller than expected, the jpeg2k unpacker functions will write outside the allocation and onto the heap, corrupting memory.

This issue was found by Alyssa Besseling at Atlassian.

1.6.62 3.1.1 (2016-02-04)

Security

CVE 2016-0740: Buffer overflow in `TiffDecode.c`

Pillow 3.1.0 and earlier when linked against libtiff >= 4.0.0 on x64 may overflow a buffer when reading a specially crafted tiff file.

Specifically, libtiff >= 4.0.0 changed the return type of `TIFFScanlineSize` from `int32` to machine dependent `int32|64`. If the scanline is sized so that it overflows an `int32`, it may be interpreted as a negative number, which will then pass the size check in `TiffDecode.c` line 236. To do this, the logical scanline size has to be > 2gb, and for the test file, the allocated buffer size is 64k against a roughly 4gb scan line size. Any image data over 64k is written over the heap, causing a segfault.

This issue was found by security researcher FourOne.

CVE 2016-0775: Buffer overflow in `FliDecode.c`

In all versions of Pillow, dating back at least to the last PIL 1.1.7 release, FliDecode.c has a buffer overflow error (CVE 2016-0775).

Around line 192:

```
case 16:
    /* COPY chunk */
    for (y = 0; y < state->ysize; y++) {
        UINT8* buf = (UINT8*) im->image[y];
        memcpy(buf+x, data, state->xsize);
        data += state->xsize;
    }
    break;
```

The memcpy has error where `x` is added to the target buffer address. `X` is used in several internal temporary variable roles, but can take a value up to the width of the image. `Im->image[y]` is a set of row pointers to segments of memory that are the size of the row. At the max `y`, this will write the contents of the line off the end of the memory buffer, causing a segfault.

This issue was found by Alyssa Besseling at Atlassian.

CVE 2016-2533: Buffer overflow in `PcdDecode.c`

In all versions of Pillow, dating back at least to the last PIL 1.1.7 release, `PcdDecode.c` has a buffer overflow error (CVE 2016-2533).

The `state.buffer` for `PcdDecode.c` is allocated based on a 3 bytes per pixel sizing, where `PcdDecode.c` wrote into the buffer assuming 4 bytes per pixel. This writes 768 bytes beyond the end of the buffer into other Python object storage. In some cases, this causes a segfault, in others an internal Python malloc error.

Integer overflow in `Resample.c`

If a large value was passed into the new size for an image, it is possible to overflow an `int32` value passed into malloc.

```
kk = malloc(xsize * kmax * sizeof(float));
...
xbounds = malloc(xsize * 2 * sizeof(int));
```

`xsize` is trusted user input. These multiplications can overflow, leading the `malloc`'d buffer to be undersized. These allocations are followed by a loop that writes out of bounds. This can lead to corruption on the heap of the Python process with attacker controlled float data.

This issue was found by Ned Williamson.

1.6.63 3.1.0 (2016-01-04)

ImageDraw arc, chord and pieslice can now use floats

There is no longer a need to ensure that the start and end arguments for `arc`, `chord` and `pieslice` are integers.

Note that these numbers are not simply rounded internally, but are actually utilised in the drawing process.

Consistent multiline text spacing

When using the `ImageDraw` multiline methods, the spacing between lines was inconsistent, based on the combination on ascenders and descenders.

This has now been fixed, so that lines are offset by their baselines, not the absolute height of each line.

There is also now a default spacing of 4px between lines.

EXIF, JPEG and TIFF metadata

There were major changes in the TIFF ImageFileDirectory support in Pillow 3.0 that led to a number of regressions. Some of them have been fixed in Pillow 3.1, and some of them have been extended to have different behavior.

TiffImagePlugin.IFDRational

Pillow 3.0 changed rational metadata to use a float. In Pillow 3.1, this has changed to allow the expression of 0/0 as a valid piece of rational metadata to reflect usage in the wild.

Rational metadata is now encapsulated in an `IFDRational` instance. This class extends the Rational class to allow a denominator of 0. It compares as a float or a number, but does allow access to the raw numerator and denominator values through attributes.

When used in a `ImageFileDirectory_v1`, a 2 item tuple is returned of the numerator and denominator, as was done previously.

This class should be used when adding a rational value to an ImageFileDirectory for saving to image metadata.

JpegImagePlugin._getexif

In Pillow 3.0, the dictionary returned from the private, experimental, but generally widely used `_getexif` function changed to reflect the ImageFileDirectory_v2 format, without a fallback to the previous format.

In Pillow 3.1, `_getexif` now returns a dictionary compatible with Pillow 2.9 and earlier, built with `ImageFileDirectory_v1` instances. Additionally, any single item tuples have been unwrapped and return a bare element.

The format returned by Pillow 3.0 has been abandoned. A more fully featured interface for EXIF is anticipated in a future release.

Out of spec metadata

In Pillow 3.0 and 3.1, images that contain metadata that is internally consistent, but not in agreement with the TIFF spec, may cause an exception when reading the metadata. This can happen when a tag that is specified to have a single value is stored with an array of values.

It is anticipated that this behavior will change in future releases.

1.6.64 3.0.0 (2015-10-01)

Backwards incompatible changes

Several methods that have been marked as deprecated for many releases have been removed in this release:

- `Image.tostring()`
- `Image.fromstring()`
- `Image.offset()`
- `ImageDraw.setink()`
- `ImageDraw.setfill()`
- The `ImageFileIO` module
- The `ImageFont.FreeTypeFont` and `ImageFont.truetype` `file` keyword arg
- The `ImagePalette` private `_make` functions
- `ImageWin.fromstring()`
- `ImageWin.tostring()`

Other changes

Saving multipage images

There is now support for saving multipage images in the `GIF` and `PDF` formats. To enable this functionality, pass in `save_all=True` as a keyword argument to the save:

```
im.save('test.pdf', save_all=True)
```

TIFF ImageFileDirectory rewrite

The TIFF ImageFileDirectory metadata code has been rewritten. Where previously it returned a somewhat arbitrary set of values and tuples, it now returns bare values where appropriate and tuples when the metadata item is a sequence or collection.

The original metadata is still available in the TiffImage.tags, the new values are available in the TiffImage.tags_v2 member. The old structures will be deprecated at some point in the future. When saving TIFF metadata, new code should use the TiffImagePlugin.ImageFileDirectory_v2 class.

LibJpeg and Zlib are required by default

The external dependencies on libjpeg and zlib are now required by default. If the headers or libraries are not found, then installation will abort with an error. This behaviour can be disabled with the `--disable-libjpeg` and `--disable-zlib` flags.

1.6.65 2.8.0 (2015-04-01)

Open HTTP response objects with Image.open

HTTP response objects returned from `urllib2.urlopen(url)` or `requests.get(url, stream=True).raw` are 'file-like' but do not support `.seek()` operations. As a result PIL was unable to open them as images, requiring a wrap in `cStringIO` or `BytesIO`.

Now new functionality has been added to `Image.open()` by way of an `.seek(0)` check and catch on exception `AttributeError` or `io.UnsupportedOperation`. If this is caught we attempt to wrap the object using `io.BytesIO` (which will only work on buffer-file-like objects).

This allows opening of files using both `urllib2` and `requests`, e.g.:

```
Image.open(urllib2.urlopen(url))
Image.open(requests.get(url, stream=True).raw)
```

If the response uses content-encoding (compression, either gzip or deflate) then this will fail as both the urllib2 and requests raw file object will produce compressed data in that case. Using Content-Encoding on images is rather nonsensical as most images are already compressed, but it can still happen.

For requests the work-around is to set the decode_content attribute on the raw object to True:

```
response = requests.get(url, stream=True)
response.raw.decode_content = True
image = Image.open(response.raw)
```

1.6.66 2.7.0 (2014-12-31)

Sane plugin

The Sane plugin has now been split into its own repo: https://github.com/python-pillow/Sane .

PNG text chunk size limits

To prevent potential denial of service attacks using compressed text chunks, there are now limits to the decompressed size of text chunks decoded from PNG images. If the limits are exceeded when opening a PNG image a `ValueError` will be raised.

Individual text chunks are limited to *PIL.PngImagePlugin.MAX_TEXT_CHUNK*, set to 1MB by default. The total decompressed size of all text chunks is limited to *PIL.PngImagePlugin.MAX_TEXT_MEMORY*, which defaults to 64MB. These values can be changed prior to opening PNG images if you know that there are large text blocks that are desired.

Image resizing filters

Image resizing methods *resize()* and *thumbnail()* take a `resample` argument, which tells which filter should be used for resampling. Possible values are: `NEAREST`, `BILINEAR`, `BICUBIC` and `ANTIALIAS`. Almost all of them were changed in this version.

Bicubic and bilinear downscaling

From the beginning `BILINEAR` and `BICUBIC` filters were based on affine transformations and used a fixed number of pixels from the source image for every destination pixel (2x2 pixels for `BILINEAR` and 4x4 for `BICUBIC`). This gave an unsatisfactory result for downscaling. At the same time, a high quality convolutions-based algorithm with flexible kernel was used for `ANTIALIAS` filter.

Starting from Pillow 2.7.0, a high quality convolutions-based algorithm is used for all of these three filters.

If you have previously used any tricks to maintain quality when downscaling with `BILINEAR` and `BICUBIC` filters (for example, reducing within several steps), they are unnecessary now.

Antialias renamed to Lanczos

A new `LANCZOS` constant was added instead of `ANTIALIAS`.

When `ANTIALIAS` was initially added, it was the only high-quality filter based on convolutions. It's name was supposed to reflect this. Starting from Pillow 2.7.0 all resize method are based on convolutions. All of them are antialias from now on. And the real name of the `ANTIALIAS` filter is Lanczos filter.

The `ANTIALIAS` constant is left for backward compatibility and is an alias for `LANCZOS`.

Lanczos upscaling quality

The image upscaling quality with `LANCZOS` filter was almost the same as `BILINEAR` due to a bug. This has been fixed.

Bicubic upscaling quality

The `BICUBIC` filter for affine transformations produced sharp, slightly pixelated image for upscaling. Bicubic for convolutions is more soft.

Resize performance

In most cases, convolution is more a expensive algorithm for downscaling because it takes into account all the pixels of source image. Therefore `BILINEAR` and `BICUBIC` filters' performance can be lower than before. On the other hand the quality of `BILINEAR` and `BICUBIC` was close to `NEAREST`. So if such quality is suitable for your tasks you can switch to `NEAREST` filter for downscaling, which will give a huge improvement in performance.

At the same time performance of convolution resampling for downscaling has been improved by around a factor of two compared to the previous version. The upscaling performance of the `LANCZOS` filter has remained the same. For `BILINEAR` filter it has improved by 1.5 times and for `BICUBIC` by four times.

Default filter for thumbnails

In Pillow 2.5 the default filter for *thumbnail()* was changed from `NEAREST` to `ANTIALIAS`. Antialias was chosen because all the other filters gave poor quality for reduction. Starting from Pillow 2.7.0, `ANTIALIAS` has been replaced with `BICUBIC`, because it's faster and `ANTIALIAS` doesn't give any advantages after downscaling with libjpeg, which uses supersampling internally, not convolutions.

Image transposition

A new method TRANSPOSE has been added for the *transpose()* operation in addition to FLIP_LEFT_RIGHT, FLIP_TOP_BOTTOM, ROTATE_90, ROTATE_180, ROTATE_270. TRANSPOSE is an algebra transpose, with an image reflected across its main diagonal.

The speed of ROTATE_90, ROTATE_270 and TRANSPOSE has been significantly improved for large images which don't fit in the processor cache.

Gaussian blur and unsharp mask

The *GaussianBlur()* implementation has been replaced with a sequential application of box filters. The new implementation is based on "Theoretical foundations of Gaussian convolution by extended box filtering" from the Mathematical Image Analysis Group. As *UnsharpMask()* implementations use Gaussian blur internally, all changes from this chapter are also applicable to it.

Blur radius

There was an error in the previous version of Pillow, where blur radius (the standard deviation of Gaussian) actually meant blur diameter. For example, to blur an image with actual radius 5 you were forced to use value 10. This has been fixed. Now the meaning of the radius is the same as in other software.

If you used a Gaussian blur with some radius value, you need to divide this value by two.

Blur performance

Box filter computation time is constant relative to the radius and depends on source image size only. Because the new Gaussian blur implementation is based on box filter, its computation time also doesn't depend on the blur radius.

For example, previously, if the execution time for a given test image was 1 second for radius 1, 3.6 seconds for radius 10 and 17 seconds for 50, now blur with any radius on same image is executed for 0.2 seconds.

Blur quality

The previous implementation takes into account only source pixels within 2 * standard deviation radius for every destination pixel. This was not enough, so the quality was worse compared to other Gaussian blur software.

The new implementation does not have this drawback.

TIFF parameter changes

Several kwarg parameters for saving TIFF images were previously specified as strings with included spaces (e.g. 'x resolution'). This was difficult to use as kwargs without constructing and passing a dictionary. These parameters now use the underscore character instead of space. (e.g. 'x_resolution')

1.6.67 2.6.0 (2014-10-01)

Security

CVE 2014-3589: Fix DOS attack

PIL/IcnsImagePlugin.py in Pillow before 2.3.2 and 2.5.x before 2.5.2 allows remote attackers to cause a denial of service via a crafted block size.

Found and reported by Andrew Drake of Dropbox.

1.6.68 2.5.2 (2014-08-12)

Security

CVE 2014-3589: Fix DOS attack

`PIL/IcnsImagePlugin.py` in Pillow before 2.3.2 and 2.5.x before 2.5.2 allows remote attackers to cause a denial of service via a crafted block size.

Found and reported by Andrew Drake of Dropbox.

1.6.69 2.3.2 (2014-08-12)

Security

CVE 2014-3589: Fix DOS attack

`PIL/IcnsImagePlugin.py` in Pillow before 2.3.2 and 2.5.x before 2.5.2 allows remote attackers to cause a denial of service via a crafted block size.

Found and reported by Andrew Drake of Dropbox.

1.6.70 2.3.1 (2014-03-14)

Security

These issues were reported in Debian bug #737059.

CVE 2014-1932: Fix insecure use of `tempfile.mktemp()`

The (1) `load_djpeg` function in `JpegImagePlugin.py`, (2) Ghostscript function in `EpsImagePlugin.py`, (3) `load` function in `IptcImagePlugin.py`, and (4) `_copy` function in `Image.py` in Pillow before 2.3.1 do not properly create temporary files, which allow local users to overwrite arbitrary files and obtain sensitive information via a symlink attack on the temporary file.

CVE 2014-1933: Fix insecure use of `tempfile.mktemp()`

The (1) `JpegImagePlugin.py` and (2) `EpsImagePlugin.py` scripts in Pillow before 2.3.1 uses the names of temporary files on the command line, which makes it easier for local users to conduct symlink attacks by listing the processes.

1.7 Deprecations and removals

This page lists Pillow features that are deprecated, or have been removed in past major releases, and gives the alternatives to use instead.

1.7.1 Deprecated features

Below are features which are considered deprecated. Where appropriate, a `DeprecationWarning` is issued.

ExifTags.IFD.Makernote

Deprecated since version 11.1.0.

`ExifTags.IFD.Makernote` has been deprecated. Instead, use `ExifTags.IFD.MakerNote`.

Image.Image.get_child_images()

Deprecated since version 11.2.1.

`Image.Image.get_child_images()` has been deprecated, and will be removed in Pillow 13 (2026-10-15). It will be moved to `ImageFile.ImageFile.get_child_images()`. The method uses an image's file pointer, and so child images could only be retrieved from an *PIL.ImageFile.ImageFile* instance.

Image.fromarray mode parameter

Deprecated since version 11.3.0.

Using the `mode` parameter in *fromarray()* was deprecated in Pillow 11.3.0. In Pillow 12.0.0, this was partially reverted, and it is now only deprecated when changing data types. Since pixel values do not contain information about palettes or color spaces, the parameter can still be used to place grayscale L mode data within a P mode image, or read RGB data as YCbCr for example. If omitted, the mode will be automatically determined from the object's shape and type.

Saving I mode images as PNG

Deprecated since version 11.3.0.

In order to fit the 32 bits of I mode images into PNG, when PNG images can only contain at most 16 bits for a channel, Pillow has been clipping the values. Rather than quietly changing the data, this is now deprecated. Instead, the image can be converted to another mode before saving:

```
from PIL import Image
im = Image.new("I", (1, 1))
im.convert("I;16").save("out.png")
```

ImageCms.ImageCmsProfile.product_name and .product_info

Deprecated since version 12.0.0.

`ImageCms.ImageCmsProfile.product_name` and the corresponding `.product_info` attributes have been deprecated, and will be removed in Pillow 13 (2026-10-15). They have been set to `None` since Pillow 2.3.0.

Image._show

Deprecated since version 12.0.0.

`Image._show` has been deprecated, and will be removed in Pillow 13 (2026-10-15). Use *show()* instead.

Image getdata()

Deprecated since version 12.1.0.

getdata() has been deprecated. *get_flattened_data()* can be used instead. This new method is identical, except that it returns a tuple of pixel values, instead of an internal Pillow data type.

1.7.2 Removed features

Deprecated features are only removed in major releases after an appropriate period of deprecation has passed.

ImageFile.raise_oserror

Deprecated since version 10.2.0.

Removed in version 12.0.0.

`ImageFile.raise_oserror()` has been removed. The function was undocumented and was only useful for translating error codes returned by a codec's `decode()` method, which ImageFile already did automatically.

IptcImageFile helper functions

Deprecated since version 10.2.0.

Removed in version 12.0.0.

The functions `IptcImageFile.dump` and `IptcImageFile.i`, and the constant `IptcImageFile.PAD` have been removed. These were undocumented helper functions intended for internal use, so there is no replacement. They can each be replaced by a single line of code using builtin functions in Python.

ImageCms constants and versions() function

Deprecated since version 10.3.0.

Removed in version 12.0.0.

A number of constants and a function in *ImageCms* have been removed. This includes a table of flags based on LittleCMS version 1 which has been replaced with a new class *ImageCms.Flags* based on LittleCMS 2 flags.

Deprecated	Use instead
ImageCms.DESCRIPTION	No replacement
ImageCms.VERSION	PIL.__version__
ImageCms.FLAGS["MATRIXINPUT"]	*ImageCms.Flags.CLUT_POST_LINEARIZATION*
ImageCms.FLAGS["MATRIXOUTPUT"]	*ImageCms.Flags.FORCE_CLUT*
ImageCms.FLAGS["MATRIXONLY"]	No replacement
ImageCms.FLAGS["NOWHITEONWHITEFIXU	*ImageCms.Flags.NOWHITEONWHITEFIXUP*
ImageCms.FLAGS["NOPRELINEARIZATION	*ImageCms.Flags.CLUT_PRE_LINEARIZATION*
ImageCms.FLAGS["GUESSDEVICECLASS"]	*ImageCms.Flags.GUESSDEVICECLASS*
ImageCms.FLAGS["NOTCACHE"]	*ImageCms.Flags.NOCACHE*
ImageCms.FLAGS["NOTPRECALC"]	*ImageCms.Flags.NOOPTIMIZE*
ImageCms.FLAGS["NULLTRANSFORM"]	*ImageCms.Flags.NULLTRANSFORM*
ImageCms.FLAGS["HIGHRESPRECALC"]	*ImageCms.Flags.HIGHRESPRECALC*
ImageCms.FLAGS["LOWRESPRECALC"]	*ImageCms.Flags.LOWRESPRECALC*
ImageCms.FLAGS["GAMUTCHECK"]	*ImageCms.Flags.GAMUTCHECK*
ImageCms.FLAGS["WHITEBLACKCOMPENSA	*ImageCms.Flags.BLACKPOINTCOMPENSATION*
ImageCms.FLAGS["BLACKPOINTCOMPENSA	*ImageCms.Flags.BLACKPOINTCOMPENSATION*
ImageCms.FLAGS["SOFTPROOFING"]	*ImageCms.Flags.SOFTPROOFING*
ImageCms.FLAGS["PRESERVEBLACK"]	*ImageCms.Flags.NONEGATIVES*
ImageCms.FLAGS["NODEFAULTRESOURCED	*ImageCms.Flags.NODEFAULTRESOURCEDEF*
ImageCms.FLAGS["GRIDPOINTS"]	*ImageCms.Flags.GRIDPOINTS()*
ImageCms.versions()	*PIL.features.version_module()* with feature="littlecms2", sys.version or sys.version_info, and PIL.__version__

ImageMath eval()

Deprecated since version 10.3.0.

Removed in version 12.0.0.

ImageMath.eval() has been removed. Use *lambda_eval()* or *unsafe_eval()* instead.

BGR;15, BGR;16 and BGR;24

Deprecated since version 10.4.0.

Removed in version 12.0.0.

The experimental BGR;15, BGR;16 and BGR;24 modes have been removed.

Non-image modes in ImageCms

Deprecated since version 10.4.0.

Removed in version 12.0.0.

The use in *ImageCms* of input modes and output modes that are not Pillow image modes has been removed. Defaulting to "L" or "1" if the mode cannot be mapped has also been removed.

Support for LibTIFF earlier than 4

Deprecated since version 10.4.0.

Removed in version 12.0.0.

Support for LibTIFF earlier than version 4 has been removed. Upgrade to a newer version of LibTIFF instead.

ImageDraw.getdraw hints parameter

Deprecated since version 10.4.0.

Removed in version 12.0.0.

The `hints` parameter in *getdraw()* has been removed.

FreeType 2.9.0

Deprecated since version 11.0.0.

Removed in version 12.0.0.

Support for FreeType 2.9.0 has been removed. FreeType 2.9.1 is the minimum version supported.

We recommend upgrading to at least FreeType 2.10.4, which fixed a severe vulnerability introduced in FreeType 2.6 (**CVE 2020-15999**).

ICNS (width, height, scale) sizes

Deprecated since version 11.0.0.

Removed in version 12.0.0.

Setting an ICNS image size to `(width, height, scale)` before loading has been removed. Instead, `load(scale)` can be used.

Image.isImageType()

Deprecated since version 11.0.0.

Removed in version 12.0.0.

`Image.isImageType(im)` has been removed. Use `isinstance(im, Image.Image)` instead.

ImageMath.lambda_eval and ImageMath.unsafe_eval options parameter

Deprecated since version 11.0.0.

Removed in version 12.0.0.

The `options` parameter in *`lambda_eval()`* and *`unsafe_eval()`* has been removed. One or more keyword arguments can be used instead.

JpegImageFile.huffman_ac and JpegImageFile.huffman_dc

Deprecated since version 11.0.0.

Removed in version 12.0.0.

The `huffman_ac` and `huffman_dc` dictionaries on JPEG images were unused. They have been removed.

Specific WebP feature checks

Deprecated since version 11.0.0.

Removed in version 12.0.0.

`features.check("transp_webp")`, `features.check("webp_mux")` and `features.check("webp_anim")` have been removed.

Get internal pointers to objects

Deprecated since version 11.0.0.

Removed in version 12.0.0.

`Image.core.ImagingCore.id` and `Image.core.ImagingCore.unsafe_ptrs` have been removed. They were used for obtaining raw pointers to `ImagingCore` internals. To interact with C code, you can use `Image.Image.getim()`, which returns a `Capsule` object.

TiffImagePlugin IFD_LEGACY_API

Removed in version 11.0.0.

`TiffImagePlugin.IFD_LEGACY_API` has been removed, as it was an unused setting.

PSFile

Deprecated since version 9.5.0.

Removed in version 11.0.0.

The `PSFile` class was removed in Pillow 11 (2024-10-15). This class was only made as a helper to be used internally, so there is no replacement. If you need this functionality though, it is a very short class that can easily be recreated in your own code.

PyAccess and Image.USE_CFFI_ACCESS

Deprecated since version 10.0.0.

Removed in version 11.0.0.

Since Pillow's C API is now faster than PyAccess on PyPy, `PyAccess` has been removed. Pillow's C API will now be used on PyPy instead.

`Image.USE_CFFI_ACCESS`, for switching from the C API to PyAccess, was similarly removed.

Tk/Tcl 8.4

Deprecated since version 8.2.0.

Removed in version 10.0.0.

Support for Tk/Tcl 8.4 was removed in Pillow 10.0.0 (2023-07-01).

Categories

Deprecated since version 8.2.0.

Removed in version 10.0.0.

`im.category` was removed along with the related `Image.NORMAL`, `Image.SEQUENCE` and `Image.CONTAINER` attributes.

To determine if an image has multiple frames or not, `getattr(im, "is_animated", False)` can be used instead.

JpegImagePlugin.convert_dict_qtables

Deprecated since version 8.3.0.

Removed in version 10.0.0.

Since deprecation in Pillow 8.3.0, the `convert_dict_qtables` method no longer performed any operations on the data given to it, and has been removed.

ImagePalette size parameter

Deprecated since version 8.4.0.

Removed in version 10.0.0.

Before Pillow 8.3.0, `ImagePalette` required palette data of particular lengths by default, and the `size` parameter could be used to override that. Pillow 8.3.0 removed the default required length, also removing the need for the `size` parameter.

ImageShow.Viewer.show_file file argument

Deprecated since version 9.1.0.

Removed in version 10.0.0.

The `file` argument in *`show_file()`* has been removed and replaced by `path`.

In effect, `viewer.show_file("test.jpg")` will continue to work unchanged.

Constants

Deprecated since version 9.1.0.

Removed in version 10.0.0.

A number of constants have been removed. Instead, `enum.IntEnum` classes have been added.

Note

Additional `Image` constants were deprecated in Pillow 9.1.0, but that was reversed in Pillow 9.4.0 and those constants will now remain available. See *Constants*

Removed	Use instead
`Image.LINEAR`	`Image.BILINEAR` or `Image.Resampling.BILINEAR`
`Image.CUBIC`	`Image.BICUBIC` or `Image.Resampling.BICUBIC`
`Image.ANTIALIAS`	`Image.LANCZOS` or `Image.Resampling.LANCZOS`
`ImageCms.INTENT_PERCEPTUAL`	`ImageCms.Intent.PERCEPTUAL`
`ImageCms.INTENT_RELATIVE_COLORMETRIC`	`ImageCms.Intent.RELATIVE_COLORMETRIC`
`ImageCms.INTENT_SATURATION`	`ImageCms.Intent.SATURATION`
`ImageCms.INTENT_ABSOLUTE_COLORIMETRIC`	`ImageCms.Intent.ABSOLUTE_COLORIMETRIC`
`ImageCms.DIRECTION_INPUT`	`ImageCms.Direction.INPUT`
`ImageCms.DIRECTION_OUTPUT`	`ImageCms.Direction.OUTPUT`
`ImageCms.DIRECTION_PROOF`	`ImageCms.Direction.PROOF`
`ImageFont.LAYOUT_BASIC`	`ImageFont.Layout.BASIC`
`ImageFont.LAYOUT_RAQM`	`ImageFont.Layout.RAQM`
`BlpImagePlugin.BLP_FORMAT_JPEG`	`BlpImagePlugin.Format.JPEG`
`BlpImagePlugin.BLP_ENCODING_UNCOMPRESSED`	`BlpImagePlugin.Encoding.UNCOMPRESSED`
`BlpImagePlugin.BLP_ENCODING_DXT`	`BlpImagePlugin.Encoding.DXT`
`BlpImagePlugin.BLP_ENCODING_UNCOMPRESSED_RAW_I`	`BlpImagePlugin.Encoding.UNCOMPRESSED_RAW_RGBA`
`BlpImagePlugin.BLP_ALPHA_ENCODING_DXT1`	`BlpImagePlugin.AlphaEncoding.DXT1`
`BlpImagePlugin.BLP_ALPHA_ENCODING_DXT3`	`BlpImagePlugin.AlphaEncoding.DXT3`
`BlpImagePlugin.BLP_ALPHA_ENCODING_DXT5`	`BlpImagePlugin.AlphaEncoding.DXT5`
`FtexImagePlugin.FORMAT_DXT1`	`FtexImagePlugin.Format.DXT1`
`FtexImagePlugin.FORMAT_UNCOMPRESSED`	`FtexImagePlugin.Format.UNCOMPRESSED`
`PngImagePlugin.APNG_DISPOSE_OP_NONE`	`PngImagePlugin.Disposal.OP_NONE`
`PngImagePlugin.APNG_DISPOSE_OP_BACKGROUND`	`PngImagePlugin.Disposal.OP_BACKGROUND`
`PngImagePlugin.APNG_DISPOSE_OP_PREVIOUS`	`PngImagePlugin.Disposal.OP_PREVIOUS`
`PngImagePlugin.APNG_BLEND_OP_SOURCE`	`PngImagePlugin.Blend.OP_SOURCE`
`PngImagePlugin.APNG_BLEND_OP_OVER`	`PngImagePlugin.Blend.OP_OVER`

FitsStubImagePlugin

Deprecated since version 9.1.0.

Removed in version 10.0.0.

The stub image plugin `FitsStubImagePlugin` has been removed. FITS images can be read without a handler through *`FitsImagePlugin`* instead.

Font size and offset methods

Deprecated since version 9.2.0.

Removed in version 10.0.0.

Several functions for computing the size and offset of rendered text have been removed:

Removed	Use instead
FreeTypeFont.getsize() and FreeTypeFont.getoffset()	*FreeTypeFont.getbbox()* and *FreeTypeFont.getlength()*
FreeTypeFont.getsize_multiline()	*ImageDraw.multiline_textbbox()*
ImageFont.getsize()	*ImageFont.getbbox()* and *ImageFont.getlength()*
TransposedFont.getsize()	*TransposedFont.getbbox()* and *TransposedFont.getlength()*
ImageDraw.textsize() and ImageDraw.multiline_textsize()	*ImageDraw.textbbox()*, *ImageDraw.textlength()* and *ImageDraw.multiline_textbbox()*
ImageDraw2.Draw.textsize()	*ImageDraw2.Draw.textbbox()* and *ImageDraw2.Draw.textlength()*

Previous code:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
width, height = font.getsize("Hello world")
left, top = font.getoffset("Hello world")

im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
width, height = draw.textsize("Hello world", font)

width, height = font.getsize_multiline("Hello\nworld")
width, height = draw.multiline_textsize("Hello\nworld", font)
```

Use instead:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
left, top, right, bottom = font.getbbox("Hello world")
width, height = right - left, bottom - top

im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
width = draw.textlength("Hello world", font)

left, top, right, bottom = draw.multiline_textbbox((0, 0), "Hello\nworld", font)
width, height = right - left, bottom - top
```

Previously, the `size` methods returned a `height` that included the vertical offset of the text, while the new `bbox` methods distinguish this as a `top` offset.

If you are using these methods for aligning text, consider using *Text anchors* instead which avoid issues that can occur with non-English text or unusual fonts. For example, instead of the following code:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")
```

(continues on next page)

(continued from previous page)

```
im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
width, height = draw.textsize("Hello world", font)
x, y = (100 - width) / 2, (100 - height) / 2
draw.text((x, y), "Hello world", font=font)
```

Use instead:

```
from PIL import Image, ImageDraw, ImageFont

font = ImageFont.truetype("Tests/fonts/FreeMono.ttf")

im = Image.new("RGB", (100, 100))
draw = ImageDraw.Draw(im)
draw.text((100 / 2, 100 / 2), "Hello world", font=font, anchor="mm")
```

FreeTypeFont.getmask2 fill parameter

Deprecated since version 9.2.0.

Removed in version 10.0.0.

The undocumented `fill` parameter of *`FreeTypeFont.getmask2()`* has been removed.

PhotoImage.paste box parameter

Deprecated since version 9.2.0.

Removed in version 10.0.0.

The `box` parameter was unused and has been removed.

PyQt5 and PySide2

Deprecated since version 9.2.0.

Removed in version 10.0.0.

Qt 5 reached end-of-life on 2020-12-08 for open-source users (and will reach EOL on 2023-12-08 for commercial licence holders).

Support for PyQt5 and PySide2 has been removed from `ImageQt`. Upgrade to PyQt6 or PySide6 instead.

Image.coerce_e

Deprecated since version 9.2.0.

Removed in version 10.0.0.

This undocumented method has been removed.

PILLOW_VERSION constant

Deprecated since version 5.2.0.

Removed in version 9.0.0.

Use `__version__` instead.

It was initially removed in Pillow 7.0.0, but temporarily brought back in 7.1.0 to give projects more time to upgrade.

Image.show command parameter

Deprecated since version 7.2.0.

Removed in version 9.0.0.

The `command` parameter has been removed. Use a subclass of *ImageShow.Viewer* instead.

Image._showxv

Deprecated since version 7.2.0.

Removed in version 9.0.0.

Use *Image.Image.show()* instead. If custom behaviour is required, use *ImageShow.register()* to add a custom *ImageShow.Viewer* class.

ImageFile.raise_ioerror

Deprecated since version 7.2.0.

Removed in version 9.0.0.

`IOError` was merged into `OSError` in Python 3.3. So, `ImageFile.raise_ioerror` has been removed. Use `ImageFile.raise_oserror` instead.

FreeType 2.7

Deprecated since version 8.1.0.

Removed in version 9.0.0.

Support for FreeType 2.7 has been removed.

We recommend upgrading to at least FreeType 2.10.4, which fixed a severe vulnerability introduced in FreeType 2.6 (**CVE 2020-15999**).

im.offset

Deprecated since version 1.1.2.

Removed in version 8.0.0.

`im.offset()` has been removed, call *ImageChops.offset()* instead.

It was documented as deprecated in PIL 1.1.2, raised a `DeprecationWarning` since 1.1.5, an `Exception` since Pillow 3.0.0 and `NotImplementedError` since 3.3.0.

Image.fromstring, im.fromstring and im.tostring

Deprecated since version 2.0.0.

Removed in version 8.0.0.

- `Image.fromstring()` has been removed, call *Image.frombytes()* instead.
- `im.fromstring()` has been removed, call *frombytes()* instead.
- `im.tostring()` has been removed, call *tobytes()* instead.

They issued a DeprecationWarning since 2.0.0, an Exception since 3.0.0 and NotImplementedError since 3.3.0.

ImageCms.CmsProfile attributes

Deprecated since version 3.2.0.

Removed in version 8.0.0.

Some attributes in *PIL.ImageCms.core.CmsProfile* have been removed. From 6.0.0, they issued a DeprecationWarning:

Removed	Use instead
color_space	Padded *xcolor_space*
pcs	Padded *connection_space*
product_copyright	Unicode *copyright*
product_desc	Unicode *profile_description*
product_description	Unicode *profile_description*
product_manufacturer	Unicode *manufacturer*
product_model	Unicode *model*

Python 2.7

Deprecated since version 6.0.0.

Removed in version 7.0.0.

Python 2.7 reached end-of-life on 2020-01-01. Pillow 6.x was the last series to support Python 2.

Image.__del__

Deprecated since version 6.1.0.

Removed in version 7.0.0.

Implicitly closing the image's underlying file in Image.__del__ has been removed. Use a context manager or call Image.close() instead to close the file in a deterministic way.

Previous method:

```
im = Image.open("hopper.png")
im.save("out.jpg")
```

Use instead:

```
with Image.open("hopper.png") as im:
    im.save("out.jpg")
```

PIL.*ImagePlugin.__version__ attributes

Deprecated since version 6.0.0.

Removed in version 7.0.0.

The version constants of individual plugins have been removed. Use PIL.__version__ instead.

Removed	Removed	Removed
BmpImagePlugin.__version__	Jpeg2KImagePlugin.__version__	PngImagePlugin.__version__
CurImagePlugin.__version__	JpegImagePlugin.__version__	PpmImagePlugin.__version__
DcxImagePlugin.__version__	McIdasImagePlugin.__version__	PsdImagePlugin.__version__
EpsImagePlugin.__version__	MicImagePlugin.__version__	SgiImagePlugin.__version__
FliImagePlugin.__version__	MpegImagePlugin.__version__	SunImagePlugin.__version__
FpxImagePlugin.__version__	MpoImagePlugin.__version__	TgaImagePlugin.__version__
GdImageFile.__version__	MspImagePlugin.__version__	TiffImagePlugin.__version__
GifImagePlugin.__version__	PalmImagePlugin.__version__	WmfImagePlugin.__version__
IcoImagePlugin.__version__	PcdImagePlugin.__version__	XbmImagePlugin.__version__
ImImagePlugin.__version__	PcxImagePlugin.__version__	XpmImagePlugin.__version__
ImtImagePlugin.__version__	PdfImagePlugin.__version__	XVThumbImagePlugin.__version__
IptcImagePlugin.__version__	PixarImagePlugin.__version__	

PyQt4 and PySide

Deprecated since version 6.0.0.

Removed in version 7.0.0.

Qt 4 reached end-of-life on 2015-12-19. Its Python bindings are also EOL: PyQt4 since 2018-08-31 and PySide since 2015-10-14.

Support for PyQt4 and PySide has been removed from `ImageQt`. Please upgrade to PyQt5 or PySide2.

Setting the size of TIFF images

Deprecated since version 5.3.0.

Removed in version 7.0.0.

Setting the size of a TIFF image directly (eg. `im.size = (256, 256)`) throws an error. Use `Image.resize` instead.

VERSION constant

Deprecated since version 5.2.0.

Removed in version 6.0.0.

`VERSION` (the old PIL version, always 1.1.7) has been removed. Use `__version__` instead.

Undocumented ImageOps functions

Deprecated since version 4.3.0.

Removed in version 6.0.0.

Several undocumented functions in `ImageOps` have been removed. Use the equivalents in `ImageFilter` instead:

Removed	Use instead
`ImageOps.box_blur`	`ImageFilter.BoxBlur`
`ImageOps.gaussian_blur`	`ImageFilter.GaussianBlur`
`ImageOps.gblur`	`ImageFilter.GaussianBlur`
`ImageOps.usm`	`ImageFilter.UnsharpMask`
`ImageOps.unsharp_mask`	`ImageFilter.UnsharpMask`

PIL.OleFileIO

Deprecated since version 4.0.0.

Removed in version 6.0.0.

`PIL.OleFileIO` was removed as a vendored file in Pillow 4.0.0 (2017-01) in favour of the upstream olefile Python package, and replaced with an `ImportError` in 5.0.0 (2018-01). The deprecated file has now been removed from Pillow. If needed, install from PyPI (eg. `python3 -m pip install olefile`).

import _imaging

Removed in version 2.1.0.

Pillow >= 2.1.0 no longer supports `import _imaging`. Please use `from PIL.Image import core as _imaging` instead.

Pillow and PIL

Removed in version 1.0.0.

Pillow and PIL cannot co-exist in the same environment. Before installing Pillow, please uninstall PIL.

import Image

Removed in version 1.0.0.

Pillow >= 1.0 no longer supports `import Image`. Please use `from PIL import Image` instead.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

p

PIL, 172
PIL._binary, 217
PIL._deprecate, 218
PIL._imaging, 220
PIL._tkinter_finder, 219
PIL._typing, 219
PIL._util, 219
PIL._version, 220
PIL.AvifImagePlugin, 180
PIL.BdfFontFile, 172
PIL.BmpImagePlugin, 181
PIL.BufrStubImagePlugin, 182
PIL.ContainerIO, 172
PIL.CurImagePlugin, 182
PIL.DcxImagePlugin, 182
PIL.DdsImagePlugin, 183
PIL.EpsImagePlugin, 190
PIL.ExifTags, 164
PIL.features, 169
PIL.FitsImagePlugin, 191
PIL.FliImagePlugin, 191
PIL.FontFile, 173
PIL.FpxImagePlugin, 192
PIL.GbrImagePlugin, 192
PIL.GdImageFile, 174
PIL.GifImagePlugin, 192
PIL.GimpGradientFile, 174
PIL.GimpPaletteFile, 175
PIL.GribStubImagePlugin, 194
PIL.Hdf5StubImagePlugin, 194
PIL.IcnsImagePlugin, 194
PIL.IcoImagePlugin, 196
PIL.Image, 64
PIL.Image.core, 220
PIL.ImageChops, 90
PIL.ImageCms, 94
PIL.ImageColor, 109
PIL.ImageDraw, 110
PIL.ImageDraw2, 175
PIL.ImageEnhance, 123
PIL.ImageFile, 124
PIL.ImageFilter, 128
PIL.ImageFont, 131
PIL.ImageGrab, 140
PIL.ImageMath, 141
PIL.ImageMode, 177
PIL.ImageMorph, 144
PIL.ImageOps, 146
PIL.ImagePalette, 151
PIL.ImagePath, 153
PIL.ImageQt, 154
PIL.ImageSequence, 154
PIL.ImageShow, 155
PIL.ImageStat, 156
PIL.ImageText, 157
PIL.ImageTk, 160
PIL.ImageTransform, 161
PIL.ImageWin, 162
PIL.ImImagePlugin, 197
PIL.ImtImagePlugin, 198
PIL.IptcImagePlugin, 198
PIL.Jpeg2KImagePlugin, 199
PIL.JpegImagePlugin, 198
PIL.JpegPresets, 166
PIL.McIdasImagePlugin, 200
PIL.MicImagePlugin, 200
PIL.MpegImagePlugin, 200
PIL.MpoImagePlugin, 201
PIL.MspImagePlugin, 201
PIL.PaletteFile, 178
PIL.PalmImagePlugin, 202
PIL.PcdImagePlugin, 202
PIL.PcfFontFile, 178
PIL.PcxImagePlugin, 202
PIL.PdfImagePlugin, 202
PIL.PixarImagePlugin, 202
PIL.PngImagePlugin, 202
PIL.PpmImagePlugin, 205
PIL.PsdImagePlugin, 206
PIL.PSDraw, 167
PIL.SgiImagePlugin, 206
PIL.SpiderImagePlugin, 207
PIL.SunImagePlugin, 208

PIL.TarIO, 179
PIL.TgaImagePlugin, 208
PIL.TiffImagePlugin, 208
PIL.TiffTags, 165
PIL.WalImageFile, 179
PIL.WebPImagePlugin, 213
PIL.WmfImagePlugin, 214
PIL.XbmImagePlugin, 214
PIL.XpmImagePlugin, 214
PIL.XVThumbImagePlugin, 214

INDEX

Symbols

_Enhance (*class in PIL.ImageEnhance*), 123
_Tile (*class in PIL.ImageFile*), 124
__getitem__() (*PixelAccess method*), 168
__init__() (*PIL.ImageCms.ImageCmsProfile method*), 94
__init__() (*PIL.ImageStat.Stat method*), 156
__init__() (*PIL.TiffTags.TagInfo method*), 165
__new__() (*PIL.PngImagePlugin.iTXt method*), 178
__setitem__() (*PixelAccess method*), 169
__version__ (*in module PIL._version*), 220

A

A1 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A16B16G16R16 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A16B16G16R16F (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A1R5G5B5 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A2B10G10R10 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A2B10G10R10_XR_BIAS (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A2R10G10B10 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A2W10V10U10 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A32B32G32R32F (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A4L4 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A4R4G4B4 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
A8B8G8R8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A8L8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A8P8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A8P8 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
A8R3G3B2 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
A8R8G8B8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
ABSOLUTE_COLORIMETRIC (*PIL.ImageCms.Intent attribute*), 94
ADAPTIVE (*PIL.Image.Palette attribute*), 90
add() (*in module PIL.ImageChops*), 90
add() (*PIL.PngImagePlugin.PngInfo method*), 179
add_itxt() (*PIL.PngImagePlugin.PngInfo method*), 179
add_modulo() (*in module PIL.ImageChops*), 91
add_patterns() (*PIL.ImageMorph.LutBuilder method*), 145
add_text() (*PIL.PngImagePlugin.PngInfo method*), 179
adopt() (*PIL.MpoImagePlugin.MpoImageFile static method*), 201
AFFINE (*PIL.Image.Transform attribute*), 89
AffineTransform (*class in PIL.ImageTransform*), 161
AI44 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
all_frames() (*in module PIL.ImageSequence*), 154
ALPHA (*PIL.DdsImagePlugin.DDPF attribute*), 185
alpha_composite() (*in module PIL.Image*), 65
alpha_composite() (*PIL.Image.Image method*), 72
ALPHAPIXELS (*PIL.DdsImagePlugin.DDPF attribute*), 185
APP() (*in module PIL.JpegImagePlugin*), 198
AppendingTiffWriter (*class in PIL.TiffImagePlugin*), 208
apply() (*PIL.ImageCms.ImageCmsTransform method*), 94
apply() (*PIL.ImageMorph.MorphOp method*), 145
apply_in_place() (*PIL.ImageCms.ImageCmsTransform method*), 94
apply_transparency() (*PIL.Image.Image method*), 72
applyTransform() (*in module PIL.ImageCms*), 96
arc() (*PIL.ImageDraw.ImageDraw method*), 113
arc() (*PIL.ImageDraw2.Draw method*), 176

args (*PIL.ImageFile._Tile attribute*), 125
ATI1 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
ATI2 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
attributes (*PIL.ImageCms.core.CmsProfile attribute*), 106
autocontrast() (*in module PIL.ImageOps*), 146
AvifImageFile (*class in PIL.AvifImagePlugin*), 180
Axis (*class in PIL.ImageFont*), 140
AYUV (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186

B

B4G4R4A4_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B5G5R5A1_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B5G6R5_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B8G8R8A8_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B8G8R8A8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B8G8R8A8_UNORM_SRGB (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B8G8R8X8_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B8G8R8X8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
B8G8R8X8_UNORM_SRGB (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
bands (*PIL.ImageMode.ModeDescriptor attribute*), 177
Base (*in module PIL.ExifTags*), 164
BaseImageFont (*class in PIL.ImageFont*), 134
basemode (*PIL.ImageMode.ModeDescriptor attribute*), 178
basetype (*PIL.ImageMode.ModeDescriptor attribute*), 178
BASIC (*PIL.ImageFont.Layout attribute*), 140
BC1_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 186
BC1_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC1_UNORM_SRGB (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC2_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC2_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC2_UNORM_SRGB (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC3_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC3_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC3_UNORM_SRGB (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC4_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC4_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC4_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC4S (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
BC4U (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
BC5_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC5_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC5_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC5S (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
BC5U (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
BC6H_SF16 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC6H_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC6H_UF16 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC7_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC7_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
BC7_UNORM_SRGB (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
bdf_char() (*in module PIL.BdfFontFile*), 172
BdfFontFile (*class in PIL.BdfFontFile*), 172
begin_document() (*PIL.PSDraw.PSDraw method*), 167
bestsize() (*PIL.IcnsImagePlugin.IcnsFile method*), 195
BICUBIC (*PIL.Image.Resampling attribute*), 89
bigtiff (*PIL.Image.Exif attribute*), 87
BILINEAR (*PIL.Image.Resampling attribute*), 89
BINARYBUFFER (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
BITFIELDS (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
bitmap (*PIL.FontFile.FontFile attribute*), 173
bitmap() (*PIL.ImageDraw.ImageDraw method*), 113
BitmapImage (*class in PIL.ImageTk*), 160
BitStream (*class in PIL.MpegImagePlugin*), 200
BLACKPOINTCOMPENSATION (*PIL.ImageCms.Flags attribute*), 95
Blend (*class in PIL.PngImagePlugin*), 202
blend() (*in module PIL.Image*), 66
blend() (*in module PIL.ImageChops*), 91
blue_colorant (*PIL.ImageCms.core.CmsProfile*

attribute), 107
blue_primary (*PIL.ImageCms.core.CmsProfile attribute*), 108
BmpImageFile (*class in PIL.BmpImagePlugin*), 181
BmpRleDecoder (*class in PIL.BmpImagePlugin*), 181
BOX (*PIL.Image.Resampling attribute*), 89
BoxBlur (*class in PIL.ImageFilter*), 129
BoxReader (*class in PIL.Jpeg2KImagePlugin*), 199
bpp (*PIL.IcoImagePlugin.IconHeader attribute*), 196
Brightness (*class in PIL.ImageEnhance*), 124
Brush (*class in PIL.ImageDraw2*), 175
Buffer (*class in PIL._typing*), 219
BufrStubImageFile (*class in PIL.BufrStubImagePlugin*), 182
build_default_lut() (*PIL.ImageMorph.LutBuilder method*), 145
build_lut() (*PIL.ImageMorph.LutBuilder method*), 145
build_prototype_image() (*in module PIL.PalmImagePlugin*), 202
buildProofTransform() (*in module PIL.ImageCms*), 96
buildProofTransformFromOpenProfiles() (*in module PIL.ImageCms*), 98
buildTransform() (*in module PIL.ImageCms*), 99
buildTransformFromOpenProfiles() (*in module PIL.ImageCms*), 100

C

call() (*PIL.PngImagePlugin.ChunkStream method*), 203
CAPS (*PIL.DdsImagePlugin.DDSD attribute*), 186
check() (*in module PIL.features*), 169
check_codec() (*in module PIL.features*), 170
check_feature() (*in module PIL.features*), 171
check_module() (*in module PIL.features*), 169
check_text_memory() (*PIL.PngImagePlugin.PngStream method*), 204
chord() (*PIL.ImageDraw.ImageDraw method*), 114
chord() (*PIL.ImageDraw2.Draw method*), 176
chromatic_adaption (*PIL.ImageCms.core.CmsProfile attribute*), 107
chromaticity (*PIL.ImageCms.core.CmsProfile attribute*), 107
chunk_acTL() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_cHRM() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_eXIf() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_fcTL() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_fdAT() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_gAMA() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_iCCP() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_IDAT() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_IEND() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_IHDR() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_iTXt() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_pHYs() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_PLTE() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_sRGB() (*PIL.PngImagePlugin.PngStream method*), 204
chunk_tEXt() (*PIL.PngImagePlugin.PngStream method*), 205
chunk_tRNS() (*PIL.PngImagePlugin.PngStream method*), 205
chunk_zTXt() (*PIL.PngImagePlugin.PngStream method*), 205
chunks (*PIL.PngImagePlugin.PngInfo attribute*), 179
ChunkStream (*class in PIL.PngImagePlugin*), 202
circle() (*PIL.ImageDraw.ImageDraw method*), 114
cleanup() (*PIL.ImageFile.PyCodec method*), 125
close() (*PIL.ContainerIO.ContainerIO method*), 172
close() (*PIL.FpxImagePlugin.FpxImageFile method*), 192
close() (*PIL.Image.Image method*), 85
close() (*PIL.ImageFile.ImageFile method*), 127
close() (*PIL.ImageFile.Parser method*), 125
close() (*PIL.MicImagePlugin.MicImageFile method*), 200
close() (*PIL.PngImagePlugin.ChunkStream method*), 203
close() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
clut (*PIL.ImageCms.core.CmsProfile attribute*), 109
CLUT_POST_LINEARIZATION (*PIL.ImageCms.Flags attribute*), 95
CLUT_PRE_LINEARIZATION (*PIL.ImageCms.Flags attribute*), 95
CmsProfile (*class in PIL.ImageCms.core*), 106
codec_name (*PIL.ImageFile._Tile attribute*), 124
Color (*class in PIL.ImageEnhance*), 123
Color3DLUT (*class in PIL.ImageFilter*), 128
color_depth (*PIL.IcoImagePlugin.IconHeader attribute*), 196
colorant_table (*PIL.ImageCms.core.CmsProfile at-*

tribute), 107
colorant_table_out (*PIL.ImageCms.core.CmsProfile attribute*), 107
colorimetric_intent (*PIL.ImageCms.core.CmsProfile attribute*), 108
colorize() (*in module PIL.ImageOps*), 147
COM() (*in module PIL.JpegImagePlugin*), 198
Common Vulnerabilities and Exposures
CVE 2014-1932, 317
CVE 2014-1933, 317
CVE 2014-3589, 316, 317
CVE 2016-0740, 311
CVE 2016-0775, 311
CVE 2016-2533, 312
CVE 2016-3076, 310
CVE 2019-16865, 290, 291
CVE 2019-19911, 290
CVE 2020-10177, 286
CVE 2020-10378, 286
CVE 2020-10379, 286
CVE 2020-10994, 286
CVE 2020-11538, 286
CVE 2020-15999, 240, 246, 272, 281, 321, 327
CVE 2020-35653, 280
CVE 2020-35654, 279, 280
CVE 2020-35655, 280
CVE 2020-5310, 290
CVE 2020-5311, 286, 290
CVE 2020-5312, 290
CVE 2020-5313, 290
CVE 2021-23437, 274
CVE 2021-25287, 276
CVE 2021-25288, 276
CVE 2021-25289, 279
CVE 2021-25290, 280
CVE 2021-25291, 280
CVE 2021-25292, 280
CVE 2021-25293, 280
CVE 2021-27921, 279
CVE 2021-27922, 279
CVE 2021-27923, 279
CVE 2021-28675, 276
CVE 2021-28676, 277
CVE 2021-28677, 277
CVE 2021-28678, 277
CVE 2021-34552, 275
CVE 2022-22815, 271
CVE 2022-22816, 271
CVE 2022-22817, 270, 271
CVE 2022-24303, 270
CVE 2022-30595, 266
CVE 2023-44271, 256
CVE 2023-4863, 255
CVE 2023-50447, 252
CVE 2024-28219, 249
CVE 2025-48379, 241
CVE 2026-25990, 236, 237
CVE 2026-40192, 236
CVE 2026-42308, 236
CVE 2026-42309, 236
CVE 2026-42310, 236
CVE 2026-42311, 236
CVE 2026-54059, 235
CVE 2026-54060, 235
CVE 2026-55379, 235
CVE 2026-55380, 235
CVE 2026-55798, 234
Common Weakness Enumeration
CWE 126, 271
CWE 665, 271
compact() (*PIL.ImagePath.PIL.ImagePath.Path method*), 153
compile() (*PIL.FontFile.FontFile method*), 173
COMPLEX (*PIL.DdsImagePlugin.DDSCAPS attribute*), 185
composite() (*in module PIL.Image*), 66
composite() (*in module PIL.ImageChops*), 91
COMPRESSIONS (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
connection_space (*PIL.ImageCms.core.CmsProfile attribute*), 106
constant() (*in module PIL.ImageChops*), 91
contain() (*in module PIL.ImageOps*), 150
ContainerIO (*class in PIL.ContainerIO*), 172
Contrast (*class in PIL.ImageEnhance*), 123
convert() (*PIL.Image.Image method*), 72
convert2byte() (*PIL.SpiderImagePlugin.SpiderImageFile method*), 207
copy() (*PIL.Image.Image method*), 73
COPY_ALPHA (*PIL.ImageCms.Flags attribute*), 96
copyright (*PIL.ImageCms.core.CmsProfile attribute*), 106
count (*PIL.ImageStat.Stat property*), 156
cover() (*in module PIL.ImageOps*), 150
crc() (*PIL.PngImagePlugin.ChunkStream method*), 203
crc_skip() (*PIL.PngImagePlugin.ChunkStream method*), 203
createProfile() (*in module PIL.ImageCms*), 101
creation_date (*PIL.ImageCms.core.CmsProfile attribute*), 106
crop() (*in module PIL.ImageOps*), 147
crop() (*PIL.Image.Image method*), 73
CUBEMAP (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 185
CUBEMAP_NEGATIVEX (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 185

CUBEMAP_NEGATIVEY (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 186
CUBEMAP_NEGATIVEZ (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 186
CUBEMAP_POSITIVEX (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 186
CUBEMAP_POSITIVEY (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 186
CUBEMAP_POSITIVEZ (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 186
CurImageFile (*class in PIL.CurImagePlugin*), 182
curved() (*in module PIL.GimpGradientFile*), 175
custom_mimetype (*PIL.ImageFile.ImageFile attribute*), 127
cvt_enum() (*PIL.TiffTags.TagInfo method*), 165
CxV8U8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183

D

D15S1 (*PIL.DdsImagePlugin.D3DFMT attribute*), 183
D16 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D16_LOCKABLE (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D16_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
D24_UNORM_S8_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
D24FS8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D24S8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D24X4S4 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D24X8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D32 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D32_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
D32_FLOAT_S8X24_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
D32_LOCKABLE (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D32F_LOCKABLE (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
D3DFMT (*class in PIL.DdsImagePlugin*), 183
darker() (*in module PIL.ImageChops*), 91
data() (*PIL.GifImagePlugin.GifImageFile method*), 192
dataforsize() (*PIL.IcnsImagePlugin.IcnsFile method*), 195
DcxImageFile (*class in PIL.DcxImagePlugin*), 182
DDPF (*class in PIL.DdsImagePlugin*), 185
DDSCAPS (*class in PIL.DdsImagePlugin*), 185
DDSCAPS2 (*class in PIL.DdsImagePlugin*), 185
DDSD (*class in PIL.DdsImagePlugin*), 186
DdsImageFile (*class in PIL.DdsImagePlugin*), 190
DdsRgbDecoder (*class in PIL.DdsImagePlugin*), 190
decode() (*PIL.BmpImagePlugin.BmpRleDecoder method*), 181
decode() (*PIL.DdsImagePlugin.DdsRgbDecoder method*), 190
decode() (*PIL.FitsImagePlugin.FitsGzipDecoder method*), 191
decode() (*PIL.ImageFile.PyDecoder method*), 126
decode() (*PIL.MspImagePlugin.MspDecoder method*), 201
decode() (*PIL.PpmImagePlugin.PpmDecoder method*), 205
decode() (*PIL.PpmImagePlugin.PpmPlainDecoder method*), 206
decode() (*PIL.SgiImagePlugin.SGI16Decoder method*), 206
decode() (*PIL.XpmImagePlugin.XpmDecoder method*), 214
decoderconfig (*PIL.ImageFile.ImageFile attribute*), 127
DecoderInput (*in module PIL.Image*), 88
default (*PIL.ImageFont.Axis attribute*), 140
DeferredError (*class in PIL._util*), 219
deform() (*in module PIL.ImageOps*), 148
denominator (*PIL.TiffImagePlugin.IFDRational property*), 209
deprecate() (*in module PIL._deprecate*), 218
DEPTH (*PIL.DdsImagePlugin.DDSD attribute*), 186
device_class (*PIL.ImageCms.core.CmsProfile attribute*), 106
Dib (*class in PIL.ImageWin*), 163
DibImageFile (*class in PIL.BmpImagePlugin*), 181
difference() (*in module PIL.ImageChops*), 91
dim (*PIL.IcoImagePlugin.IconHeader attribute*), 196
Direction (*class in PIL.ImageCms*), 95
Disposal (*class in PIL.PngImagePlugin*), 203
Dither (*class in PIL.Image*), 89
DQT() (*in module PIL.JpegImagePlugin*), 198
draft() (*PIL.Image.Image method*), 73
draft() (*PIL.JpegImagePlugin.JpegImageFile method*), 198
Draw (*class in PIL.ImageDraw2*), 175
Draw() (*in module PIL.ImageDraw*), 112
draw() (*PIL.ImageWin.Dib method*), 163
duplicate() (*in module PIL.ImageChops*), 91
DX10 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
DXGI_FORMAT (*class in PIL.DdsImagePlugin*), 186
DXT1 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
DXT2 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
DXT3 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
DXT4 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
DXT5 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184

E

effect_mandelbrot() (*in module PIL.Image*), 69
effect_noise() (*in module PIL.Image*), 69
effect_spread() (*PIL.Image.Image method*), 74

ellipse() (*PIL.ImageDraw.ImageDraw method*), 114
ellipse() (*PIL.ImageDraw2.Draw method*), 176
embed_color() (*PIL.ImageText.Text method*), 158
encode() (*PIL.ImageFile.PyEncoder method*), 126
encode_to_file() (*PIL.ImageFile.PyEncoder method*), 126
encode_to_pyfd() (*PIL.ImageFile.PyEncoder method*), 127
end_document() (*PIL.PSDraw.PSDraw method*), 167
endian (*PIL.Image.Exif attribute*), 87
enhance() (*PIL.ImageEnhance._Enhance method*), 123
entropy() (*PIL.Image.Image method*), 74
EPSILON (*in module PIL.GimpGradientFile*), 174
EpsImageFile (*class in PIL.EpsImagePlugin*), 190
equalize() (*in module PIL.ImageOps*), 148
ERRORS (*in module PIL.ImageFile*), 128
eval() (*in module PIL.Image*), 66
Exif (*class in PIL.Image*), 87
exif_transpose() (*in module PIL.ImageOps*), 149
expand() (*in module PIL.ImageOps*), 148
expose() (*PIL.ImageWin.Dib method*), 163
EXTENT (*PIL.Image.Transform attribute*), 89
extents (*PIL.ImageFile._Tile attribute*), 124
ExtentTransform (*class in PIL.ImageTransform*), 162
extrema (*PIL.ImageStat.Stat property*), 156

F

f (*PIL.TiffImagePlugin.AppendingTiffWriter attribute*), 208
FASTOCTREE (*PIL.Image.Quantize attribute*), 90
feed() (*PIL.ImageFile.Parser method*), 125
field() (*PIL.IptcImagePlugin.IptcImageFile method*), 198
fieldSizes (*PIL.TiffImagePlugin.AppendingTiffWriter attribute*), 208
filename (*PIL.Image.Image attribute*), 86
fileno() (*PIL.ContainerIO.ContainerIO method*), 172
fill (*PIL.ImageDraw.ImageDraw attribute*), 113
Filter (*class in PIL.ImageFilter*), 131
filter() (*PIL.Image.Image method*), 74
filter() (*PIL.ImageFilter.Filter method*), 131
filter() (*PIL.ImageFilter.MultibandFilter method*), 131
finalize() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
fit() (*in module PIL.ImageOps*), 151
FitsGzipDecoder (*class in PIL.FitsImagePlugin*), 191
FitsImageFile (*class in PIL.FitsImagePlugin*), 191
fixIFD() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
fixOffsets() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
Flags (*class in PIL.ImageCms*), 95
FliImageFile (*class in PIL.FliImagePlugin*), 191
flip() (*in module PIL.ImageOps*), 149
FLIP_LEFT_RIGHT (*PIL.Image.Transpose attribute*), 88
FLIP_TOP_BOTTOM (*PIL.Image.Transpose attribute*), 88
floodfill() (*in module PIL.ImageDraw*), 122
FLOYDSTEINBERG (*PIL.Image.Dither attribute*), 90
flush() (*PIL.ContainerIO.ContainerIO method*), 172
flush() (*PIL.ImageDraw2.Draw method*), 175
Font (*class in PIL.ImageDraw2*), 175
font (*PIL.ImageDraw.ImageDraw attribute*), 113
font_variant() (*PIL.ImageFont.FreeTypeFont method*), 135
FontFile (*class in PIL.FontFile*), 173
fontmode (*PIL.ImageDraw.ImageDraw attribute*), 113
FORCE_CLUT (*PIL.ImageCms.Flags attribute*), 95
format (*PIL.AvifImagePlugin.AvifImageFile attribute*), 180
format (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
format (*PIL.BmpImagePlugin.DibImageFile attribute*), 181
format (*PIL.BufrStubImagePlugin.BufrStubImageFile attribute*), 182
format (*PIL.CurImagePlugin.CurImageFile attribute*), 182
format (*PIL.DcxImagePlugin.DcxImageFile attribute*), 182
format (*PIL.DdsImagePlugin.DdsImageFile attribute*), 190
format (*PIL.EpsImagePlugin.EpsImageFile attribute*), 190
format (*PIL.FitsImagePlugin.FitsImageFile attribute*), 191
format (*PIL.FliImagePlugin.FliImageFile attribute*), 191
format (*PIL.FpxImagePlugin.FpxImageFile attribute*), 192
format (*PIL.GbrImagePlugin.GbrImageFile attribute*), 192
format (*PIL.GdImageFile.GdImageFile attribute*), 174
format (*PIL.GifImagePlugin.GifImageFile attribute*), 192
format (*PIL.GribStubImagePlugin.GribStubImageFile attribute*), 194
format (*PIL.Hdf5StubImagePlugin.HDF5StubImageFile attribute*), 194
format (*PIL.IcnsImagePlugin.IcnsImageFile attribute*), 195
format (*PIL.IcoImagePlugin.IcoImageFile attribute*), 196
format (*PIL.Image.Image attribute*), 86
format (*PIL.ImageShow.Viewer attribute*), 156
format (*PIL.ImImagePlugin.ImImageFile attribute*), 197
format (*PIL.ImtImagePlugin.ImtImageFile attribute*), 198

format (*PIL.IptcImagePlugin.IptcImageFile attribute*), 198
format (*PIL.Jpeg2KImagePlugin.Jpeg2KImageFile attribute*), 199
format (*PIL.JpegImagePlugin.JpegImageFile attribute*), 199
format (*PIL.McIdasImagePlugin.McIdasImageFile attribute*), 200
format (*PIL.MicImagePlugin.MicImageFile attribute*), 200
format (*PIL.MpegImagePlugin.MpegImageFile attribute*), 200
format (*PIL.MpoImagePlugin.MpoImageFile attribute*), 201
format (*PIL.MspImagePlugin.MspImageFile attribute*), 201
format (*PIL.PcdImagePlugin.PcdImageFile attribute*), 202
format (*PIL.PcxImagePlugin.PcxImageFile attribute*), 202
format (*PIL.PixarImagePlugin.PixarImageFile attribute*), 202
format (*PIL.PngImagePlugin.PngImageFile attribute*), 204
format (*PIL.PpmImagePlugin.PpmImageFile attribute*), 205
format (*PIL.PsdImagePlugin.PsdImageFile attribute*), 206
format (*PIL.SgiImagePlugin.SgiImageFile attribute*), 207
format (*PIL.SpiderImagePlugin.SpiderImageFile attribute*), 207
format (*PIL.SunImagePlugin.SunImageFile attribute*), 208
format (*PIL.TgaImagePlugin.TgaImageFile attribute*), 208
format (*PIL.TiffImagePlugin.TiffImageFile attribute*), 212
format (*PIL.WalImageFile.WalImageFile attribute*), 180
format (*PIL.WebPImagePlugin.WebPImageFile attribute*), 213
format (*PIL.WmfImagePlugin.WmfStubImageFile attribute*), 214
format (*PIL.XbmImagePlugin.XbmImageFile attribute*), 214
format (*PIL.XpmImagePlugin.XpmImageFile attribute*), 214
format (*PIL.XVThumbImagePlugin.XVThumbImageFile attribute*), 214
format_description (*PIL.AvifImagePlugin.AvifImageFile attribute*), 180
format_description (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
format_description (*PIL.BmpImagePlugin.DibImageFile attribute*), 181
format_description (*PIL.BufrStubImagePlugin.BufrStubImageFile attribute*), 182
format_description (*PIL.CurImagePlugin.CurImageFile attribute*), 182
format_description (*PIL.DcxImagePlugin.DcxImageFile attribute*), 182
format_description (*PIL.DdsImagePlugin.DdsImageFile attribute*), 190
format_description (*PIL.EpsImagePlugin.EpsImageFile attribute*), 191
format_description (*PIL.FitsImagePlugin.FitsImageFile attribute*), 191
format_description (*PIL.FliImagePlugin.FliImageFile attribute*), 191
format_description (*PIL.FpxImagePlugin.FpxImageFile attribute*), 192
format_description (*PIL.GbrImagePlugin.GbrImageFile attribute*), 192
format_description (*PIL.GdImageFile.GdImageFile attribute*), 174
format_description (*PIL.GifImagePlugin.GifImageFile attribute*), 192
format_description (*PIL.GribStubImagePlugin.GribStubImageFile attribute*), 194
format_description (*PIL.Hdf5StubImagePlugin.HDF5StubImageFile attribute*), 194
format_description (*PIL.IcnsImagePlugin.IcnsImageFile attribute*), 195
format_description (*PIL.IcoImagePlugin.IcoImageFile attribute*), 196
format_description (*PIL.ImImagePlugin.ImImageFile attribute*), 197
format_description (*PIL.ImtImagePlugin.ImtImageFile attribute*), 198
format_description (*PIL.IptcImagePlugin.IptcImageFile attribute*), 198
format_description (*PIL.Jpeg2KImagePlugin.Jpeg2KImageFile attribute*), 199
format_description (*PIL.JpegImagePlugin.JpegImageFile attribute*), 199
format_description (*PIL.McIdasImagePlugin.McIdasImageFile attribute*), 200
format_description (*PIL.MicImagePlugin.MicImageFile attribute*), 200
format_description (*PIL.MpegImagePlugin.MpegImageFile attribute*), 200
format_description (*PIL.MpoImagePlugin.MpoImageFile attribute*), 201
format_description (*PIL.MspImagePlugin.MspImageFile attribute*), 202
format_description (*PIL.PcdImagePlugin.PcdImageFile attribute*), 202
format_description (*PIL.PcxImagePlugin.PcxImageFile*

attribute), 202
format_description (*PIL.PixarImagePlugin.PixarImageFile attribute*), 202
format_description (*PIL.PngImagePlugin.PngImageFile attribute*), 204
format_description (*PIL.PpmImagePlugin.PpmImageFile attribute*), 206
format_description (*PIL.PsdImagePlugin.PsdImageFile attribute*), 206
format_description (*PIL.SgiImagePlugin.SgiImageFile attribute*), 207
format_description (*PIL.SpiderImagePlugin.SpiderImageFile attribute*), 207
format_description (*PIL.SunImagePlugin.SunImageFile attribute*), 208
format_description (*PIL.TgaImagePlugin.TgaImageFile attribute*), 208
format_description (*PIL.TiffImagePlugin.TiffImageFile attribute*), 212
format_description (*PIL.WalImageFile.WalImageFile attribute*), 180
format_description (*PIL.WebPImagePlugin.WebPImageFile attribute*), 213
format_description (*PIL.WmfImagePlugin.WmfStubImageFile attribute*), 214
format_description (*PIL.XbmImagePlugin.XbmImageFile attribute*), 214
format_description (*PIL.XpmImagePlugin.XpmImageFile attribute*), 215
format_description (*PIL.XVThumbImagePlugin.XVThumbImageFile attribute*), 214
FOURCC (*PIL.DdsImagePlugin.DDPF attribute*), 185
fp (*PIL.ImageFile.ImageFile attribute*), 127
fp (*PIL.PngImagePlugin.ChunkStream attribute*), 203
FpxImageFile (*class in PIL.FpxImagePlugin*), 192
frame() (*PIL.IcoImagePlugin.IcoFile method*), 196
FreeTypeFont (*class in PIL.ImageFont*), 135
from_v2() (*PIL.TiffImagePlugin.ImageFileDirectory_v1 class method*), 210
fromarray() (*in module PIL.Image*), 67
fromarrow() (*in module PIL.Image*), 67
frombuffer() (*in module PIL.Image*), 68
frombytes() (*in module PIL.Image*), 68
frombytes() (*PIL.GimpPaletteFile.GimpPaletteFile class method*), 175
frombytes() (*PIL.Image.Image method*), 74
frombytes() (*PIL.ImageWin.Dib method*), 163

G

G16R16 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
G16R16F (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
G32R32F (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
G8R8_G8B8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
G8R8_G8B8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
GAMUTCHECK (*PIL.ImageCms.Flags attribute*), 95
GaussianBlur (*class in PIL.ImageFilter*), 130
GbrImageFile (*class in PIL.GbrImagePlugin*), 192
GdImageFile (*class in PIL.GdImageFile*), 174
generate() (*PIL.ImageFilter.Color3DLUT class method*), 129
get_bbox() (*PIL.ImageText.Text method*), 158
get_child_images() (*PIL.ImageFile.ImageFile method*), 127
get_codec_version() (*in module PIL.AvifImagePlugin*), 181
get_command() (*PIL.ImageShow.Viewer method*), 156
get_display_profile() (*in module PIL.ImageCms*), 104
get_flattened_data() (*PIL.Image.Image method*), 76
get_format() (*PIL.ImageShow.Viewer method*), 156
get_format_mimetype() (*PIL.ImageFile.ImageFile method*), 127
get_ifd() (*PIL.Image.Exif method*), 87
get_interlace() (*in module PIL.GifImagePlugin*), 193
get_length() (*PIL.ImageText.Text method*), 158
get_lut() (*PIL.ImageMorph.LutBuilder method*), 145
get_on_pixels() (*PIL.ImageMorph.MorphOp method*), 146
get_photoshop_blocks() (*PIL.TiffImagePlugin.TiffImageFile method*), 212
get_sampling() (*in module PIL.JpegImagePlugin*), 199
get_supported() (*in module PIL.features*), 169
get_supported_codecs() (*in module PIL.features*), 171
get_supported_features() (*in module PIL.features*), 171
get_supported_modules() (*in module PIL.features*), 170
get_variation_axes() (*PIL.ImageFont.FreeTypeFont method*), 135
get_variation_names() (*PIL.ImageFont.FreeTypeFont method*), 135
getbands() (*PIL.Image.Image method*), 74
getbbox() (*PIL.Image.Image method*), 75
getbbox() (*PIL.ImageFont.FreeTypeFont method*), 135
getbbox() (*PIL.ImageFont.ImageFont method*), 134
getbbox() (*PIL.ImageFont.TransposedFont method*), 139
getbbox() (*PIL.ImagePath.PIL.ImagePath.Path method*), 153
getchannel() (*PIL.Image.Image method*), 75
getchunks() (*in module PIL.PngImagePlugin*), 205
getcolor() (*in module PIL.ImageColor*), 110
getcolor() (*PIL.ImagePalette.ImagePalette method*), 152

getcolors() (*PIL.Image.Image method*), 75
getdata() (*in module PIL.GifImagePlugin*), 193
getdata() (*PIL.Image.Image method*), 75
getdata() (*PIL.ImagePalette.ImagePalette method*), 152
getdata() (*PIL.ImageTransform.Transform method*), 161
getDefaultIntent() (*in module PIL.ImageCms*), 101
getdraw() (*in module PIL.ImageDraw*), 122
getentryindex() (*PIL.IcoImagePlugin.IcoFile method*), 196
getexif() (*PIL.Image.Image method*), 76
getexif() (*PIL.PngImagePlugin.PngImageFile method*), 203
getextrema() (*PIL.Image.Image method*), 76
getfont() (*PIL.ImageDraw.ImageDraw method*), 113
getheader() (*in module PIL.GifImagePlugin*), 194
getimage() (*PIL.IcnsImagePlugin.IcnsFile method*), 195
getimage() (*PIL.IcoImagePlugin.IcoFile method*), 196
getint() (*PIL.IptcImagePlugin.IptcImageFile method*), 198
getiptcinfo() (*in module PIL.IptcImagePlugin*), 198
getlength() (*PIL.ImageFont.FreeTypeFont method*), 136
getlength() (*PIL.ImageFont.ImageFont method*), 134
getlength() (*PIL.ImageFont.TransposedFont method*), 139
getmask() (*PIL.ImageFont.FreeTypeFont method*), 137
getmask() (*PIL.ImageFont.ImageFont method*), 134
getmask() (*PIL.ImageFont.TransposedFont method*), 139
getmask2() (*PIL.ImageFont.FreeTypeFont method*), 138
getmetrics() (*PIL.ImageFont.FreeTypeFont method*), 139
getmode() (*in module PIL.ImageMode*), 178
getname() (*PIL.ImageFont.FreeTypeFont method*), 139
getOpenProfile() (*in module PIL.ImageCms*), 102
getpalette() (*PIL.GimpGradientFile.GradientFile method*), 174
getpalette() (*PIL.GimpPaletteFile.GimpPaletteFile method*), 175
getpalette() (*PIL.Image.Image method*), 76
getpalette() (*PIL.PaletteFile.PaletteFile method*), 178
getpixel() (*PIL.Image.Image method*), 76
getProfileCopyright() (*in module PIL.ImageCms*), 102
getProfileDescription() (*in module PIL.ImageCms*), 102
getProfileInfo() (*in module PIL.ImageCms*), 103
getProfileManufacturer() (*in module PIL.ImageCms*), 103
getProfileModel() (*in module PIL.ImageCms*), 103
getProfileName() (*in module PIL.ImageCms*), 104
getprojection() (*PIL.Image.Image method*), 76
getrgb() (*in module PIL.ImageColor*), 110
getxmp() (*PIL.Image.Image method*), 76
Ghostscript() (*in module PIL.EpsImagePlugin*), 191
GifImageFile (*class in PIL.GifImagePlugin*), 192
GimpGradientFile (*class in PIL.GimpGradientFile*), 174
GimpPaletteFile (*class in PIL.GimpPaletteFile*), 175
global_palette (*PIL.GifImagePlugin.GifImageFile attribute*), 192
goToEnd() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
GPS (*in module PIL.ExifTags*), 164
GPSTAGS (*in module PIL.ExifTags*), 165
grab() (*in module PIL.ImageGrab*), 140
grabclipboard() (*in module PIL.ImageGrab*), 141
gradient (*PIL.GimpGradientFile.GradientFile attribute*), 174
GradientFile (*class in PIL.GimpGradientFile*), 174
grayscale() (*in module PIL.ImageOps*), 149
green_colorant (*PIL.ImageCms.core.CmsProfile attribute*), 107
green_primary (*PIL.ImageCms.core.CmsProfile attribute*), 108
GribStubImageFile (*class in PIL.GribStubImagePlugin*), 194
GRIDPOINTS() (*PIL.ImageCms.Flags static method*), 96
GUESSDEVICECLASS (*PIL.ImageCms.Flags attribute*), 95

H

HAMMING (*PIL.Image.Resampling attribute*), 89
hard_light() (*in module PIL.ImageChops*), 93
has_ghostscript() (*in module PIL.EpsImagePlugin*), 191
has_next_box() (*PIL.Jpeg2KImagePlugin.BoxReader method*), 199
has_transparency_data (*PIL.Image.Image attribute*), 87
HDC (*class in PIL.ImageWin*), 164
HDF5StubImageFile (*class in PIL.Hdf5StubImagePlugin*), 194
header_flags (*PIL.ImageCms.core.CmsProfile attribute*), 106
header_manufacturer (*PIL.ImageCms.core.CmsProfile attribute*), 106
header_model (*PIL.ImageCms.core.CmsProfile attribute*), 106
HEIGHT (*PIL.DdsImagePlugin.DDSD attribute*), 186
height (*PIL.IcoImagePlugin.IconHeader attribute*), 196
height (*PIL.Image.Image attribute*), 86
height() (*PIL.ImageTk.BitmapImage method*), 160
height() (*PIL.ImageTk.PhotoImage method*), 160
hide_offsets() (*PIL.Image.Exif method*), 88

HIGHRESPRECALC (*PIL.ImageCms.Flags attribute*), 95
histogram() (*PIL.Image.Image method*), 77
HWND (*class in PIL.ImageWin*), 164

I

i16be() (*in module PIL._binary*), 217
i16le() (*in module PIL._binary*), 217
i32be() (*in module PIL._binary*), 217
i32le() (*in module PIL._binary*), 217
i8() (*in module PIL._binary*), 218
IA44 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
icc_version (*PIL.ImageCms.core.CmsProfile attribute*), 106
IcnsFile (*class in PIL.IcnsImagePlugin*), 194
IcnsImageFile (*class in PIL.IcnsImagePlugin*), 195
IcoFile (*class in PIL.IcoImagePlugin*), 196
IcoImageFile (*class in PIL.IcoImagePlugin*), 196
IconHeader (*class in PIL.IcoImagePlugin*), 196
IFD (*in module PIL.ExifTags*), 164
IFDRational (*class in PIL.TiffImagePlugin*), 209
im_custom_mimetype (*PIL.PngImagePlugin.PngStream attribute*), 205
im_info (*PIL.PngImagePlugin.PngStream attribute*), 205
im_n_frames (*PIL.PngImagePlugin.PngStream attribute*), 205
im_palette (*PIL.PngImagePlugin.PngStream attribute*), 205
im_text (*PIL.PngImagePlugin.PngStream attribute*), 205
im_tile (*PIL.PngImagePlugin.PngStream attribute*), 205
Image (*class in PIL.Image*), 71
image() (*PIL.PSDraw.PSDraw method*), 167
ImageCmsProfile (*class in PIL.ImageCms*), 94
ImageCmsTransform (*class in PIL.ImageCms*), 94
ImageFile (*class in PIL.ImageFile*), 127
ImageFileDirectory (*in module PIL.TiffImagePlugin*), 209
ImageFileDirectory_v1 (*class in PIL.TiffImagePlugin*), 209
ImageFileDirectory_v2 (*class in PIL.TiffImagePlugin*), 210
ImageFont (*class in PIL.ImageFont*), 134
ImagePalette (*class in PIL.ImagePalette*), 152
ImagePointHandler (*class in PIL.Image*), 88
ImagePointTransform (*class in PIL.Image*), 88
ImageQt (*class in PIL.ImageQt*), 154
ImageTransformHandler (*class in PIL.Image*), 88
ImagingCore (*class in PIL.Image.core*), 220
ImImageFile (*class in PIL.ImImagePlugin*), 197
ImtImageFile (*class in PIL.ImtImagePlugin*), 198
INDEX16 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
INDEX32 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
info (*PIL.Image.Image attribute*), 86
init() (*in module PIL.Image*), 70
init() (*PIL.ImageFile.PyCodec method*), 125
ink (*PIL.ImageDraw.ImageDraw attribute*), 113
INPUT (*PIL.ImageCms.Direction attribute*), 95
IntegralLike (*class in PIL._typing*), 219
Intent (*class in PIL.ImageCms*), 94
intent_supported (*PIL.ImageCms.core.CmsProfile attribute*), 109
Interop (*in module PIL.ExifTags*), 164
invert() (*in module PIL.ImageChops*), 91
invert() (*in module PIL.ImageOps*), 149
IptcImageFile (*class in PIL.IptcImagePlugin*), 198
IPythonViewer (*class in PIL.ImageShow*), 155
is_animated (*PIL.GifImagePlugin.GifImageFile property*), 193
is_animated (*PIL.Image.Image attribute*), 86
is_animated (*PIL.ImImagePlugin.ImImageFile property*), 197
is_animated (*PIL.PsdImagePlugin.PsdImageFile property*), 206
is_animated (*PIL.SpiderImagePlugin.SpiderImageFile property*), 207
is_cid() (*in module PIL.PngImagePlugin*), 205
is_intent_supported() (*PIL.ImageCms.core.CmsProfile method*), 109
is_matrix_shaper (*PIL.ImageCms.core.CmsProfile attribute*), 108
is_path() (*in module PIL._util*), 219
isatty() (*PIL.ContainerIO.ContainerIO method*), 172
isInt() (*in module PIL.SpiderImagePlugin*), 207
isIntentSupported() (*in module PIL.ImageCms*), 104
isSpiderHeader() (*in module PIL.SpiderImagePlugin*), 207
isSpiderImage() (*in module PIL.SpiderImagePlugin*), 208
Iterator (*class in PIL.ImageSequence*), 154
itersizes() (*PIL.IcnsImagePlugin.IcnsFile method*), 195
iTXt (*class in PIL.PngImagePlugin*), 178

J

JPEG (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
Jpeg2KImageFile (*class in PIL.Jpeg2KImagePlugin*), 199
jpeg_factory() (*in module PIL.JpegImagePlugin*), 199
JpegImageFile (*class in PIL.JpegImagePlugin*), 198

K

k (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
KEEP_SEQUENCE (*PIL.ImageCms.Flags attribute*), 95

Kernel (*class in PIL.ImageFilter*), 130

L

L16 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
L6V5U5 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
L8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
lambda_eval() (*in module PIL.ImageMath*), 142
LANCZOS (*PIL.Image.Resampling attribute*), 89
lang (*PIL.PngImagePlugin.iTXt attribute*), 178
layers (*PIL.PsdImagePlugin.PsdImageFile property*), 206
Layout (*class in PIL.ImageFont*), 140
legacy_api (*PIL.TiffImagePlugin.ImageFileDirectory_v2 property*), 211
LIBIMAGEQUANT (*PIL.Image.Quantize attribute*), 90
lighter() (*in module PIL.ImageChops*), 92
LightSource (*in module PIL.ExifTags*), 164
limit_rational() (*PIL.TiffImagePlugin.IFDRational method*), 209
line() (*PIL.ImageDraw.ImageDraw method*), 114
line() (*PIL.ImageDraw2.Draw method*), 176
line() (*PIL.PSDraw.PSDraw method*), 167
linear() (*in module PIL.GimpGradientFile*), 175
linear_gradient() (*in module PIL.Image*), 69
LINEARSIZE (*PIL.DdsImagePlugin.DDSD attribute*), 186
load() (*in module PIL.ImageFont*), 132
load() (*PIL.AvifImagePlugin.AvifImageFile method*), 180
load() (*PIL.EpsImagePlugin.EpsImageFile method*), 191
load() (*PIL.FpxImagePlugin.FpxImageFile method*), 192
load() (*PIL.GbrImagePlugin.GbrImageFile method*), 192
load() (*PIL.IcnsImagePlugin.IcnsImageFile method*), 195
load() (*PIL.IcoImagePlugin.IcoImageFile method*), 196
load() (*PIL.Image.Exif method*), 88
load() (*PIL.Image.Image method*), 85
load() (*PIL.ImageFile.ImageFile method*), 127
load() (*PIL.ImageFile.StubImageFile method*), 127
load() (*PIL.IptcImagePlugin.IptcImageFile method*), 198
load() (*PIL.Jpeg2KImagePlugin.Jpeg2KImageFile method*), 199
load() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load() (*PIL.TiffImagePlugin.TiffImageFile method*), 212
load() (*PIL.WalImageFile.WalImageFile method*), 180
load() (*PIL.WebPImagePlugin.WebPImageFile method*), 213
load() (*PIL.WmfImagePlugin.WmfStubImageFile method*), 214
load_byte() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_default() (*in module PIL.ImageFont*), 134
load_default_imagefont() (*in module PIL.ImageFont*), 134
load_djpeg() (*PIL.JpegImagePlugin.JpegImageFile method*), 199
load_double() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_end() (*PIL.CurImagePlugin.CurImageFile method*), 182
load_end() (*PIL.GifImagePlugin.GifImageFile method*), 193
load_end() (*PIL.ImageFile.ImageFile method*), 127
load_end() (*PIL.PcdImagePlugin.PcdImageFile method*), 202
load_end() (*PIL.PngImagePlugin.PngImageFile method*), 203
load_end() (*PIL.TgaImagePlugin.TgaImageFile method*), 208
load_end() (*PIL.TiffImagePlugin.TiffImageFile method*), 213
load_float() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_from_fp() (*PIL.Image.Exif method*), 88
load_long() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_long8() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_lut() (*PIL.ImageMorph.MorphOp method*), 146
load_path() (*in module PIL.ImageFont*), 132
load_prepare() (*PIL.CurImagePlugin.CurImageFile method*), 182
load_prepare() (*PIL.GifImagePlugin.GifImageFile method*), 193
load_prepare() (*PIL.ImageFile.ImageFile method*), 127
load_prepare() (*PIL.PcdImagePlugin.PcdImageFile method*), 202
load_prepare() (*PIL.PngImagePlugin.PngImageFile method*), 203
load_prepare() (*PIL.TiffImagePlugin.TiffImageFile method*), 213
load_rational() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_read() (*PIL.JpegImagePlugin.JpegImageFile method*), 199
load_read() (*PIL.PngImagePlugin.PngImageFile method*), 203
load_read() (*PIL.XpmImagePlugin.XpmImageFile method*), 215
load_seek() (*PIL.AvifImagePlugin.AvifImageFile method*), 180
load_seek() (*PIL.DdsImagePlugin.DdsImageFile*

method), 190
load_seek() (*PIL.EpsImagePlugin.EpsImageFile method*), 191
load_seek() (*PIL.IcoImagePlugin.IcoImageFile method*), 196
load_seek() (*PIL.MpoImagePlugin.MpoImageFile method*), 201
load_seek() (*PIL.WebPImagePlugin.WebPImageFile method*), 213
load_short() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_signed_byte() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_signed_long() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_signed_rational() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_signed_short() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
load_string() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
LOAD_TRUNCATED_IMAGES (*in module PIL.ImageFile*), 128
load_undefined() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
loadImageSeries() (*in module PIL.SpiderImagePlugin*), 208
LOADING_STRATEGY (*in module PIL.GifImagePlugin*), 193
LoadingStrategy (*class in PIL.GifImagePlugin*), 193
logical_and() (*in module PIL.ImageChops*), 92
logical_or() (*in module PIL.ImageChops*), 92
logical_xor() (*in module PIL.ImageChops*), 92
lookup() (*in module PIL.TiffTags*), 165
LOWRESPRECALC (*PIL.ImageCms.Flags attribute*), 95
LUMINANCE (*PIL.DdsImagePlugin.DDPF attribute*), 185
luminance (*PIL.ImageCms.core.CmsProfile attribute*), 107
LutBuilder (*class in PIL.ImageMorph*), 144

M

MacViewer (*class in PIL.ImageShow*), 155
makeSpiderHeader() (*in module PIL.SpiderImagePlugin*), 208
manufacturer (*PIL.ImageCms.core.CmsProfile attribute*), 107
map() (*PIL.ImagePath.PIL.ImagePath.Path method*), 153
match() (*PIL.ImageMorph.MorphOp method*), 146
MAX_IMAGE_PIXELS (*in module PIL.Image*), 88
MAX_STRING_LENGTH (*in module PIL.ImageFont*), 140
MAX_TEXT_CHUNK (*in module PIL.PngImagePlugin*), 205
MAX_TEXT_MEMORY (*in module PIL.PngImagePlugin*), 205
MAXBLOCK (*in module PIL.ImageFile*), 128
MAXCOVERAGE (*PIL.Image.Quantize attribute*), 90
MaxFilter (*class in PIL.ImageFilter*), 131
maximum (*PIL.ImageFont.Axis attribute*), 140
McIdasImageFile (*class in PIL.McIdasImagePlugin*), 200
mean (*PIL.ImageStat.Stat property*), 157
media_black_point (*PIL.ImageCms.core.CmsProfile attribute*), 108
media_white_point (*PIL.ImageCms.core.CmsProfile attribute*), 108
media_white_point_temperature (*PIL.ImageCms.core.CmsProfile attribute*), 108
median (*PIL.ImageStat.Stat property*), 157
MEDIANCUT (*PIL.Image.Quantize attribute*), 90
MedianFilter (*class in PIL.ImageFilter*), 130
merge() (*in module PIL.Image*), 66
MESH (*PIL.Image.Transform attribute*), 89
MeshTransform (*class in PIL.ImageTransform*), 162
method (*PIL.ImageTransform.AffineTransform attribute*), 161
method (*PIL.ImageTransform.ExtentTransform attribute*), 162
method (*PIL.ImageTransform.MeshTransform attribute*), 162
method (*PIL.ImageTransform.PerspectiveTransform attribute*), 162
method (*PIL.ImageTransform.QuadTransform attribute*), 162
method (*PIL.ImageTransform.Transform attribute*), 161
MicImageFile (*class in PIL.MicImagePlugin*), 200
MinFilter (*class in PIL.ImageFilter*), 130
minimum (*PIL.ImageFont.Axis attribute*), 140
MIPMAP (*PIL.DdsImagePlugin.DDSCAPS attribute*), 185
MIPMAPCOUNT (*PIL.DdsImagePlugin.DDSD attribute*), 186
mirror() (*in module PIL.ImageOps*), 149
mode (*PIL.Image.Image attribute*), 86
mode (*PIL.ImageMode.ModeDescriptor attribute*), 178
mode_map (*PIL.EpsImagePlugin.EpsImageFile attribute*), 191
ModeDescriptor (*class in PIL.ImageMode*), 177
ModeFilter (*class in PIL.ImageFilter*), 131
model (*PIL.ImageCms.core.CmsProfile attribute*), 107
module
    PIL, 172
    PIL._binary, 217
    PIL._deprecate, 218
    PIL._imaging, 220
    PIL._tkinter_finder, 219
    PIL._typing, 219

PIL._util, 219
PIL._version, 220
PIL.AvifImagePlugin, 180
PIL.BdfFontFile, 172
PIL.BmpImagePlugin, 181
PIL.BufrStubImagePlugin, 182
PIL.ContainerIO, 172
PIL.CurImagePlugin, 182
PIL.DcxImagePlugin, 182
PIL.DdsImagePlugin, 183
PIL.EpsImagePlugin, 190
PIL.ExifTags, 164
PIL.features, 169
PIL.FitsImagePlugin, 191
PIL.FliImagePlugin, 191
PIL.FontFile, 173
PIL.FpxImagePlugin, 192
PIL.GbrImagePlugin, 192
PIL.GdImageFile, 174
PIL.GifImagePlugin, 192
PIL.GimpGradientFile, 174
PIL.GimpPaletteFile, 175
PIL.GribStubImagePlugin, 194
PIL.Hdf5StubImagePlugin, 194
PIL.IcnsImagePlugin, 194
PIL.IcoImagePlugin, 196
PIL.Image, 64
PIL.Image.core, 220
PIL.ImageChops, 90
PIL.ImageCms, 94
PIL.ImageColor, 109
PIL.ImageDraw, 110
PIL.ImageDraw2, 175
PIL.ImageEnhance, 123
PIL.ImageFile, 124
PIL.ImageFilter, 128
PIL.ImageFont, 131
PIL.ImageGrab, 140
PIL.ImageMath, 141
PIL.ImageMode, 177
PIL.ImageMorph, 144
PIL.ImageOps, 146
PIL.ImagePalette, 151
PIL.ImagePath, 153
PIL.ImageQt, 153
PIL.ImageSequence, 154
PIL.ImageShow, 154
PIL.ImageStat, 156
PIL.ImageText, 157
PIL.ImageTk, 160
PIL.ImageTransform, 161
PIL.ImageWin, 162
PIL.ImImagePlugin, 197
PIL.ImtImagePlugin, 198
PIL.IptcImagePlugin, 198
PIL.Jpeg2KImagePlugin, 199
PIL.JpegImagePlugin, 198
PIL.JpegPresets, 166
PIL.McIdasImagePlugin, 200
PIL.MicImagePlugin, 200
PIL.MpegImagePlugin, 200
PIL.MpoImagePlugin, 201
PIL.MspImagePlugin, 201
PIL.PaletteFile, 178
PIL.PalmImagePlugin, 202
PIL.PcdImagePlugin, 202
PIL.PcfFontFile, 178
PIL.PcxImagePlugin, 202
PIL.PdfImagePlugin, 202
PIL.PixarImagePlugin, 202
PIL.PngImagePlugin, 202
PIL.PpmImagePlugin, 205
PIL.PsdImagePlugin, 206
PIL.PSDraw, 167
PIL.SgiImagePlugin, 206
PIL.SpiderImagePlugin, 207
PIL.SunImagePlugin, 208
PIL.TarIO, 179
PIL.TgaImagePlugin, 208
PIL.TiffImagePlugin, 208
PIL.TiffTags, 165
PIL.WalImageFile, 179
PIL.WebPImagePlugin, 213
PIL.WmfImagePlugin, 214
PIL.XbmImagePlugin, 214
PIL.XpmImagePlugin, 214
PIL.XVThumbImagePlugin, 214
MorphOp (*class in PIL.ImageMorph*), 145
MpegImageFile (*class in PIL.MpegImagePlugin*), 200
MpoImageFile (*class in PIL.MpoImagePlugin*), 201
MspDecoder (*class in PIL.MspImagePlugin*), 201
MspImageFile (*class in PIL.MspImagePlugin*), 201
MULTI2_ARGB8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
MultibandFilter (*class in PIL.ImageFilter*), 131
multiline_text() (*PIL.ImageDraw.ImageDraw method*), 118
multiline_textbbox() (*PIL.ImageDraw.ImageDraw method*), 121
multiply() (*in module PIL.ImageChops*), 92

N

n_frames (*PIL.GifImagePlugin.GifImageFile property*), 193
n_frames (*PIL.Image.Image attribute*), 86
n_frames (*PIL.ImImagePlugin.ImImageFile property*), 197

n_frames (*PIL.PsdImagePlugin.PsdImageFile property*), 206
n_frames (*PIL.SpiderImagePlugin.SpiderImageFile property*), 207
n_frames (*PIL.TiffImagePlugin.TiffImageFile property*), 213
name (*PIL.ImageFont.Axis attribute*), 140
name (*PIL.PcfFontFile.PcfFontFile attribute*), 178
named() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 211
nb_color (*PIL.IcoImagePlugin.IconHeader attribute*), 196
NEAREST (*PIL.Image.Resampling attribute*), 89
new() (*in module PIL.Image*), 67
new() (*PIL._util.DeferredError static method*), 219
newFrame() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
next() (*PIL.MpegImagePlugin.BitStream method*), 200
next_box_type() (*PIL.Jpeg2KImagePlugin.BoxReader method*), 199
nextheader() (*in module PIL.IcnsImagePlugin*), 195
NOCACHE (*PIL.ImageCms.Flags attribute*), 95
NODEFAULTRESOURCEDEF (*PIL.ImageCms.Flags attribute*), 96
NONE (*in module PIL.Image*), 88
NONE (*PIL.Image.Dither attribute*), 89
NONE (*PIL.ImageCms.Flags attribute*), 95
NONEGATIVES (*PIL.ImageCms.Flags attribute*), 96
NOOPTIMIZE (*PIL.ImageCms.Flags attribute*), 95
NOWHITEONWHITEFIXUP (*PIL.ImageCms.Flags attribute*), 95
NULLTRANSFORM (*PIL.ImageCms.Flags attribute*), 95
number() (*in module PIL.ImImagePlugin*), 197
numerator (*PIL.TiffImagePlugin.IFDRational property*), 209
NumpyArray (*class in PIL._typing*), 219
NV11 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
NV12 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187

O

o16be() (*in module PIL._binary*), 218
o16le() (*in module PIL._binary*), 218
o32be() (*in module PIL._binary*), 218
o32le() (*in module PIL._binary*), 218
o8() (*in module PIL._binary*), 218
offset (*PIL.IcoImagePlugin.IconHeader attribute*), 197
offset (*PIL.ImageFile._Tile attribute*), 124
offset (*PIL.TiffImagePlugin.ImageFileDirectory_v2 property*), 212
offset() (*in module PIL.ImageChops*), 93
OP_BACKGROUND (*PIL.PngImagePlugin.Disposal attribute*), 203
OP_NONE (*PIL.PngImagePlugin.Disposal attribute*), 203
OP_OVER (*PIL.PngImagePlugin.Blend attribute*), 202
OP_PREVIOUS (*PIL.PngImagePlugin.Disposal attribute*), 203
OP_SOURCE (*PIL.PngImagePlugin.Blend attribute*), 202
OPAQUE_420 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
open() (*in module PIL.GdImageFile*), 174
open() (*in module PIL.Image*), 65
open() (*in module PIL.WalImageFile*), 180
options (*PIL.ImageShow.Viewer attribute*), 156
ORDERED (*PIL.Image.Dither attribute*), 89
OUTPUT (*PIL.ImageCms.Direction attribute*), 95
overlay() (*in module PIL.ImageChops*), 93

P

P010 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
P016 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 187
P208 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
P8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
P8 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
pad() (*in module PIL.ImageOps*), 151
Palette (*class in PIL.Image*), 90
palette (*PIL.Image.Image attribute*), 86
PaletteFile (*class in PIL.PaletteFile*), 178
PALETTEINDEXED8 (*PIL.DdsImagePlugin.DDPF attribute*), 185
Parser (*class in PIL.ImageFile*), 125
paste() (*PIL.Image.Image method*), 77
paste() (*PIL.ImageTk.PhotoImage method*), 160
paste() (*PIL.ImageWin.Dib method*), 163
PcdImageFile (*class in PIL.PcdImagePlugin*), 202
PcfFontFile (*class in PIL.PcfFontFile*), 178
PcxImageFile (*class in PIL.PcxImagePlugin*), 202
peek() (*PIL.MpegImagePlugin.BitStream method*), 200
Pen (*class in PIL.ImageDraw2*), 175
PERCEPTUAL (*PIL.ImageCms.Intent attribute*), 94
perceptual_rendering_intent_gamut (*PIL.ImageCms.core.CmsProfile attribute*), 108
PERSPECTIVE (*PIL.Image.Transform attribute*), 89
PerspectiveTransform (*class in PIL.ImageTransform*), 161
PhotoImage (*class in PIL.ImageTk*), 160
pieslice() (*PIL.ImageDraw.ImageDraw method*), 115
pieslice() (*PIL.ImageDraw2.Draw method*), 176
PIL
    module, 172
PIL._binary
    module, 217
PIL._deprecate

module, 218
PIL._imaging
module, 220
PIL._tkinter_finder
module, 219
PIL._typing
module, 219
PIL._util
module, 219
PIL._version
module, 220
PIL.AvifImagePlugin
module, 180
PIL.BdfFontFile
module, 172
PIL.BmpImagePlugin
module, 181
PIL.BufrStubImagePlugin
module, 182
PIL.ContainerIO
module, 172
PIL.CurImagePlugin
module, 182
PIL.DcxImagePlugin
module, 182
PIL.DdsImagePlugin
module, 183
PIL.EpsImagePlugin
module, 190
PIL.ExifTags
module, 164
PIL.features
module, 169
PIL.FitsImagePlugin
module, 191
PIL.FliImagePlugin
module, 191
PIL.FontFile
module, 173
PIL.FpxImagePlugin
module, 192
PIL.GbrImagePlugin
module, 192
PIL.GdImageFile
module, 174
PIL.GifImagePlugin
module, 192
PIL.GimpGradientFile
module, 174
PIL.GimpPaletteFile
module, 175
PIL.GribStubImagePlugin
module, 194
PIL.Hdf5StubImagePlugin
module, 194
PIL.IcnsImagePlugin
module, 194
PIL.IcoImagePlugin
module, 196
PIL.Image
module, 64
PIL.Image.core
module, 220
PIL.ImageChops
module, 90
PIL.ImageCms
module, 94
PIL.ImageColor
module, 109
PIL.ImageDraw
module, 110
PIL.ImageDraw2
module, 175
PIL.ImageEnhance
module, 123
PIL.ImageFile
module, 124
PIL.ImageFilter
module, 128
PIL.ImageFont
module, 131
PIL.ImageGrab
module, 140
PIL.ImageMath
module, 141
PIL.ImageMode
module, 177
PIL.ImageMorph
module, 144
PIL.ImageOps
module, 146
PIL.ImagePalette
module, 151
PIL.ImagePath
module, 153
PIL.ImagePath.Path (*class in PIL.ImagePath*), 153
PIL.ImageQt
module, 153
PIL.ImageSequence
module, 154
PIL.ImageShow
module, 154
PIL.ImageStat
module, 156
PIL.ImageText
module, 157
PIL.ImageTk
module, 160

PIL.ImageTransform
    module, 161
PIL.ImageWin
    module, 162
PIL.ImImagePlugin
    module, 197
PIL.ImtImagePlugin
    module, 198
PIL.IptcImagePlugin
    module, 198
PIL.Jpeg2KImagePlugin
    module, 199
PIL.JpegImagePlugin
    module, 198
PIL.JpegPresets
    module, 166
PIL.McIdasImagePlugin
    module, 200
PIL.MicImagePlugin
    module, 200
PIL.MpegImagePlugin
    module, 200
PIL.MpoImagePlugin
    module, 201
PIL.MspImagePlugin
    module, 201
PIL.PaletteFile
    module, 178
PIL.PalmImagePlugin
    module, 202
PIL.PcdImagePlugin
    module, 202
PIL.PcfFontFile
    module, 178
PIL.PcxImagePlugin
    module, 202
PIL.PdfImagePlugin
    module, 202
PIL.PixarImagePlugin
    module, 202
PIL.PngImagePlugin
    module, 202
PIL.PpmImagePlugin
    module, 205
PIL.PsdImagePlugin
    module, 206
PIL.PSDraw
    module, 167
PIL.SgiImagePlugin
    module, 206
PIL.SpiderImagePlugin
    module, 207
PIL.SunImagePlugin
    module, 208
PIL.TarIO
    module, 179
PIL.TgaImagePlugin
    module, 208
PIL.TiffImagePlugin
    module, 208
PIL.TiffTags
    module, 165
PIL.TiffTags.LIBTIFF_CORE (*in module PIL.TiffTags*), 166
PIL.TiffTags.TAGS (*in module PIL.TiffTags*), 166
PIL.TiffTags.TAGS_V2 (*in module PIL.TiffTags*), 166
PIL.TiffTags.TAGS_V2_GROUPS (*in module PIL.TiffTags*), 166
PIL.TiffTags.TYPES (*in module PIL.TiffTags*), 166
PIL.WalImageFile
    module, 179
PIL.WebPImagePlugin
    module, 213
PIL.WmfImagePlugin
    module, 214
PIL.XbmImagePlugin
    module, 214
PIL.XpmImagePlugin
    module, 214
PIL.XVThumbImagePlugin
    module, 214
pilinfo() (*in module PIL.features*), 169
PITCH (*PIL.DdsImagePlugin.DDSD attribute*), 186
PixarImageFile (*class in PIL.PixarImagePlugin*), 202
PixelAccess (*built-in class*), 168
PIXELFORMAT (*PIL.DdsImagePlugin.DDSD attribute*), 186
planes (*PIL.IcoImagePlugin.IconHeader attribute*), 197
PNG (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
PngImageFile (*class in PIL.PngImagePlugin*), 203
PngInfo (*class in PIL.PngImagePlugin*), 179
PngStream (*class in PIL.PngImagePlugin*), 204
point() (*PIL.Image.Image method*), 77
point() (*PIL.ImageCms.ImageCmsTransform method*), 94
point() (*PIL.ImageDraw.ImageDraw method*), 115
polygon() (*PIL.ImageDraw.ImageDraw method*), 115
polygon() (*PIL.ImageDraw2.Draw method*), 176
posterize() (*in module PIL.ImageOps*), 149
PpmDecoder (*class in PIL.PpmImagePlugin*), 205
PpmImageFile (*class in PIL.PpmImagePlugin*), 205
PpmPlainDecoder (*class in PIL.PpmImagePlugin*), 206
prefix (*PIL.TiffImagePlugin.ImageFileDirectory_v2 property*), 212
preinit() (*in module PIL.Image*), 70
presets (*in module PIL.JpegPresets*), 167
profile_description (*PIL.ImageCms.core.CmsProfile attribute*),

107
profile_id (*PIL.ImageCms.core.CmsProfile attribute*), 106
profileToProfile() (*in module PIL.ImageCms*), 105
PROOF (*PIL.ImageCms.Direction attribute*), 95
PsdImageFile (*class in PIL.PsdImagePlugin*), 206
PSDraw (*class in PIL.PSDraw*), 167
push() (*PIL.PngImagePlugin.ChunkStream method*), 203
putalpha() (*PIL.Image.Image method*), 78
putchunk() (*in module PIL.PngImagePlugin*), 205
putdata() (*PIL.Image.Image method*), 78
puti16() (*in module PIL.FontFile*), 173
putpalette() (*PIL.Image.Image method*), 78
putpixel() (*PIL.Image.Image method*), 79
PyCMSError, 94
PyCodec (*class in PIL.ImageFile*), 125
PyDecoder (*class in PIL.ImageFile*), 126
PyEncoder (*class in PIL.ImageFile*), 126
Python Enhancement Proposals
PEP 527, 292
PEP 619, 274
PEP 656, 270
PEP 664, 263
PEP 693, 255
PEP 703, 248
PEP 719, 248
PEP 745, 241
PEP 770, 235

Q

Q16W16V16U16 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
Q8W8V8U8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
QUAD (*PIL.Image.Transform attribute*), 89
QuadTransform (*class in PIL.ImageTransform*), 162
Quantize (*class in PIL.Image*), 90
quantize() (*PIL.Image.Image method*), 79
query_palette() (*PIL.ImageWin.Dib method*), 163
queue (*PIL.PngImagePlugin.ChunkStream attribute*), 203

R

R10G10B10_XR_BIAS_A2_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R10G10B10A2_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R10G10B10A2_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R10G10B10A2_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R11G11B10_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16F (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
R16G16_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16B16A16_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16B16A16_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16B16A16_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16B16A16_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16B16A16_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R16G16B16A16_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R1_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R24_UNORM_X8_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R24G8_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R32_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32_FLOAT_X8X24_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189

R32_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32F (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
R32G32_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32B32_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32B32_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32B32_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32B32_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G32B32A32_FLOAT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R32G32B32A32_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R32G32B32A32_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 188
R32G32B32A32_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R32G8X24_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R3G3B2 (*PIL.DdsImagePlugin.D3DFMT attribute*), 184
R5G6B5 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
R8_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
R8G8_B8G8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
R8G8_B8G8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8B8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
R8G8B8A8_SINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8B8A8_SNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8B8A8_TYPELESS (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8B8A8_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8B8A8_UNORM (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R8G8B8A8_UNORM_SRGB (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 189
R9G9B9E5_SHAREDEXP (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
radial_gradient() (*in module PIL.Image*), 69
RankFilter (*class in PIL.ImageFilter*), 130
RAQM (*PIL.ImageFont.Layout attribute*), 140
RASTERIZE (*PIL.Image.Dither attribute*), 90
RAW (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
rawmode (*PIL.GimpPaletteFile.GimpPaletteFile attribute*), 175
rawmode (*PIL.PaletteFile.PaletteFile attribute*), 178
read() (*PIL.ContainerIO.ContainerIO method*), 172
read() (*PIL.MpegImagePlugin.BitStream method*), 200
read() (*PIL.PngImagePlugin.ChunkStream method*), 203
read_32() (*in module PIL.IcnsImagePlugin*), 195
read_32t() (*in module PIL.IcnsImagePlugin*), 195
read_boxes() (*PIL.Jpeg2KImagePlugin.BoxReader method*), 199
read_fields() (*PIL.Jpeg2KImagePlugin.BoxReader method*), 199
read_mk() (*in module PIL.IcnsImagePlugin*), 195
read_png_or_jpeg2000() (*in module PIL.IcnsImagePlugin*), 195
readable() (*PIL.ContainerIO.ContainerIO method*), 172
readline() (*PIL.ContainerIO.ContainerIO method*), 172
readlines() (*PIL.ContainerIO.ContainerIO method*), 172
readLong() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
readShort() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
rectangle() (*PIL.ImageDraw.ImageDraw method*), 116
rectangle() (*PIL.ImageDraw2.Draw method*), 177

rectangle() (*PIL.PSDraw.PSDraw method*), 167
red_colorant (*PIL.ImageCms.core.CmsProfile attribute*), 107
red_primary (*PIL.ImageCms.core.CmsProfile attribute*), 108
reduce (*PIL.Jpeg2KImagePlugin.Jpeg2KImageFile property*), 199
reduce() (*PIL.Image.Image method*), 79
register() (*in module PIL.ImageShow*), 155
register_decoder() (*in module PIL.Image*), 71
register_encoder() (*in module PIL.Image*), 71
register_extension() (*in module PIL.Image*), 71
register_extensions() (*in module PIL.Image*), 71
register_handler() (*in module PIL.BufrStubImagePlugin*), 182
register_handler() (*in module PIL.GribStubImagePlugin*), 194
register_handler() (*in module PIL.Hdf5StubImagePlugin*), 194
register_handler() (*in module PIL.WmfImagePlugin*), 214
register_mime() (*in module PIL.Image*), 70
register_open() (*in module PIL.Image*), 70
register_save() (*in module PIL.Image*), 70
register_save_all() (*in module PIL.Image*), 70
registered_extensions() (*in module PIL.Image*), 71
regular_polygon() (*PIL.ImageDraw.ImageDraw method*), 115
RELATIVE_COLORIMETRIC (*PIL.ImageCms.Intent attribute*), 94
remap_palette() (*PIL.Image.Image method*), 80
render() (*PIL.ImageDraw2.Draw method*), 176
rendering_intent (*PIL.ImageCms.core.CmsProfile attribute*), 106
Resampling (*class in PIL.Image*), 89
reserved (*PIL.IcoImagePlugin.IconHeader attribute*), 197
reset() (*PIL.ImageFile.Parser method*), 125
reset() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
resize() (*PIL.Image.Image method*), 80
rewind() (*PIL.PngImagePlugin.PngStream method*), 205
rewriteLastLong() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 208
rewriteLastShort() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
rewriteLastShortToLong() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
RGB (*PIL.DdsImagePlugin.DDPF attribute*), 185
RGB_AFTER_DIFFERENT_PALETTE_ONLY (*PIL.GifImagePlugin.LoadingStrategy attribute*), 193
RGB_AFTER_FIRST (*PIL.GifImagePlugin.LoadingStrategy attribute*), 193
RGB_ALWAYS (*PIL.GifImagePlugin.LoadingStrategy attribute*), 193
RLE4 (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
RLE8 (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
rms (*PIL.ImageStat.Stat property*), 157
rotate() (*PIL.Image.Image method*), 80
ROTATE_180 (*PIL.Image.Transpose attribute*), 89
ROTATE_270 (*PIL.Image.Transpose attribute*), 89
ROTATE_90 (*PIL.Image.Transpose attribute*), 89
rounded_rectangle() (*PIL.ImageDraw.ImageDraw method*), 116

S

S8_LOCKABLE (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
SAMPLER_FEEDBACK_MIN_MIP_OPAQUE (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
SAMPLER_FEEDBACK_MIP_REGION_USED_OPAQUE (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
SATURATION (*PIL.ImageCms.Intent attribute*), 94
saturation_rendering_intent_gamut (*PIL.ImageCms.core.CmsProfile attribute*), 108
save() (*PIL.FontFile.FontFile method*), 173
save() (*PIL.Image.Image method*), 81
save() (*PIL.ImagePalette.ImagePalette method*), 152
save() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
save_image() (*PIL.ImageShow.Viewer method*), 156
save_lut() (*PIL.ImageMorph.MorphOp method*), 146
save_rewind() (*PIL.PngImagePlugin.PngStream method*), 205
scale() (*in module PIL.ImageOps*), 148
screen() (*in module PIL.ImageChops*), 93
screening_description (*PIL.ImageCms.core.CmsProfile attribute*), 108
seek() (*PIL.AvifImagePlugin.AvifImageFile method*), 180
seek() (*PIL.ContainerIO.ContainerIO method*), 173
seek() (*PIL.DcxImagePlugin.DcxImageFile method*), 182
seek() (*PIL.FliImagePlugin.FliImageFile method*), 191
seek() (*PIL.GifImagePlugin.GifImageFile method*), 193
seek() (*PIL.Image.Image method*), 82
seek() (*PIL.ImImagePlugin.ImImageFile method*), 197
seek() (*PIL.MicImagePlugin.MicImageFile method*), 200

seek() (*PIL.MpoImagePlugin.MpoImageFile method*), 201
seek() (*PIL.PngImagePlugin.PngImageFile method*), 203
seek() (*PIL.PsdImagePlugin.PsdImageFile method*), 206
seek() (*PIL.SpiderImagePlugin.SpiderImageFile method*), 207
seek() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
seek() (*PIL.TiffImagePlugin.TiffImageFile method*), 213
seek() (*PIL.WebPImagePlugin.WebPImageFile method*), 213
seekable() (*PIL.ContainerIO.ContainerIO method*), 173
SEGMENTS (*in module PIL.GimpGradientFile*), 175
set_as_raw() (*PIL.ImageFile.PyDecoder method*), 126
set_lut() (*PIL.ImageMorph.MorphOp method*), 146
set_variation_by_axes() (*PIL.ImageFont.FreeTypeFont method*), 139
set_variation_by_name() (*PIL.ImageFont.FreeTypeFont method*), 139
setEndian() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
setfd() (*PIL.ImageFile.PyCodec method*), 125
setfont() (*PIL.PSDraw.PSDraw method*), 167
setimage() (*PIL.ImageFile.PyCodec method*), 125
settransform() (*PIL.ImageDraw2.Draw method*), 176
setup() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
SGI16Decoder (*class in PIL.SgiImagePlugin*), 206
SgiImageFile (*class in PIL.SgiImagePlugin*), 207
shape() (*PIL.ImageDraw.ImageDraw method*), 116
Sharpness (*class in PIL.ImageEnhance*), 124
show() (*in module PIL.ImageShow*), 155
show() (*PIL.Image.Image method*), 82
show() (*PIL.ImageShow.Viewer method*), 156
show_file() (*PIL.ImageShow.Viewer method*), 156
show_image() (*PIL.ImageShow.Viewer method*), 156
si16be() (*in module PIL._binary*), 218
si16le() (*in module PIL._binary*), 218
si32be() (*in module PIL._binary*), 218
si32le() (*in module PIL._binary*), 218
sine() (*in module PIL.GimpGradientFile*), 175
size (*PIL.IcnsImagePlugin.IcnsImageFile property*), 195
size (*PIL.IcoImagePlugin.IcoImageFile property*), 196
size (*PIL.IcoImagePlugin.IconHeader attribute*), 197
size (*PIL.Image.Image attribute*), 86
SIZES (*PIL.IcnsImagePlugin.IcnsFile attribute*), 194
sizes() (*PIL.IcoImagePlugin.IcoFile method*), 196
Skip() (*in module PIL.JpegImagePlugin*), 199
skip() (*PIL.MpegImagePlugin.BitStream method*), 200
skipIFDs() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
SOF() (*in module PIL.JpegImagePlugin*), 199
soft_light() (*in module PIL.ImageChops*), 93
SOFTPROOFING (*PIL.ImageCms.Flags attribute*), 95
solarize() (*in module PIL.ImageOps*), 149
sphere_decreasing() (*in module PIL.GimpGradientFile*), 175
sphere_increasing() (*in module PIL.GimpGradientFile*), 175
SpiderImageFile (*class in PIL.SpiderImagePlugin*), 207
split() (*PIL.Image.Image method*), 82
square (*PIL.IcoImagePlugin.IconHeader attribute*), 197
Stat (*class in PIL.ImageStat*), 156
stddev (*PIL.ImageStat.Stat property*), 157
stroke() (*PIL.ImageText.Text method*), 159
StrOrBytesPath (*class in PIL._typing*), 219
StubHandler (*class in PIL.ImageFile*), 127
StubImageFile (*class in PIL.ImageFile*), 127
subtract() (*in module PIL.ImageChops*), 93
subtract_modulo() (*in module PIL.ImageChops*), 93
sum (*PIL.ImageStat.Stat property*), 157
sum2 (*PIL.ImageStat.Stat property*), 157
SunImageFile (*class in PIL.SunImagePlugin*), 208
SupportsArrayInterface (*class in PIL.Image*), 88
SupportsArrowArrayInterface (*class in PIL.Image*), 88
SupportsGetData (*class in PIL.Image*), 88
SupportsGetMesh (*class in PIL.ImageOps*), 148
SupportsRead (*class in PIL._typing*), 219
sz() (*in module PIL.PcfFontFile*), 178

T

tag (*PIL.TiffImagePlugin.TiffImageFile attribute*), 213
tag_v2 (*PIL.TiffImagePlugin.TiffImageFile attribute*), 213
tagdata (*PIL.TiffImagePlugin.ImageFileDirectory_v1 property*), 210
TagInfo (*class in PIL.TiffTags*), 165
TAGS (*in module PIL.ExifTags*), 165
Tags (*PIL.TiffImagePlugin.AppendingTiffWriter attribute*), 208
tags (*PIL.TiffImagePlugin.ImageFileDirectory_v1 property*), 210
tagtype (*PIL.TiffImagePlugin.ImageFileDirectory_v1 attribute*), 210
tagtype (*PIL.TiffImagePlugin.ImageFileDirectory_v2 attribute*), 212
target (*PIL.ImageCms.core.CmsProfile attribute*), 107
TarIO (*class in PIL.TarIO*), 179
technology (*PIL.ImageCms.core.CmsProfile attribute*), 108
tell() (*PIL.AvifImagePlugin.AvifImageFile method*), 180
tell() (*PIL.ContainerIO.ContainerIO method*), 173

tell() (*PIL.DcxImagePlugin.DcxImageFile method*), 182
tell() (*PIL.FliImagePlugin.FliImageFile method*), 192
tell() (*PIL.GifImagePlugin.GifImageFile method*), 193
tell() (*PIL.Image.Image method*), 83
tell() (*PIL.ImImagePlugin.ImImageFile method*), 197
tell() (*PIL.MicImagePlugin.MicImageFile method*), 200
tell() (*PIL.MpoImagePlugin.MpoImageFile method*), 201
tell() (*PIL.PngImagePlugin.PngImageFile method*), 204
tell() (*PIL.PsdImagePlugin.PsdImageFile method*), 206
tell() (*PIL.SpiderImagePlugin.SpiderImageFile method*), 207
tell() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
tell() (*PIL.TiffImagePlugin.TiffImageFile method*), 213
tell() (*PIL.WebPImagePlugin.WebPImageFile method*), 213
Text (*class in PIL.ImageText*), 158
text (*PIL.PngImagePlugin.PngImageFile property*), 204
text() (*PIL.ImageDraw.ImageDraw method*), 117
text() (*PIL.ImageDraw2.Draw method*), 177
text() (*PIL.PSDraw.PSDraw method*), 168
textbbox() (*PIL.ImageDraw.ImageDraw method*), 120
textbbox() (*PIL.ImageDraw2.Draw method*), 177
textlength() (*PIL.ImageDraw.ImageDraw method*), 119
textlength() (*PIL.ImageDraw2.Draw method*), 177
TEXTURE (*PIL.DdsImagePlugin.DDSCAPS attribute*), 185
TgaImageFile (*class in PIL.TgaImagePlugin*), 208
thumbnail() (*PIL.Image.Image method*), 83
TiffImageFile (*class in PIL.TiffImagePlugin*), 212
tile (*PIL.ImageFile.ImageFile attribute*), 127
tkey (*PIL.PngImagePlugin.iTXt attribute*), 178
tkPhotoImage() (*PIL.SpiderImagePlugin.SpiderImageFile method*), 207
to_imagefont() (*PIL.FontFile.FontFile method*), 173
to_v2() (*PIL.TiffImagePlugin.ImageFileDirectory_v1 method*), 210
tobitmap() (*PIL.Image.Image method*), 83
tobytes() (*PIL.Image.Exif method*), 88
tobytes() (*PIL.Image.Image method*), 83
tobytes() (*PIL.ImageCms.ImageCmsProfile method*), 94
tobytes() (*PIL.ImagePalette.ImagePalette method*), 152
tobytes() (*PIL.ImageWin.Dib method*), 164
tobytes() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
tolist() (*PIL.ImagePath.PIL.ImagePath.Path method*), 153
tostring() (*PIL.ImagePalette.ImagePalette method*), 152
Transform (*class in PIL.Image*), 89
Transform (*class in PIL.ImageTransform*), 161
transform() (*PIL.Image.Image method*), 84
transform() (*PIL.ImageFilter.Color3DLUT method*), 129
transform() (*PIL.ImagePath.PIL.ImagePath.Path method*), 153
transform() (*PIL.ImageTransform.Transform method*), 161
Transpose (*class in PIL.Image*), 88
TRANSPOSE (*PIL.Image.Transpose attribute*), 89
transpose() (*PIL.Image.Image method*), 85
TransposedFont (*class in PIL.ImageFont*), 139
TRANSVERSE (*PIL.Image.Transpose attribute*), 89
truetype() (*in module PIL.ImageFont*), 132
truncate() (*PIL.ContainerIO.ContainerIO method*), 173
typestr (*PIL.ImageMode.ModeDescriptor attribute*), 178

U

UnidentifiedImageError, 172
UnixViewer (*class in PIL.ImageShow*), 155
UnixViewer.DisplayViewer (*class in PIL.ImageShow*), 155
UnixViewer.EogViewer (*class in PIL.ImageShow*), 155
UnixViewer.GmDisplayViewer (*class in PIL.ImageShow*), 155
UnixViewer.XDGViewer (*class in PIL.ImageShow*), 155
UnixViewer.XVViewer (*class in PIL.ImageShow*), 155
UNKNOWN (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
UNKNOWN (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
unsafe_eval() (*in module PIL.ImageMath*), 142
UnsharpMask (*class in PIL.ImageFilter*), 130
USE_8BITS_DEVICELINK (*PIL.ImageCms.Flags attribute*), 95
UYVY (*PIL.DdsImagePlugin.D3DFMT attribute*), 185

V

v (*PIL.BmpImagePlugin.BmpImageFile attribute*), 181
V16U16 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
V208 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
V408 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
V8U8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
var (*PIL.ImageStat.Stat property*), 157
verify() (*PIL.Image.Image method*), 85
verify() (*PIL.ImageFile.ImageFile method*), 127

verify() (*PIL.PngImagePlugin.ChunkStream method*), 203
verify() (*PIL.PngImagePlugin.PngImageFile method*), 204
version (*PIL.ImageCms.core.CmsProfile attribute*), 106
version() (*in module PIL.features*), 169
version_codec() (*in module PIL.features*), 170
version_feature() (*in module PIL.features*), 171
version_module() (*in module PIL.features*), 170
VERTEXDATA (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
Viewer (*class in PIL.ImageShow*), 156
viewing_condition (*PIL.ImageCms.core.CmsProfile attribute*), 108
VOLUME (*PIL.DdsImagePlugin.DDSCAPS2 attribute*), 186

W

WalImageFile (*class in PIL.WalImageFile*), 180
WARN_POSSIBLE_FORMATS (*in module PIL.Image*), 88
WEB (*PIL.Image.Palette attribute*), 90
WebPImageFile (*class in PIL.WebPImagePlugin*), 213
WIDTH (*PIL.DdsImagePlugin.DDSD attribute*), 186
width (*PIL.IcoImagePlugin.IconHeader attribute*), 197
width (*PIL.Image.Image attribute*), 86
width() (*PIL.ImageTk.BitmapImage method*), 160
width() (*PIL.ImageTk.PhotoImage method*), 161
WindowsViewer (*class in PIL.ImageShow*), 155
WmfStubImageFile (*class in PIL.WmfImagePlugin*), 214
wrap() (*PIL.ImageText.Text method*), 159
writable() (*PIL.ContainerIO.ContainerIO method*), 173
write() (*PIL.ContainerIO.ContainerIO method*), 173
write() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
write_byte() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_double() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_float() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_long() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_long8() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_rational() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_short() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_signed_byte() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_signed_long() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_signed_rational() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_signed_short() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_string() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
write_undefined() (*PIL.TiffImagePlugin.ImageFileDirectory_v2 method*), 212
writelines() (*PIL.ContainerIO.ContainerIO method*), 173
writeLong() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209
writeShort() (*PIL.TiffImagePlugin.AppendingTiffWriter method*), 209

X

X1R5G5B5 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
X24_TYPELESS_G8_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
X32_TYPELESS_G8X24_UINT (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
X4R4G4B4 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
X8B8G8R8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
X8L8V8U8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
X8R8G8B8 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185
XbmImageFile (*class in PIL.XbmImagePlugin*), 214
xcolor_space (*PIL.ImageCms.core.CmsProfile attribute*), 106
XpmDecoder (*class in PIL.XpmImagePlugin*), 214
XpmImageFile (*class in PIL.XpmImagePlugin*), 214
XVThumbImageFile (*class in PIL.XVThumbImagePlugin*), 214

Y

Y210 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
Y216 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
Y410 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
Y416 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190
YUY2 (*PIL.DdsImagePlugin.D3DFMT attribute*), 185

YUY2 (*PIL.DdsImagePlugin.DXGI_FORMAT attribute*), 190