Structure

Calendar

A definition of the relationships between calendar units and absolute points in time, providing features for calculation and comparison of dates.

iOS 7.0+ iPadOS 7.0+ macOS 10.9+ Mac Catalyst 13.0+ tvOS 9.0+ watchOS 2.0+ Xcode 8.0+

Declaration

```
struct Calendar
```

Overview

`Calendar` encapsulates information about systems of reckoning time in which the beginning, length, and divisions of a year are defined. It provides information about the calendar and support for calendrical computations such as determining the range of a given calendrical unit and adding units to a given absolute time.

Topics

Creating a Calendar

`enum Calendar.Identifier`

An enumeration for the available calendars.

Getting the User's Calendar

`static var autoupdatingCurrent: Calendar`

A calendar that tracks changes to user's preferred calendar.

`static var current: Calendar`

The user's current calendar.

Extracting Components

`func date(Date, matchesComponents: DateComponents) -> Bool`

Determines if the date has all of the specified date components.

`func component(Calendar.Component, from: Date) -> Int`

Returns the value for one component of a date.

`func dateComponents(Set<Calendar.Component>, from: Date) -> DateComponents`

Returns all the date components of a date, using the calendar time zone.

`func dateComponents(Set<Calendar.Component>, from: Date, to: Date) -> DateComponents`

Returns the difference between two dates.

`func dateComponents(Set<Calendar.Component>, from: DateComponents, to: DateComponents) -> DateComponents`

Returns the difference between two dates specified as `DateComponents`.

`func dateComponents(in: TimeZone, from: Date) -> DateComponents`

Returns all the date components of a date, as if in a given time zone (instead of the `Calendar` time zone).

`enum Calendar.Component`

An enumeration for the various components of a calendar date.

Getting Calendar Information

`var identifier: Calendar.Identifier`

The identifier of the calendar.

`var locale: Locale?`

The locale of the calendar.

`var firstWeekday: Int`

The first day of the week for the calendar.

`var minimumDaysInFirstWeek: Int`

The number of minimum days in the first week.

`var timeZone: TimeZone`

The time zone of the calendar.

`func maximumRange(of: Calendar.Component) -> Range<Int>?`

The maximum range limits of the values that a given component can take on.

`func minimumRange(of: Calendar.Component) -> Range<Int>?`

Returns the minimum range limits of the values that a given component can take on.

`func ordinality(of: Calendar.Component, in: Calendar.Component, for: Date) -> Int?`

Returns, for a given absolute time, the ordinal number of a smaller calendar component (such as a day) within a specified larger calendar component (such as a week).

`func range(of: Calendar.Component, in: Calendar.Component, for: Date) -> Range<Int>?`

Returns the range of absolute time values that a smaller calendar component (such as a day) can take on in a larger calendar component (such as a month) that includes a specified absolute time.

Scanning Dates

`func startOfDay(for: Date) -> Date`

Returns the first moment of a given Date, as a Date.

`func enumerateDates(startingAfter: Date, matching: DateComponents, matchingPolicy: Calendar.MatchingPolicy, repeatedTimePolicy: Calendar.RepeatedTimePolicy, direction: Calendar.SearchDirection, using: (Date?, Bool, inout Bool) -> Void)`

Computes the dates which match (or most closely match) a given set of components, and calls the closure once for each of them, until the enumeration is stopped.

`func nextDate(after: Date, matching: DateComponents, matchingPolicy: Calendar.MatchingPolicy, repeatedTimePolicy: Calendar.RepeatedTimePolicy, direction: Calendar.SearchDirection) -> Date?`

Computes the next date which matches (or most closely matches) a given set of components.

`enum Calendar.MatchingPolicy`

A hint to the search algorithm to control the method used for searching for dates.

`enum Calendar.RepeatedTimePolicy`

Determines which result to use when a time is repeated on a day in a calendar (for example, during a daylight saving transition when the times between 2:00am and 3:00am may happen twice).

Calculating Dates from Components

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

Returns a date created from the specified components.

`func date(byAdding: DateComponents, to: Date, wrappingComponents: Bool) -> Date?`

Returns a new `Date` representing the date calculated by adding components to a given date.

`func date(byAdding: Calendar.Component, value: Int, to: Date, wrappingComponents: Bool) -> Date?`

Returns a new `Date` representing the date calculated by adding an amount of a specific component to a given date.

`func date(bySetting: Calendar.Component, value: Int, of: Date) -> Date?`

Returns a new `Date` representing the date calculated by setting a specific component to a given time, and trying to keep lower components the same. If the component already has that value, this may result in a date which is the same as the given date.

```
func date(bySettingHour: Int, minute: Int, second: Int, of: Date, matchingPolicy: Calendar.MatchingPolicy, repeatedTimePolicy: Calendar.RepeatedTimePolicy, direction: Calendar.SearchDirection) -> Date?
```

Returns a new `Date` representing the date calculated by setting hour, minute, and second to a given time on a specified `Date`.

Calculating Intervals

```
func dateInterval(of: Calendar.Component, for: Date) -> DateInterval?
```

Returns the starting time and duration of a given calendar component that contains a given date.

```
func dateInterval(of: Calendar.Component, start: inout Date, interval: inout TimeInterval, for: Date) -> Bool
```

Returns, via two inout parameters, the starting time and duration of a given calendar component that contains a given date.

```
func dateIntervalOfWeekend(containing: Date) -> DateInterval?
```

Returns a `DateInterval` of the weekend contained by the given date, or `nil` if the date is not in a weekend.

```
func dateIntervalOfWeekend(containing: Date, start: inout Date, interval: inout TimeInterval) -> Bool
```

Find the range of the weekend around the given date, returned via two by-reference parameters.

```
func nextWeekend(startingAfter: Date, direction: Calendar.SearchDirection) -> DateInterval?
```

Returns a `DateInterval` of the next weekend, which starts strictly after the given date.

```
func nextWeekend(startingAfter: Date, start: inout Date, interval: inout TimeInterval, direction: Calendar.SearchDirection) -> Bool
```

Returns the range of the next weekend via two inout parameters. The weekend starts strictly after the given date.

`enum Calendar.SearchDirection`

The direction in time to search.

Comparing Dates

`func compare(Date, to: Date, toGranularity: Calendar.Component) -> ComparisonResult`

Compares two dates down to the specified component.

`func isDate(Date, equalTo: Date, toGranularity: Calendar.Component) -> Bool`

Returns a Boolean value indicating whether two dates are equal down to the specified component.

`func isDate(Date, inSameDayAs: Date) -> Bool`

Returns a Boolean value indicating whether a date is within the same day as another date.

`func isDateInToday(Date) -> Bool`

Returns a Boolean value indicating whether the given date is within today.

`func isDateInTomorrow(Date) -> Bool`

Returns a Boolean value indicating whether the given date is within tomorrow.

`func isDateInYesterday(Date) -> Bool`

Returns a Boolean value indicating whether the given date is within yesterday.

`func isDateInWeekend(Date) -> Bool`

Returns a Boolean value indicating whether the given date is within a weekend period.

Comparing Calendars

`static func == (Calendar, Calendar) -> Bool`

Returns a Boolean indicating whether two calendars are the same.

`static func != (Calendar, Calendar) -> Bool`

Returns a Boolean value indicating whether two values are not equal.

Getting AM and PM symbols

`var amSymbol: String`

The symbol used to represent “AM”, localized to the Calendar’s `locale`.

`var pmSymbol: String`

The symbol used to represent “PM”, localized to the Calendar’s `locale`.

Getting Weekday Symbols

`var weekdaySymbols: [String]`

A list of weekdays in this calendar, localized to the Calendar’s `locale`.

`var shortWeekdaySymbols: [String]`

A list of shorter-named weekdays in this calendar, localized to the Calendar’s `locale`.

`var veryShortWeekdaySymbols: [String]`

A list of very-shortly-named weekdays in this calendar, localized to the Calendar’s `locale`.

`var standaloneWeekdaySymbols: [String]`

A list of standalone weekday names in this calendar, localized to the Calendar’s `locale`.

`var shortStandaloneWeekdaySymbols: [String]`

A list of shorter-named standalone weekdays in this calendar, localized to the Calendar’s `locale`.

`var veryShortStandaloneWeekdaySymbols: [String]`

A list of very-shortly-named weekdays in this calendar, localized to the Calendar’s `locale`.

Getting Month Symbols

`var monthSymbols: [String]`

A list of months in this calendar, localized to the Calendar’s `locale`.

`var shortMonthSymbols: [String]`

A list of shorter-named months in this calendar, localized to the Calendar’s `locale`.

`var veryShortMonthSymbols: [String]`

A list of very-shortly-named months in this calendar, localized to the Calendar's `locale`.

`var standaloneMonthSymbols: [String]`

A list of standalone months in this calendar, localized to the Calendar's `locale`.

`var shortStandaloneMonthSymbols: [String]`

A list of shorter-named standalone months in this calendar, localized to the Calendar's `locale`.

`var veryShortStandaloneMonthSymbols: [String]`

A list of very-shortly-named standalone months in this calendar, localized to the Calendar's `locale`.

Getting Quarter Symbols

`var quarterSymbols: [String]`

A list of quarter names in this calendar, localized to the Calendar's `locale`.

`var shortQuarterSymbols: [String]`

A list of shorter-named quarters in this calendar, localized to the Calendar's `locale`.

`var standaloneQuarterSymbols: [String]`

A list of standalone quarter names in this calendar, localized to the Calendar's `locale`.

`var shortStandaloneQuarterSymbols: [String]`

A list of shorter-named standalone quarters in this calendar, localized to the Calendar's `locale`.

Getting Era Symbols

`var eraSymbols: [String]`

A list of eras in this calendar, localized to the Calendar's `locale`.

`var longEraSymbols: [String]`

A list of longer-named eras in this calendar, localized to the Calendar's `locale`.

Describing a Calendar

`var description: String`

A textual description of the calendar.

`var debugDescription: String`

A textual description of the locale suitable for debugging.

`var customMirror: Mirror`

A mirror that reflects the calendar.

`var hashValue: Int`

The computed hash value for the calendar.

Encoding and Decoding

`func encode(to: Encoder)`

Encodes this calendar into the given encoder.

`init(from: Decoder)`

Creates a new calendar instance by decoding from the given decoder.

Using Reference Types

`class NSCalendar`

An encapsulation of calendar information and calculations that bridges to `Calendar`; use `NSCalendar` when you need reference semantics or other Foundation-specific behavior.

`typealias Calendar.ReferenceType`

An alias for this value type's equivalent reference type.

Initializers

`init(identifier: Calendar.Identifier)`

Instance Methods

`func hash(into: inout Hasher)`

Relationships

Conforms To

`CustomDebugStringConvertible`
`CustomReflectable`
`CustomStringConvertible`
`Decodable`
`Encodable`
`Hashable`
`ReferenceConvertible`
`Sendable`

See Also

Calendrical Calculations

`struct DateComponents`

A date or time specified in terms of units (such as year, month, day, hour, and minute) to be evaluated in a calendar system and time zone.

`struct TimeZone`

Information about standard time conventions associated with a specific geopolitical region.