Class

DateFormatter

A formatter that converts between dates and their textual representations.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.0+ | macOS 10.0+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class DateFormatter
```

Overview

Instances of `DateFormatter` create string representations of `NSDate` objects, and convert textual representations of dates and times into `NSDate` objects. For user-visible representations of dates and times, `DateFormatter` provides a variety of localized presets and configuration options. For fixed format representations of dates and times, you can specify a custom format string.

When working with date representations in ISO 8601 format, use `ISO8601DateFormatter` instead.

To represent an interval between two `NSDate` objects, use `DateIntervalFormatter` instead.

To represent a quantity of time specified by an `NSDateComponents` object, use `DateComponentsFormatter` instead.

Tip

In Swift, you can use `Date.FormatStyle` or `Date.VerbatimFormatStyle` rather than `DateFormatter`. The `FormatStyle` API offers a declarative idiom for customizing the formatting of various types. Also, Foundation caches identical `FormatStyle` instances, so you don't need to pass them around your app, or risk wasting memory with duplicate formatters.

Working With User-Visible Representations of Dates and Times

When displaying a date to a user, you set the `dateStyle` and `timeStyle` properties of the date formatter according to your particular needs. For example, if you want to show the month, day, and year without showing the time, you would set the `dateStyle` property to `DateFormatter.Style.long` and the `timeStyle` property to `DateFormatter.Style.none`. Conversely, if you want to show only the time, you would set the `dateStyle` property to `DateFormatter.Style.none` and the `timeStyle` property to `DateFormatter.Style.short`. Based on the values of the `dateStyle` and `timeStyle` properties, `DateFormatter` provides a representation of a specified date that is appropriate for a given locale.

Swift Objective-C

```swift
let dateFormatter = DateFormatter()
dateFormatter.dateStyle = .medium
dateFormatter.timeStyle = .none

let date = Date(timeIntervalSinceReferenceDate: 118800)

// US English Locale (en_US)
dateFormatter.locale = Locale(identifier: "en_US")
print(dateFormatter.string(from: date)) // Jan 2, 2001

// French Locale (fr_FR)
dateFormatter.locale = Locale(identifier: "fr_FR")
print(dateFormatter.string(from: date)) // 2 janv. 2001

// Japanese Locale (ja_JP)
dateFormatter.locale = Locale(identifier: "ja_JP")
print(dateFormatter.string(from: date)) // 2001/01/02
```

If you need to define a format that cannot be achieved using the predefined styles, you can use the `setLocalizedDateFormatFromTemplate(_:)` to specify a localized date format from a template.

Swift Objective-C

```swift
let dateFormatter = DateFormatter()
let date = Date(timeIntervalSinceReferenceDate: 410220000)
```

```
// US English Locale (en_US)
dateFormatter.locale = Locale(identifier: "en_US")
dateFormatter.setLocalizedDateFormatFromTemplate("MMMMd") // set template after sett
print(dateFormatter.string(from: date)) // December 31

// British English Locale (en_GB)
dateFormatter.locale = Locale(identifier: "en_GB")
dateFormatter.setLocalizedDateFormatFromTemplate("MMMMd") // // set template after s
print(dateFormatter.string(from: date)) // 31 December
```

Working With Fixed Format Date Representations

> **Important**
>
> In macOS 10.12 and later or iOS 10 and later, use the `ISO8601DateFormatter` class when working with ISO 8601 date representations.

When working with fixed format dates, such as RFC 3339, you set the `dateFormat` property to specify a format string. For most fixed formats, you should also set the `locale` property to a POSIX locale (`"en_US_POSIX"`), and set the `timeZone` property to UTC.

Swift | Objective-C

```
let RFC3339DateFormatter = DateFormatter()
RFC3339DateFormatter.locale = Locale(identifier: "en_US_POSIX")
RFC3339DateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ssZZZZZ"
RFC3339DateFormatter.timeZone = TimeZone(secondsFromGMT: 0)

/* 39 minutes and 57 seconds after the 16th hour of December 19th, 1996 with an offs
let string = "1996-12-19T16:39:57-08:00"
let date = RFC3339DateFormatter.date(from: string)
```

For more information, see Technical Q&A QA1480 “NSDateFormatter and Internet Dates”.

Thread Safety

On iOS 7 and later `NSDateFormatter` is thread safe.

In macOS 10.9 and later `NSDateFormatter` is thread safe so long as you are using the modern behavior in a 64-bit app.

On earlier versions of the operating system, or when using the legacy formatter behavior or running in 32-bit in macOS, `NSDateFormatter` is not thread safe, and you therefore must not mutate a date formatter simultaneously from multiple threads.

Topics

Converting Objects

```
func date(from: String) -> Date?
```

Returns a date representation of a specified string that the system interprets using the receiver's current settings.

```
func string(from: Date) -> String
```

Returns a string representation of a specified date that the system formats using the receiver's current settings.

```
class func localizedString(from: Date, dateStyle: DateFormatter.Style, timeStyle: DateFormatter.Style) -> String
```

Returns a string representation of a specified date, that the system formats for the current locale using the specified date and time styles.

```
func getObjectValue(AutoreleasingUnsafeMutablePointer<AnyObject?>?, for: String, range: UnsafeMutablePointer<NSRange>?) throws
```

Returns by reference a date representation of a specified string and its date range, as well as a Boolean value that indicates whether the system can parse the string.

Managing Formats and Styles

```
var dateStyle: DateFormatter.Style
```

The date style of the receiver.

```
var timeStyle: DateFormatter.Style
```

The time style of the receiver.

```
var dateFormat: String!
```

The date format string used by the receiver.

```
func setLocalizedDateFormatFromTemplate(String)
```

Sets the date format from a template using the specified locale for the receiver.

```
class func dateFormat(fromTemplate: String, options: Int, locale: Locale?) -> String?
```

Returns a localized date format string representing the given date format components arranged appropriately for the specified locale.

```
var formattingContext: Formatter.Context
```

The capitalization formatting context used when formatting a date.

Managing Attributes

```
var calendar: Calendar!
```

The calendar for the receiver.

```
var defaultDate: Date?
```

The default date for the receiver.

```
var locale: Locale!
```

The locale for the receiver.

```
var timeZone: TimeZone!
```

The time zone for the receiver.

```
var twoDigitStartDate: Date?
```

The earliest date that can be denoted by a two-digit year specifier.

```
var gregorianStartDate: Date?
```

The start date of the Gregorian calendar for the receiver.

Managing Behavior Version

```
var formatterBehavior: DateFormatter.Behavior
```

The formatter behavior for the receiver.

```
class var defaultFormatterBehavior: DateFormatter.Behavior
```

Returns the default formatting behavior for instances of the class.

Managing Natural Language Support

```
var isLenient: Bool
```

A Boolean value that indicates whether the receiver uses heuristics when parsing a string.

`var doesRelativeDateFormatting: Bool`

A Boolean value that indicates whether the receiver uses phrases such as “today” and “tomorrow” for the date component.

Managing AM and PM Symbols

`var amSymbol: String!`

The AM symbol for the receiver.

`var pmSymbol: String!`

The PM symbol for the receiver.

Managing Weekday Symbols

`var weekdaySymbols: [String]!`

The array of weekday symbols for the receiver.

`var shortWeekdaySymbols: [String]!`

The array of short weekday symbols for the receiver.

`var veryShortWeekdaySymbols: [String]!`

The array of very short weekday symbols for the receiver.

`var standaloneWeekdaySymbols: [String]!`

The array of standalone weekday symbols for the receiver.

`var shortStandaloneWeekdaySymbols: [String]!`

The array of short standalone weekday symbols for the receiver.

`var veryShortStandaloneWeekdaySymbols: [String]!`

The array of very short standalone weekday symbols for the receiver.

Managing Month Symbols

`var monthSymbols: [String]!`

The month symbols for the receiver.

`var shortMonthSymbols: [String]!`

The array of short month symbols for the receiver.

`var veryShortMonthSymbols: [String]!`

The very short month symbols for the receiver.

`var standaloneMonthSymbols: [String]!`

The standalone month symbols for the receiver.

`var shortStandaloneMonthSymbols: [String]!`

The short standalone month symbols for the receiver.

`var veryShortStandaloneMonthSymbols: [String]!`

The very short month symbols for the receiver.

Managing Quarter Symbols

`var quarterSymbols: [String]!`

The quarter symbols for the receiver.

`var shortQuarterSymbols: [String]!`

The short quarter symbols for the receiver.

`var standaloneQuarterSymbols: [String]!`

The standalone quarter symbols for the receiver.

`var shortStandaloneQuarterSymbols: [String]!`

The short standalone quarter symbols for the receiver.

Managing Era Symbols

`var eraSymbols: [String]!`

The era symbols for the receiver.

`var longEraSymbols: [String]!`

The long era symbols for the receiver

Deprecated

`var generatesCalendarDates: Bool`

Indicates whether the formatter generates the deprecated calendar date type.

Constants

`enum` `Style`

The following constants specify predefined format styles for dates and times.

`enum` `Behavior`

Constants that specify the behavior `NSDateFormatter` should exhibit.

Relationships

Inherits From

`Formatter`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSCoding`
`NSCopying`
`NSObjectProtocol`
`Sendable`
`SendableMetatype`

See Also

Dates and times

`class` `DateComponentsFormatter`

A formatter that creates string representations of quantities of time.

`class` `RelativeDateTimeFormatter`

A formatter that creates locale-aware string representations of a relative date or time.

`class` `DateIntervalFormatter`

A formatter that creates string representations of time intervals.

`class` `ISO8601DateFormatter`

A formatter that converts between dates and their ISO 8601 string representations.