Class

Formatter

An abstract class that declares an interface for objects that create, interpret, and validate the textual representation of values.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.0+ | macOS 10.0+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class Formatter
```

Overview

The Foundation framework provides several concrete subclasses of `Formatter`, including `ByteCountFormatter`, `DateFormatter`, `DateComponentsFormatter`, `DateIntervalFormatter`, `MeasurementFormatter`, `NumberFormatter`, and `PersonNameComponentsFormatter`.

> **Tip**
>
> In Swift, you can use implementations of `FormatStyle` rather than `Formatter`. 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.

Subclassing Notes

`Formatter` is intended for subclassing. A custom formatter can restrict the input and enhance the display of data in novel ways. For example, you could have a custom formatter that ensures that serial numbers entered by a user conform to predefined formats. Before you decide to create

a custom formatter, make sure that you cannot configure the public subclasses to satisfy your requirements.

For instructions on how to create your own custom formatter, see Creating a Custom Formatter.

Topics

Getting Textual Representations of Object Values

`func string(for: Any?) -> String?`

The default implementation of this method raises an exception.

`func attributedString(for: Any, withDefaultAttributes: [NSAttributedString.Key : Any]?) -> NSAttributedString?`

The default implementation returns `nil` to indicate that the formatter object does not provide an attributed string.

`func editingString(for: Any) -> String?`

The default implementation of this method invokes `string(for:)`.

Getting Object Values for Textual Representations

`func getObjectValue(AutoreleasingUnsafeMutablePointer<AnyObject?>?, for: String, errorDescription: AutoreleasingUnsafeMutablePointer<NSString?>?) -> Bool`

The default implementation of this method raises an exception.

Validating Partial Strings

`func isPartialStringValid(String, newEditingString: AutoreleasingUnsafeMutablePointer<NSString?>?, errorDescription: AutoreleasingUnsafeMutablePointer<NSString?>?) -> Bool`

Returns a Boolean value that indicates whether a partial string is valid.

`func isPartialStringValid(AutoreleasingUnsafeMutablePointer<NSString>, proposedSelectedRange: NSRangePointer?, originalString: String, originalSelectedRange: NSRange, errorDescription: AutoreleasingUnsafeMutablePointer<NSString?>?) -> Bool`

This method should be implemented in subclasses that want to validate user changes to a string in a field, where the user changes are not necessarily at the end of the string, and

preserve the selection (or set a different one, such as selecting the erroneous part of the string the user has typed).

Constants

`enum Context`

The formatting context for a formatter.

`enum UnitStyle`

Specifies the width of the unit, determining the textual representation.

Relationships

Inherits From

`NSObject`

Inherited By

`ByteCountFormatter`
`DateComponentsFormatter`
`DateFormatter`
`DateIntervalFormatter`
`EnergyFormatter`
`ISO8601DateFormatter`
`LengthFormatter`
`ListFormatter`
`MassFormatter`
`MeasurementFormatter`
`NumberFormatter`
`PersonNameComponentsFormatter`
`RelativeDateTimeFormatter`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`

Hashable
NSCoding
NSCopying
NSObjectProtocol