Structure

FloatingPointFormatStyle

A structure that converts between floating-point values and their textual representations.

iOS 15.0+ | iPadOS 15.0+ | Mac Catalyst 15.0+ | macOS 12.0+ | tvOS 15.0+ | visionOS 1.0+ | watchOS 8.0+

```
struct FloatingPointFormatStyle<Value> where Value : BinaryFloatingPoint
```

Overview

Instances of `FloatingPointFormatStyle` create localized, human-readable text from `BinaryFloatingPoint` numbers and parse string representations of numbers into instances of `BinaryFloatingPoint` types. All of the Swift standard library's floating-point types, such as `Double`, `Float`, and `Float80`, conform to `BinaryFloatingPoint`, and therefore work with this format style.

`FloatingPointFormatStyle` includes two nested types, `FloatingPointFormatStyle.Percent` and `FloatingPointFormatStyle.Currency`, for working with percentages and currencies, respectively. Each format style includes a configuration that determines how it represents numeric values, for things like grouping, displaying signs, and variant presentations like scientific notation. `FloatingPointFormatStyle` and `FloatingPointFormatStyle.Percent` include a `NumberFormatStyleConfiguration`, and `FloatingPointFormatStyle.Currency` includes a `CurrencyFormatStyleConfiguration`. You can customize numeric formatting for a style by adjusting its backing configuration. The system automatically caches unique configurations of a format style to enhance performance.

Note

Foundation provides another format style type, IntegerFormatStyle, for working with numbers that conform to BinaryInteger. For Foundation's Decimal type, use Decimal.FormatStyle.

Formatting floating-point values

Use the formatted() method to create a string representation of a floating-point value using the default FloatingPointFormatStyle configuration.

```
let formattedDefault = 12345.67.formatted()
// formattedDefault is "12,345.67" in the en_US locale.
// Other locales may use different separator and grouping behavior.
```

You can specify a format style by providing an argument to the formatted(_:) method. The following example shows the number 0.1 represented in each of the available styles, in the en_US locale:

```
let number = 0.1

let formattedNumber = number.formatted(.number)
// formattedNumber is "0.1".

let formattedPercent = number.formatted(.percent)
// formattedPercent is "10%".

let formattedCurrency = number.formatted(.currency(code: "USD"))
// formattedCurrency is "$0.10".
```

Each style provides methods for updating its numeric configuration, including the number of significant digits, grouping length, and more. You can specify a numeric configuration by calling as many of these methods as you need in any order you choose. The following example shows the same number with default and custom configurations:

```
let exampleNumber = 123456.78

let defaultFormatting = exampleNumber.formatted(.number)
// defaultFormatting is "123 456,78" for the "fr_FR" locale.
// defaultFormatting is "123,456.78" for the "en_US" locale.
```

```swift
let customFormatting = exampleNumber.formatted(
    .number
        .grouping(.never)
        .sign(strategy: .always()))
// customFormatting is "+123456.78"
```

Creating a floating-point format style instance

The previous examples use static factory methods like `number` to create format styles within the call to the `formatted(_:)` method. You can also create a `FloatingPointFormatStyle` instance and use it to repeatedly format different values, with the `format(_:)` method:

```swift
let percentFormatStyle = FloatingPointFormatStyle<Double>.Percent()

percentFormatStyle.format(0.5) // "50%"
percentFormatStyle.format(0.855) // "85.5%"
percentFormatStyle.format(1.0) // "100%"
```

Parsing floating-point values

You can use `FloatingPointFormatStyle` to parse strings into floating-point values. You can define the format style within the type's initializer or pass in a format style created outside the function, as shown here:

```swift
let price = try? Double("$3,500.63",
                         format: .currency(code: "USD")) // 3500.63

let priceFormatStyle = FloatingPointFormatStyle<Double>.Currency(code: "USD")
let salePrice = try? Double("$731.67",
                             format: priceFormatStyle) // 731.67
```

Matching regular expressions

Along with parsing numeric values in strings, you can use theSwift regular expression domain-specific language to match and capture numeric substrings. The following example defines a percentage format style to match a percentage value using en_US numeric conventions. The rest of the regular expression ignores any characters prior to a `": "` sequence that precedes the percentage substring.

```
import RegexBuilder
let source = "Percentage complete: 55.1%"
let matcher = Regex {
    OneOrMore(.any)
    ": "
    Capture {
        One(.localizedDoublePercentage(locale: Locale(identifier: "en_US")))
    }
}
let match = source.firstMatch(of: matcher)
let localizedPercentage = match?.1
print("\(localizedPercentage!)") // 0.551
```

Topics

Creating a floating-point format style

`init(locale: Locale)`

Creates a floating-point format style that uses the given locale.

Customizing style behavior

`func decimalSeparator(strategy: FloatingPointFormatStyle<Value>.Configuration.DecimalSeparatorDisplayStrategy) -> FloatingPointFormatStyle<Value>`

Modifies the format style to use the specified decimal separator display strategy.

`func grouping(FloatingPointFormatStyle<Value>.Configuration.Grouping) -> FloatingPointFormatStyle<Value>`

Modifies the format style to use the specified grouping.

`func notation(FloatingPointFormatStyle<Value>.Configuration.Notation) -> FloatingPointFormatStyle<Value>`

Modifies the format style to use the specified notation.

`func precision(FloatingPointFormatStyle<Value>.Configuration.Precision) -> FloatingPointFormatStyle<Value>`

Modifies the format style to use the specified precision.

```
func rounded(rule: FloatingPointFormatStyle<Value>.Configuration.
RoundingRule, increment: Double?) -> FloatingPointFormatStyle<Value>
```

Modifies the format style to use the specified rounding rule and increment.

```
func scale(Double) -> FloatingPointFormatStyle<Value>
```

Modifies the format style to use the specified scale.

```
func sign(strategy: FloatingPointFormatStyle<Value>.Configuration.Sign
DisplayStrategy) -> FloatingPointFormatStyle<Value>
```

Modifies the format style to use the specified sign display strategy for displaying or omitting sign symbols.

```
typealias Configuration
```

The type the format style uses for configuration settings.

```
enum NumberFormatStyleConfiguration
```

Configuration settings for formatting numbers of different types.

Accessing style locale

```
var locale: Locale
```

The locale of the format style.

Applying currency styles

```
struct Currency
```

A format style that converts between floating-point currency values and their textual representations.

Applying measurement styles

```
struct FormatStyle
```

A type that provides localized representations of measurements.

Applying list styles

```
struct ListFormatStyle
```

A type that formats lists of items with a separator and conjunction appropriate for a given locale.

Creating attributed strings

`var attributed: FloatingPointFormatStyle<Value>.Attributed`

An attributed format style based on the floating-point format style.

`struct Attributed`

A format style that converts integers into attributed strings.

Parsing floating-point numbers

`struct FloatingPointParseStrategy`

A parse strategy for creating floating-point values from formatted strings.

Supporting types

`struct Currency`

A format style that converts between floating-point currency values and their textual representations.

`struct Percent`

A format style that converts between floating-point percentage values and their textual representations.

Relationships

Conforms To

`Copyable`
`CustomConsumingRegexComponent`
`Decodable`
`Encodable`
`Equatable`
`FormatStyle`

Conforms when `Value` conforms to `BinaryFloatingPoint`.

`Hashable`

`ParseableFormatStyle`

Conforms when `Value` conforms to `BinaryFloatingPoint`.

`RegexComponent`

`Sendable`

`SendableMetatype`

See Also

Data formatting in Swift

{} Language Introspector

Converts data into human-readable text using formatters and locales.

`protocol FormatStyle`

A type that converts a given data type into a representation in another type, such as a string.

`struct IntegerFormatStyle`

A structure that converts between integer values and their textual representations.

`struct FormatStyle`

A structure that converts between decimal values and their textual representations.

`struct ListFormatStyle`

A type that formats lists of items with a separator and conjunction appropriate for a given locale.

`struct StringStyle`

`struct FormatStyle`

A structure that converts between URL instances and their textual representations.

`struct FormatStyleCapitalizationContext`

The capitalization formatting context used when formatting dates and times.

Format Style Configurations

Behaviors for traits like numeric precision, rounding, and scale, used for formatting and parsing numeric values.