Plugin reference

`BmpImagePlugin` Module

class **`PIL.BmpImagePlugin.BmpImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.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, *args*) [source]

Bases: `PIL.ImageFile.PyDecoder`

decode(*buffer*) [source]

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=None, filename=None*) [source]

Bases: `PIL.BmpImagePlugin.BmpImageFile`

`format`= *'DIB'*

`format_description`= *'Windows Bitmap'*

`BufrStubImagePlugin` Module

class **`PIL.BufrStubImagePlugin.BufrStubImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.StubImageFile`

`format`= *'BUFR'*

`format_description`= *'BUFR'*

`PIL.BufrStubImagePlugin.register_handler`(*handler*) [source]

Install application-specific BUFR image handler.

Parameters: **handler** – Handler object.

`CurImagePlugin` Module

class **`PIL.CurImagePlugin.CurImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.BmpImagePlugin.BmpImageFile`

`format`= *'CUR'*

`format_description`= *'Windows Cursor'*

`DcxImagePlugin` Module

class **`PIL.DcxImagePlugin.DcxImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.PcxImagePlugin.PcxImageFile`

`format`= *'DCX'*

`format_description`= *'Intel DCX'*

`seek`(*frame*) [source]

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`() [source]

Returns the current frame number. See `seek()`.

If defined, `n_frames` refers to the number of available frames.

Returns: Frame number, starting with 0.

`EpsImagePlugin` Module

class **`PIL.EpsImagePlugin.EpsImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

EPS File Parser for the Python Imaging Library

`format`= *'EPS'*

`format_description`= *'Encapsulated Postscript'*

load(*scale=1*, *transparency=False*) [source]

Load image data based on tile list

load_seek(**args*, ***kwargs*) [source]

mode_map= *{1: 'L', 2: 'LAB', 3: 'RGB', 4: 'CMYK'}*

PIL.EpsImagePlugin.Ghostscript(*tile*, *size*, *fp*, *scale=1*, *transparency=False*) [source]

Render an image using Ghostscript

class **PIL.EpsImagePlugin.PSFile**(*fp*) [source]

Bases: `object`

Wrapper for bytesio object that treats either CR or LF as end of line.

readline() [source]

seek(*offset*, *whence=0*) [source]

PIL.EpsImagePlugin.has_ghostscript() [source]

`FitsImagePlugin` Module

class **PIL.FitsImagePlugin.FitsImageFile**(*fp=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

format= *'FITS'*

format_description= *'FITS'*

`FliImagePlugin` Module

class **PIL.FliImagePlugin.FliImageFile**(*fp=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

format= *'FLI'*

format_description= *'Autodesk FLI/FLC Animation'*

seek(*frame*) [source]

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() [source]

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=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

format= *'FPX'*

format_description= *'FlashPix'*

load() [source]

Load image data based on tile list

`GbrImagePlugin` Module

class **PIL.GbrImagePlugin.GbrImageFile**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

format= *'GBR'*

`format_description`= *'GIMP brush file'*

`load()` [source]

Load image data based on tile list

`GifImagePlugin` Module

class **`PIL.GifImagePlugin.GifImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`data()` [source]

`format`= *'GIF'*

`format_description`= *'Compuserve GIF'*

`global_palette`= *None*

property **`is_animated`**

`load_end()` [source]

`load_prepare()` [source]

property **`n_frames`**

`seek`(*frame*) [source]

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()` [source]

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*

New in version 9.1.0.

class `PIL.GifImagePlugin.LoadingStrategy`*(value)* [source]

Bases: `enum.IntEnum`

New in version 9.1.0.

`RGB_AFTER_DIFFERENT_PALETTE_ONLY`*= 1*

`RGB_AFTER_FIRST`*= 0*

`RGB_ALWAYS`*= 2*

`PIL.GifImagePlugin.get_interlace`*(im)* [source]

`PIL.GifImagePlugin.getdata`*(im, offset=(0, 0), **params)* [source]

Legacy Method

Return a list of strings representing this image. The first string is a local image header, the rest contains encoded image data.

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, palette=None, info=None)* [source]

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=None, filename=None*) [source]

Bases: `PIL.ImageFile.StubImageFile`

`format`= *'GRIB'*

`format_description`= *'GRIB'*

`PIL.GribStubImagePlugin.register_handler`(*handler*) [source]

Install application-specific GRIB image handler.

Parameters: **handler** – Handler object.

`Hdf5StubImagePlugin` Module

class **`PIL.Hdf5StubImagePlugin.HDF5StubImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.StubImageFile`

`format`= *'HDF5'*

`format_description`= *'HDF5'*

`PIL.Hdf5StubImagePlugin.register_handler`(*handler*) [source]

Install application-specific HDF5 image handler.

Parameters: **handler** – Handler object.

`IcnsImagePlugin` Module

class **`PIL.IcnsImagePlugin.IcnsFile`**(*fobj*) [source]

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()` [source]

`dataforsize`(*size*) [source]

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=None*) [source]

`itersizes()` [source]

class **`PIL.IcnsImagePlugin.IcnsImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.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()` [source]

Load image data based on tile list

property **`size`**

`PIL.IcnsImagePlugin.nextheader`(*fobj*) [source]

`PIL.IcnsImagePlugin.read_32`(*fobj*, *start_length*, *size*) [source]

Read a 32bit RGB icon resource. Seems to be either uncompressed or an RLE packbits-like scheme.

`PIL.IcnsImagePlugin.read_32t`(*fobj*, *start_length*, *size*) [source]

`PIL.IcnsImagePlugin.read_mk`(*fobj*, *start_length*, *size*) [source]

`PIL.IcnsImagePlugin.read_png_or_jpeg2000`(*fobj*, *start_length*, *size*) [source]

`IcoImagePlugin` Module

class `PIL.IcoImagePlugin.IcoFile`(*buf*) [source]

Bases: `object`

`frame`(*idx*) [source]

Get an image from frame idx

`getentryindex`(*size*, *bpp=False*) [source]

`getimage`(*size*, *bpp=False*) [source]

Get an image from the icon

`sizes`() [source]

Get a list of all available icon sizes and color depths.

class `PIL.IcoImagePlugin.IcoImageFile`(*fp=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.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()` [source]

Load image data based on tile list

`load_seek()` [source]

property **`size`**

`ImImagePlugin` Module

class **`PIL.ImImagePlugin.ImImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'IM'*

`format_description`= *'IFUNC Image Memory'*

property **`is_animated`**

property **`n_frames`**

`seek`(*frame*) [source]

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()` [source]

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*) [source]

`ImtImagePlugin` Module

class **`PIL.ImtImagePlugin.ImtImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'IMT'*

`format_description`= *'IM Tools'*

`IptcImagePlugin` Module

class **`PIL.IptcImagePlugin.IptcImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`field()` [source]

`format`= *'IPTC'*

`format_description`= *'IPTC/NAA'*

`getint`(*key*) [source]

`load()` [source]

Load image data based on tile list

`PIL.IptcImagePlugin.dump`(*c*) [source]

`PIL.IptcImagePlugin.getiptcinfo`(*im*) [source]

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.

`PIL.IptcImagePlugin.i`(*c*) [source]

`JpegImagePlugin` Module

`PIL.JpegImagePlugin.APP`(*self*, *marker*) [source]

`PIL.JpegImagePlugin.COM`(*self*, *marker*) [source]

`PIL.JpegImagePlugin.DQT`(*self*, *marker*) [source]

class **`PIL.JpegImagePlugin.JpegImageFile`**(*fp=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`draft`(*mode*, *size*) [source]

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 greyscale 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.

format= *'JPEG'*

format_description= *'JPEG (ISO 10918)'*

getxmp() [source]

Returns a dictionary containing the XMP tags. Requires defusedxml to be installed.

Returns: XMP tags in a dictionary.

load_djpeg() [source]

load_read(*read_bytes*) [source]

internal: read more image data For premature EOF and LOAD_TRUNCATED_IMAGES adds EOI marker so libjpeg can finish decoding

PIL.JpegImagePlugin.SOF(*self*, *marker*) [source]

PIL.JpegImagePlugin.Skip(*self*, *marker*) [source]

PIL.JpegImagePlugin.convert_dict_qtables(*qtables*) [source]

PIL.JpegImagePlugin.get_sampling(*im*) [source]

PIL.JpegImagePlugin.jpeg_factory(*fp=None*, *filename=None*) [source]

`Jpeg2KImagePlugin` Module

class **PIL.Jpeg2KImagePlugin.BoxReader**(*fp*, *length=- 1*) [source]

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() [source]

next_box_type() [source]

read_boxes() [source]

`read_fields`(*field_format*) [source]

class **`PIL.Jpeg2KImagePlugin.Jpeg2KImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'JPEG2000'*

`format_description`= *'JPEG 2000 (ISO 15444)'*

`load`() [source]

Load image data based on tile list

property **`reduce`**

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=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'MCIDAS'*

`format_description`= *'McIdas area file'*

`MicImagePlugin` Module

class **`PIL.MicImagePlugin.MicImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.TiffImagePlugin.TiffImageFile`

format= *'MIC'*

format_description= *'Microsoft Image Composer'*

seek(*frame*) [source]

Select a given frame as current image

tell() [source]

Return the current frame number

`MpegImagePlugin` Module

class **PIL.MpegImagePlugin.BitStream**(*fp*) [source]

Bases: `object`

next() [source]

peek(*bits*) [source]

read(*bits*) [source]

skip(*bits*) [source]

class **PIL.MpegImagePlugin.MpegImageFile**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

format= *'MPEG'*

format_description= *'MPEG'*

`MspImagePlugin` Module

class **PIL.MspImagePlugin.MspDecoder**(*mode, *args*) [source]

Bases: `PIL.ImageFile.PyDecoder`

decode(*buffer*) [source]

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=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'MSP'*

`format_description`= *'Windows Paint'*

`PalmImagePlugin` Module

`PIL.PalmImagePlugin.build_prototype_image`() [source]

`PcdImagePlugin` Module

class **`PIL.PcdImagePlugin.PcdImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'PCD'*

`format_description`= *'Kodak PhotoCD'*

`load_end`() [source]

`PcxImagePlugin` Module

class **`PIL.PcxImagePlugin.PcxImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'PCX'*

`format_description`= *'Paintbrush'*

`PdfImagePlugin` Module

`PixarImagePlugin` Module

class **`PIL.PixarImagePlugin.PixarImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'PIXAR'*

`format_description`= *'PIXAR raster image'*

`PngImagePlugin` Module

class **`PIL.PngImagePlugin.Blend`**(*value*) [source]

Bases: `enum.IntEnum`

An enumeration.

`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*) [source]

Bases: `object`

`call`(*cid, pos, length*) [source]

Call the appropriate chunk handler

`close`() [source]

`crc`(*cid, data*) [source]

Read and verify checksum

`crc_skip`(*cid, data*) [source]

Read checksum. Used if the C module is not present

push(*cid, pos, length*) [source]

read() [source]

Fetch a new chunk. Returns header information.

verify(*endchunk=b'IEND'*) [source]

class **PIL.PngImagePlugin.Disposal**(*value*) [source]

Bases: `enum.IntEnum`

An enumeration.

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=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

getexif() [source]

getxmp() [source]

Returns a dictionary containing the XMP tags. Requires defusedxml to be installed.

Returns: XMP tags in a dictionary.

load_end() [source]

internal: finished reading image data

load_prepare() [source]

internal: prepare to read PNG file

`load_read`(*read_bytes*) [source]

internal: read more image data

`seek`(*frame*) [source]

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`() [source]

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`() [source]

Verify PNG file

`format`= *'PNG'*

`format_description`= *'Portable network graphics'*

property **`text`**

class **`PIL.PngImagePlugin.PngStream`**(*fp*) [source]

Bases: `PIL.PngImagePlugin.ChunkStream`

`check_text_memory`(*chunklen*) [source]

`chunk_IDAT`(*pos*, *length*) [source]

`chunk_IEND`(*pos*, *length*) [source]

`chunk_IHDR`(*pos*, *length*) [source]

`chunk_PLTE`(*pos*, *length*) [source]

`chunk_acTL`(*pos*, *length*) [source]

`chunk_cHRM`(*pos*, *length*) [source]

`chunk_eXIf`(*pos*, *length*) [source]

`chunk_fcTL`(*pos*, *length*) [source]

`chunk_fdAT`(*pos*, *length*) [source]

`chunk_gAMA`(*pos*, *length*) [source]

`chunk_iCCP`(*pos*, *length*) [source]

`chunk_iTXt`(*pos*, *length*) [source]

`chunk_pHYs`(*pos*, *length*) [source]

`chunk_sRGB`(*pos*, *length*) [source]

`chunk_tEXt`(*pos*, *length*) [source]

`chunk_tRNS`(*pos*, *length*) [source]

`chunk_zTXt`(*pos*, *length*) [source]

`rewind`() [source]

`save_rewind`() [source]

`PIL.PngImagePlugin.getchunks`(*im*, ***params*) [source]

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*, *cid*, **data*) [source]

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*, **args*) [source]

Bases: `PIL.ImageFile.PyDecoder`

decode(*buffer*) [source]

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=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'PPM'*

`format_description`= *'Pbmplus image'*

`PsdImagePlugin` Module

class **`PIL.PsdImagePlugin.PsdImageFile`**(*fp=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'PSD'*

`format_description`= *'Adobe Photoshop'*

seek(*layer*) [source]

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`() [source]

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*, **args*) [source]

Bases: `PIL.ImageFile.PyDecoder`

decode(*buffer*) [source]

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=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'SGI'*

`format_description`= *'SGI Image File Format'*

`SpiderImagePlugin` Module

class **`PIL.SpiderImagePlugin.SpiderImageFile`**(*fp=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`convert2byte`(*depth=255*) [source]

`format`= *'SPIDER'*

`format_description`= *'Spider 2D image'*

property **`is_animated`**

property **`n_frames`**

`seek`(*frame*) [source]

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`() [source]

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() [source]

PIL.SpiderImagePlugin.isInt(*f*) [source]

PIL.SpiderImagePlugin.isSpiderHeader(*t*) [source]

PIL.SpiderImagePlugin.isSpiderImage(*filename*) [source]

PIL.SpiderImagePlugin.loadImageSeries(*filelist=None*) [source]

create a list of `Image` objects for use in a montage

PIL.SpiderImagePlugin.makeSpiderHeader(*im*) [source]

`SunImagePlugin` Module

class **PIL.SunImagePlugin.SunImageFile**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

format= *'SUN'*

format_description= *'Sun Raster File'*

`TgaImagePlugin` Module

class **PIL.TgaImagePlugin.TgaImageFile**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

format= *'TGA'*

format_description= *'Targa'*

load_end() [source]

`TiffImagePlugin` Module

class **PIL.TiffImagePlugin.AppendingTiffWriter**(*fn, new=False*) [source]

Bases: `object`

`Tags`= *{273, 288, 324, 519, 520, 521}*

`close`() [source]

`fieldSizes`= *[0, 1, 1, 2, 4, 8, 1, 1, 2, 4, 8, 4, 8]*

`finalize`() [source]

`fixIFD`() [source]

`fixOffsets`(*count*, *isShort=False*, *isLong=False*) [source]

`goToEnd`() [source]

`newFrame`() [source]

`readLong`() [source]

`readShort`() [source]

`rewriteLastLong`(*value*) [source]

`rewriteLastShort`(*value*) [source]

`rewriteLastShortToLong`(*value*) [source]

`seek`(*offset*, *whence=0*) [source]

`setEndian`(*endian*) [source]

`setup`() [source]

`skipIFDs`() [source]

`tell`() [source]

`write`(*data*) [source]

`writeLong`(*value*) [source]

`writeShort`(*value*) [source]

class **`PIL.TiffImagePlugin.IFDRational`**(*value*, *denominator=1*) [source]

Bases: `numbers.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`**

`limit_rational`(*max_denominator*) [source]

Parameters: **max_denominator** – Integer, the maximum denominator value

Returns: Tuple of (numerator, denominator)

property **`numerator`**

`PIL.TiffImagePlugin.ImageFileDirectory`

alias of `PIL.TiffImagePlugin.ImageFileDirectory_v1`

class **`PIL.TiffImagePlugin.ImageFileDirectory_v1`**(**args*, ***kwargs*) [source]

Bases: `PIL.TiffImagePlugin.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*) [source]

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*

Dictionary of tag types

`to_v2`() [source]

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=b'II*\x00\x00\x00\x00\x00', prefix=None, group=None*) [source]

Bases: `collections.abc.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`

  New 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`**

`load`(*fp*) [source]

`load_byte`(*data*, *legacy_api=True*) [source]

`load_double`(*data*, *legacy_api=True*)

`load_float`(*data*, *legacy_api=True*)

`load_long`(*data*, *legacy_api=True*)

`load_long8`(*data*, *legacy_api=True*)

`load_rational`(*data*, *legacy_api=True*) [source]

`load_short`(*data*, *legacy_api=True*)

`load_signed_byte`(*data*, *legacy_api=True*)

`load_signed_long`(*data*, *legacy_api=True*)

`load_signed_rational`(*data*, *legacy_api=True*) [source]

`load_signed_short`(*data*, *legacy_api=True*)

`load_string`(*data*, *legacy_api=True*) [source]

`load_undefined`(*data*, *legacy_api=True*) [source]

`named`() [source]

Returns: dict of name|key: value

Returns the complete tag dictionary, with named tags where possible.

property **`offset`**

property **`prefix`**

`reset`() [source]

`save`(*fp*) [source]

`tagtype`

Dictionary of tag types

`tobytes`(*offset=0*) [source]

`write_byte`(*data*) [source]

`write_double`(**values*)

`write_float`(**values*)

`write_long`(**values*)

`write_long8`(**values*)

`write_rational`(**values*) [source]

`write_short`(**values*)

`write_signed_byte`(**values*)

`write_signed_long`(**values*)

`write_signed_rational`(**values*) [source]

`write_signed_short`(**values*)

`write_string`(*value*) [source]

`write_undefined`(*value*) [source]

class **`PIL.TiffImagePlugin.TiffImageFile`**(*fp=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'TIFF'*

`format_description`= *'Adobe TIFF'*

`get_photoshop_blocks`() [source]

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.

`getxmp()` [source]

Returns a dictionary containing the XMP tags. Requires defusedxml to be installed.

Returns: XMP tags in a dictionary.

`load()` [source]

Load image data based on tile list

`load_end()` [source]

property **`n_frames`**

`seek`(*frame*) [source]

Select a given frame as current image

`tag`

Legacy tag entries

`tag_v2`

Image file directory (tag dictionary)

`tell()` [source]

Return the current frame number

`WebPImagePlugin` Module

class **`PIL.WebPImagePlugin.WebPImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'WEBP'*

`format_description`= *'WebP image'*

`load()` [source]

Load image data based on tile list

seek(*frame*) [source]

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() [source]

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=None*, *filename=None*) [source]

Bases: `PIL.ImageFile.StubImageFile`

format= *'WMF'*

format_description= *'Windows Metafile'*

load(*dpi=None*) [source]

Load image data based on tile list

PIL.WmfImagePlugin.register_handler(*handler*) [source]

Install application-specific WMF image handler.

Parameters: **handler** – Handler object.

`XVThumbImagePlugin` Module

class **`PIL.XVThumbImagePlugin.XVThumbImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'XVThumb'*

`format_description`= *'XV thumbnail image'*

`XbmImagePlugin` Module

class **`PIL.XbmImagePlugin.XbmImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'XBM'*

`format_description`= *'X11 Bitmap'*

`XpmImagePlugin` Module

class **`PIL.XpmImagePlugin.XpmImageFile`**(*fp=None, filename=None*) [source]

Bases: `PIL.ImageFile.ImageFile`

`format`= *'XPM'*

`format_description`= *'X11 Pixel Map'*

`load_read`(*bytes*) [source]