Protocol

FormatStyle

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

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

```
protocol FormatStyle<FormatInput, FormatOutput> : Decodable, Encodable, Hashable
```

Overview

Types conforming to the `FormatStyle` protocol take their input type and produce formatted instances of their output type. The formatting process accounts for locale-specific conventions, like grouping and separators for numbers, and presentation of units for measurements. The format styles Foundation provides produce their output as `String` or `AttributedString` instances. You can also create custom styles that format their output as any type, like XML or JSON `Data` or an image.

There are two basic approaches to using a `FormatStyle`:

- Create an instance of a type that conforms to `FormatStyle` and apply it to one or more instances of the input type, by calling the style's `format(_:)` method. Use this when you want to customize a style once and apply it repeatedly to many instances.
- Pass an instance of a type that conforms to `FormatStyle` to the data type's `formatted(_:)` method, which takes the style as a parameter. Use this for one-off formatting scenarios, or when you want to apply different format styles to the same data value. For the simplest cases, most types that support formatting also have a no-argument `formatted()` method that applies a locale-appropriate default format style.

Foundation provides format styles for integers (`IntegerFormatStyle`), floating-point numbers (`FloatingPointFormatStyle`), decimals (`Decimal.FormatStyle`), measurements

(Measurement.FormatStyle), arrays (ListFormatStyle), and more. The "Conforming types" section below shows all the format styles available from Foundation and any system frameworks that implement the FormatStyle protocol. The numeric format styles also provide supporting format styles to format currency and percent values, like IntegerFormatStyle.Currency and Decimal.FormatStyle.Percent.

Modifying a format style

Format styles include modifier methods that return a new format style with an adjusted behavior. The following example creates an IntegerFormatStyle, then applies modifiers to round values down to the nearest 1,000 and applies formatting appropriate to the fr_FR locale:

```
let style = IntegerFormatStyle<Int>()
    .rounded(rule: .down, increment: 1000)
    .locale(Locale(identifier: "fr_FR"))
let rounded = 123456789.formatted(style) // "123 456 000"
```

Foundation caches identical instances of a customized format style, so you don't need to pass format style instances around unrelated parts of your app's source code.

Accessing static instances

Types that conform to FormatStyle typically extend the base protocol with type properties or type methods to provide convenience instances. These are available for use in a data type's formatted(_:) method when the format style's input type matches the data type. For example, the various numeric format styles define number properties with generic constraints to match the different numeric types (Double, Int, Float16, and so on).

To see how this works, consider this example of a default formatter for an Int value. Because 123456789 is a BinaryInteger, its formatted(_:) method accepts an IntegerFormatStyle parameter. The following example shows the style's default behavior in the en_US locale.

```
let formatted = 123456789.formatted(IntegerFormatStyle()) // "123,456,789"
```

IntegerFormatStyle extends FormatStyle with multiple type properties called number, each of which is an IntegerFormatStyle instance; these properties differ by which BinaryInteger-conforming type they take as input. Since one of these statically-defined properties (number) takes Int as its input, you can use this type property instead of instantiating a new format style instance. Using dot notation to access this property on the inferred FormatStyle makes the call point much easier to read, as seen here:

```
let formatted = 123456789.formatted( .number) // "123,456,789"
```

Furthermore, since you can customize these statically-accessed format style instances, you can rewrite the example from the previous section without instantiating a new `IntegerFormatStyle`, like this:

```
let rounded = 123456789.formatted( .number
    .rounded(rule: .down, increment: 1000)
    .locale(Locale(identifier: "fr_FR"))) // "123 456 000"
```

Parsing with a format style

To perform the opposite conversion — from formatted output type to input data type — some format styles provide a corresponding `ParseStrategy` type. These format styles typically expose an instance of this type as a variable, called `parseStrategy`.

You can use a `ParseStrategy` one of two ways:

- Initialize the data type by calling an initializer of that type that takes a formatted instance and a parse strategy as parameters. For example, you can create a `Decimal` from a formatted string with the initializer `init(_:format:lenient:)`.
- Create a parse strategy and call its `parse(_:)` method on one or more formatted instances.

Topics

Performing formatting

`func format(Self.FormatInput) -> Self.FormatOutput`

Formats a value, using this style.

Required

Setting style Locale

`func locale(Locale) -> Self`

Modifies the format style to use the specified locale.

Required Default implementation provided.

Applying numeric styles for integers

`static var number: IntegerFormatStyle<Int>`

A style for formatting the Swift default integer type.

`static var number: IntegerFormatStyle<UInt>`

A style for formatting the Swift unsigned integer type.

`static var number: IntegerFormatStyle<Int8>`

A style for formatting 8-bit signed integers.

`static var number: IntegerFormatStyle<Int16>`

A style for formatting 16-bit signed integers.

`static var number: IntegerFormatStyle<Int32>`

A style for formatting 32-bit signed integers.

`static var number: IntegerFormatStyle<Int64>`

A style for formatting 64-bit signed integers.

`static var number: IntegerFormatStyle<UInt8>`

A style for formatting 8-bit unsigned integers.

`static var number: IntegerFormatStyle<UInt16>`

A style for formatting 16-bit unsigned integers.

`static var number: IntegerFormatStyle<UInt32>`

A style for formatting 32-bit unsigned integers.

`static var number: IntegerFormatStyle<UInt64>`

A style for formatting 64-bit unsigned integers.

`struct IntegerFormatStyle`

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

Applying numeric styles for floating-point values

`static var number: FloatingPointFormatStyle<Float>`

A style for formatting the Swift standard single-precision floating-point type.

`static var number: FloatingPointFormatStyle<Double>`

A style for formatting the Swift standard double-precision floating-point type.

`static var number: FloatingPointFormatStyle<Float16>`

A style for formatting 16-bit floating-point values.

`struct FloatingPointFormatStyle`

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

Applying numeric styles for decimals

`static var number: Decimal.FormatStyle`

A style for formatting decimal values.

`struct FormatStyle`

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

Applying percentage styles for integers

`static var percent: IntegerFormatStyle<Int>.Percent`

A style for formatting signed integer types in Swift as a percent representation.

`static var percent: IntegerFormatStyle<UInt>.Percent`

A style for formatting signed integer types in Swift as a percent representation.

`static var percent: IntegerFormatStyle<Int8>.Percent`

A style for formatting 8-bit signed integers as a percent representation.

`static var percent: IntegerFormatStyle<Int16>.Percent`

A style for formatting 16-bit signed integers as a percent representation.

`static var percent: IntegerFormatStyle<Int32>.Percent`

A style for formatting 32-bit signed integers as a percent representation.

`static var percent: IntegerFormatStyle<Int64>.Percent`

A style for formatting 64-bit signed integers as a percent representation.

`static var percent: IntegerFormatStyle<UInt8>.Percent`

A style for formatting 8-bit unsigned integers as a percent representation.

`static var percent: IntegerFormatStyle<UInt16>.Percent`

A style for formatting 16-bit unsigned integers as a percent representation.

`static var percent: IntegerFormatStyle<UInt32>.Percent`

A style for formatting 32-bit unsigned integers as a percent representation.

`static var percent: IntegerFormatStyle<UInt64>.Percent`

A style for formatting 64-bit unsigned integers as a percent representation.

`struct Percent`

A format style that converts between integer percentage values and their textual representations.

Applying percentage styles for floating-point values

`static var percent: FloatingPointFormatStyle<Float>.Percent`

A style for formatting the Swift standard single-precision floating-point type as a percent representation.

`static var percent: FloatingPointFormatStyle<Double>.Percent`

A style for formatting the Swift standard single-precision floating-point type as a percent representation.

`static var percent: FloatingPointFormatStyle<Float16>.Percent`

A style for formatting 16-bit floating-point values as a percent representation.

`struct Percent`

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

Applying percentage styles for decimals

`static var percent: Decimal.FormatStyle.Percent`

A style for formatting decimal values as a percent represntation.

`struct Percent`

A format style that converts between decimal percentage values and their textual representations.

Applying date and time styles

`static var dateTime: Date.FormatStyle`

A style for formatting a date and time.

`struct FormatStyle`

A structure that creates a locale-appropriate string representation of a date instance and converts strings of dates and times into date instances.

```
struct ISO8601FormatStyle
```

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

```
static func verbatim(Date.FormatString, locale: Locale?, timeZone: TimeZone, calendar: Calendar) -> Date.VerbatimFormatStyle
```

Returns a style for formatting a date with an explicitly-specified style.

```
struct VerbatimFormatStyle
```

A style that formats a date with an explicitly-specified style.

```
static var interval: Date.IntervalFormatStyle
```

A style for formatting a date interval.

```
struct IntervalFormatStyle
```

A format style that creates string representations of date intervals.

```
static func relative(presentation: Date.RelativeFormatStyle.Presentation, unitsStyle: Date.RelativeFormatStyle.UnitsStyle) -> Self
```

Returns a style for formatting a date as relative to the current date.

```
struct RelativeFormatStyle
```

A format style that forms locale-aware string representations of a relative date or time.

```
static func components(style: Date.ComponentsFormatStyle.Style, fields: Set<Date.ComponentsFormatStyle.Field>?) -> Self
```

Returns a style for formatting a date interval in terms of specific date components.

```
struct ComponentsFormatStyle
```

A style for formatting a date interval in terms of specific date components.

Applying duration styles

```
static var timeDuration: Date.ComponentsFormatStyle
```

A style for formatting a duration expressed as a range of dates.

```
struct ComponentsFormatStyle
```

A style for formatting a date interval in terms of specific date components.

```
static func time(pattern: Duration.TimeFormatStyle.Pattern) -> Self
```

Returns a style for formatting a duration using a provided pattern.

```
static func units(allowed: Set<Duration.UnitsFormatStyle.Unit>, width: Duration.UnitsFormatStyle.UnitWidth, maximumUnitCount: Int?, zeroValueUnits: Duration.UnitsFormatStyle.ZeroValueUnitsDisplayStrategy, valueLength: Int?, fractionalPart: Duration.UnitsFormatStyle.FractionalPartDisplayStrategy) -> Self
```

Returns a style for formatting a duration that uses the specified units.

```
static func units<ValueRange>(allowed: Set<Duration.UnitsFormatStyle.Unit>, width: Duration.UnitsFormatStyle.UnitWidth, maximumUnitCount: Int?, zeroValueUnits: Duration.UnitsFormatStyle.ZeroValueUnitsDisplayStrategy, valueLengthLimits: ValueRange, fractionalPart: Duration.UnitsFormatStyle.FractionalPartDisplayStrategy) -> Self
```

Returns a style for formatting a duration range that uses the specified units, with padding/truncating behavior defined as a range.

Applying currency styles

```
static func currency<V>(code: String) -> Self
```

Returns a format style to use integer currency notation.

```
static func currency<Value>(code: String) -> Self
```

Returns a format style to use floating-point currency notation.

```
static func currency(code: String) -> Self
```

Returns a format style to use decimal currency notation.

Applying measurement styles

```
static func measurement<UnitType>(width: Measurement<UnitType>.FormatStyle.UnitWidth, usage: MeasurementFormatUnitUsage<UnitType>, numberFormatStyle: FloatingPointFormatStyle<Double>?) -> Self
```

Returns a format style to format measurement units.

```
static func measurement(width: Measurement<UnitTemperature>.FormatStyle.UnitWidth, usage: MeasurementFormatUnitUsage<UnitTemperature>, hidesScaleName: Bool, numberFormatStyle: FloatingPointFormatStyle<Double>?) -> Self
```

Returns a format style to format temperature units.

Applying person name styles

```
static func name(style: PersonNameComponents.FormatStyle.Style) -> Self
```

Returns a format style to use the given name style for formatting a name from its components.

Applying list styles

```
static func list<MemberStyle, Base>(memberStyle: MemberStyle, type: ListFormatStyle<MemberStyle, Base>.ListType, width: ListFormatStyle<MemberStyle, Base>.Width) -> Self
```

Returns a format style to format a list of items.

```
static func list<Base>(type: ListFormatStyle<StringStyle, Base>.ListType, width: ListFormatStyle<StringStyle, Base>.Width) -> Self
```

Returns a format style to format a list of strings.

Applying byte-count styles

```
static func byteCount(style: ByteCountFormatStyle.Style, allowedUnits: ByteCountFormatStyle.Units, spellsOutZero: Bool, includesActualByteCount: Bool) -> Self
```

Returns a format style to format a data storage value.

```
struct ByteCountFormatStyle
```

A format style that provides string representations of byte counts.

```
static func byteCount(style: Measurement<UnitInformationStorage>.FormatStyle.ByteCount.Style, allowedUnits: Measurement<UnitInformationStorage>.FormatStyle.ByteCount.Units, spellsOutZero: Bool, includesActualByteCount: Bool) -> Self
```

Returns a format style to format a data storage value represented with Foundation's measurement type.

```
struct ByteCount
```

A format style that provides string representations of byte counts, expressed as measurements of information storage.

Applying URL styles

```
static var url: URL.FormatStyle
```

A style for formatting a URL.

```
struct FormatStyle
```

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

Declaring input and output types

```
associatedtype FormatInput
```

The type this format style accepts as input.

Required

```
associatedtype FormatOutput
```

The type this format style produces as output.

Required

Type Properties

```
static var http: Date.HTTPFormatStyle
```

```
static var http: DateComponents.HTTPFormatStyle
```

```
static var iso8601: DateComponents.ISO8601FormatStyle
```

```
static var iso8601: Date.ISO8601FormatStyle
```

Type Methods

```
static func offset(to: Date, allowedFields: Set<Date.ComponentsFormat
Style.Field>, maxFieldCount: Int, sign: NumberFormatStyleConfiguration.
SignDisplayStrategy) -> SystemFormatStyle.DateOffset
```

```
static func reference(to: Date, allowedFields: Set<Date.RelativeFormat
Style.Field>, maxFieldCount: Int, thresholdField: Date.RelativeFormat
Style.Field) -> SystemFormatStyle.DateReference
```

```
static func stopwatch(startingAt: Date, showsHours: Bool, maxFieldCount
: Int, maxPrecision: Duration) -> SystemFormatStyle.Stopwatch
```

```
static func timer(countingDownIn: Range<Date>, showsHours: Bool, max
FieldCount: Int, maxPrecision: Duration) -> SystemFormatStyle.Timer
```

```
static func timer(countingUpIn: Range<Date>, showsHours: Bool, maxFieldCount: Int, maxPrecision: Duration) -> SystemFormatStyle.Timer
```

Relationships

Inherits From

Decodable
Encodable
Equatable
Hashable

Inherited By

DiscreteFormatStyle, ParseableFormatStyle

Conforming Types

ByteCountFormatStyle
ByteCountFormatStyle.Attributed
Date.AnchoredRelativeFormatStyle
Date.AttributedStyle
Date.ComponentsFormatStyle
Date.FormatStyle
Date.FormatStyle.Attributed
Date.HTTPFormatStyle
Date.ISO8601FormatStyle
Date.IntervalFormatStyle
Date.RelativeFormatStyle
Date.VerbatimFormatStyle
Date.VerbatimFormatStyle.Attributed
DateComponents.HTTPFormatStyle
DateComponents.ISO8601FormatStyle
Decimal.FormatStyle
Decimal.FormatStyle.Attributed
Decimal.FormatStyle.Currency
Decimal.FormatStyle.Percent
FloatingPointFormatStyle

Conforms when `Value` conforms to `BinaryFloatingPoint`.

`FloatingPointFormatStyle.Attributed`

Conforms when `Value` conforms to `BinaryFloatingPoint`.

`FloatingPointFormatStyle.Currency`

Conforms when `Value` conforms to `BinaryFloatingPoint`.

`FloatingPointFormatStyle.Percent`

Conforms when `Value` conforms to `BinaryFloatingPoint`.

`IntegerFormatStyle`

Conforms when `Value` conforms to `BinaryInteger`.

`IntegerFormatStyle.Attributed`

Conforms when `Value` conforms to `BinaryInteger`.

`IntegerFormatStyle.Currency`

Conforms when `Value` conforms to `BinaryInteger`.

`IntegerFormatStyle.Percent`

Conforms when `Value` conforms to `BinaryInteger`.

`ListFormatStyle`
`Measurement.AttributedStyle`

Conforms when `UnitType` inherits `Dimension`.

`Measurement.AttributedStyle.ByteCount`

Conforms when `UnitType` is `UnitInformationStorage`.

`Measurement.FormatStyle`

Conforms when `UnitType` inherits `Dimension`.

`Measurement.FormatStyle.ByteCount`

Conforms when `UnitType` is `UnitInformationStorage`.

`PersonNameComponents.AttributedStyle`
`PersonNameComponents.FormatStyle`
`StringStyle`
`URL.FormatStyle`

See Also

Data formatting in Swift

{} Language Introspector

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

`struct IntegerFormatStyle`

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

`struct` `FloatingPointFormatStyle`

A structure that converts between floating-point 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.