NSFormatter Class Reference

Cocoa > User Experience

2007-07-09

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

Simultaneously published in the United States and Canada.

Contents

NSFormatter Class Reference

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

Overview

`NSFormatter` 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 `NSFormatter` to generate these objects: `NSNumberFormatter` and `NSDateFormatter`.

Subclassing Notes

`NSFormatter` 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 `editingStringForObjectValue:` (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 *)stringForObjectValue:(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
- `attributedStringForObjectValue:withDefaultAttributes:` (page 6)
- `editingStringForObjectValue:` (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.

Document Revision History

Index