Swift Jun 16, 2021 • 11 min read

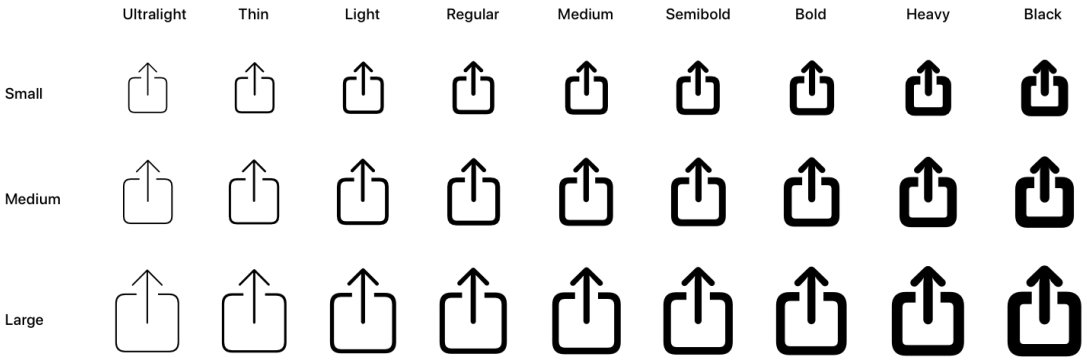
SF Symbols: The benefits and how to use them guide

SF Symbols were introduced during WWDC 2019 and are a big present for us developers. Apple basically gave us free symbols to use in our app, and it's straightforward to use them as well! With SF Symbols 2.0 being introduced in WWDC 2020 and 3.0 at WWDC 2021, we've got even more possibilities to show beautiful icons in our apps.

It's time to dive into the details of what they are and how you can use them to make your app look nice and natively aligned with the system apps.
You can find Swift code examples at the bottom of this post.

What are SF Symbols?

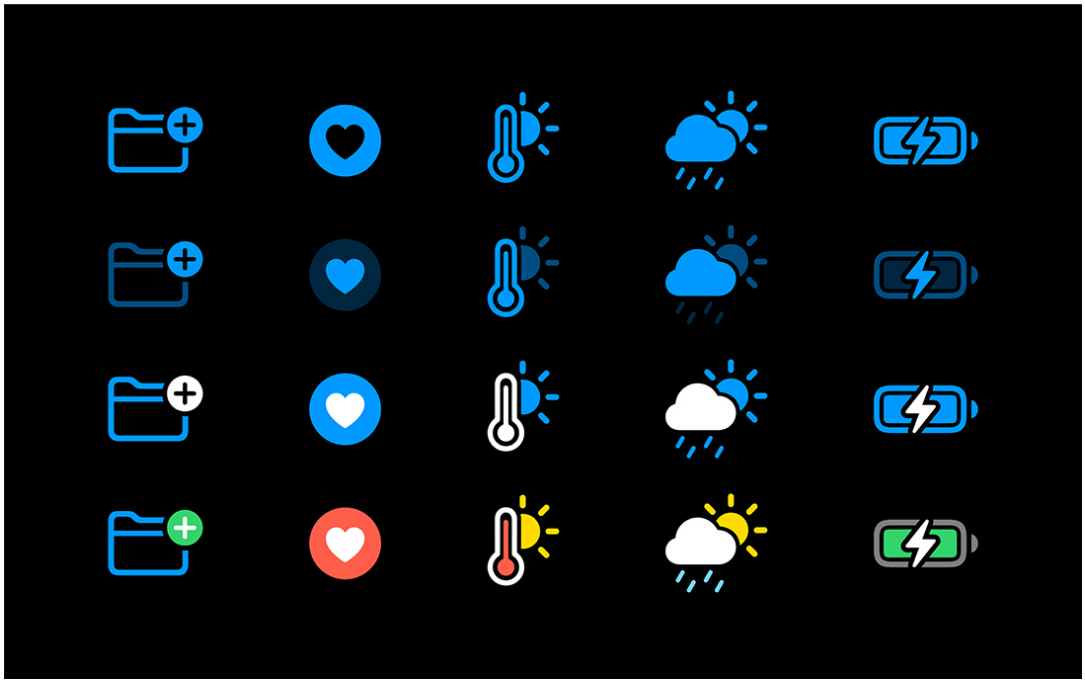
SF Symbols is a set of over 3,100 symbols that you can use in your app. They're aligned and configurable in a wide range of weights and scales to adapt to your designs. As they are integrated into the San Francisco system font, they automatically ensure optical vertical alignment with text for all weights and sizes.



An SF Symbol example with all its scales and weights

Enhanced color customisation

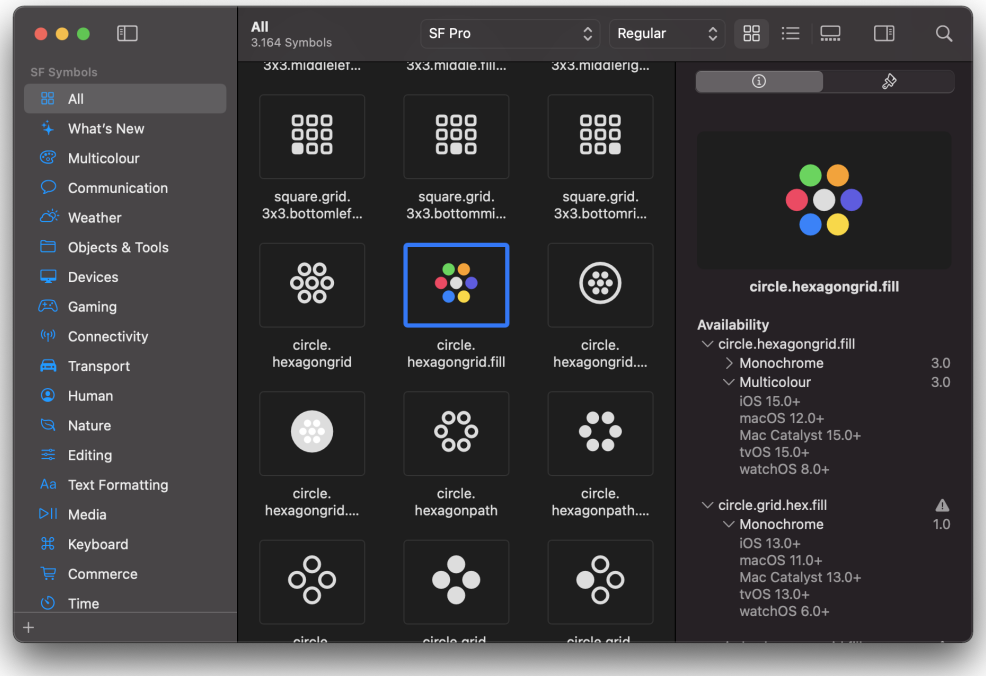
SF Symbols 3.0 introduced the option to configure a symbol with different rendering modes, providing greater control over how color is applied to symbols.



SF Symbols Rendering modes

- Hierarchical rendering to add depth and emphasis using a single tint color with multiple levels of opacity
- Palette rendering using multiple contrasting colors
- Multicolor rendering provides intrinsic colors across hundreds of symbols

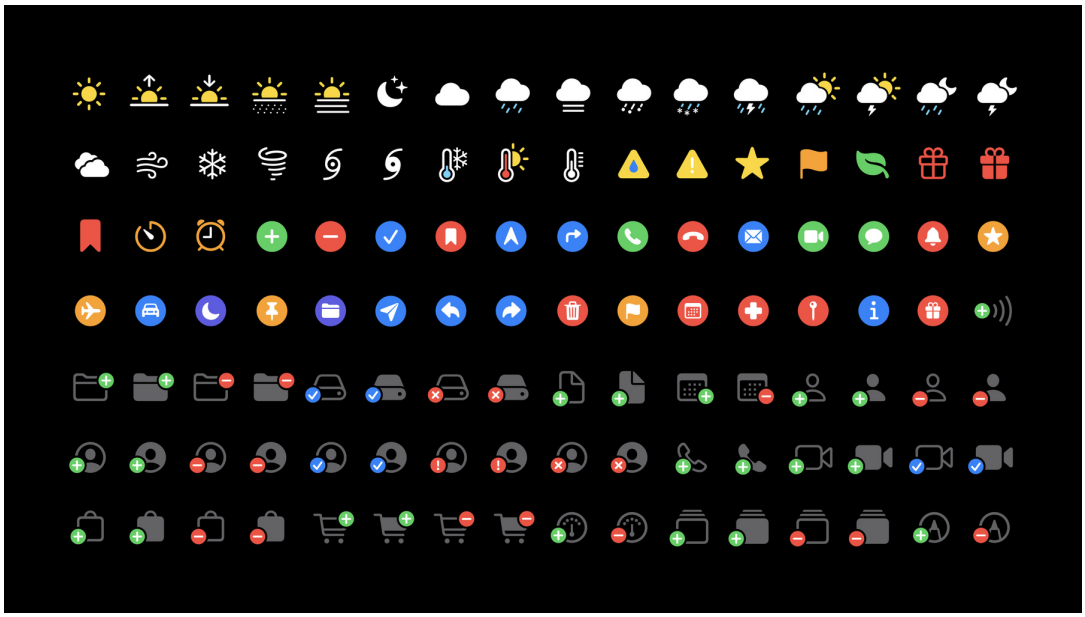
Not all symbols support every rendering mode. Verify supported modes from within the SF Symbols 3.0 app inside the availability section:



SF Symbols rendering mode availability as shown in the app.

By default, monochrome rendering is applied.

MULTICOLOR SYMBOLS



SF Symbols 2.0 adds support for multicolor symbols

SF Symbols 2.0 already introduced multicolor symbols. Multicolor symbols are unique compared to tinted monochrome symbols as they automatically adapt to appearance modes, accessibility settings, and vibrancy. This can be a great way to support many different scenarios quickly. Support for multicolor is added to more symbols in every release.

Optical alignment in symbols

One of the challenges of using SF Symbols is optical alignment. Vertical alignment is automatically ensured if used with the San Fransisco system font. Still, up until version 1.1, it wasn't possible to horizontally align symbols to make them look good in all cases.

With the 2.0 update, you can now give symbols negative side margins supported by standard and custom symbols. This should give you enough control to visually align symbols horizontally. For example, you might have a symbol that includes a badge, in which case you want to move the symbol a bit horizontally visually.

A small caveat is that two of those symbols could be placed next to each other, with negative margins overlapping both symbols. Although it's a rare case, you might need to add extra space or other content in between to avoid collisions.

Localized symbols variants

SF Symbols 2.0 introduced localized symbols that support right-to-left writing systems and script-specific symbols for Hebrew, Arabic, and Devanagari. These variants are optimized to look great in those systems. The 3.0 release added new designs for Thai, Chinese, Japanese, and Korean and expanded coverage for Arabic, Hebrew, and Devanagari.

Which platforms support SF Symbols?

The supported platforms for SF Symbols are:

- iOS 13 and later
- watchOS 6 and later
- macOS 11 and later
- tvOS 13 and later

Each system version can affect the availability of individual symbols and features. As an alternative, you can export new symbols from the latest SF Symbols app as SVG and bundle them within your app so that you can use them in older OS versions. In this case, you won't be able to benefit from features like different rendering modes.

As the symbols are included in the San Francisco system font, you can also use them in a Mac app, but license agreements apply.

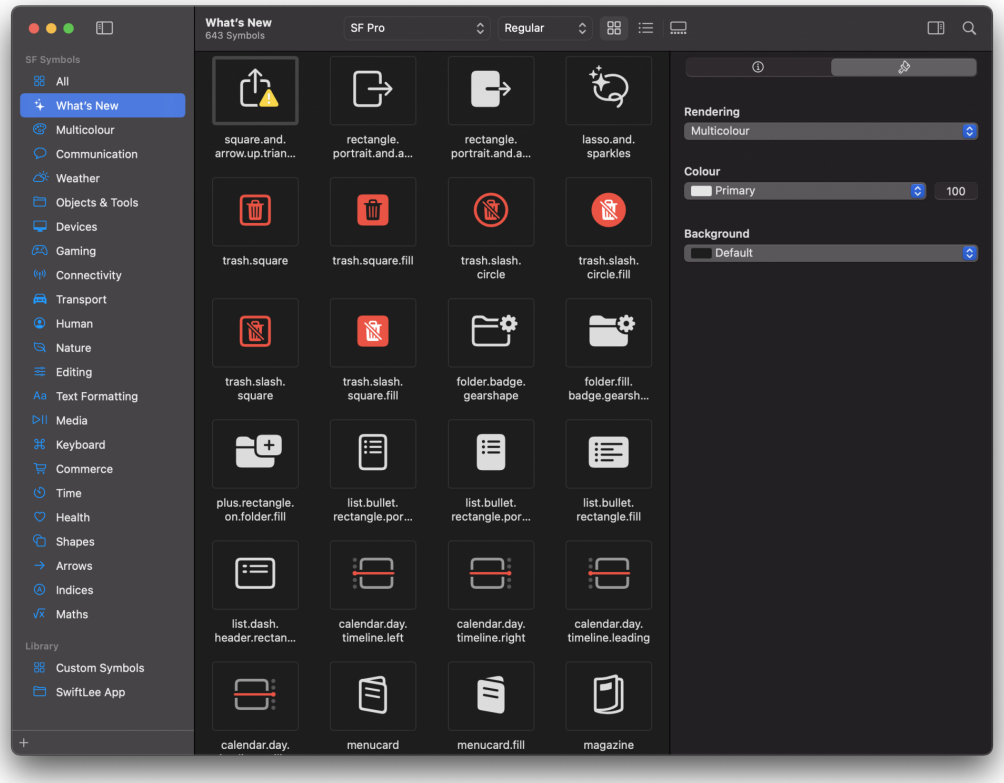
Can I use the symbols everywhere?

No, definitely not! Keep an eye sharp on the license agreements that apply. As quoted from Apple:

> *You may not use SF Symbols — or glyphs that are substantially or confusingly similar — in your app icons, logos, or any other trademark-related use. Apple reserves the right to review and, in its sole discretion, require modification or discontinuance of use of any Symbol used in violation of the foregoing restrictions, and you agree to promptly comply with any such request.*

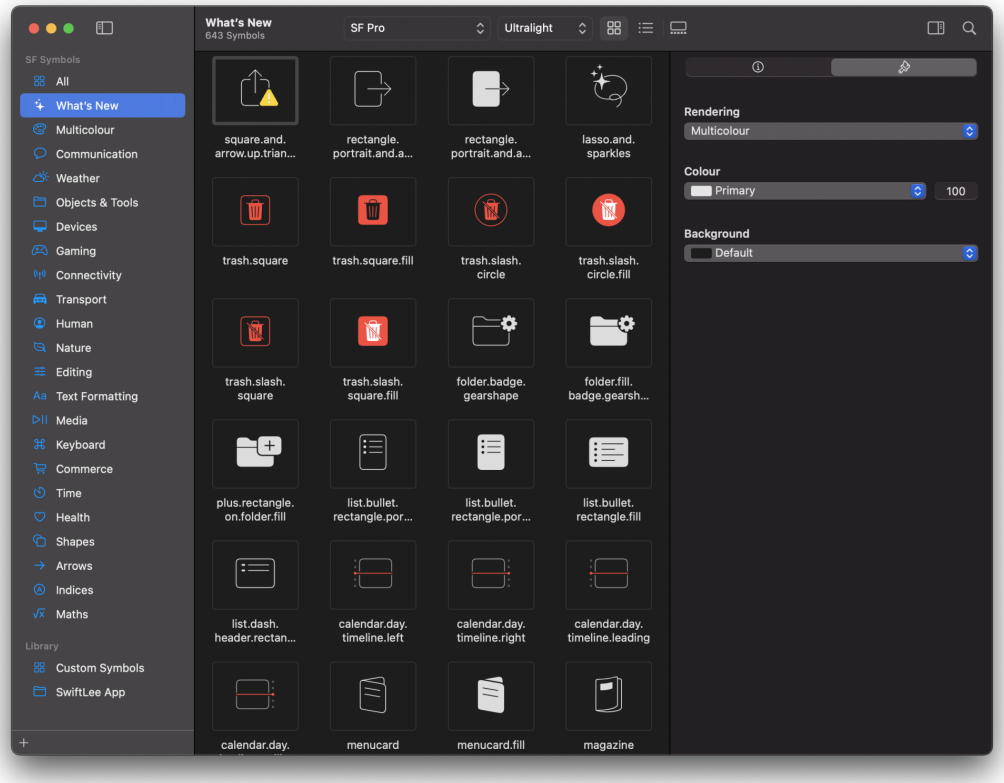
How can I browse the available symbols?

Apple provides an SF Symbols app that allows you to browse, copy, and export any available symbols. The app can be downloaded here and is available for macOS 10.14 and later.



The SF Symbols Mac app with an overview of all the symbols.

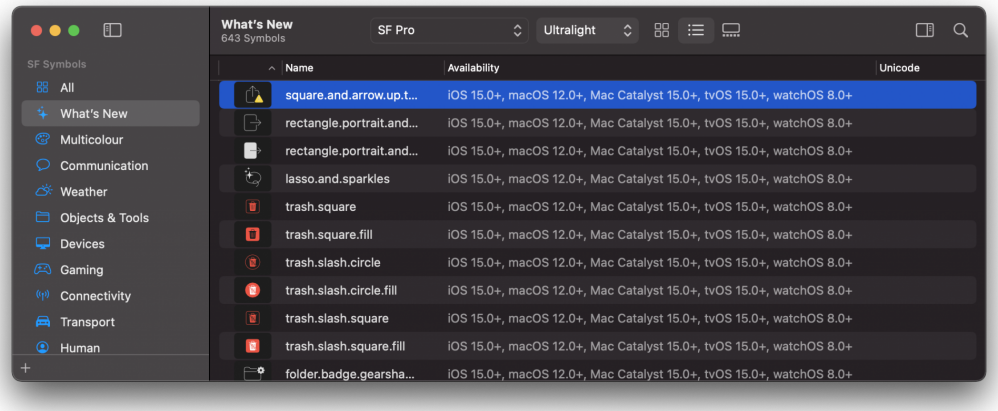
The app allows you to browse the symbols and display them in the selected weight. The above image shows the icon in the regular weight, while the next image shows how the same icons look in the UltraLight weight:



An overview of the SF Symbols with the ultra-lightweight

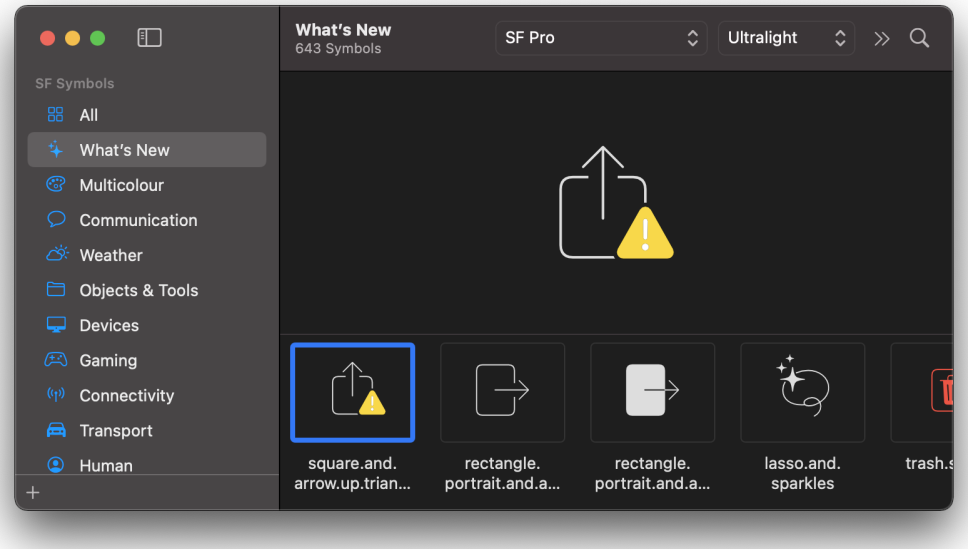
This is great for finding the right icon in the weight you want to use.

If you like, you could also use the list view to browse the available symbols.



An overview of the symbols in a list view

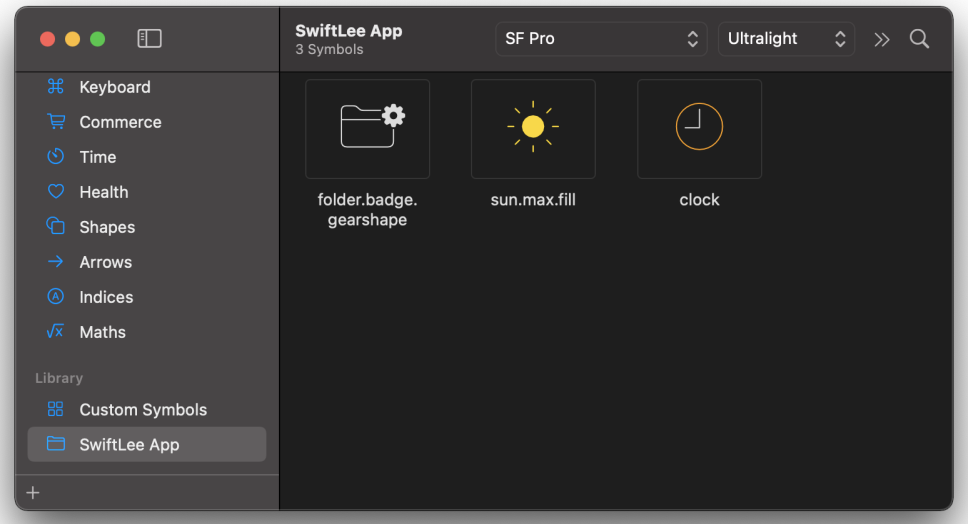
And lastly, you can also decide to show symbols in the gallery view, displaying the selected symbol in large:



Display symbols in the gallery view to get a large preview.

Creating a collection of symbols

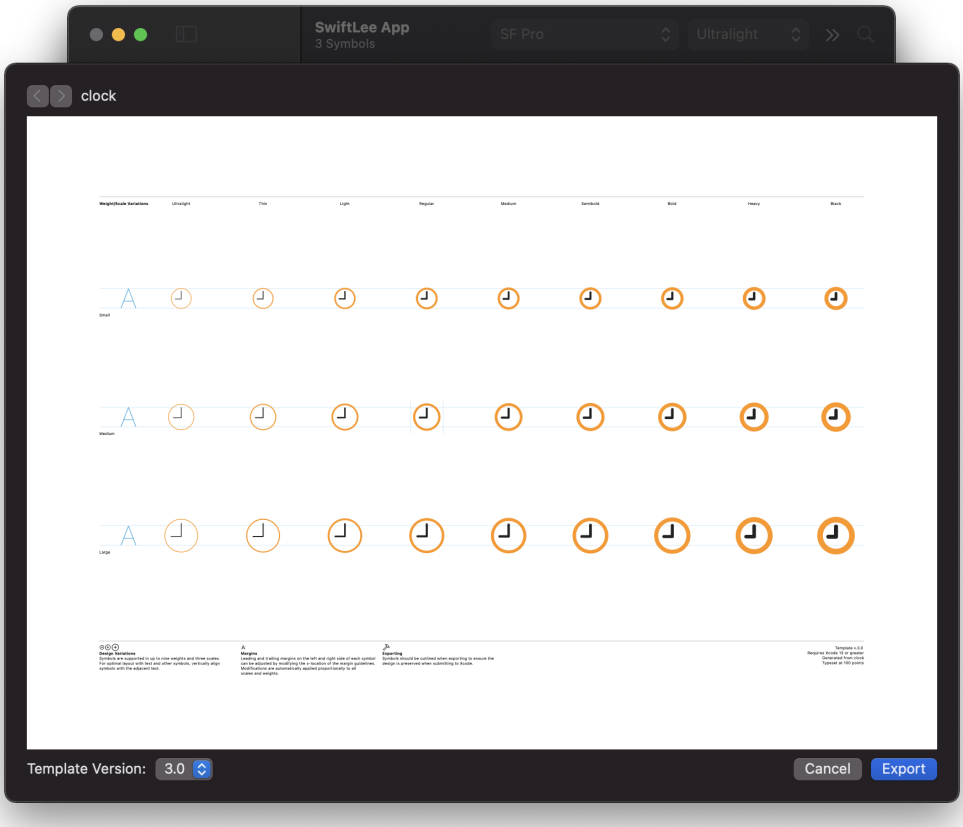
The 3.0 version of the app allows creating a collection of symbols. This can be a great way to manage symbols for your app by making an overview of all used symbols:



Create a collection to create an overview of your app's symbols.

Exporting a symbol

The Mac app allows you to support all symbols as an SVG using `File -> Export Symbol...` . This is a great way to include the symbol in places where you can not use the font itself. However, do keep in mind the license agreements.



An example export of a symbol using the 3.0 template version.

Browsing the symbols on the web

It's worth mentioning that you can also browse the symbols at sfsymbols.com.

One benefit you have by searching on this website is that you can see the applying restrictions for each icon if they exist:

sfsymbols.com showing the restrictions per symbol

However, due to the earlier described license restrictions, they can only display the names of the symbols and you are still better off by using the Mac app.

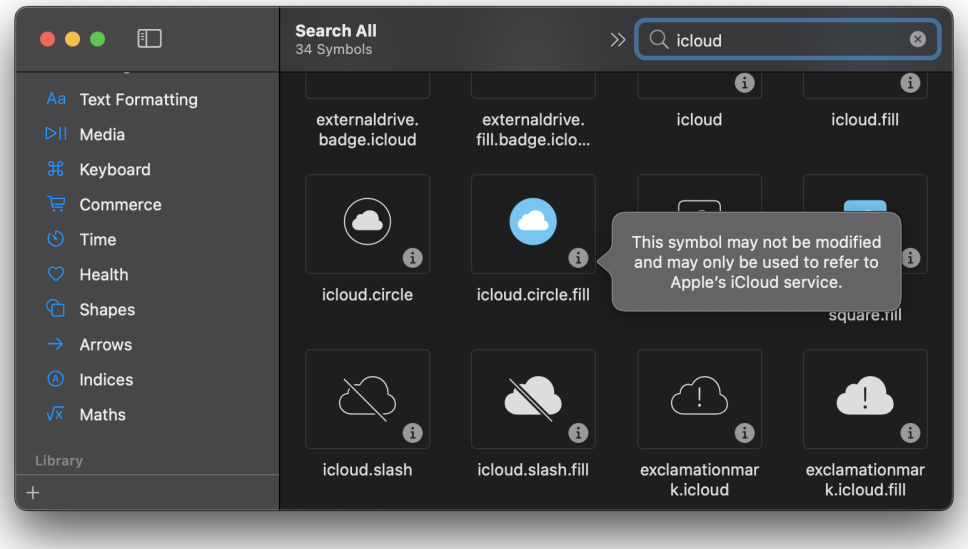
Creating a custom symbol

A custom symbol can be created whenever there's no symbol available matching your requirements. The exported SVG version of a symbol is well structured and can be used as a base for your custom symbol.

> *The Create custom symbols session from WWDC 2021 gives an in-depth explanation of how to create custom symbols*

Choosing the right symbol as a base reference

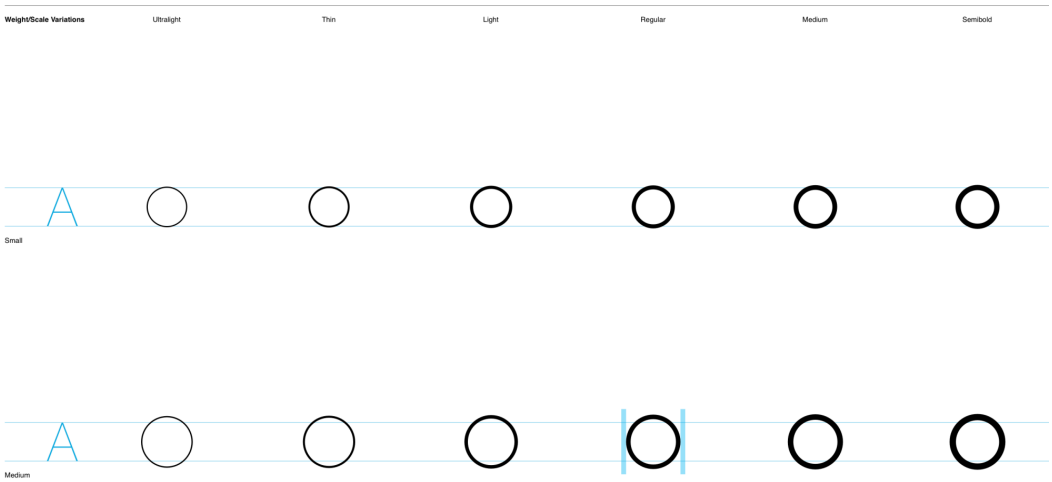
It's best to start with a symbol that is close to the one you'd like to have. Do know that SF Symbols includes copyrighted ones and you can't use replicas of Apple products and features or reproduce them in your custom symbols. Such symbols are marked with an info glyph badge and can also be identified within the app inside the inspector pane.



Some symbols are restricted to use and come with a copyright.

Once you found a symbol that's close to what you need, you can right-click it and select "Duplicate as Custom Symbol". This will move the symbol into the custom symbols collection from which you can start editing, right inside the SF Symbols app.

Most of the shapes are also available without an icon in them. For example the `circle` or `circle.fill`:



The circle symbol is a great base for custom symbols

This is a great starting point for your custom symbol!

Editing a custom symbol from within the SF Symbols App

You don't always need to go into Sketch or any other SVG editing app to create a custom symbol. You can also decide to alter an existing SF Symbol from within the SF Symbols App. For example, you can decide to change the colors for the Multicolour rendering:

Alternate an existing symbol and change its colours.

Each layer of the symbol can be dragged to the right-side layers list. Once added as a layer, you can alter its colours for each rendering mode. Another option could be to change its style for the Hierarchical rendering mode. By setting the second layer to Secondary, you'll see that a slightly lighter gray is applied automatically:

Adjusting the style of a symbol for the Hierarchical rendering mode.

These adjustments can also be done for your custom symbols and makes it easy to apply different colours for different layers.

Do I need to support all weights and scales?

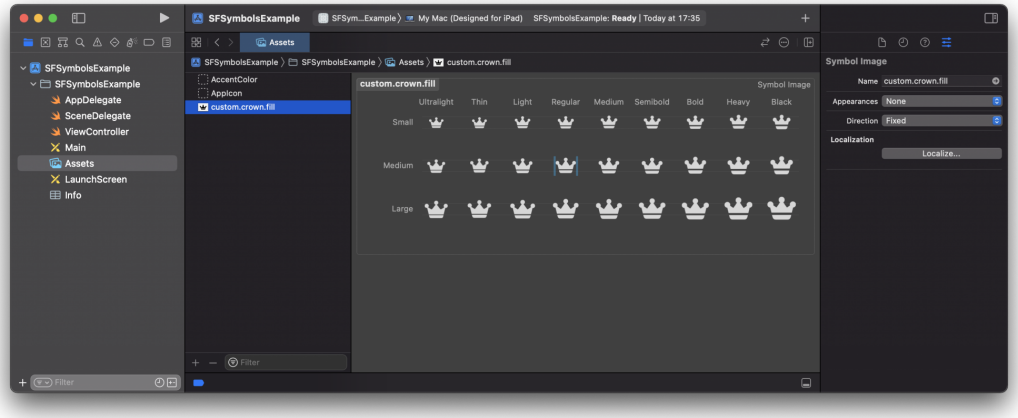
Well, the more weights and scales you implement, the wider the range of text settings you'll support. The minimum should be regular, medium, semibold, and bold at all scales so you can support Dynamic Type and the bold text setting.

How can I validate my custom SF Symbol?

The app allows you to validate the custom SF Symbol through the menu: `File → Validate Templates...`. Navigate and select your custom symbol to start the validation.

Exporting and using the custom symbol

Once you exported your symbol to an SVG you can simply drag the symbol into your Asset Catalog in Xcode.



A custom symbol added to an asset catalog in Xcode

It shows a preview of your symbol and it allows you to configure its name, appearance, direction, and localization.

Note that at the moment of writing this article, SF Symbols template version 3.0 isn't correctly recognized in Xcode 13 yet. This is expected to change in a future update.

Using the custom symbol on iOS 12 and below

Symbols are only supported on iOS 13 and up. Apple did a great job to make it easy for us to fall back on an image in the case of an older system version. Simply give your image the same name as your custom symbol and the following line of code will fallback to the image if needed:

```
let customSymbol = UIImage(named: "my_custom_symbol")
```

How to use SF Symbols in Swift

UIKIT

To use an SF Symbol in Swift you can make use of the new `UIImage(systemName:)` initializer:

```
let image = UIImage(systemName: "square.and.pencil")
```

As you can see, Apple made it really easy to use the SF symbols:

- Browse and find your icon in the SF Symbols Mac app

- Use `⇧⌘C` to copy the name of the symbol
- Use the name inside the `UIImage(systemName:)` initializer
- Use the image inside a UIImageView, UIButton, or any other UI element

SWIFTUI

Using symbols in SwiftUI can be done as follows:

```
Image(systemName: "square.and.pencil")
```

Or combine an image with a label:

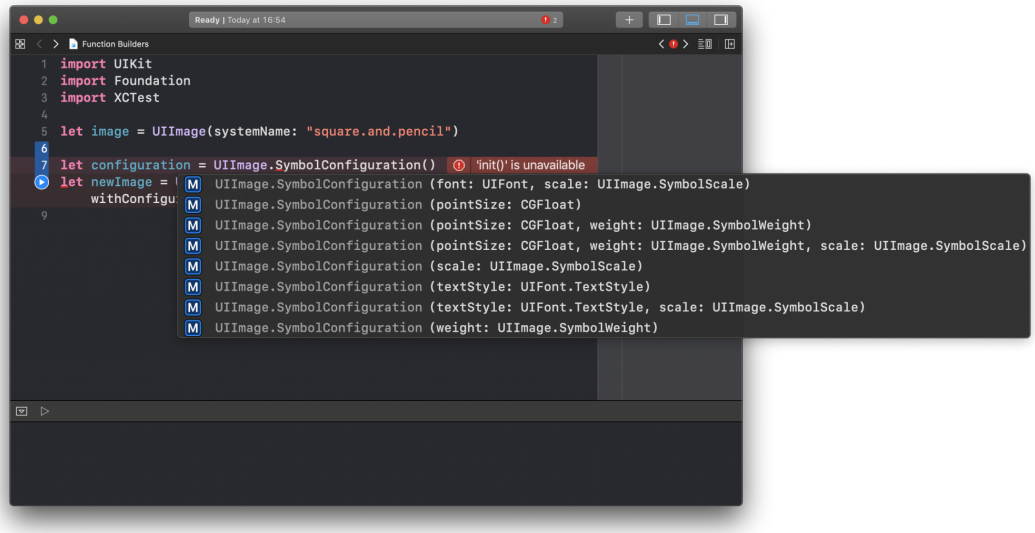
```
Label("Heart", systemImage: "heart")
```

You can even combine multiline strings with symbols:

```
Text("""
    This SwiftLee example is combined with
    a symbol \(Image(systemName: "chevron"))
""")
```

How to change the scale of an SF Symbol in Swift?

To change the scale of an SF Symbol you can make use of a UIImage Symbol Configuration.

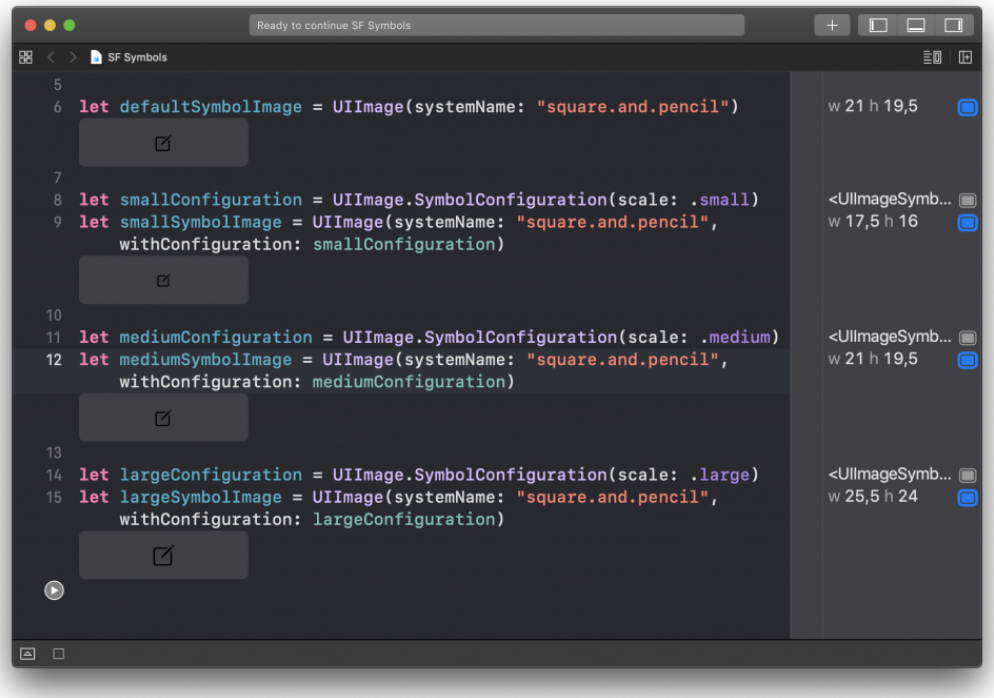


The available initializers for a UIImage Symbol Configuration

It allows you to set the font, scale, point size, weight, and text style.

```
let smallConfiguration = UIImage.SymbolConfiguration(scale: .small)
let smallSymbolImage = UIImage(systemName: "square.and.pencil", withConfigur
```

The scale is set to `medium` by default, as you can see in the following Swift Playground:



An SF Symbol can be set to a different scale using the Symbol Configuration

SWIFTUI

In SwiftUI, you can set the scale as follows:

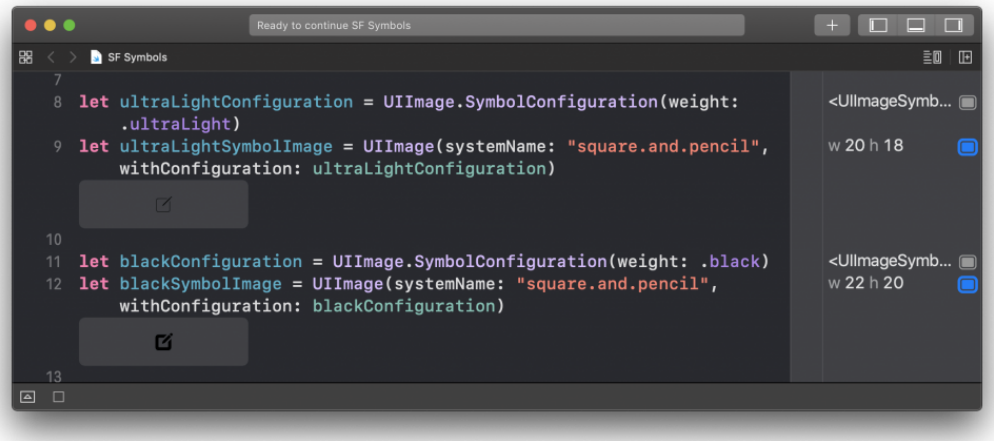
```
Image(systemName: "square.and.pencil")
    .imageScale(.large)
```

How to change the weight of an SF Symbol in Swift?

The weight can be changed with a UIImage Symbol Configuration as well.

```
let ultraLightConfiguration = UIImage.SymbolConfiguration(weight: .ultraLigh
let ultraLightSymbolImage = UIImage(systemName: "square.and.pencil", withCon
```

It's a great way to make your symbol align better with your design.



Examples of weight configurations for an SF Symbol

SWIFTUI

In SwiftUI, you can change the font and colours using view modifiers:

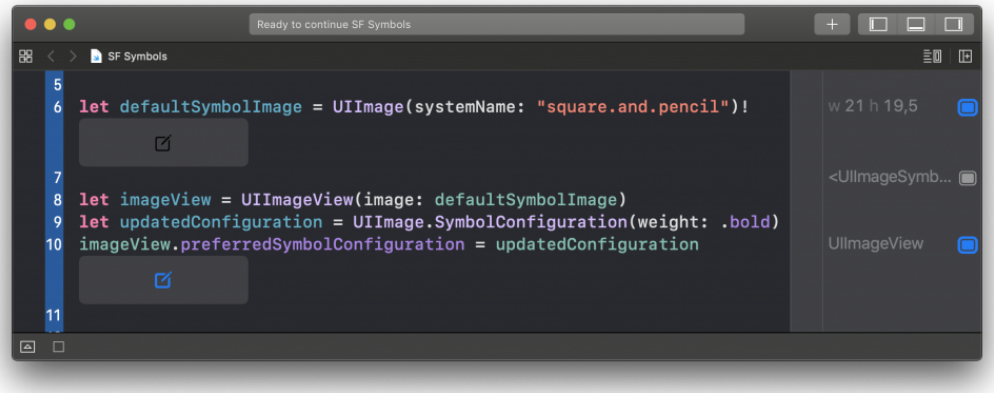
```
Image(systemName: "heart")
    .foregroundColor(.red)
    .font(.body)
```

Changing the Symbol Configuration in an UIImageView and UIButton

The `UIImageView` and `UIButton` class comes with a new property `preferredSymbolConfiguration` which allows you to apply a specific configuration to any symbols set in the image view.

```
let imageView = UIImageView(image: defaultSymbolImage)
let updatedConfiguration = UIImage.SymbolConfiguration(weight: .bold)
imageView.preferredSymbolConfiguration = updatedConfiguration
```

This will change the symbol whenever it's set in the image view. As you can see in the following Swift Playground image the symbol is also changed into blue color.



A configured UIImageView with an SF Symbol

This is inherited from the tint color set on the image view. If you want to make sure a symbol always appears in a specific color, you can make use of the `withTintColor` method:

```
let heartImage = UIImage(systemName: "heart.fill")!
let redHeartImage = heartImage.withTintColor(.red, renderingMode: .alwaysOri
```

Setting a fixed tint color on an SF Symbol

For buttons, it works more or less the same by making use of the `UIButton.setPreferredSymbolConfiguration(_:forImageIn:)` method. Default styles applied to system buttons are `.body` and `.large`.

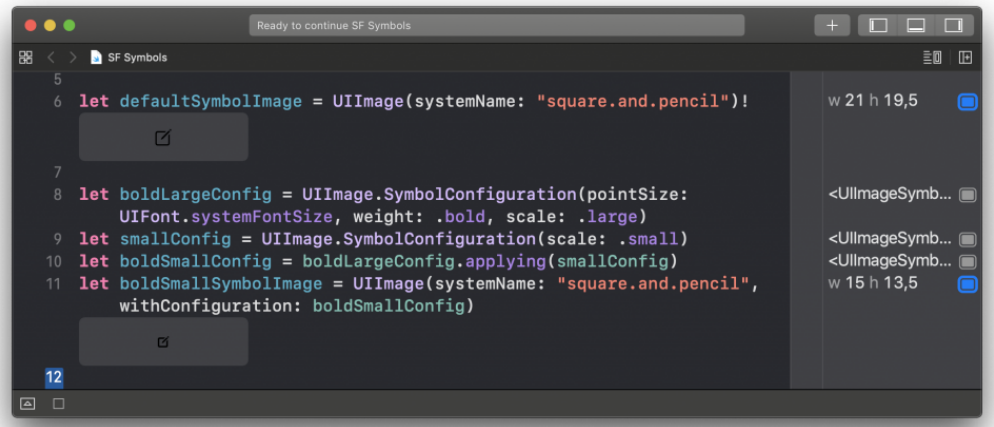
Combining configurations

In the case you're saving multiple configurations to use throughout your app, it could be that you want to apply a small adjustment on top of a shared configuration.

You can do this easily using the `applying` method.

```
let boldLargeConfig = UIImage.SymbolConfiguration(pointSize: UIFont.systemFo
let smallConfig = UIImage.SymbolConfiguration(scale: .small)
let boldSmallConfig = boldLargeConfig.applying(smallConfig)
let boldSmallSymbolImage = UIImage(systemName: "square.and.pencil", withConf
```

As you can see in the following Playground, the image is smaller but bold:



By combining two Symbol Configurations you can be flexible

Aligning symbols

The new `UIImage.withBaselineOffset(fromBottom:)` method allows you to apply a new offset to an image from the bottom of the image. This method can also be useful in the general use of images.

Configuring symbol variants

Symbol variants allow to change the body of symbols.

Symbols can be configured in many different variants as the above image shows. You can use the following code to do so in SwiftUI:

```
Image(systemName: "square.and.pencil")
    .symbolVariant(.rectangle)
```

As you can see, you no longer have to define `.fill` inside the symbol name. This is a new feature starting from iOS 15 and makes it possible for the system to adjust icons accordingly based on where they are used. For example, the outline variant can be used in tab bars on iOS, while the filled variant will be used within tabs on macOS. This is a great addition and takes away the need for us to think about using the right variant in the right elements.

Changing the symbol rendering mode

UIKIT

The rendering mode of symbols in UIKit can be changed using the earlier used symbol configuration:

```
let image = UIImage(systemName: "thermometer.sun.fill")
let config = UIImage.SymbolConfiguration(paletteColors: [.red, .yellow])

let imageView = UIImageView(image: image)
imageView.preferredSymbolConfiguration = config
imageView.preferredImageVariantShape = UIImage.VariantShape.circle
imageView.preferredImageVariantFill = UIImage.VariantFill.filled
```

SWIFTUI

The rendering mode of symbols can be changed using the following code in SwiftUI:

```
Image(systemName: "thermometer.sun.fill")
    .symbolVariant(.circle.fill)
    .symbolRenderingMode(.palette)
    .foregroundStyle(.red, .yellow)
```

Here, we've changed the rendering mode to palette and updated the foreground style colours accordingly.

In this ar
What are
Which pla

How to use SF Symbols in iOS 12 and below?

Unfortunately, you can't use SF Symbols directly on iOS 12 and below as you can on iOS 13 and up. If you'd like to use any of the symbols on iOS 12 and below you need to export them from the SF Symbols Mac app and import them as regular assets in your Asset Catalog.

Conclusion

With over 3000 symbols to use for free in our apps, we can feel fortunate as Apple engineers. Especially if you don't have a designer or want to move fast, you can use SF symbols to build a beautiful native app. The SF Symbols app is a great companion in finding the right symbol for your task.

If you like to improve your Swift knowledge, even more, check out the Swift category page. Feel free to contact me or tweet to me on Twitter if you have any additional tips or feedback.

Thanks!

WRITTEN BY
ANTOINE VAN DER LEE

GitHub Sponsor Follow

iOS Developer since 2010. Lead developer of the Collect by WeTransfer app. Writing a new blog post every week related to Swift, iOS and Xcode. Regular speaker and workshop host.

Async let explained: call async functions in parallel

Async let is part of Swift's concurrency framework and allows instantiating a constant asynchronously. The concurrency framework introduced the concept ...

Aug 03, 2021 Concurrency Swift

Improve discoverability using Static Member Lookup in Generic Contexts

Static Member Lookup is extended to Generic Contexts since the release of SE-0299. It might seem to be a minor ...

Jun 15, 2021 Swift

AnyObject, Any, and any: When to use which?

AnyObject and Any got a new option any as introduced in SE-355, making it harder for us developers to know ...

Mar 22, 2022 Swift