

---

## **lilliput**

lilliput resizes images in Go.

Lilliput relies on mature, high-performance C libraries to do most of the work of decompressing, resizing and compressing images. It aims to do as little memory allocation as possible and especially not to create garbage in Go. As a result, it is suitable for very high throughput image resizing services.

Lilliput supports resizing JPEG, PNG, WEBP and animated GIFs. It can also convert formats. Lilliput also has some support for getting the first frame from MOV and WEBM videos.

**Lilliput presently only supports OSX and Linux.**

### **Example**

Lilliput comes with a fully working example that runs on the command line. The example takes a user supplied filename and prints some basic info about the file. It then resizes and transcodes the image (if flags are supplied) and saves the resulting file.

To use the example, `go get github.com/discord/lilliput` and then run `go build` from the `examples/` directory.

### **License**

Lilliput is released under MIT license (see LICENSE). Additionally, lilliput ships with other libraries, each provided under its own license. See `third-party-licenses` for more info.

### **Usage**

First, `import "github.com/discord/lilliput"`.

### **Decoder**

Lilliput is concerned with in-memory images, so the decoder requires image data to be in a `[]byte` buffer.

```
1 func lilliput.NewDecoder([]byte buf) (lilliput.Decoder, error)
```

Create a new `Decoder` object from the compressed image contained by `buf`. This will return an error when the magic bytes of the buffer don't match one of the supported image types.

---

```
1 func (d llliput.Decoder) Header() (llliput.ImageHeader, error)
```

Read and return the image's header. The header contains the image's metadata. Returns error if the image has a malformed header. An image with a malformed header cannot be decoded.

```
1 func (d llliput.Decoder) Description() string
```

Returns a string describing the image's type, e.g. "JPEG" or "PNG".

```
1 func (h llliput.Decoder) Duration() time.Duration
```

Returns the length of the content. Returns 0 for static images and animated GIFs.

```
1 func (d llliput.Decoder) DecodeTo(f *llliput.Framebuffer) error
```

Fully decodes the image and writes its pixel data to `f`. Returns an error if the decoding process fails. If the image contains multiple frames, then each call returns a subsequent frame. `io.EOF` is returned when the image does not contain any more data to be decoded.

**Users of llliput generally should not call `DecodeTo` and should instead use an `ImageOps` object.**

```
1 func (d llliput.Decoder) Close()
```

Closes the decoder and releases resources. The Decoder object must have `.Close()` called when it is no longer in use.

## ImageOps

Llliput provides a convenience object to handle image resizing and encoding from an open Decoder object. The ImageOps object can be created and then reused, which reduces memory allocations. Generally, users should prefer the ImageOps object over manually controlling the resize and encode process.

```
1 func llliput.NewImageOps(dimension int) *llliput.ImageOps
```

Create an ImageOps object that can operate on images up to `dimension x dimension` pixels in size. This object can be reused for multiple operations.

```
1 func (o *llliput.ImageOps) Transform(decoder llliput.Decoder, opts *llliput.ImageOptions, dst []byte) ([]byte, error)
```

Transform the compressed image contained in a Decoder object into the desired output type. **The decoder must not have `DecodeTo()` called on it already.** However, it is ok to call `decoder.Header`

---

( ) if you would like to check image properties before transforming the image. Returns an error if the resize or encoding process fails.

The resulting compressed image will be written into `dst`. The returned []byte slice will point to the same region as `dst` but with a different length, so that you can tell where the image ends.

Fields for `lilliput.ImageOptions` are as follows

- `FileType`: file extension type, e.g. `".jpeg"`
- `Width`: number of pixels of width of output image
- `Height`: number of pixels of height of output image
- `ResizeMethod`: one of `lilliput.ImageOpsNoResize` or `lilliput.ImageOpsFit`. `Fit` behavior is the same as `Framebuffer.Fit()` – it performs a cropping resize that does not stretch the image.
- `NormalizeOrientation`: If `true`, `Transform()` will inspect the image orientation and normalize the output so that it is facing in the standard orientation. This will undo JPEG EXIF-based orientation.
- `EncodeOptions`: Of type `map[int]int`, same options accepted as `Encoder.Encode()`. This controls output encode quality.

```
1 func (o *lilliput.ImageOps) Clear()
```

Clear out all pixel data contained in `ImageOps` object from any previous operations. This function does not need to be called between `Transform()` calls. The user may choose to do this if they want to remove image data from memory.

```
1 func (o *lilliput.ImageOps) Close()
```

Close the `ImageOps` object and release resources. The `ImageOps` object must have `.Close()` called when it is no longer in use.

## ImageHeader

This interface returns basic metadata about an image. It is created by calling `Decoder.Header()`.

```
1 func (h lilliput.ImageHeader) Width() int
```

Returns the image's width in number of pixels.

```
1 func (h lilliput.ImageHeader) Height() int
```

---

Returns the image's height in number of pixels.

```
1 func (h llliput.ImageHeader) PixelType() llliput.PixelType
```

Returns the basic pixel type for the image's pixels.

```
1 func (h llliput.ImageHeader) Orientation() llliput.ImageOrientation
```

Returns the metadata-based orientation of the image. This function can be called on all image types but presently only detects orientation in JPEG images. An orientation value of 1 indicates default orientation. All other values indicate some kind of rotation or mirroring.

## PixelType

```
1 func (p llliput.PixelType) Depth() int
```

Returns the number of bits per pixel.

```
1 func (p llliput.PixelType) Channels() int
```

Returns the number of channels per pixel, e.g. 3 for RGB or 4 for RGBA.

## Framebuffer

This type contains a raw array of pixels, decompressed from an image. In general, you will want to use the ImageOps object instead of operating on Framebuffers manually.

```
1 func llliput.NewFramebuffer(width, height int) *llliput.Framebuffer
```

Create a new Framebuffer with given dimensions without any pixel data.

```
1 func (f *llliput.Framebuffer) Clear()
```

Set contents of framebuffer to 0, clearing out any previous pixel data.

```
1 func (f *llliput.Framebuffer) Width() int
```

Returns the width in number of pixels of the contained pixel data, if any. This does not return the capacity of the buffer.

```
1 func (f *llliput.Framebuffer) Height() int
```

Returns the height in number of pixels of the contained pixel data, if any. This does not return the capacity of the buffer.

---

```
1 func (f *lilliput.Framebuffer) PixelType() lilliput.PixelType
```

Returns the `PixelType` of the contained pixel data, if any.

```
1 func (f *lilliput.Framebuffer) OrientationTransform(orientation
    lilliput.ImageOrientation)
```

Rotate and/or mirror framebuffer according to orientation value. If you pass the `orientation` value given by the image's `ImageHeader`, then the resulting image has its orientation normalized to the default orientation.

```
1 func (f *lilliput.Framebuffer) ResizeTo(width, height int, dst *
    lilliput.Framebuffer) error
```

Perform a resize into `dst` of `f` according to given dimensions. This function does not preserve the source's aspect ratio if the new dimensions have a different ratio. The resize can fail if the destination is not large enough to hold the new image.

```
1 func (f *lilliput.Framebuffer) Fit(width, height int, dst *lilliput.
    Framebuffer) error
```

Perform a cropping resize into `dst` of `f` according to given dimensions. This function does preserve the source's aspect ratio. The image will be cropped along one axis if the new dimensions have a different ratio than the source. The cropping will occur equally on the edges, e.g. if the source image is too tall for the new ratio, then the destination will have rows of pixels from the top and bottom removed. Returns error if the destination is not large enough to contain the resized image.

```
1 func (f *lilliput.Framebuffer) Close()
```

Closes the framebuffer and releases resources. The `Framebuffer` object must have `.Close()` called when it is no longer in use.

## Encoder

The Encoder takes a Framebuffer and writes the pixels into a compressed format.

```
1 func lilliput.NewEncoder(extension string, decodedBy lilliput.Decoder,
    dst []byte) (lilliput.Encoder, error)
```

Create a new Encoder object that writes to `dst`. `extension` should be a file extension-like string, e.g. `".jpeg"` or `".png"`. `decodedBy` should be the `Decoder` used to decompress the image, if any. `decodedBy` may be left as `nil` in most cases but is required when creating a `.gif` encoder. That is, `.gif` outputs can only be created from source GIFs.

---

```
1 func (e lilliput.Encoder) Encode(buffer lilliput.Framebuffer, opts map[
    int]int) ([]byte, error)
```

Encodes the Framebuffer supplied into the output `dst` given when the `Encoder` was created. The returned `[]byte` will point to the same buffer as `dst` but can be a shorter slice, so that if `dst` has 50MB of capacity but the image only occupies 30KB, you can tell where the image data ends. This function returns an error if the encoding process fails.

`opts` is optional and may be left `nil`. It is used to control encoder behavior e.g. `map[int]int{ lilliput.JpegQuality: 80}` to set JPEG outputquality to 80.

Valid keys/values for `opts` are

- `JpegQuality` (1 - 100)
- `PngCompression` (0 - 9)
- `WebpQuality` (0 - 100).

```
1 func (e lilliput.Encoder) Close()
```

Close the Encoder and release resources. The `Encoder` object must have `.Close()` called when it is no longer in use.

## Building Dependencies

Go does not provide any mechanism for arbitrary building of dependencies, e.g. invoking `make` or `cmake`. In order to make lilliput usable as a standard Go package, prebuilt static libraries have been provided for all of lilliput's dependencies on Linux and OSX. In order to automate this process, lilliput ships with build scripts alongside compressed archives of the sources of its dependencies. These build scripts are provided for OSX and Linux.