# nativeImage

> *Create tray, dock, and application icons using PNG or JPG files.*

Process: Main, Renderer

In Electron, for the APIs that take images, you can pass either file paths or `NativeImage` instances. An empty image will be used when `null` is passed.

For example, when creating a tray or setting a window's icon, you can pass an image file path as a `string`:

```
const { BrowserWindow, Tray } = require('electron')

const appIcon = new Tray('/Users/somebody/images/icon.png')
const win = new BrowserWindow({ icon: '/Users/somebody/images/window.png' })
console.log(appIcon, win)
```

Or read the image from the clipboard, which returns a `NativeImage`:

```
const { clipboard, Tray } = require('electron')
const image = clipboard.readImage()
const appIcon = new Tray(image)
console.log(appIcon)
```

## Supported Formats

Currently `PNG` and `JPEG` image formats are supported. `PNG` is recommended because of its support for transparency and lossless compression.

On Windows, you can also load `ICO` icons from file paths. For best visual quality, it is recommended to include at least the following sizes in the:

- Small icon
  - 16x16 (100% DPI scale)
  - 20x20 (125% DPI scale)
  - 24x24 (150% DPI scale)
  - 32x32 (200% DPI scale)
- Large icon
  - 32x32 (100% DPI scale)
  - 40x40 (125% DPI scale)
  - 48x48 (150% DPI scale)
  - 64x64 (200% DPI scale)
  - 256x256

Check the *Size requirements* section in this article.

## High Resolution Image

On platforms that have high-DPI support such as Apple Retina displays, you can append `@2x` after image's base filename to mark it as a high resolution image.

For example, if `icon.png` is a normal image that has standard resolution, then `icon@2x.png` will be treated as a high resolution image that has double DPI density.

If you want to support displays with different DPI densities at the same time, you can put images with different sizes in the same folder and use the filename without DPI suffixes. For example:

```
images/
├── icon.png
├── icon@2x.png
└── icon@3x.png
```

```
const { Tray } = require('electron')
const appIcon = new Tray('/Users/somebody/images/icon.png')
console.log(appIcon)
```

The following suffixes for DPI are also supported:

- `@1x`
- `@1.25x`
- `@1.33x`
- `@1.4x`
- `@1.5x`
- `@1.8x`
- `@2x`
- `@2.5x`
- `@3x`
- `@4x`
- `@5x`

## Template Image

Template images consist of black and an alpha channel. Template images are not intended to be used as standalone images and are usually mixed with other content to create the desired final appearance.

The most common case is to use template images for a menu bar icon, so it can adapt to both light and dark menu bars.

**Note:** Template image is only supported on macOS.

To mark an image as a template image, its filename should end with the word `Template`. For example:

- `xxxTemplate.png`
- `xxxTemplate@2x.png`

## Methods

The `nativeImage` module has the following methods, all of which return an instance of the `NativeImage` class:

### `nativeImage.createEmpty()`

Returns `NativeImage`

Creates an empty `NativeImage` instance.

### `nativeImage.createThumbnailFromPath(path, maxSize)` *macOS Windows*

- `path` string - path to a file that we intend to construct a thumbnail out of.
- `maxSize` Size - the maximum width and height (positive numbers) the thumbnail returned can be. The Windows implementation will ignore `maxSize.height` and scale the height according to `maxSize.width`.

Returns `Promise<NativeImage>` - fulfilled with the file's thumbnail preview image, which is a NativeImage.

### `nativeImage.createFromPath(path)`

- `path` string

Returns `NativeImage`

Creates a new `NativeImage` instance from a file located at `path`. This method returns an empty image if the `path` does not exist, cannot be read, or is not a valid image.

```
const nativeImage = require('electron').nativeImage

const image = nativeImage.createFromPath('/Users/somebody/images/icon.png')
console.log(image)
```

### `nativeImage.createFromBitmap(buffer, options)`

- `buffer` Buffer
- `options` Object
  - `width` Integer
  - `height` Integer
  - `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `NativeImage`

Creates a new `NativeImage` instance from `buffer` that contains the raw bitmap pixel data returned by `toBitmap()`. The specific format is platform-dependent.

### `nativeImage.createFromBuffer(buffer[, options])`

- `buffer` Buffer
- `options` Object (optional)
  - `width` Integer (optional) - Required for bitmap buffers.
  - `height` Integer (optional) - Required for bitmap buffers.
  - `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `NativeImage`

Creates a new `NativeImage` instance from `buffer`. Tries to decode as PNG or JPEG first.

### `nativeImage.createFromDataURL(dataURL)`

- `dataURL` string

Returns `NativeImage`

Creates a new `NativeImage` instance from `dataURL`.

### `nativeImage.createFromNamedImage(imageName[, hslShift])` *macOS*

- `imageName` string
- `hslShift` number[] (optional)

Returns `NativeImage`

Creates a new `NativeImage` instance from the NSImage that maps to the given image name. See `System Icons` for a list of possible values.

The `hslShift` is applied to the image with the following rules:

- `hsl_shift[0]` (hue): The absolute hue value for the image - 0 and 1 map to 0 and 360 on the hue color wheel (red).
- `hsl_shift[1]` (saturation): A saturation shift for the image, with the following key values: 0 = remove all color. 0.5 = leave unchanged. 1 = fully saturate the image.
- `hsl_shift[2]` (lightness): A lightness shift for the image, with the following key values: 0 = remove all lightness (make all pixels black). 0.5 = leave unchanged. 1 = full lightness (make all pixels white).

This means that `[-1, 0, 1]` will make the image completely white and `[-1, 1, 0]` will make the image completely black.

In some cases, the `NSImageName` doesn't match its string representation; one example of this is `NSFolderImageName`, whose string representation would actually be `NSFolder`. Therefore, you'll need to determine the correct string representation for your image before passing it in. This can be done with the following:

```
echo -e '#import <Cocoa/Cocoa.h>\nint main() { NSLog(@"%@", SYSTEM_IMAGE_NAME); }' | clang -otest -x objective-c -framework Cocoa - && ./test
```

where `SYSTEM_IMAGE_NAME` should be replaced with any value from this list.

## Class: NativeImage

> *Natively wrap images such as tray, dock, and application icons.*

Process: Main, Renderer
*This class is not exported from the `'electron'` module. It is only available as a return value of other methods in the Electron API.*

### Instance Methods

The following methods are available on instances of the `NativeImage` class:

#### `image.toPNG([options])`

- `options` Object (optional)
  - `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `Buffer` - A Buffer that contains the image's `PNG` encoded data.

### `image.toJPEG(quality)`

- `quality` Integer - Between 0 - 100.

Returns `Buffer` - A Buffer that contains the image's `JPEG` encoded data.

### `image.toBitmap([options])`

- `options` Object (optional)
  - `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `Buffer` - A Buffer that contains a copy of the image's raw bitmap pixel data.

### `image.toDataURL([options])`

- `options` Object (optional)
  - `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `string` - The data URL of the image.

### `image.getBitmap([options])`

- `options` Object (optional)
  - `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `Buffer` - A Buffer that contains the image's raw bitmap pixel data.

The difference between `getBitmap()` and `toBitmap()` is that `getBitmap()` does not copy the bitmap data, so you have to use the returned Buffer immediately in current event loop tick; otherwise the data might be changed or destroyed.

### `image.getNativeHandle()` *macOS*

Returns `Buffer` - A Buffer that stores C pointer to underlying native handle of the image. On macOS, a pointer to `NSImage` instance would be returned.

Notice that the returned pointer is a weak pointer to the underlying native image instead of a copy, so you *must* ensure that the associated `nativeImage` instance is kept around.

### `image.isEmpty()`

Returns `boolean` - Whether the image is empty.

### `image.getSize([scaleFactor])`

- `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `Size` .

If `scaleFactor` is passed, this will return the size corresponding to the image representation most closely matching the passed value.

### `image.setTemplateImage(option)`

- `option` boolean

Marks the image as a template image.

**`image.isTemplateImage()`**

Returns `boolean` - Whether the image is a template image.

**`image.crop(rect)`**

- `rect` Rectangle - The area of the image to crop.

Returns `NativeImage` - The cropped image.

**`image.resize(options)`**

- `options` Object
  - `width` Integer (optional) - Defaults to the image's width.
  - `height` Integer (optional) - Defaults to the image's height.
  - `quality` string (optional) - The desired quality of the resize image. Possible values are `good`, `better`, or `best`. The default is `best`. These values express a desired quality/speed tradeoff. They are translated into an algorithm-specific method that depends on the capabilities (CPU, GPU) of the underlying platform. It is possible for all three methods to be mapped to the same algorithm on a given platform.

Returns `NativeImage` - The resized image.

If only the `height` or the `width` are specified then the current aspect ratio will be preserved in the resized image.

**`image.getAspectRatio([scaleFactor])`**

- `scaleFactor` Double (optional) - Defaults to 1.0.

Returns `Float` - The image's aspect ratio.

If `scaleFactor` is passed, this will return the aspect ratio corresponding to the image representation most closely matching the passed value.

**`image.getScaleFactors()`**

Returns `Float[]` - An array of all scale factors corresponding to representations for a given nativeImage.

**`image.addRepresentation(options)`**

- `options` Object
  - `scaleFactor` Double - The scale factor to add the image representation for.
  - `width` Integer (optional) - Defaults to 0. Required if a bitmap buffer is specified as `buffer`.
  - `height` Integer (optional) - Defaults to 0. Required if a bitmap buffer is specified as `buffer`.
  - `buffer` Buffer (optional) - The buffer containing the raw image data.
  - `dataURL` string (optional) - The data URL containing either a base 64 encoded PNG or JPEG image.

Add an image representation for a specific scale factor. This can be used to explicitly add different scale factor representations to an image. This can be called on empty images.

### Instance Properties

**`nativeImage.isMacTemplateImage`** ***macOS***

A `boolean` property that determines whether the image is considered a template image.

Please note that this property only has an effect on macOS.