Class

NumberFormatter

A formatter that converts between numeric values 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 NumberFormatter
```

Overview

Instances of `NumberFormatter` format the textual representation of cells that contain `NSNumber` objects and convert textual representations of numeric values into `NSNumber` objects. The representation encompasses integers, floats, and doubles; floats and doubles can be formatted to a specified decimal position. `NumberFormatter` objects can also impose ranges on the numeric values cells can accept.

> **Tip**
>
> In Swift, you can use `IntegerFormatStyle`, `FloatingPointFormatStyle`, or `Decimal.FormatStyle` rather than `NumberFormatter`. 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.

Significant Digits and Fraction Digits

The `NumberFormatter` class provides flexible options for displaying non-zero fractional parts of numbers.

If you set the `usesSignificantDigits` property to `true`, you can configure `NumberFormatter` to display significant digits using the `minimumSignificantDigits` and `maximumSignificantDigits` properties. If `usesSignificantDigits` is `false`, these properties are ignored. See Configuring Significant Digits.

Otherwise, you can configure the minimum and maximum number of integer and fraction digits, or the numbers before and after the decimal separator, respectively, using the `minimumIntegerDigits`, `maximumIntegerDigits`, `minimumFractionDigits`, and `maximumFractionDigits` properties. See Configuring Integer and Fraction Digits.

Thread Safety

On iOS 7 and later `NumberFormatter` is thread-safe.

In macOS 10.9 and later `NumberFormatter` 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, `NumberFormatter` is not thread-safe, and you therefore must not mutate a number formatter simultaneously from multiple threads.

Topics

Configuring Formatter Behavior and Style

`var formatterBehavior: NumberFormatter.Behavior`

The formatter behavior of the receiver.

`class func setDefaultFormatterBehavior(NumberFormatter.Behavior)`

Sets the default formatter behavior for new instances of `NSNumberFormatter`.

`class func defaultFormatterBehavior() -> NumberFormatter.Behavior`

Returns an `NSNumberFormatterBehavior` constant that indicates default formatter behavior for new instances of `NSNumberFormatter`.

`var numberStyle: NumberFormatter.Style`

The number style used by the receiver.

`var generatesDecimalNumbers: Bool`

Determines whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.

Converting Between Numbers and Strings

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

Returns by reference a cell-content object after creating it from a range of characters in a given string.

```
func number(from: String) -> NSNumber?
```

Returns an NSNumber object created by parsing a given string.

```
func string(from: NSNumber) -> String?
```

Returns a string containing the formatted value of the provided number object.

```
class func localizedString(from: NSNumber, number: NumberFormatter.
Style) -> String
```

Returns a localized number string with the specified style.

Managing Localization of Numbers

```
var localizesFormat: Bool
```

Determines whether the dollar sign character ($), decimal separator character (.), and thousand separator character (,) are converted to appropriately localized characters as specified by the user's localization preference.

```
var locale: Locale!
```

The locale of the receiver.

Configuring Rounding Behavior

```
var roundingBehavior: NSDecimalNumberHandler
```

The rounding behavior used by the receiver.

```
class NSDecimalNumberHandler
```

A class that adopts the decimal number behaviors protocol.

```
var roundingIncrement: NSNumber!
```

The rounding increment used by the receiver.

```
var roundingMode: NumberFormatter.RoundingMode
```

The rounding mode used by the receiver.

Configuring Integer and Fraction Digits

`var minimumIntegerDigits: Int`

The minimum number of digits before the decimal separator.

`var maximumIntegerDigits: Int`

The maximum number of digits before the decimal separator.

`var minimumFractionDigits: Int`

The minimum number of digits after the decimal separator.

`var maximumFractionDigits: Int`

The maximum number of digits after the decimal separator.

Configuring Significant Digits

`var usesSignificantDigits: Bool`

A Boolean value indicating whether the formatter uses minimum and maximum significant digits when formatting numbers.

`var minimumSignificantDigits: Int`

The minimum number of significant digits for the number formatter.

`var maximumSignificantDigits: Int`

The maximum number of significant digits for the number formatter.

Configuring Numeric Formats

`var format: String`

The receiver's format.

`var formattingContext: Formatter.Context`

The capitalization formatting context used when formatting a number.

`var formatWidth: Int`

The format width used by the receiver.

`var negativeFormat: String!`

The format the receiver uses to display negative values.

`var positiveFormat: String!`

The format the receiver uses to display positive values.

`var multiplier: NSNumber?`

The multiplier of the receiver.

Configuring Numeric Symbols

`var percentSymbol: String!`

The string used to represent a percent symbol.

`var perMillSymbol: String!`

The string used to represent a per-mill (per-thousand) symbol.

`var minusSign: String!`

The string used to represent a minus sign.

`var plusSign: String!`

The string used to represent a plus sign.

`var exponentSymbol: String!`

The string used to represent an exponent symbol.

`var zeroSymbol: String?`

The string used to represent a zero value.

`var nilSymbol: String`

The string used to represent a `nil` value.

`var notANumberSymbol: String!`

The string used to represent a NaN (“not a number”) value.

`var negativeInfinitySymbol: String`

The string used to represent a negative infinity symbol.

`var positiveInfinitySymbol: String`

The string used to represent a positive infinity symbol.

Configuring the Format of Currency

`var currencySymbol: String!`

The string used by the receiver as a local currency symbol.

`var currencyCode: String!`

The receiver's currency code.

`var internationalCurrencySymbol: String!`

The international currency symbol used by the receiver.

`var currencyGroupingSeparator: String!`

The currency grouping separator for the receiver.

Configuring Numeric Prefixes and Suffixes

`var positivePrefix: String!`

The string the receiver uses as the prefix for positive values.

`var positiveSuffix: String!`

The string the receiver uses as the suffix for positive values.

`var negativePrefix: String!`

The string the receiver uses as a prefix for negative values.

`var negativeSuffix: String!`

The string the receiver uses as a suffix for negative values.

Configuring the Display of Numeric Values

`var textAttributesForNegativeValues: [String : Any]?`

The text attributes to be used in displaying negative values.

`var textAttributesForPositiveValues: [String : Any]?`

The text attributes to be used in displaying positive values.

`var attributedStringForZero: NSAttributedString`

The attributed string that the receiver uses to display zero values.

`var textAttributesForZero: [String : Any]?`

The text attributes used to display a zero value.

`var attributedStringForNil: NSAttributedString`

The attributed string the receiver uses to display `nil` values.

`var textAttributesForNil: [String : Any]?`

The text attributes used to display the `nil` symbol.

`var attributedStringForNotANumber: NSAttributedString`

The attributed string the receiver uses to display “not a number” values.

`var textAttributesForNotANumber: [String : Any]?`

The text attributes used to display the NaN (“not a number”) string.

`var textAttributesForPositiveInfinity: [String : Any]?`

The text attributes used to display the positive infinity symbol.

`var textAttributesForNegativeInfinity: [String : Any]?`

The text attributes used to display the negative infinity symbol.

Configuring Separators and Grouping Size

`var groupingSeparator: String!`

The string used by the receiver for a grouping separator.

`var usesGroupingSeparator: Bool`

Determines whether the receiver displays the group separator.

`var thousandSeparator: String!`

The character the receiver uses as a thousand separator.

`var hasThousandSeparators: Bool`

Determines whether the receiver uses thousand separators.

`var decimalSeparator: String!`

The character the receiver uses as a decimal separator.

`var alwaysShowsDecimalSeparator: Bool`

Determines whether the receiver always shows the decimal separator, even for integer numbers.

`var currencyDecimalSeparator: String!`

The string used by the receiver as a currency decimal separator.

`var groupingSize: Int`

The grouping size of the receiver.

`var secondaryGroupingSize: Int`

The secondary grouping size of the receiver.

Managing the Padding of Numbers

`var paddingCharacter: String!`

The string that the receiver uses to pad numbers in the formatted string representation.

`var paddingPosition: NumberFormatter.PadPosition`

The padding position used by the receiver.

Managing Input and Output Attributes

`var allowsFloats: Bool`

Determines whether the receiver allows as input floating-point values (that is, values that include the period character [.]).

`var minimum: NSNumber?`

The lowest number allowed as input by the receiver.

`var maximum: NSNumber?`

The highest number allowed as input by the receiver.

Managing Leniency Behavior

`var isLenient: Bool`

Determines whether the receiver will use heuristics to guess at the number which is intended by a string.

Managing the Validation of Partial Numeric Strings

`var isPartialStringValidationEnabled: Bool`

Determines whether partial string validation is enabled for the receiver.

Constants

`enum Style`

The predefined number format styles used by the `numberStyle` property.

`enum` `Behavior`

These constants specify the behavior of a number formatter. These constants are returned by the `defaultFormatterBehavior()` class method and the `formatterBehavior` property.

`enum` `PadPosition`

These constants are used to specify how numbers should be padded. These constants are used by the `paddingPosition` property.

`enum` `RoundingMode`

These constants are used to specify how numbers should be rounded. These constants are used by the `roundingMode` property.

Instance Properties

```
var minimumGroupingDigits: Int
```

Relationships

Inherits From

`Formatter`

Conforms To

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