

---

# NSFormatter Class Reference

[Cocoa](#) > [User Experience](#)



2007-07-09



Apple Inc.  
© 2007 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

Apple, the Apple logo, and Cocoa are trademarks of Apple Inc., registered in the United States and other countries.

iPhone is a trademark of Apple Inc.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR**

**CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

---

## **NSFormatter Class Reference 5**

---

Overview 5

    Subclassing Notes 5

Tasks 6

    Textual Representation of Cell Content 6

    Object Equivalent to Textual Representation 6

    Dynamic Cell Editing 6

Instance Methods 6

    attributedStringForObjectValue:withDefaultAttributes: 6

    editingStringForObjectValue: 7

    getObjectValue:forString:errorDescription: 7

    isPartialStringValid:newEditingString:errorDescription: 9

    isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange: errorDescription:  
    9

    stringForObjectValue: 10

---

## **Document Revision History 13**

---

## **Index 15**

---



# NSNumberFormatter Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSCoding NSCopying NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/Foundation.framework
<b>Availability</b>	Available in Mac OS X v10.0 and later.
<b>Companion guide</b>	Data Formatting Programming Guide for Cocoa
<b>Declared in</b>	NSNumberFormatter.h
<b>Related sample code</b>	bMoviePalette bMoviePaletteCocoa QTMetadataEditor QTSSConnectionMonitor QTSSInspector

## Overview

`NSNumberFormatter` is an abstract class that declares an interface for objects that create, interpret, and validate the textual representation of cell contents. The Foundation framework provides two concrete subclasses of `NSNumberFormatter` to generate these objects: `NSNumberFormatter` and `NSDateFormatter`.

## Subclassing Notes

---

`NSNumberFormatter` 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 `NSDateFormatter` and `NSNumberFormatter` to satisfy your requirements.

For instructions on how to create your own custom formatter, see [Creating a Custom Formatter](#).

## Tasks

### Textual Representation of Cell Content

- `stringForObjectValue:` (page 10)  
The default implementation of this method raises an exception.
- `attributedStringForObjectValue:withDefaultAttributes:` (page 6)  
The default implementation returns `nil` to indicate that the formatter object does not provide an attributed string.
- `editingStringForObjectValue:` (page 7)  
The default implementation of this method invokes `stringForObjectValue:` (page 10).

### Object Equivalent to Textual Representation

- `getObjectValue:forString:errorDescription:` (page 7)  
The default implementation of this method raises an exception.

### Dynamic Cell Editing

- `isPartialStringValid:newEditingString:errorDescription:` (page 9)  
Returns a Boolean value that indicates whether a partial string is valid.
- `isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 9)  
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).

## Instance Methods

### **attributedStringForObjectValue:withDefaultAttributes:**

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

```
(NSAttributedString *)attributedStringForObjectValue:(id)anObject
withDefaultAttributes:(NSDictionary *)attributes
```

#### Parameters

*anObject*

The object for which a textual representation is returned.

*attributes*

The default attributes to use for the returned attributed string.

**Return Value**

An attributed string that represents *anObject*.

**Discussion**

When implementing a subclass, return an `NSAttributedString` object if the string for display should have some attributes. For instance, you might want negative values in a financial application to appear in red text. Invoke your implementation of `stringForObjectValue:` (page 10) to get the non-attributed string, then create an `NSAttributedString` object with it (see `initWithString:`). Use the `attributes` default dictionary to reset the attributes of the string when a change in value warrants it (for example, a negative value becomes positive) For information on creating attributed strings, see *Attributed Strings Programming Guide*.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [editingStringForObjectValue:](#) (page 7)

**Declared In**

`NSFormatter.h`

**editingStringForObjectValue:**

The default implementation of this method invokes `stringForObjectValue:` (page 10).

```
- (NSString *)editingStringForObjectValue:(id)anObject
```

**Parameters**

*anObject*

The object for which to return an editing string.

**Return Value**

An `NSString` object that is used for editing the textual representation of *anObject*.

**Discussion**

When implementing a subclass, override this method only when the string that users see and the string that they edit are different. In your implementation, return an `NSString` object that is used for editing, following the logic recommended for implementing `stringForObjectValue:` (page 10). As an example, you would implement this method if you want the dollar signs in displayed strings removed for editing.

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [attributedStringForObjectValue:withDefaultAttributes:](#) (page 6)

**Declared In**

`NSFormatter.h`

**getObjectValue:forString:errorDescription:**

The default implementation of this method raises an exception.

- (BOOL)getObjectValue:(id \*)*anObject* forString:(NSString \*)*string*  
errorDescription:(NSString \*\*)*error*

### Parameters

*anObject*

If conversion is successful, upon return contains the object created from *string*.

*string*

The string to parse.

*error*

If non-*nil*, if there is a error during the conversion, upon return contains an NSString object that describes the problem.

### Return Value

YES if the conversion from string to cell content object was successful, otherwise NO.

### Discussion

When implementing a subclass, return by reference the object *anObject* after creating it from *string*. Return YES if the conversion is successful. If you return NO, also return by indirection (in *error*) a localized user-presentable NSString object that explains the reason why the conversion failed; the delegate (if any) of the NSControl object managing the cell can then respond to the failure in `control:didFailToFormatString:errorDescription:.` However, if *error* is *nil*, the sender is not interested in the error description, and you should not attempt to assign one.

The following example (which is paired with the example given in [stringForObjectValue: \(page 10\)](#)) converts a string representation of a dollar amount that includes the dollar sign; it uses an NSScanner instance to convert this amount to a float after stripping out the initial dollar sign.

```
- (BOOL)getObjectValue:(id *)obj forString:(NSString *)string
errorDescription:(NSString **)error
{
    float floatResult;
    NSScanner *scanner;
    BOOL returnValue = NO;

    scanner = [NSScanner scannerWithString: string];
    [scanner scanString:@"$" intoString: NULL]; //ignore return value
    if ([scanner scanFloat:&floatResult] && ([scanner isAtEnd])) {
        returnValue = YES;
        if (obj)
            *obj = [NSNumber numberWithFloat:floatResult];
    } else {
        if (error)
            *error = NSLocalizedString(@"Couldn't convert to float", @"Error
converting");
    }
    return returnValue;
}
```

### Availability

Available in Mac OS X v10.0 and later.

### See Also

- [stringForObjectValue: \(page 10\)](#)

### Declared In

NSFormatter.h



**isPartialStringValid:newEditingString:errorDescription:**

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

```
- (BOOL)isPartialStringValid:(NSString *)partialString newEditingString:(NSString **)newString errorDescription:(NSString **)error
```

**Parameters**

*partialString*

The text currently in a cell.

*newString*

If *partialString* needs to be modified, upon return contains the replacement string.

*error*

If non-*nil*, if validation fails contains an `NSString` object that describes the problem.

**Return Value**

YES if *partialString* is an acceptable value, otherwise NO.

**Discussion**

This method is invoked each time the user presses a key while the cell has the keyboard focus—it lets you verify and edit the cell text as the user types it.

In a subclass implementation, evaluate *partialString* according to the context, edit the text if necessary, and return by reference any edited string in *newString*. Return YES if *partialString* is acceptable and NO if *partialString* is unacceptable. If you return NO and *newString* is *nil*, the cell displays *partialString* minus the last character typed. If you return NO, you can also return by indirection an `NSString` object (in *error*) that explains the reason why the validation failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToValidatePartialString:errorDescription:.` The selection range will always be set to the end of the text if replacement occurs.

This method is a compatibility method. If a subclass overrides this method and does not override [isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription: \(page 9\)](#), this method will be called as before ([isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription: \(page 9\)](#) just calls this one by default).

**Availability**

Available in Mac OS X v10.0 and later.

**Declared In**

`NSFormatter.h`

**isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:**

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).

```
- (BOOL)isPartialStringValid:(NSString **)partialStringPtr
    proposedSelectedRange:(NSRangePointer)proposedSelRangePtr
    originalString:(NSString *)origString originalSelectedRange:(NSRange)origSelRange
    errorDescription:(NSString **)error
```

**Parameters**

*partialStringPtr*

The new string to validate.

*proposedSelRangePtr*

The selection range that will be used if the string is accepted or replaced.

*origString*

The original string, before the proposed change.

*origSelRange*

The selection range over which the change is to take place.

*error*

If non-*nil*, if validation fails contains an `NSString` object that describes the problem.

**Return Value**

YES if *partialStringPtr* is acceptable, otherwise NO.

**Discussion**

In a subclass implementation, evaluate *partialString* according to the context. Return YES if *partialStringPtr* is acceptable and NO if *partialStringPtr* is unacceptable. Assign a new string to *partialStringPtr* and a new range to *proposedSelRangePtr* and return NO if you want to replace the string and change the selection range. If you return NO, you can also return by indirection an `NSString` object (in *error*) that explains the reason why the validation failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToValidatePartialString:errorDescription:.`

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [isPartialStringValid:newEditingString:errorDescription:](#) (page 9)

**Declared In**

`NSFormatter.h`

**stringForObjectValue:**

The default implementation of this method raises an exception.

```
- (NSString *)stringForObjectValue:(id)anObject
```

**Parameters**

*anObject*

The object for which a textual representation is returned.

**Return Value**

An `NSString` object that textually represents *object* for display. Returns *nil* if *object* is not of the correct class.

**Discussion**

When implementing a subclass, return the `NSString` object that textually represents the cell's object for display and—if `editingStringValueForObjectValue:` (page 7) is unimplemented—for editing. First test the passed-in object to see if it's of the correct class. If it isn't, return `nil`; but if it is of the right class, return a properly formatted and, if necessary, localized string. (See the specification of the `NSString` class for formatting and localizing details.)

The following implementation (which is paired with the `getObjectValue:forString:errorDescription:` (page 7) example above) prefixes a two-digit float representation with a dollar sign:

```
- (NSString *)stringValueForObjectValue:(id)anObject
{
    if (![anObject isKindOfClass:[NSNumber class]]) {
        return nil;
    }
    return [NSString stringWithFormat:@"$%.2f", [anObject floatValue]];
}
```

**Availability**

Available in Mac OS X v10.0 and later.

**See Also**

- [attributedStringValueForObjectValue:withDefaultAttributes:](#) (page 6)
- [editingStringValueForObjectValue:](#) (page 7)
- [getObjectValue:forString:errorDescription:](#) (page 7)

**Declared In**

NSFormatter.h



# Document Revision History

---

This table describes the changes to *NSFormatter Class Reference*.

Date	Notes
2007-07-09	Corrected minor typographical error.
2006-05-23	First publication of this content as a separate document.
	First publication of this content as a separate document.

## REVISION HISTORY

### Document Revision History

# Index

---

## A

---

attributedStringForObjectValue:  
withDefaultAttributes: **instance method** [6](#)

## E

---

editingStringForObjectValue: **instance method** [7](#)

## G

---

getObjectValue:forString:errorDescription:  
**instance method** [7](#)

## I

---

isPartialStringValid:newEditingString:  
errorDescription: **instance method** [9](#)  
isPartialStringValid:proposedSelectedRange:  
originalString:originalSelectedRange:  
errorDescription: **instance method** [9](#)

## S

---

stringForObjectValue: **instance method** [10](#)