NSTextStorage

Inherits From:	NSMutableAttributedString : NSAttributedString : NSObject
Conforms To:	NSCopying NSMutableCopying NSObject (NSObject)
Declared In:	AppKit/NSTextStorage.h

Class Description

NSTextStorage is a semi-concrete subclass of NSMutableAttributedString that manages a set of client NSLayoutManagers, notifying of them of any changes to its characters or attributes so that they can re-lay and redisplay the text as needed. NSTextStorage defines the fundamental storage mechanism of NeXT's extended text-handling system.

Like an abstract class of a class cluster, allocating and initializing an NSTextStorage actually produces an instance of a private subclass. You can use any of NSAttributedString and NSMutableAttributedString's initialization methods to create an NSTextStorage object. Following this, you add NSLayoutManagers to it using **addLayoutManager:**.

The behavior of an NSTextStorage object is best illustrated by following the methods it invokes while being changed. There are three stages to editing a text storage object programmatically. The first stage is to send it a **beginEditing** message to announce a group of changes. In the second stage, you send it some editing messages, such as **deleteCharactersInRange:** and **addAttributes:range:**, to effect the changes in characters or attributes. Each time you send such a method, the text storage object invokes **edited:range:changeInLength:** to record the range of its characters affected since it received the **beginEditing** message. For the third stage, when you're done changing the text storage object, you send it an **endEditing** message. This causes it to invoke its own **processEditing** method, fixing attributes within the recorded range of changed characters. After fixing its attributes, the text storage object sends a message to each NSLayoutManager indicating the range in the text storage object that has changed, along with the nature of those changes. The NSLayoutManagers in turn use this information to re-lay their glyphs and redisplay if necessary. NSTextStorage also keeps a delegate and sends it messages before and after processing edits.

Creating a Subclass of NSTextStorage

As indicated above, NSTextStorage isn't a fully concrete class. It defines the storage for its NSLayoutManagers and implements all of the methods described in this specification, but doesn't provide the primitive attributed string methods to subclasses. A subclass must define the storage for its attributed string, typically as an instance variable of type NSMutableAttributedString, override **init** and define its own

initialization methods, and implement the primitive methods of both NSAttributedString and NSMutableAttributedString. For the record, these methods are:

– string
– attributesAtIndex:effectiveRange:
– replaceCharactersInRange:withString:
– setAttributes:range:

Beyond these requirements, if a subclass overrides or adds any methods that change its characters or attributes directly (not using the primitive methods or making extra changes after invoking the primitives), those methods must invoke **edited:range:changeInLength:** after performing the change in order to keep the change-tracking information up to date. See the description of this method for more information.

Method Types

Managing NSLayoutManagers	– addLayoutManager: – removeLayoutManager: – layoutManagers
Handling text edited messages	– edited:range:changeInLength: – endEditing – processEditing
Determining the nature of changes	– editedMask
Determining the extent of changes	– editedRange – changeInLength
Setting the delegate	– setDelegate: – delegate

Instance Methods

addLayoutManager:

– (void)**addLayoutManager:**(NSLayoutManager *)*aLayoutManager*

Adds *aLayoutManager* to the receiver's set of NSLayoutManagers.

See also: – **removeLayoutManager:**, – **layoutManagers**

changeInLength

– (int)**changeInLength**

Returns the difference between the current length of the edited range and its length before editing began (that is, before the receiver was sent the first **beginEditing** message or a single **edited:range:changeInLength:** message). This difference is accumulated with each invocation of **edited:range:changeInLength:**, until a final **endEditing** message processes the changes.

The receiver's delegate and layout managers can use this information to determine the nature of edits in their respective notification methods.

See also: – **editedRange**, – **editedMask**

delegate

– (id)**delegate**

Returns the receiver's delegate.

See also: – **setDelegate:**

edited:range:changeInLength:

– (void)**edited:**(unsigned)*mask*
range:(NSRange)*oldRange*
changeInLength:(int)*lengthChange*

Tracks changes made to the receiver, allowing the NSTextStorage to record the full extent of changes made between a pair of **beginEditing** and **endEditing** messages. If invoked outside of such a pair, this method immediately invokes **processEditing**. NSTextStorage invokes this method automatically each time it makes a change to its attributed string. Subclasses that override or add methods that alter their attributed strings directly should invoke this method after making those changes. The information accumulated with this method is then used in an invocation of **processEditing** to report the affected portion of the receiver.

mask specifies the nature of the changes. Its value is made by combining these options with the C bitwise OR operator:

Option	Meaning
NSTextStorageEditedAttributes	Attributes were added, removed, or changed.
NSTextStorageEditedCharacters	Characters were added, removed, or replaced.

oldRange indicates the extent of characters affected before the change took place. If the NSTextStorageEditedCharacters bit of *mask* is set, *lengthChange* gives the number of characters added to or removed from *oldRange* (otherwise its value is irrelevant). For example, when replacing "The" with "Several" in the string "The files couldn't be saved", *oldRange* is {0, 3} and *lengthChange* is 4.

Note: The methods for querying changes, **editedRange** and **changeInLength**, indicate the extent of characters affected *after* the change. This method expects the characters before the change because that information is readily available as the argument to whatever method performs the change (such as **replaceCharactersInRange:withString:**).

editedMask

– (unsigned int)**editedMask**

Returns the kinds of edits pending for the receiver, as a mask containing either or both of NSTextStorageEditedAttributes and NSTextStorageEditedCharacters. Use the C bitwise AND operator to test the mask; testing for equality will fail if additional mask flags are added later. The receiver's delegate and layout managers can use this information to determine the nature of edits in their respective notification methods.

See also: – **editedRange**, – **changeInLength**

editedRange

– (NSRange)**editedRange**

Returns the range of the receiver to which pending changes have been made, whether of characters or of attributes. The receiver's delegate and layout managers can use this information to determine the nature of edits in their respective notification methods.

See also: – **changeInLength**, – **editedMask**

endEditing

– (void)**endEditing**

Clears the last recorded invocation of **beginEditing**, and if there are no more, invokes **processEditing** to clean up after changes and notify the delegate and layout managers of the edits.

layoutManagers

– (NSArray *)**layoutManagers**

Returns the receiver's NSLayoutManagers.

See also: – **addLayoutManager:**, – **removeLayoutManager:**

processEditing

– (void)**processEditing**

Cleans up changes made to the receiver and notifies its delegate and layout managers of changes. This method is automatically invoked in response to an **endEditing** or **edited:range:changeInLength:** message. You should never need to invoke it directly.

This method begins by posting an NSTextStorageWillProcessEditingNotification to the default notification center (which results in the delegate receiving a **textStorageWillProcessEditing:** message). It then invokes the inherited **fixAttributesAfterEditingRange:** method to fix up attributes after a batch of editing changes. After this, it posts an NSTextStorageDidProcessEditingNotification to the default notification center (which results in the delegate receiving a **textStorageDidProcessEditing:** message). Finally, it sends a **textStorage:edited:range:changeInLength:invalidatedRange:** message to each of the receiver's NSLayoutManagers using the argument values provided.

removeLayoutManager:

– (void)**removeLayoutManager:**(NSLayoutManager *)*aLayoutManager*

Removes *aLayoutManager* from the receiver's set of NSLayoutManagers.

See also: – **addLayoutManager:**, – **layoutManagers**

setDelegate:

– (void)**setDelegate:**(id)*anObject*

Sets the receiver's delegate to *anObject*.

See also: – **delegate**

Methods Implemented By the Delegate

textStorageDidProcessEditing:

– (void)**textStorageDidProcessEditing:**(NSNotification *)*aNotification*

Informs the delegate that an NSTextStorage object has finished processing edits. The text storage object is available by sending **object** to *aNotification*, which is always an NSTextStorageDidProcessEditingNotification. The delegate can use this notification to verify the final state of the text storage object; it can't change the text storage object's characters without leaving it in an inconsistent state, but if necessary it can change attributes. Note that even in this case it's possible to put a text storage object into an inconsistent state—for example by changing the font of a range to one that doesn't support the characters in that range (such as using a Latin font for Kanji text).

textStorageWillProcessEditing:

– (void)**textStorageWillProcessEditing:**(NSNotification *)*aNotification*

Informs the delegate that an NSTextStorage object is about to process edits. The text storage object is available by sending **object** to *aNotification*, which is always an NSTextStorageWillProcessEditingNotification. The delegate can use this notification to verify the changed state of the text storage object, and to make changes to the text storage object's characters or attributes to enforce whatever constraints it establishes (which doesn't result in this message being sent again, however). For example, a code editor application might add a delegate that checks after edits to make sure that all programming language keywords are set in boldface.

Notifications

NSTextStorageDidProcessEditingNotification

Posted after the NSTextStorage finishes processing edits in **processEditing**. Observers other than the delegate shouldn't make further changes to the NSTextStorage. The notification contains:

Notification Object	The NSTextStorage that processed edits.
Userinfo	None

NSTextStorageWillProcessEditingNotification

Posted before the NSTextStorage finishes processing edits in **processEditing**. Observers other than the delegate shouldn't make further changes to the NSTextStorage. The notification contains:

Notification Object	The NSTextStorage about to process edits.
Userinfo	None