NSNumberFormatter

Inherits From:	NSFormatter : NSObject
Conforms To:	NSObject (NSObject) NSCoding NSCopying
Declared In:	Foundation/NSNumberFormatter.h

lass Description

Instances of NSNumberFormatter format the textual representation of cells that contain NSDecimalNumbers and convert textual representations of numeric values into NSDecimalNumbers. The representation encompasses integers, floats, and doubles; floats and doubles can be formatted to a specified decimal position. NSNumberFormatters can also impose ranges on the numeric values that cells can accept.

NSControl provides delegation methods that permit you to validate cell contents and to handle errors in formatting. See the specification of the NSFormatter class for details.

When a cell with a NSNumberFormatter is copied, the new cell retains the NSNumberFormatter object instead of copying it. You remove an NSNumberFormatter from a cell by specifying **nil** as the argument of NSCell's **setFormatter:** method.

Instances of NSNumberFormatter are mutable.

Creating an Instance of NSNumberFormatter

The easiest way to use NSNumberFormatter is to drag a formatter onto a control in Interface Builder. However, if you're not using Interface Builder to create your user interface or if you simply want more fine-grained control over an NSNumberFormatter (for example, to change the text attributes of the values displayed), you can create and manipulate instances of the class programmatically.

To create an NSNumberFormatter, allocate an instance of NSNumberFormatter and use one or more of NSNumberFormatter's "set format" methods to set its format. You then use NSCell's **setFormatter:** method to associate the NSNumberFormatter instance with a cell.

For example, the following code excerpt creates an instance of NSNumberFormatter, sets its formatting for positive, zero, and negative values, and applies it to the cell of an NSTextField using NSCell's **setFormatter:** method:

```
NSNumberFormatter *numberFormatter =
   [[[NSNumberFormatter alloc] init] autorelease];
[numberFormatter setFormat:@"$#,###.00;0.00;($#,##0.00)"];
```

```
[[textField cell] setFormatter:numberFormatter];
```

The value of a cell (NSCell) is represented by an object, typically an NSDecimalNumber object in this case. When this value needs to be displayed or edited, the cell passes its object to the NSNumberFormatter instance, which returns the formatted string. When the user enters a string, or when a string is programmatically written in a cell (using **setStringValue:**), the cell obtains the equivalent NSDecimalNumber object from the NSNumberFormatter.

The most common technique for assigning a format to an NSNumberFormatter object is to use the method **setFormat:**, as shown above. This method takes as an argument an NSString whose contents can be one of the following:

- @”*positiveFormat*”

  For example, @”$###,##0.00” (the syntax of format strings is discussed in the following section).

- @”*positiveFormat;negativeFormat*”

  For example, @”###,##0.00;(###,##0.00)”.

- @”*positiveFormat;zeroFormat;negativeFormat*”

  For example, @”$###,###.00;0.00;($###,##0.00)”. Note that zero formats are treated as string constants.

As implied in the preceding list, you’re only required to specify a format for positive values. If you don’t specify a format for negative and zero values, a default format based on the positive value format is used. For example, if your positive value format is “#,##0.00”, an input value of “0” will be displayed as “0.00”.

If you don’t specify a format for negative values, the format specified for positive values is used, preceded by a minus sign (-).

If you specify a separate format for negative values, its separators should be parallel to those specified in the positive format string. In NSNumberFormatter, separators are either enabled or disabled for all formats—both your negative and positive formats should therefore use the same approach.

As an alternative to using the **setFormat:** method, you can use the **setPositiveFormat:** and **setNegativeFormat:** methods.

Format String Syntax

Format strings can include the following types of characters:

- Numbers

  Format strings can include numeric characters. Wherever you include a number in a format string, the number is displayed unless an input character in the same relative position “overwrites” it. For example, suppose you have the positive format string @“9,990.00”, and the value 53.88 is entered into a cell to which the format has been applied. The cell would display the value as 9,953.88.

- Separators

  Format strings can include the period character (.) as a decimal separator, and comma character (,) as a thousand separator. If you want to use different characters as separators, you can set them using the **setDecimalSeparator:** and **setThousandSeparator:** methods. When you enable localization for an NSNumberFormatter object by using the method **setLocalizesFormat:**, separators are converted to characters appropriate to the environment in which the application is running.

- Placeholders

  You use the pound sign character (#) to represent numeric characters that will be input by the user. For example, suppose you have the positive format @”$#,##0.00”. If the characters 76329 were entered into a cell to which the format has been applied, they would be displayed as $76,329.00. Strictly speaking, however, you don’t need to use placeholders. The format strings @”,0.00”, @”#,#0.00, and @”#,##0.00” are functionally equivalent. In other words, including separator characters in a format string signals NSNumberFormatter to use the separators, regardless of whether you use (or where you put) placeholders. The placeholder character’s chief virtue lies in its ability to make format strings more human-readable, which is especially useful if you’re displaying formats in the user interface.

- Spaces

  To include a space in a format string, use the underscore character (_). This character inserts a space if no numeric character has been input to occupy that position.

- Currency

  The dollar sign character ($) is normally treated just like any other character that doesn’t play a special role in NSNumberFormatter. However, when you enable localization for an NSNumberFormatter object by using the method **setLocalizesFormat:**, the dollar sign character is converted to the currency symbol appropriate for the environment in which the application is running.

All other characters specified in a format string are displayed as typed. The following table shows examples of the how the value 1019.55 is displayed for different positive formats:

Format String	**Display**
@“#,##0.00”	1,019.55
@“$#,##0.00”	$1,019.55
@“___,__0.00”	1,019.55

Working with Values as Attributed Strings

In NSNumberFormatter, positive, negative, zero, **nil**, and “not a number” values are NSAttributedStrings. NSAttributedString objects manage character strings and associated sets of attributes (for example, font and kerning) that apply to individual characters or ranges of characters in the string. An association of characters and their attributes is called an *attributed string*. For more information on NSAttributedString, see the NSAttributedString class cluster specification in the *Foundation Kit Reference*, and the NSAttributedString Class Cluster Additions specification in the *Application Kit Reference*.

Because the values displayed by NSNumberFormatter are attributed strings, you can customize aspects of their appearance, such as their color and font. The NSNumberFormatter methods you use to do this are as follows:

– textAttributesForPositiveValues
– setTextAttributesForPositiveValues:
– textAttributesForNegativeValues
– setTextAttributesForNegativeValues:
– attributedStringForZero
– setAttributedStringForZero:
– allowsFloats
– setAttributedStringForNil:
– attributedStringForNotANumber
– setAttributedStringForNotANumber:

Using Separators

NSNumberFormatter supports two different kinds of separators: thousand and decimal. By default these separators are represented by the comma (,) and period (.) characters respectively, and by default they're disabled.

All of the following statements have the effect of enabling thousand separators:

```
// use setFormat:
[numberFormatter setFormat:@"#,###"];

// use setHasThousandSeparators:
[numberFormatter setHasThousandSeparators:YES];

// use setThousandSeparator:
[numberFormatter setThousandSeparator:@"_"];
```

If you use the statement `[numberFormatter setHasThousandSeparators:NO]`, it disables thousand separators, even if you've set them through another means.

Both of the following statements have the effect of enabling decimal separators:

```
// use setFormat:
[numberFormatter setFormat:@"0.00"];

// use setDecimalSeparator:
[numberFormatter setDecimalSeparator:@"-"];
```

When you enable or disable separators, it affects both positive and negative formats. Consequently, both formats must use the same separator scheme.

You can use the **thousandSeparator** and **decimalSeparator** methods to return an NSString containing the character the receiver uses to represent each separator. However, this shouldn't be taken as an indication of whether separators are enabled—even when separators are disabled, an NSNumberFormatter still knows the characters it uses to represent separators.

Separators must be single characters. If you specify multiple characters in the arguments to **setThousandSeparator:** and **setDecimalSeparator:**, only the first character is used.

You can't use the same character to represent thousand and decimal separators.

NSCell Methods for Number Formatting

NSCell provides methods that give you almost the same behavior as instances of NSNumberFormatter. Send **setEntryType:** to a cell to associate it with an NSNumberFormatter object; specify the numeric format with one of the constants listed below. The constant is equivalent to an NSNumberFormatter initialized with a certain range and a conversion specifier:

Constant	Range	Specifier
NSIntType	MININT, MAXINT	%d
NSPositiveIntType	1, MAXINT	%d
NSFloatType	-MAXFLOAT, MAXFLOAT	%g
NSPositiveFloatType	MINFLOAT, MAXFLOAT	%g
NSDoubleType	-MAXDOUBLE, MAXDOUBLE	%g
NSPositiveDoubleType	MINDOUBLE, MAXDOUBLE	%g

Send NSCell's **isEntryAcceptable:** to a cell to determine if it can accept a numeric type as indicated by one of the above constants. Send NSCell's **setFloatingPointFormat:left:right:** to specify the number of digits that appear to the left and right of the decimal point. By invoking this method you do not lose any range of values for floats or values set either through **setEntryType:** or by initializing an NSNumberFormatter directly.

Note: The NSNumberFormatter approach is recommended over the NSCell methods because it allows you greater freedom in specifying the representation of numbers. However, NSCell's **setEntryType:**, **isEntryAcceptable:**, and **setFloatingPointFormat:left:right:** are OpenStep-compliant, whereas NSNumberFormatter is an extension to OpenStep.

Method Types

Set formats	– negativeFormat
	– setNegativeFormat:
	– positiveFormat
	– setPositiveFormat:
	– format
	– setFormat:
Set characteristics for displaying values	– textAttributesForNegativeValues
	– setTextAttributesForNegativeValues:
	– textAttributesForPositiveValues
	– setTextAttributesForPositiveValues:
	– attributedStringForZero
	– setAttributedStringForZero:
	– attributedStringForNil
	– setAttributedStringForNil:
	– attributedStringForNotANumber
	– setAttributedStringForNotANumber:
Set separators	– hasThousandSeparators
	– setHasThousandSeparators:
	– thousandSeparator
	– setThousandSeparator:
	– decimalSeparator
	– setDecimalSeparator:
Enable localization	– localizesFormat
	– setLocalizesFormat:
Set float behavior	– allowsFloats
	– setAllowsFloats:
Set rounding behavior	– roundingBehavior
	– setRoundingBehavior:
Set minimum and maximum values	– minimum
	– setMinimum:
	– maximum
	– setMaximum:

nstance Methods

allowsFloats

– (BOOL)**allowsFloats**

Returns YES if the receiver allows as input floating point values (that is, values that include the period character (.)), NO otherwise. When this is set to NO, only integer values can be provided as input. The default is YES.

See also: – **setAllowsFloats:**

attributedStringForNil

– (NSAttributedString *)**attributedStringForNil**

Returns the NSAttributedString used to display **nil** values. By default **nil** values are displayed as an empty string.

See also: – **allowsFloats**

attributedStringForNotANumber

– (NSAttributedString *)**attributedStringForNotANumber**

Returns the NSAttributedString used to display “not a number” values. By default “not a number” values are displayed as the string “NaN”.

See also: – **attributedStringForNotANumber**

attributedStringForZero

– (NSAttributedString *)**attributedStringForZero**

Returns the NSAttributedString used to display zero values. By default zero values are displayed according to the format specified for positive values; for more discussion of this subject see the section ““Creating an Instance of NSNumberFormatter”” in the Class Description.

See also: – **setAttributedStringForZero:**

decimalSeparator

– (NSString *)**decimalSeparator**

Returns an NSString containing the character the receiver uses to represent decimal separators. By default this is the period character (.). Note that the return value doesn't indicate whether decimal separators are enabled.

See also: – **setDecimalSeparator:**

format

– (NSString *)**format**

Returns an NSString containing the format being used by the receiver.

See also: – **setFormat:**

hasThousandSeparators

– (BOOL)**hasThousandSeparators**

Returns YES to indicate that the receiver's format includes thousand separators, NO otherwise. The default is NO.

See also: – **setHasThousandSeparators:**

localizesFormat

– (BOOL)**localizesFormat**

Returns YES to indicate that the receiver localizes formats, NO otherwise. The default is NO.

See also: – **setLocalizesFormat:**

maximum

– (NSDecimalNumber *)**maximum**

Returns the highest number that is allowed as input by the receiver.

See also: – **setMaximum:**

minimum

– (NSDecimalNumber *)**minimum**

Returns the lowest number that is allowed as input by the receiver.

See also: – **setMinimum:**

negativeFormat

– (NSString *)**negativeFormat**

Returns an NSString containing the format used by the receiver to display negative numbers.

See also: – **setNegativeFormat:**, – **setFormat:**

positiveFormat

– (NSString *)**positiveFormat**

Returns an NSString containing the format used by the receiver to display positive numbers.

See also: – **setPositiveFormat:**, – **setFormat:**

roundingBehavior

– (NSDecimalNumberHandler *)**roundingBehavior**

Returns an NSDecimalNumberHandler object to indicate the rounding behavior used by the receiver.

See also: – **setRoundingBehavior:**

setAllowsFloats:

– (void)**setAllowsFloats:**(BOOL)*flag*

Sets according to *flag* whether the receiver allows as input floating point values (that is, values that include the period character (.)). By default, floating point values are allowed as input.

See also: – **allowsFloats**

setAttributedStringForNil:

– (void)**setAttributedStringForNil:**(NSAttributedString *)*newAttributedString*

Sets to *newAttributedString* the NSAttributedString the receiver uses to display **nil** values.

See also: – **allowsFloats**

setAttributedStringForNotANumber:

– (void)**setAttributedStringForNotANumber:**(NSAttributedString *)*newAttributedString*

Sets to *newAttributedString* the NSAttributedString the receiver uses to display “not a number” values.

See also: – **attributedStringForNotANumber**

setAttributedStringForZero:

– (void)**setAttributedStringForZero:**(NSAttributedString *)*newAttributedString*

Sets to *newAttributedString* the NSAttributedString the receiver uses to display zero values.

See also: – **attributedStringForZero**

setDecimalSeparator:

– (void)**setDecimalSeparator:**(NSString *)*newSeparator*

Sets to *newSeparator* the character the receiver uses as a decimal separator. If *newSeparator* contains multiple characters, only the first one is used. If you don’t have decimal separators enabled through another means (such as **setFormat:**), using this method enables them.

See also: – **decimalSeparator**

setFormat:

– (void)**setFormat:**(NSString *)*aFormat*

Sets the receiver’s format to the string *aFormat*. *aFormat* can consist of one, two or three parts separated by ‘;’. The first part of the string represents the positive format, the second part of the string represents the zero value, and the last part of the string represents the negative format. If the string just has two parts, the first one becomes the positive format, and the second one becomes the negative format. If the string just has one part, it becomes the positive format, and default formats are provided for zero and negative values based on the positive format. For more discussion of this subject, see the section ““Creating an Instance of NSNumberFormatter”” in the Class Description.

For example, the following code excerpt shows the three different approaches for setting an NSNumberFormatter object's format using **setFormat:**:

```
NSNumberFormatter *numberFormatter =
   [[[NSNumberFormatter alloc] init] autorelease];

// specify just positive format
[numberFormatter setFormat:@"$#,##0.00"];

// specify positive and negative formats
[numberFormatter setFormat:@"$#,##0.00;($#,##0.00)"];

// specify positive, zero, and negative formats
[numberFormatter setFormat:@"$#,###.00;0.00;($#,##0.00)"];
```

See also: – **format**

setHasThousandSeparators:

– (void)**setHasThousandSeparators:**(BOOL)*flag*

Sets according to *flag* whether the receiver uses thousand separators. When *flag* is NO, thousand separators are disabled for both positive and negative formats (even if you've set them through another means, such as **setFormat:**). When *flag* is YES, thousand separators are used. In addition to using this method to add thousand separators to your format, you can also use it to disable thousand separators if you've set them using another method. The default is NO (though you in effect change this setting to YES when you set thousand separators through any means, such as **setFormat:**).

See also: – **hasThousandSeparators**

setLocalizesFormat:

– (void)**setLocalizesFormat:**(BOOL)*flag*

Sets according to *flag* whether the dollar sign character ($), decimal separator character (.), and thousand separator character (,) are converted to appropriately localized characters as specified by the user's localization preference. While this feature may be useful in certain types of applications, it's probably more likely that you would tie a particular application to a particular currency (that is, that you would "hard-code" the currency symbol and separators instead of having them dynamically change based on the user's configuration). The reason for this, of course, is that NSNumberFormatter doesn't perform currency conversions, it just formats numeric data. You wouldn't want one user interpreting the value "56324" as US currency and another user who's accessing the same data interpreting it as Japanese currency, simply based on each user's localization preferences.

See also: – **localizesFormat**

setMaximum:

– (void)**setMaximum:**(NSDecimalNumber *)*aMaximum*

Sets to *aMaximum* the highest number the receiver allows as input.

See also: – **maximum**

setMinimum:

– (void)**setMinimum:**(NSDecimalNumber *)*aMinimum*

Sets to *aMinimum* the lowest number the receiver allows as input.

See also: – **setMinimum:**

setNegativeFormat:

– (void)**setNegativeFormat:**(NSString *)*aFormat*

Sets to *aFormat* the format the receiver uses to display negative values.

See also: – **negativeFormat**, – **setFormat:**

setPositiveFormat:

– (void)**setPositiveFormat:**(NSString *)*aFormat*

Sets to *aFormat* the format the receiver uses to display positive values.

See also: – **positiveFormat**, – **setFormat:**

setRoundingBehavior:

– (void)**setRoundingBehavior:**(NSDecimalNumberHandler *)*newRoundingBehavior*

Sets to *newRoundingBehavior* the rounding behavior used by the receiver.

See also: – **roundingBehavior**

setTextAttributesForNegativeValues:

– (void)**setTextAttributesForNegativeValues:**(NSDictionary *)*newAttributes*

Sets to *newAttributes* the text attributes to be used in displaying negative values. For example, this code excerpt causes negative values to be displayed in red:

```
NSNumberFormatter *numberFormatter =
   [[[NSNumberFormatter alloc] init] autorelease];
NSMutableDictionary *newAttrs = [NSMutableDictionary dictionary];

[numberFormatter setFormat:@"$#,##0.00;($#,##0.00)"];
[newAttrs setObject:[NSColor redColor] forKey:@"NSColor"];
[numberFormatter setTextAttributesForNegativeValues:newAttrs];
[[textField cell] setFormatter:numberFormatter];
```

An even simpler way to cause negative values to be displayed in red is to include the constant `[Red]` in your format string, for example:

```
[numberFormatter setFormat:@"$#,##0.00;[Red]($#,##0.00)"];
```

Note: When you set a value's text attributes to use color, the color only appears when the value's cell doesn't have input focus. When the cell has input focus, the value is displayed in standard black.

See also: – **textAttributesForNegativeValues**

setTextAttributesForPositiveValues:

– (void)**setTextAttributesForPositiveValues:**(NSDictionary *)*newAttributes*

Sets to *newAttributes* the text attributes to be used in displaying positive values.

See also: – **textAttributesForPositiveValues**

setThousandSeparator:

– (void)**setThousandSeparator:**(NSString *)*newSeparator*

Sets to *newSeparator* the character the receiver uses as a thousand separator. If *newSeparator* contains multiple characters, only the first one is used. If you don't have thousand separators enabled through any other means (such as **setFormat:**), using this method enables them.

See also: – **thousandSeparator**

textAttributesForNegativeValues

– (NSDictionary *)**textAttributesForNegativeValues**

Returns an NSDictionary containing the text attributes that have been set for negative values.

See also: – **setTextAttributesForNegativeValues:**

textAttributesForPositiveValues

– (NSDictionary *)**textAttributesForPositiveValues**

Returns an NSDictionary containing the text attributes that have been set for positive values.

See also: – **setTextAttributesForPositiveValues:**

thousandSeparator

– (NSString *)**thousandSeparator**

Returns an NSString containing the character the receiver uses to represent thousand separators. By default this is the comma character (,). Note that the return value doesn't indicate whether thousand separators are enabled.

See also: – **setThousandSeparator:**