Structure

Locale

Information about linguistic, cultural, and technological conventions for use in formatting data for presentation.

iOS 8.0+ | iPadOS 8.0+ | Mac Catalyst 8.0+ | macOS 10.10+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
struct Locale
```

Overview

`Locale` encapsulates information about linguistic, cultural, and technological conventions and standards. Examples of information encapsulated by a locale include the symbol used for the decimal separator in numbers and the formatting conventions for dates and times.

Apps use locales to provide, format, and interpret information about and according to the user's customs and preferences. Data formatting APIs commonly make use of locales to present data in a locale-appropriate way.

You can create a `Locale` from a common identifier like `en–US`, or by specifying its components. More commonly, you access the current system locale with the `current` or `autoupdatingCurrent` static variables.

Working with locale components

A `Locale` exposes its various traits — the appropriate measurement system, currency symbols, date and time conventions, and more — as strongly-typed properties like `currency`, `numberingSystem`, and `firstDayOfWeek`.

In addition, the `language` property allows you examine traits of languages, through the `Locale.Language` type, in contast with `NSLocale`, where `languageCode` is just a string identifier. You

can use a locale's language to compare whether two locales use the same language, or if one language is a parent of another.

The following example creates a `Locale` from the identifier `zh-CN`, for Chinese. It then accesses this locale's `language` to get the language's `script`, and uses a US English locale to get a localized string describing the script: "Simplified Han". With the locale `zh-Hant-CN`, for Traditional Chinese, the script would be "Traditional Han" instead.

```
let zhCN = Locale(identifier: "zh-CN")
if let script = zhCN.language.script {
    let enUS = Locale(identifier: "en-US")
    let localizedScript = enUS.localizedString(forScript: script) // "Simplified Har
}
```

Creating custom locales from components

You can create a custom locale by creating a `Locale` instance from a customized `Locale.Components`. Do this when you want to tweak specific aspects of a locale. The following example creates a locale that uses language conventions of British English (language region GB), but otherwise uses US conventions for things like currency and measurement.

```
var components = Locale.Components(languageCode: "en", languageRegion: "GB")
components.region = Locale.Region("US")
let en_GB_US = Locale(components: components)
```

Creating a custom locale like this isn't necessarily common in apps, but can be useful in unit testing your app's localizations.

Topics

Creating a locale by identifier

`init(identifier: String)`

Creates a locale with the specified identifier.

Creating a locale by components

`init(components: Locale.Components)`

Creates a locale from the given components.

`struct Components`

A type that represents the components of a locale, for use when creating a locale with specific overrides.

`init(languageCode: Locale.LanguageCode?, script: Locale.Script?, languageRegion: Locale.Region?)`

Creates a locale with the specified language code, script, and region identifier.

`init(languageComponents: Locale.Language.Components)`

Creates a locale from the given language components.

`struct Components`

A type that identifies a language by its various components.

Getting the user's locale

`static var autoupdatingCurrent: Locale`

A locale which tracks the user's current preferences.

`static var current: Locale`

A locale representing the user's region settings at the time the property is read.

Getting known identifiers and codes

`static var availableIdentifiers: [String]`

A list of available identifiers.

~~`static var isoRegionCodes: [String]`~~

A list of available region codes.

Deprecated

~~`static var isoLanguageCodes: [String]`~~

A list of available language codes.

Deprecated

~~`static var isoCurrencyCodes: [String]`~~

A list of available currency codes.

Deprecated

`static var commonISOCurrencyCodes: [String]`

A list of common currency codes.

Converting between identifiers

`static func canonicalIdentifier(from: String) -> String`

Returns a canonical identifier from the given string.

~~`static func components(fromIdentifier: String) -> [String : String]`~~

Returns a dictionary that splits an identifier into its component pieces.

Deprecated

`static func identifier(fromComponents: [String : String]) -> String`

Constructs an identifier from a dictionary of components.

`static func identifier(Locale.IdentifierType, from: String) -> String`

Returns the identifier conforming to the specified standard for the specified string.

`enum IdentifierType`

A type that indicates the standard that defines a locale's identifier.

`static func canonicalLanguageIdentifier(from: String) -> String`

Returns a canonical language identifier from the given string.

`static func identifier(fromWindowsLocaleCode: Int) -> String?`

Returns the locale identifier from a given Windows locale code, or `nil` if it could not be converted.

`static func windowsLocaleCode(fromIdentifier: String) -> Int?`

Returns the Windows locale code from a given identifier, or `nil` if it could not be converted.

Getting locale components

`struct Components`

A type that represents the components of a locale, for use when creating a locale with specific overrides.

Getting language components

`var language: Locale.Language`

The language of a locale.

`struct Language`

A type that represents a language, as used in a locale.

Getting date and time components

`var firstDayOfWeek: Locale.Weekday`

The first day of the week as represented by this locale.

`enum Weekday`

A type that represents weekdays, used for indicating a locale's first day of the week.

`var hourCycle: Locale.HourCycle`

The hour cycle used by the locale, like one-to-twelve or zero-to-twenty-three.

`enum HourCycle`

A type that represents the hour cycle used in a locale, like one-to-twelve or zero-to-twenty-three.

`var timeZone: TimeZone?`

The time zone associated with the locale, if any.

Getting measurement and counting components

`var currency: Locale.Currency?`

The currency used by the locale.

`struct Currency`

A type that represents the currency system used by a locale, like dollars or euros.

`var measurementSystem: Locale.MeasurementSystem`

The measurement system used by the locale, like metric or the US system.

`struct MeasurementSystem`

A type that represents the measurement system used by a locale, like metric or the US system.

`var numberingSystem: Locale.NumberingSystem`

The numbering system used by the locale.

`var availableNumberingSystems: [Locale.NumberingSystem]`

An array containing all the valid numbering systems for the locale.

`struct NumberingSystem`

A type that represents the numbering system used in a locale.

Getting region components

`var region: Locale.Region?`

The region used by the locale.

`struct Region`

A type that represents a geographic region, for use in specifying a locale or language.

`var subdivision: Locale.Subdivision?`

The optional subdivision of the region used by this locale.

`struct Subdivision`

A type that represents a subdivision of a region, such as a state in the US or a province in Canada.

`var variant: Locale.Variant?`

An optional variant used by the locale.

`struct Variant`

A type that represents a locale's languate variant.

Getting ordering components

`var collation: Locale.Collation`

The string sort order of the locale.

`struct Collation`

A type that represents the string sort order used by the locale.

Getting information about a locale

`var identifier: String`

The identifier of the locale.

`func identifier(Locale.IdentifierType) -> String`

Returns the locale identifier, in the specified standard format.

`enum IdentifierType`

A type that indicates the standard that defines a locale's identifier.

`var calendar: Calendar`

The calendar for the locale, or the Gregorian calendar as a fallback.

`var regionCode: String?`

The region code of the locale, or `nil` if it has none.

`var languageCode: String?`

The language code of the locale, or `nil` if has none.

`var scriptCode: String?`

The script code of the locale, or `nil` if has none.

`var variantCode: String?`

The variant code for the locale, or `nil` if it has none.

`var exemplarCharacterSet: CharacterSet?`

The exemplar character set for the locale, or `nil` if has none.

`var collationIdentifier: String?`

The collation identifier for the locale, or `nil` if it has none.

`var collatorIdentifier: String?`

The collator identifier of the locale.

~~`var usesMetricSystem: Bool`~~

A Boolean that is true if the locale uses the metric system.

Deprecated

`var decimalSeparator: String?`

The decimal separator of the locale.

`var groupingSeparator: String?`

The grouping separator of the locale.

`var currencyCode: String?`

The currency code of the locale.

`var currencySymbol: String?`

The currency symbol of the locale.

```
var quotationBeginDelimiter: String?
```

The quotation begin delimiter of the locale.

```
var quotationEndDelimiter: String?
```

The quotation end delimiter of the locale.

```
var alternateQuotationBeginDelimiter: String?
```

The alternate quotation begin delimiter of the locale.

```
var alternateQuotationEndDelimiter: String?
```

The alternate quotation end delimiter of the locale.

Getting display information about a locale

```
func localizedString(for: Calendar.Identifier) -> String?
```

Returns a localized string for a specified calendar.

```
func localizedString(forCollationIdentifier: String) -> String?
```

Returns a localized string for a specified ICU collation identifier.

```
func localizedString(forCollatorIdentifier: String) -> String?
```

Returns a localized string for a specified ICU collator identifier.

```
func localizedString(forCurrencyCode: String) -> String?
```

Returns a localized string for a specified ISO 4217 currency code.

```
func localizedString(forIdentifier: String) -> String?
```

Returns a localized string for a specified locale identifier.

```
func localizedString(forLanguageCode: String) -> String?
```

Returns a localized string for a specified language code.

```
func localizedString(forRegionCode: String) -> String?
```

Returns a localized string for a specified region code.

```
func localizedString(forScriptCode: String) -> String?
```

Returns a localized string for a specified script code.

```
func localizedString(forVariantCode: String) -> String?
```

Returns a localized string for a specified variant code.

Getting the user's preferred languages

`static var preferredLanguages: [String]`

A list of the user's preferred languages.

Getting line and character direction for a language

~~`static func characterDirection(forLanguage: String) -> Locale.LanguageDirection`~~

Returns the character direction for a specified language code.

Deprecated

~~`static func lineDirection(forLanguage: String) -> Locale.LanguageDirection`~~

Returns the line direction for a specified language code.

Deprecated

`typealias LanguageDirection`

An alias for the standard set of language directions.

`enum LanguageDirection`

The directions that a language may take across a page of text.

Using reference types

`class NSLocale`

Information about linguistic, cultural, and technological conventions for use in formatting data for presentation.

Structures

`struct CurrentLocaleDidChangeMessage`

`struct LanguageCode`

An alphabetical code associated with a language.

`struct Script`

The written script used with a given language.

Type Properties

`static var preferredLocales: [Locale]`

Returns a list of the user's preferred locales, as specified in Language & Region settings, taking into account any per-app language overrides.

Relationships

Conforms To

`Copyable`
`CustomDebugStringConvertible`
`CustomReflectable`
`CustomStringConvertible`
`Decodable`
`Encodable`
`Equatable`
`Hashable`
`ReferenceConvertible`
`Sendable`
`SendableMetatype`