

[Xcode](#) / [Localization](#) / Preparing your app's text for translation

## Article

# Preparing your app's text for translation

Make your app's text translatable by leveraging the localization APIs in the Foundation framework.

## Overview

Before the system can translate your app's text, you need to prepare your app's strings for translation by creating localizable versions of these strings using the `String(localized:)` and `AttributedString(localized:)` APIs found in the Foundation framework.

Localizable strings are user-facing strings you present to people at runtime. These strings signal to the system that they should be considered for translation, and thus added to your string catalog file. You then export those strings from your string catalog and send them to a localizer for translation or you edit the string catalog yourself and enter the translations directly there.

## Make your string literals localizable

String literals — strings created only with double quotes — aren't localizable by themselves.

```
// An example of a nonlocalizable string literal.  
let name = "Lightbulbs"
```

There's no way for the system to know whether the string literal is a user-facing string in need of translation or simply a print statement there for debugging.

To make a string localizable, create a `String` object using the `init(localized:)` initializer.

```
// Localizable string with the same key and value.  
String(localized: "Lightbulbs")
```

This initializer takes the string literal passed in as a LocalizedStringResource and assigns it to a key equal to the value of the string literal itself. Your string catalog uses this key to look up translations based on the language and locale of the user's device, and then returns the translated value associated with that key. With this initializer, the key and underlying string value are the same. This means you can use the text of your development language as the keys for your translations.

To create localizable strings with different keys and values, use the init(localized:defaultValue:options:table:bundle:locale:comment:) initializer. This initializer assigns the first string literal as the string's key and makes the second parameter the default string value.

```
// Localizable string with a different key and value.  
String(localized: "LIGHTING_KEY", defaultValue: "Lightbulbs")
```

If someone else translates your strings, consider adding helpful comments to your string initializers to provide additional context about how and when the string displays.

```
// Localizable string with a comment providing additional context.  
String(localized: "Lightbulbs", comment: "Label: The icon name displayed on the cont")
```

If the number of translations in your string catalog grows too large, consider breaking the default catalog up into several smaller catalogs. Then specify which catalog a translation comes from using the table parameter in the init(localized:defaultValue:options:table:bundle:locale:comment:) initializer.

```
// Localizable string referenced from the Greetings.xcstrings string catalog file.  
String(localized: "Welcome", table: "Greetings")
```

Use `String(localized:)` and `AttributedString(localized:)` initializers to initialize UIKit and AppKit controls as well as general Swift structures containing variables of type String.

```
struct Accessory {  
    // Nonlocalizable string literal.  
    let name: String  
}  
  
// Made localizable using the String(localized:) initializer.  
Accessory(name: String(localized: "Welcome"))  
  
// Localizable string passed into a UIKit control.  
let label = UILabel()
```

```
label.text = String(localized: "Welcome")

// Localizable string passed into a AppKit UI control.
let textField = NSTextField()
```

## Localize text automatically in SwiftUI

SwiftUI views that accept string literals of type LocalizedStringKey are automatically considered localizable. For example, the following strings are all automatically considered localizable in SwiftUI:

```
// Text made localizable with LocalizedStringKey.

Label("Thanks for shopping with us!", systemImage: "bag")
    .font(.title)
HStack {
    Button("Clear Cart") {}
    Button("Checkout") {}
}
```

To help translators better understand the context for a localized string, use the init(\_:tableName:bundle:comment:) initializer of your Text view and provide a comment with additional details.

```
// Provide additional localizable data with a `TextView`.

Stepper {
    Text("Increase or decrease the item quantity", comment: "Lets the shopper increase or decrease the item quantity")
} onIncrement: {
    // ...
} onDecrement: {
    // ...
}
```

## Pass localizable strings with a localizable type

When defining or passing localizable text in your views, use the recommended type for passing strings in Swift LocalizedStringResource.

```
// Localizable strings in SwiftUI.
```

```
struct CardView: View {  
    let title: LocalizedStringResource  
    let subtitle: LocalizedStringResource  
  
    var body: some View {  
        ZStack {  
            Rectangle()  
            VStack {  
                Text(title)  
                Text(subtitle)  
            }  
            .padding()  
        }  
    }  
}
```

```
CardView(title: "Recent Purchases", subtitle: "Items you've ordered in the past week")
```

This type not only supports initialization using string literals, it also supports adding a comment, table name, or default value that's different from the string key.

LocalizedStringResource also works for strings defined in general Swift code. For example, here's a structure that defines a title of type LocalizedStringResource, which is then instantiated using a string literal and an instance of LocalizedStringResource, both localizable.

```
struct UserAction {  
    let title: LocalizedStringResource  
}
```

```
// Localizable text created with a string literal.
```

```
let action = UserAction(title: "Order items")
```

```
// Localizable text created with a `LocalizedStringResource`.
```

```
let actionWithComment = UserAction(title: LocalizedStringResource("Order items", comment: "Order items"))
```

## Define and load localizable strings from within your framework

If you define the strings that you want to localize in another module, framework, or Swift Package, use the `bundle` argument in their definition. For example, say you want to modularize your project and you create a framework called `BirdFinderUtilities`. To look up a localizable string from within that framework:

1. Identify the name of a class within the framework (in this case, `BirdSongs`).
2. Pass that class into an instance of `Bundle` using the `init(for:)` initializer.
3. Use that `Bundle` to look up the localizable string.

```
// Localizable string within a framework.
String(localized: "Songs", bundle: Bundle(for: (BirdSongs.self)))

// Localizable string within a framework in SwiftUI.
Text("Songs", bundle: Bundle(for: (BirdSongs.self)))

// Localizable string within a framework for a `LocalizedStringResource`.
LocalizedStringResource("Songs", bundle: .forClass(BirdSongs.self), comment: "Headli
```

### Important

Avoid looking up strings from bundles you don't own. Doing so may prevent automatic string extraction from properly working in the string catalog.

## See Also

### Strings and text



Preparing your interface for localization

Find text in your app that needs translation and verify that your interface adapts to translated text.



Preparing dates, currencies, and numbers for translation

Ensure that dates, currencies, and numbers display correctly across multiple languages and locales by using formatters.