Article

Preparing dates, currencies, and numbers for translation

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

Overview

Languages and regions have different formats for presenting dates and numbers. Some languages use the period (.) as a decimal separator, and others use the comma (,). Some place the percent sign before the number when formatting percentages. And many regions display time and date differently, despite being part of the same language.

Instead of trying to account for all these variations yourself, use the formatters built into Foundation to create localizable versions of the dates and numbers you want to present.

Format dates with predefined styles

To convert a date or number into a localizable string, use the Foundation formatters and styles. These APIs take instances of your date and number objects, and convert them into localizable formatted strings according to the locale of the device your app is running on.

For example, to create a localizable string from a date object, create an instance of the `Date` you want to format and then call the `formatted()` function on the date.

```swift
// The current time and date. Example output is for en_US locale.en_US locale.
let date = Date.now

// A default, formatted, localizable date string.
let defaultFormatted = date.formatted()
// "8/25/2023, 12:03 PM"
```

To vary the date components that display, or display only the time or the date, use the `formatted(date:time:)` method on the `Date` object passing in instances of `Date.FormatStyle.DateStyle` and `Date.FormatStyle.TimeStyle`.

```swift
// The date you want to format.
let meetingDate = Date.now

// A formatted date displaying only the date.
let formattedDate = meetingDate.formatted(date: .abbreviated, time: .omitted)
// "Aug 25, 2023"

// A formatted date displaying only the time.
let formattedTime = meetingDate.formatted(date: .omitted, time: .standard)
// "12:03:10 PM"

// A formatted date displaying both the date and time.
let formattedDateAndTime = meetingDate.formatted(date: .complete, time: .complete)
// "Friday, August 25, 2023 at 12:03:10 PM PDT"
```

Create your own custom date styles

To format a date to a specific style, create your own custom date style including only the date properties you want to display.

For example, to create a date that includes only the month, day, and year:

1. Create an instance of the `Date` object you want to format.
2. Create a `Date.FormatStyle` structure or use the `dateTime` factory variable, and chain together the properties you want to display in successive function calls.
3. Then pass that `Date.FormatStyle` structure as an input into the the `formatted(_:)` function on the date object.

```
// A date string with specific attributes.
let myDate = Date.now
let formatStyle = Date.FormatStyle.dateTime.year().day().month()
let formatted = date.formatted(formatStyle)
// "Sep 7, 2023"
```

You can also achieve the same result in one line.

```
// Same result in one line using the `dateTime` factory variable.
let formatted = Date.now.formatted(.dateTime.year().day().month())
// "Sep 7, 2023"
```

The order of the fields you pass into the `formatted(_:)` function doesn't matter. For example, these lines of code produce the same result.

```
// Same result.
Date.now.formatted(.dateTime.year().month().day().hour().minute().second())
Date.now.formatted(.dateTime.second().minute().hour().day().month().year())
// "Sep 7, 2023 at 10:29:52 AM"
```

Customize the date styles you want to display by chaining together instances of `Date.FormatStyle.Symbol` structures along with their respective formatting properties.

```
// A date string for a wide month format.
let formattedWide = date.formatted(.dateTime.year().day().month(.wide))
// "September 7, 2023"

// A date string for a wide weekday.
let formattedWeekday = date.formatted(.dateTime.weekday(.wide))
// "Thursday"

// A date string for the ISO 8601 time and date standard.
let logFormat = date.formatted(.iso8601)
// "2023-09-07T17:25:39Z"

// A date string representing a file format.
let fileNameFormat = date.formatted(.iso8601.year().month().day().dateSeparator(.das
// "2023-09-07"
```

Format percents and scientific numbers

If you want to create a localizable string for a number (such as `Int`, `Double`, `Decimal`, or `Float`), call `formatted()` or `formatted(_:)` on the number instance, along with the format style to display.

For example, to create a formatted version of an `Int`, call the `formatted()` function on the number.

```swift
let value = 12345
// A default, formatted, localizable date string.
var formatted = value.formatted()
// "12,345"
```

To format the number as a percent, call `formatted(_ format:)` on the number you want to display with the `NumberFormatter.Style.percent` number format style. Integers convert directly into percentages using the whole number.

```swift
let number = 25
let numberFormatted = number.formatted(.percent)
// "25%"
```

Fractions convert between the range of 0 and 1.

```swift
let fraction = 0.25
let fractionFormatted = fraction.formatted(.percent)
// "25%"
```

To display a number using scientific notation, call `formatted(_:)` on the number to display using the `scientific`, `notation(_:)`, and `number` format styles.

```swift
let scientific = 42e9
let scientificFormatted = scientific.formatted(.number.notation(.scientific))
// "4.2E10"
```

Format currencies

To present a number as a localizable currency:

1. Look up the code of the currency you want to display (such as `"CAD"` for Canada).

2. Pass that code as a parameter to the `Decimal.FormatStyle.Currency` format style initializer `init(code:locale:)`.

3. Then call `formatted(_:)` on the number passing in the currency format instance.

```
// A number formatted in different currencies.
let amount: Decimal = 12345.67
amount.formatted(.currency(code: "JPY"))
// "¥12,346"
amount.formatted(.currency(code: "EUR").presentation(.fullName))
// "12,345.67 euros"
amount.formatted(.currency(code: "USD").grouping(.automatic))
// "$12,345.67"
```

Note

To ensure accuracy, don't use `Float` or `Double` to represent currency in your app. Use `Decimal` instead.

Format times as intervals or durations

To display an interval of time as a localizable string:

1. Create two instances of the `Date` object — one representing the start of the time interval and the other the end.

2. Using these two dates, create a `Range` structure setting the upper and lower bounds of the interval.

3. Then call one of the range formatters — such as `formatted()` or `formatted(date:time:)` — passing in the time and date styles you want to display.

```
// An example of a time interval.

// The current time and date. Example output is for en_US locale.
let now = Date.now

// 5000 seconds from now.
let later = now + TimeInterval(5000)

// The default formatted display for a time interval.
let range = (now..<later).formatted()
```

```
// "9/8/2023, 10:44 AM–12:07 PM"

// A time interval formatted using a predefined date format.
let noDate = (now..<later).formatted(date: .omitted, time: .complete)
// "10:44:39 AM PDT–12:07:59 PM PDT"
```

To display time as a duration, you can similarly define a date range and convert that range into a duration.

```
// An example of a duration.

// Duration from a range of dates.
let timeDuration = (now..<later).formatted(.timeDuration)
// "1:23:20"

let components = (now..<later).formatted(.components(style: .wide))
// "1 hour, 23 minutes, 20 seconds"

let relative = later.formatted(.relative(presentation: .named, unitsStyle: .wide))
// "in 1 hour"
```

You can also use factory methods like `seconds(_:)` on the `Duration` structure to produce localizable durations from a single number.

For example, to display a given number of seconds as a duration of time:

1. Pass in the number of seconds you want to display to the `seconds(_:)` function of the `Duration` structure.

2. Then call `formatted(_:)`, passing in instances of `Duration.TimeFormatStyle` or `Duration.UnitsFormatStyle` to achieve the format and style you want.

```
// Duration formatted from a single unit of time.
Duration.seconds(2000).formatted(.time(pattern: .hourMinute))
// "0:33"
Duration.seconds(2000).formatted(.time(pattern: .hourMinuteSecond))
// "0:33:20"
Duration.seconds(2000).formatted(.time(pattern: .minuteSecond))
// "33:20"
```

Format items as lists

To create a localizable string in the form of a list, use the ListFormatStyle structure along with either the formatted() or formatted(_:) function to create a string representation of a Sequence of items.

```
// An array of strings formatted into a list.
let sizes = ["small", "medium", "large"]
sizes.formatted(.list(type: .and, width: .narrow))
// "small, medium, large"
sizes.formatted(.list(type: .and, width: .standard))
// "small, medium, and large"
sizes.formatted(.list(type: .and, width: .short))
// "small, medium, & large"
```

You can also create lists using different formatting styles by calling the list(memberStyle:type:width:) function along with specific list format styles.

```
// A list of numbers formatted as percentages.
[25, 50, 75].formatted(.list(memberStyle: .percent, type: .or))
// "25%, 50%, or 75%"
```

Convert and display measurement units across different locales

Units of measure vary significantly depending on the locale the format style uses. For example, a distance in feet for the en_US locale appears as meters using the French locale fr_FR.

To ensure your units of measure convert and display properly across different languages and regions:

1. Use the Measurement structure to define a variable representing the unit of measure you want to display.

2. Then call formatted(_:) or formatted(_:) on the variable to get the display style you want.

For example, say you want to convert and display the following measurements.

```
// Measurements to display.
let speedLimit = Measurement(value: 110, unit: UnitSpeed.kilometersPerHour)
let distanceToMoon = Measurement(value: 384400, unit: UnitLength.kilometers)
let surfBoardLength = Measurement(value: 8, unit: UnitLength.feet)
let waterTemperature = Measurement(value: 61.2, unit: UnitTemperature.fahrenheit)
```

To convert them using the default format, call `formatted(_:)` on the measurement object.

```
// Example output is for en_US locale.

// Default display for a unit of measure.
speedLimit.formatted()
// "68 mph"
distanceToMoon.formatted()
// "238,855 mi"
surfBoardLength.formatted()
// "8 ft"
waterTemperature.formatted()
// "61.2°F"
```

To customize the output, call the `formatted(_:)` function on the measurement using the `measurement(width:usage:hidesScaleName:numberFormatStyle:)` factory method to create the format and style you want.

```
// Custom display options for a unit of measure.
distanceToMoon.formatted(.measurement(width: .wide))
// "238,855 miles"
distanceToMoon.formatted(.measurement(width: .abbreviated))
// "238,855 mi"
distanceToMoon.formatted(.measurement(width: .narrow))
// "238,855mi"
```

Format dates and numbers in SwiftUI

To format dates and numbers in SwiftUI, use the `format` initializers on SwiftUI view controls to customize how those strings display.

For example, here is a SwiftUI view that displays three different localizable formats of `Date` using the `init(_:format:)` initializer from the `Text` view.

```
@State private var myDate = Date.now

var body: some View {
    VStack {
        Text(myDate, format: Date.FormatStyle(date: .numeric, time: .omitted))
        Text(myDate, format: Date.FormatStyle(date: .complete, time: .complete))
        Text(myDate, format: Date.FormatStyle().hour(.defaultDigits(amPM: .omitted))
    }
}
```

This example uses the `init(_:value:format:prompt:)` initializer of the `TextField` view to present a number as a percentage for a tip.

```
@State private var tip = 0.15

var body: some View {
    HStack {
        Text("Tip")
        Spacer()
        TextField("Amount", value: $tip, format: .percent)
    }
}
```

Test the formatters

To test and see how your formatters display in different languages and regions, create an instance of a `Locale` object, passing in the `identifier` of the region you want to test. Then set that locale on the output of your formatted string to see how that string displays in that language and region.

For example, you can see how your localizable strings display in French as follows.

```
// The locale for France French.
let frenchLocale = Locale(identifier: "fr_FR")

let stages = ["50", "75", "100"]
stages.formatted(.list(type: .and).locale(frenchLocale))
// "50, 75 et 100"
stages.formatted(.list(type: .or).locale(frenchLocale))
// "50, 75 ou 100"
```

To test your formatters in SwiftUI, set the locale in the environment variable in the `#Preview` section of your code.

```
struct ContentView: View {
    @State private var myDate = Date.now
    @Environment(\.locale) var locale
    var body: some View {
        VStack {
            Text(myDate, format: .dateTime.second().minute().hour().day().month().ye
        }
    }
}

#Preview {
    Group {
        ContentView()
            .environment(\.locale, Locale(identifier: "fr_FR"))
        ContentView()
            .environment(\.locale, Locale(identifier: "pt_BR"))
    }
}
```

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 your app’s text for translation

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