NSFormatter

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

Class Description

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

Cells, which are instances of NSCell and its subclasses, can have any arbitrary object as their content. However, when cells are to be displayed or edited, they must convert this object to an NSString. If no formatting object is associated with a cell, the cell displays its content by invoking the localized **description** method of the object it contains. But if the cell has a formatting object, the cell invokes this object's **stringForObjectValue:** method to obtain the correctly formatted string. Conversely, when the user enters text into a cell, the cell needs to convert the text to the underlying object; formatting objects handle this conversion as well.

To use a formatting object, you must create an instance of NSNumberFormatter, NSDateFormatter, or a custom NSFormatter subclass and associate the object with a cell. The cell invokes the formatting behavior of this instance every time it needs to display its object or have it edited, and every time it needs to convert a textual representation to its object. See the class description of NSDateFormatter for the details of using formatting objects.

Instances of NSFormatter subclasses are immutable. In addition, when a cell with a formatter object is copied, the new cell retains the formatter object instead of copying it.

Note: NSCell provides two methods that operate almost the same as instances of NSNumberFormatter. One method, **setEntryType:**, takes a constant that specifies a typical numeric format (integer, float, positive float, double, and so on). With **isEntryAcceptable:**, you can ask a cell for the type of value it expects. Another method, **setFloatingPointFormat:left:right:**, allows you to specify the digits that appear to the left and right of the decimal point. See the NSCell specification for further details.

Delegation Methods for Error Handling

NSControl has delegation methods for handling errors returned in implementations of NSFormatter's **objectValue:forString:errorDescription:** and

isPartialStringValid:newEditingString:errorDescription:. These delegation methods are, respectively, **control:didFailToFormatString:errorDescription:** and **control:didFailToValidatePartialString:errorDescription:**.

Making a Subclass of NSFormatter

There are many possibilities for custom subclasses of NSFormatter. You might find use for a custom formatter of telephone numbers, or a custom formatter of part numbers.

To subclass NSFormatter, you must, at the least, override the two primitive methods s**tringForObjectValue:** and **getObjectValue:forString:errorDescription:**. In the first method you convert the cell's object to a string representation; in the second method you convert the string to the object associated with the cell.

Implement **attributedStringForObjectValue:withDefaultAttributes:** in addition to **stringForObjectValue:** when the display string has attributes associated with it. For example, if you want negative financial amounts to appear in red, you would return a string with an attribute of red text. In **attributedStringForObjectValue:withDefaultAttributes:** get the non-attributed NSString by invoking **stringForObjectValue:** and then apply the proper attributes to that NSString.

If the string for editing is different than the string for display—for example, the display version of a currency field should show a dollar sign but the editing version shouldn't—implement **editingStringForObjectValue:** in addition to **stringForObjectValue:**.

The method **isPartialStringValid:newEditingString:errorDescription:** allows you to edit the textual contents of a cell at each key press or to prevent entry of invalid characters. You might apply this on-the-fly editing to things like telephone numbers or social security numbers; the person entering data only needs to enter the number since the formatter automatically inserts the separator characters.

Method Types

Textual representation of cell content
- stringForObjectValue:
- attributedStringForObjectValue:withDefaultAttributes:
- editingStringForObjectValue:

Object equivalent to textual representation
- getObjectValue:forString:errorDescription:

Dynamic cell editing
- isPartialStringValid:newEditingString:errorDescription:

Instance Methods

attributedStringForObjectValue:withDefaultAttributes:

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

The default implementation returns **nil** to indicate that the formatter object does not provide an attributed string. In a subclass implementation, return an NSAttributedString 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:** to get the non-attributed string. Then create an NSAttributedString with it. The default attributes for text in the cell is passed in with *attributes*; use this NSDictionary 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 the specification for the NSAttributedString class.

See also: – **editingStringForObjectValue:**

editingStringForObjectValue:

– (NSString *)**editingStringForObjectValue:**(id)*anObject*

The default implementation of this method invokes **stringForObjectValue:**. 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 that is used for editing, following the logic recommended for implementing **stringForObjectValue:** (see below). As an example, you would implement this method if you want the dollar signs in displayed strings removed for editing.

See also: – **attributedStringValueForObject:**

getObjectValue:forString:errorDescription:

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

The default implementation of this method raises an exception. In your subclass implementation, return by reference the object *anObject* after creating it from the *string* passed in. Return YES if the conversion from string to cell-content object was successful and NO if any error prevented the conversion. If you return NO, also return by indirection a localized user-presentable NSString (in *error*) that explains the reason why the conversion failed; the delegate (if any) of the NSControl managing the cell can then respond to the failure in **control:didFailToFormatString:errorDescription:**.

The following implementation example (which is paired with the **stringForObjectValue:** example below) converts an NSString representation of a dollar amount that includes the dollar sign; it uses an NSScanner 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 retval = NO;
    NSString *err = nil;

    scanner = [NSScanner scannerWithString:string];
    if ([string hasPrefix:@"$"]) [scanner setScanLocation:1];
    if ([scanner scanFloat:&floatResult]
      && ([scanner scanLocation] == [string length] )) {
       if (obj) {
          *obj = [NSNumber numberWithFloat:floatResult];
          {
             retval = YES;
          }
       } else {
          err = NSLocalizedString(@"Couldn't convert to float");
       }
    }
    if (error) {
        *error = err;
    }
    return retval;
}
```

See also: – **stringForObjectValue:**

isPartialStringValid:newEditingString:errorDescription:

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

Since this method is invoked at each key press in the cell, it permits editing or evaluation of cell text as it is typed. The text as currently typed (*partialString*) is passed in. Evaluate this text according to the context, edit the text if necessary, and return by reference any edited NSString in *newString*. Return YES if *partialString* is acceptable and NO if *partialString* is unacceptable. If you return NO and *newString* is **nil**, *partialString* minus the last character typed is displayed. If you return NO, you can also return by indirection an NSString (in *error*) that explains the reason why the validation failed; the delegate (if any) of the NSControl managing the cell can then respond to the failure in **control:didFailToValidatePartialString:errorDescription:**.

stringForObjectValue:

– (NSString *)**stringForObjectValue:**(id)*anObject*

The default implementation of this method raises an exception. When subclassing, return the NSString that textually represents the cell's object for display and—if **editingStringForObjectValue:** 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:** 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]];
}
```

See also: – **attributedStringForObjectValue:withDefaultAttributes:**, – **editingStringForObjectValue:**, – **getObjectValue:forString:errorDescription:**