

[Foundation](#) / [IntegerFormatStyle](#)

Structure

IntegerFormatStyle

A structure that converts between integer 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 IntegerFormatStyle<Value> where Value : BinaryInteger
```

Overview

Instances of [IntegerFormatStyle](#) create localized, human-readable text from [BinaryInteger](#) numbers and parse string representations of numbers into instances of [BinaryInteger](#) types. All of the Swift standard library's integer types, such as [Int](#) and [UInt32](#), conform to [BinaryInteger](#), and therefore work with this format style.

[IntegerFormatStyle](#) includes two nested types, [IntegerFormatStyle.Percent](#) and [IntegerFormatStyle.Currency](#), for working with percentages and currencies. 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. [IntegerFormatStyle](#) and [IntegerFormatStyle.Percent](#) include a [NumberFormatStyleConfiguration](#), and [IntegerFormatStyle.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, [FloatingPointFormatStyle](#), for working with numbers that conform to [BinaryFloatingPoint](#). For Foundation's [Decimal](#) type, use [Decimal.FormatStyle](#).

Formatting integers

Use the [formatted\(\)](#) method to create a string representation of an integer using the default [IntegerFormatStyle](#) configuration.

```
let formattedDefault = 123456.formatted()  
// formattedDefault is "123,456" in 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 12345 represented in each of the available styles, in the en_US locale:

```
let number = 123456  
  
let formattedNumber = number.formatted(.number)  
// formattedNumber is "123,456".  
  
let formattedPercent = number.formatted(.percent)  
// formattedPercent is "123,456%".  
  
let formattedCurrency = number.formatted(.currency(code: "USD"))  
// formattedCurrency is "$123,456.00".
```

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  
  
let defaultFormatting = exampleNumber.formatted(.number)  
// defaultFormatting is "125 000" for the "fr_FR" locale  
// defaultFormatting is "125000" for the "jp_JP" locale
```

```
// defaultFormatting is "125,000" for the "en_US" locale

let customFormatting = exampleNumber.formatted(
    .number
    .grouping(.never)
    .sign(strategy: .always()))
// customFormatting is "+123456"
```

Creating an integer 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 an `IntegerFormatStyle` instance and use it to repeatedly format different values with the `format(_:_:)` method:

```
let percentFormatStyle = IntegerFormatStyle<Int>.Percent()

percentFormatStyle.format(50) // "50%"
percentFormatStyle.format(85) // "85%"
percentFormatStyle.format(100) // "100%"
```

Parsing integers

You can use `IntegerFormatStyle` to parse strings into integer values. You can define the format style within the type's initializer or pass in a format style you create prior to calling the method, as shown here:

```
let price = try? Int("$123,456",
                    format: .currency(code: "USD")) // 123456

let priceFormatStyle = IntegerFormatStyle<Int>.Currency(code: "USD")
let salePrice = try? Int("$120,000",
                        format: priceFormatStyle) // 120000
```

Matching regular expressions

Along with parsing numeric values in strings, you can use the Swift regular expression domain-specific language to match and capture numeric substrings. The following example defines a currency format style to match and capture a currency value using US dollars and en_US numeric

conventions. The rest of the regular expression ignores any characters prior to a ":" sequence that precedes the currency substring.

```
import RegexBuilder

let source = "Payment due: $123,456"
let matcher = Regex {
    OneOrMore(.any)
    ":"
    Capture {
        One(.localizedIntegerCurrency(code: Locale.Currency("USD"),
                                     locale: Locale(identifier: "en_US")))
    }
}

let match = source.firstMatch(of: matcher)
let localizedInteger = match?.1 // 123456
```

Topics

Creating an integer format style

```
init(locale: Locale)
```

Creates an integer format style that uses the given locale.

Formatting integer values

```
func format(Value) -> String
```

Formats an integer, using this style.

Customizing style behavior

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

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

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

Modifies the format style to use the specified grouping.

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

Modifies the format style to use the specified notation.

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

Modifies the format style to use the specified precision.

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

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

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

Modifies the format style to use the specified scale.

```
func sign(strategy: IntegerFormatStyle<Value>.Configuration.SignDisplayStrategy) -> IntegerFormatStyle<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.

Acessing style locale

```
var locale: Locale
```

The locale of the format style.

Applying currency styles

```
struct Currency
```

A format style that converts between integer 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: IntegerFormatStyle<Value>.Attributed
```

An attributed format style based on the integer format style.

```
struct Attributed
```

A format style that converts integers into attributed strings.

Parsing integers

```
struct IntegerParseStrategy
```

A parse strategy for creating integer values from formatted strings.

Supporting types

```
struct Currency
```

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

```
struct Percent
```

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

Default Implementations

 FormatStyle Implementations

Relationships

Conforms To

Copyable

CustomConsumingRegexComponent

Decodable

Encodable

Equatable

FormatStyle

Conforms when `Value` conforms to `BinaryInteger`.

Hashable

ParseableFormatStyle

Conforms when `Value` conforms to `BinaryInteger`.

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` `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.