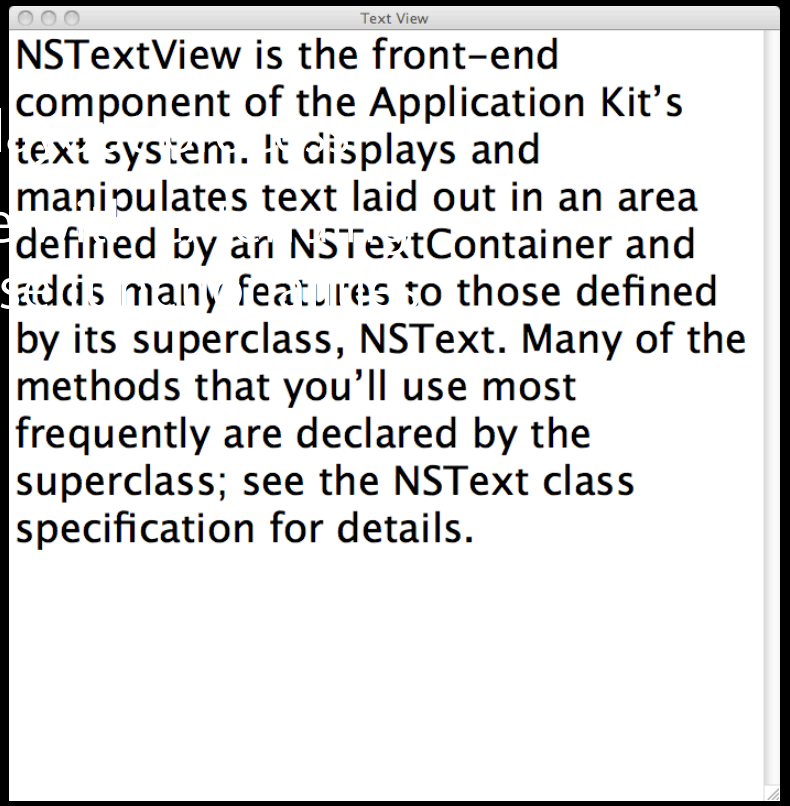
# Advanced Cocoa Text Tips and Tricks

**Aki Inoue**
Cocoa Engineer

# Introduction

Only on Mac OS

- Diving deeper
- Understanding the l
- Getting comfortable
  and customizing bas



# Prerequisite Download Required for This Session

## Advanced Cocoa Text Tips and Tricks

- To be ready for this session:
  - Open "WWDC_2010_114.xcodeproj"

**Materials available at:**
http://developer.apple.com/wwdc/attendee/

# What You'll Learn

- Multiple text view linking
- Line numbering and ruler view customization
- Line folding
- Providing inline UI widgets for user interaction
- Text animation

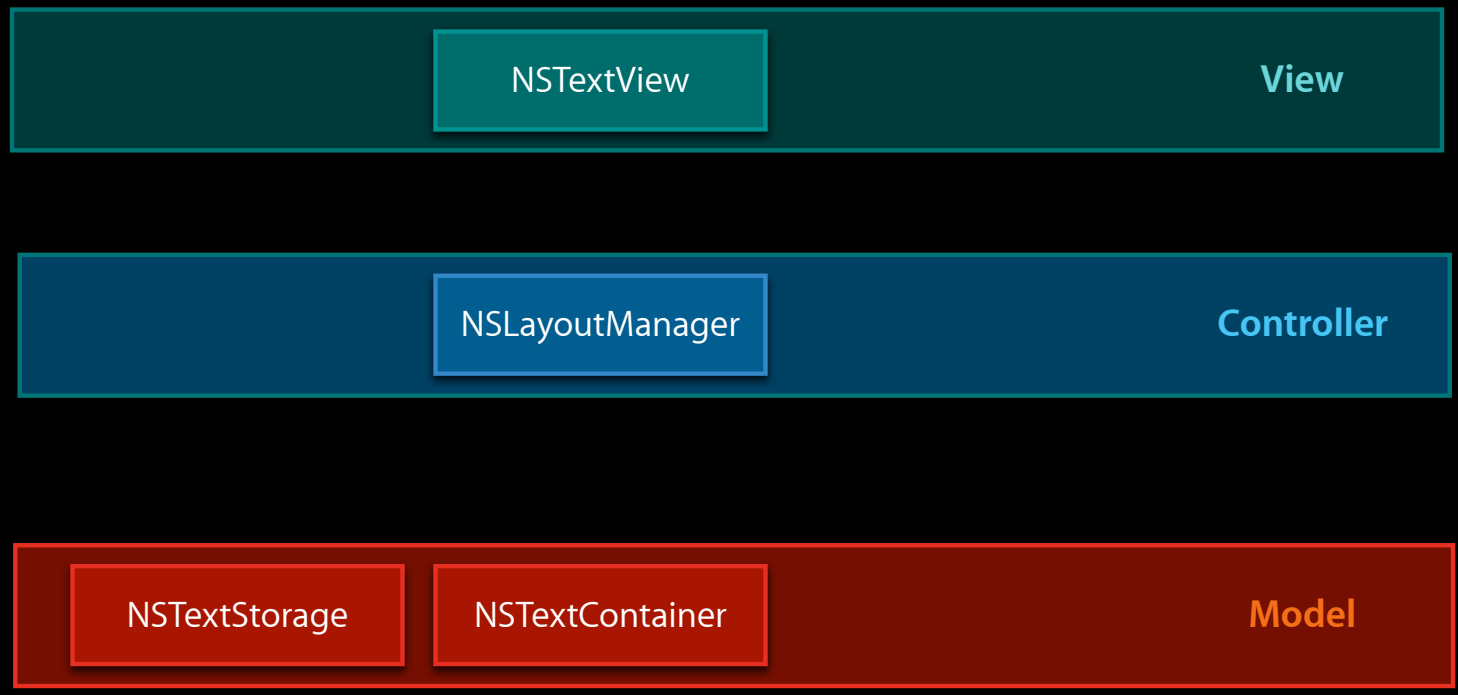
## Features found in advanced text editors

# Cocoa Text System

- Easy to access
- Designed to scale
- Advanced typography built in
- MVC design pattern
- Many customization points

# Cocoa Text System

## The Cocoa Text System primary classes



# Customizing with Primary Text Classes

- Flexible MVC configuration
  - Multipage, multipane, multicolumn, multiframe, etc.

# Customizing with Primary Text Classes

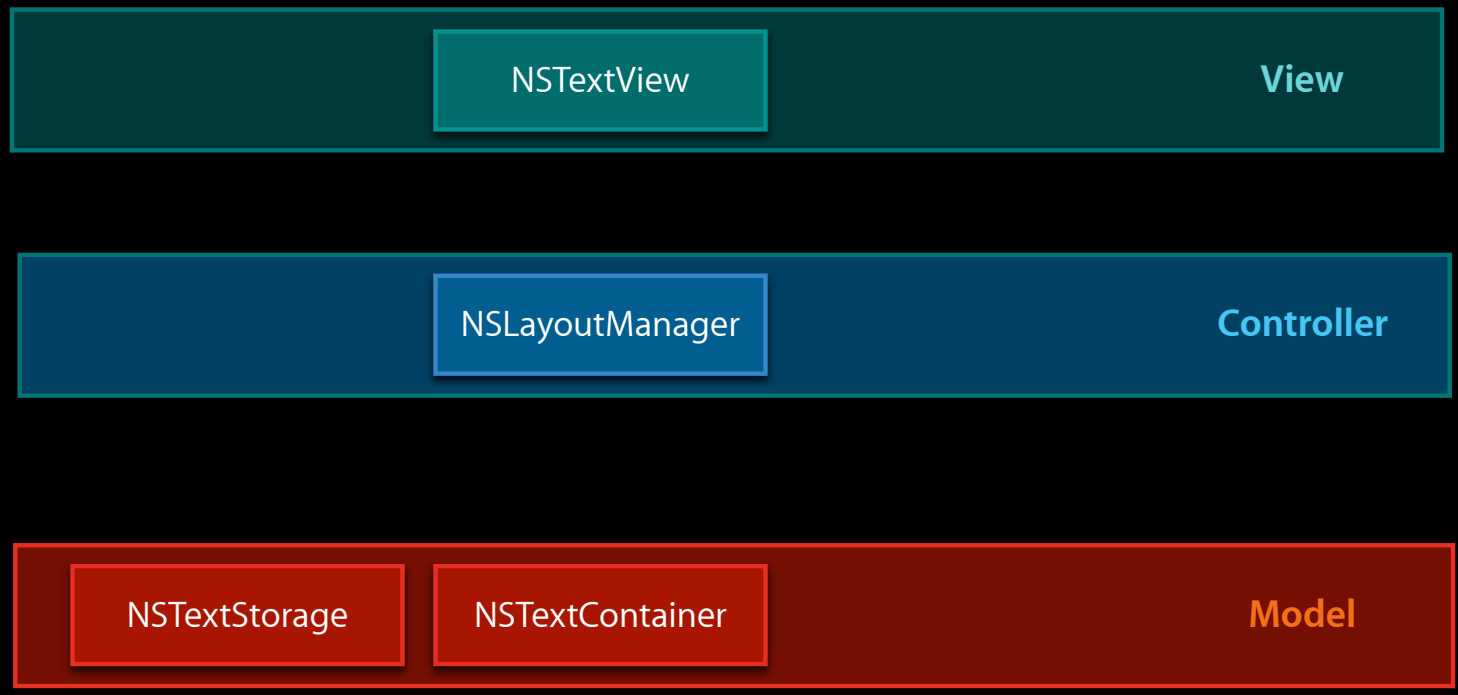
- Flexible MVC configuration
- Rich delegation interface points
  - NSTextDelegate, NSTextViewDelegate, NSLayoutManagerDelegate, etc.
  - Editing validation, mouse handling, pasteboard interface overrides, selection validation, typing attributes validation, key command overrides, context menu overrides…

# Customizing with Primary Text Classes

- Flexible MVC configuration
- Rich delegation interface points
- Subclassing
  - Custom text container shape
  - Custom text attributes

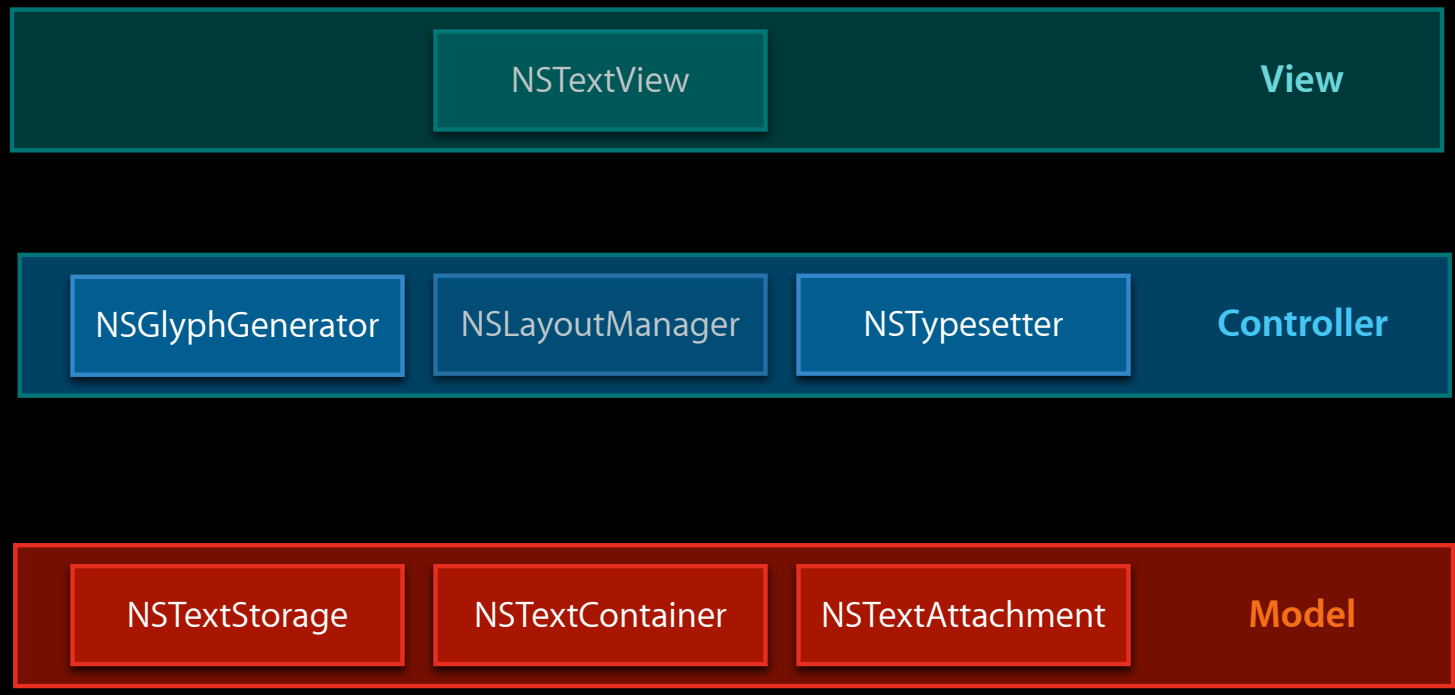
# Cocoa Text System

## The Cocoa Text System primary classes



# Cocoa Text System

## The Cocoa Text System helper classes



# Advanced Cocoa Text Tips and Tricks

**Dan Schimpf**
Cocoa Engineer

# NSTextStorage

- Model of the Cocoa text system
- Subclass of NSMutableAttributedString
- Can be reused across many views

# NSLayoutManager

- Lays out text in NSTextStorage
- Uses instance of NSTypesetter to lay out text
- Arranges glyphs in regions described by NSTextContainer

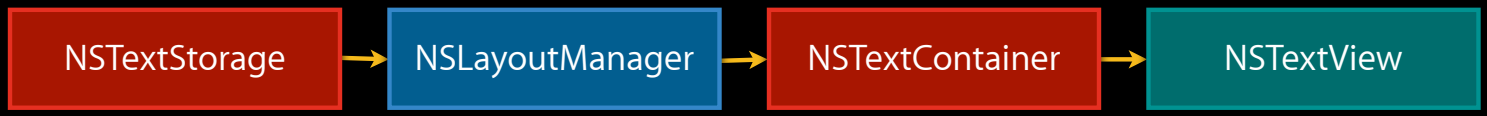
# NSTextContainer

- Defines a region of text
- Rectangular by default, but can be customized
- Hit testing

# NSTextView

- Subclass of NSView
- Uses NSLayoutManager to draw text in drawRect:
- Edits text in NSTextStorage
- Resizes NSTextContainer as the view resizes

# Basic Setup


NSTextStorage
NSLayoutManager
NSTextContainer
NSTextView

# Tip: How to Synchronize Multiple Text Views

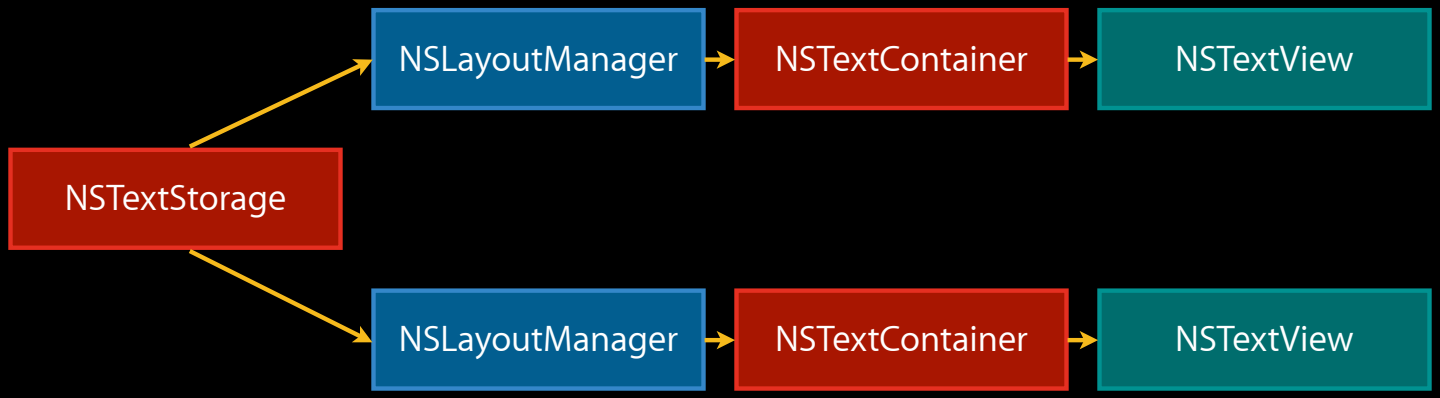
# Reusing Text Storages

- Tell all layout managers to use the same text storage

```
- (void)replaceTextStorage:(NSTextStorage *)aTextStorage;
```

- Can be done using text views configured in Interface Builder

# Multiple Text Views



# Demo

Synchronized multiple text views

# Tip: How to Add Line Numbers

# Adding Line Numbers to NSTextView

- Text views are normally inside an NSScrollView
- NSScrollView can have custom rulers
- Custom ruler view to show line numbers
- Driven by changes to text storage
- Uses information from layout manager to draw

NSRulerView

```
NSTextStorageDidProcessEditingNotification
```

```
- (NSRectArray)rectArrayForCharacterRange:(NSRange)charRange
  withinSelectedCharacterRange:(NSRange)selCharRange
  inTextContainer:(NSTextContainer *)container
  rectCount:(NSUInteger *)rectCoun
```

# Demo

## Line number view

# Advanced Cocoa Text Tips and Tricks

**Aki Inoue**
Cocoa Engineer

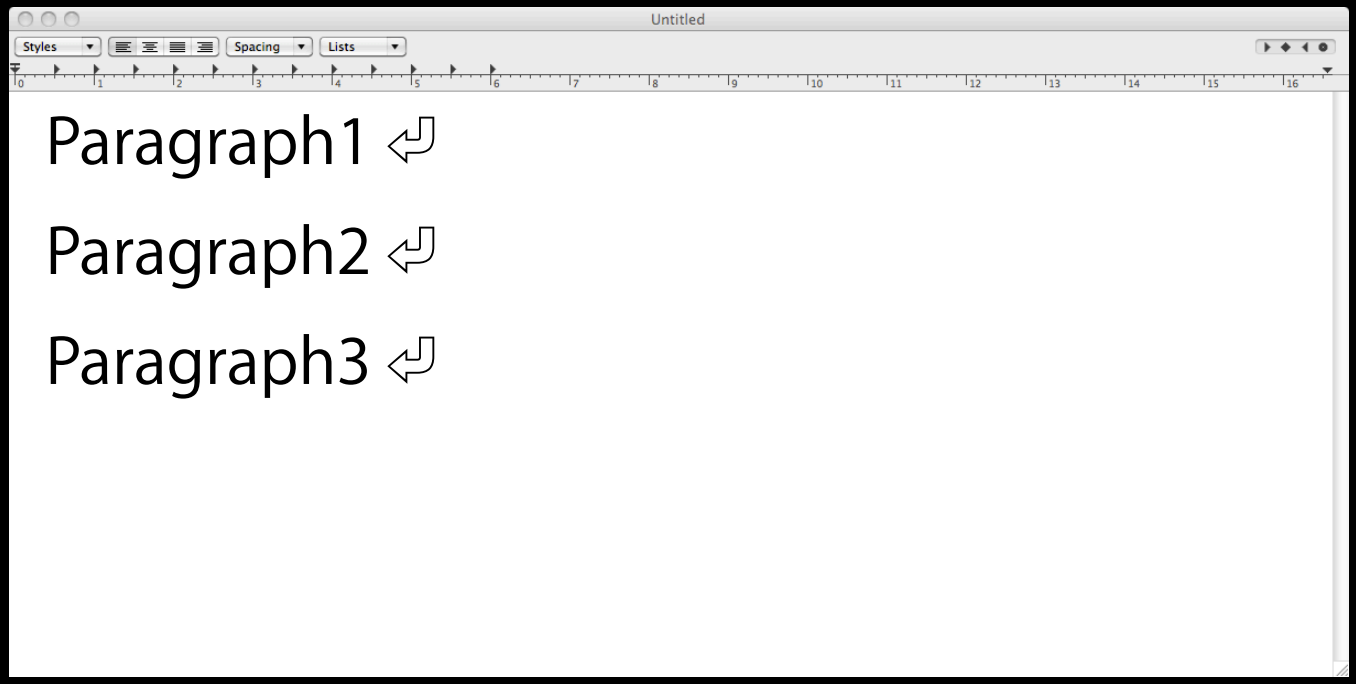
# Demo

## Text-folding editor

# Tip: How to Customize Control Character Behavior

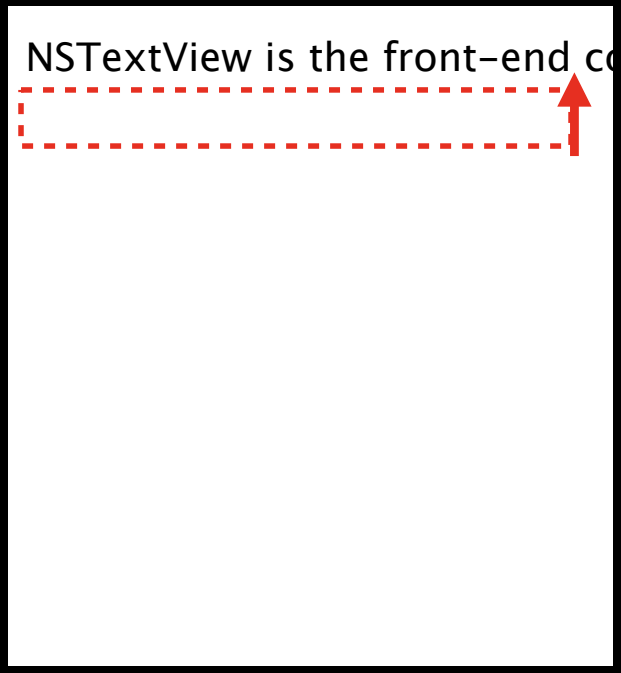
# Concatenate Paragraphs

## Need to concatenate all paragraphs in a folded text range



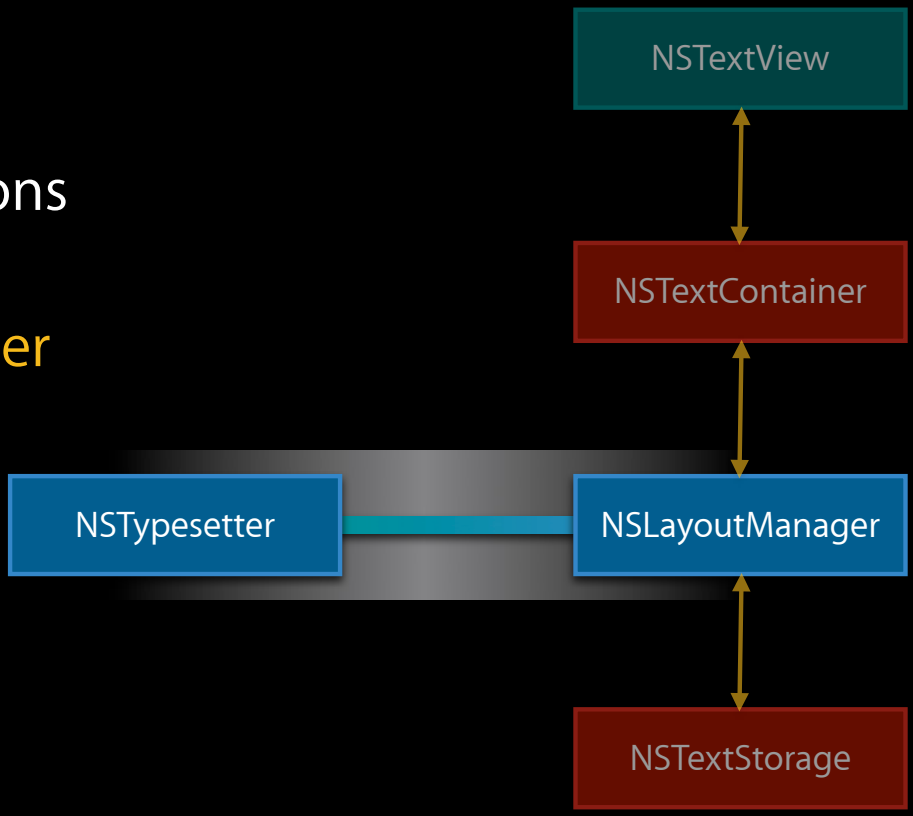
# NSTypesetter

- Performs actual line layout operations
- Determines…
  - Soft line breaks
  - Line positions in text container
  - Glyph positions in line



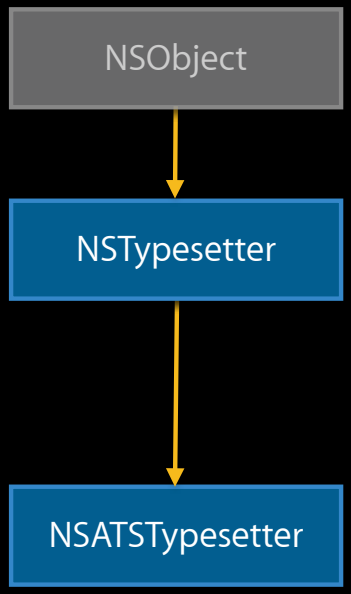
# NSTypesetter

- Performs actual line layout operations
- Abstract class declaring interface communicating with layout manager



# NSTypesetter

- Performs actual line layout operations
- Abstract class declaring interface communicating with layout manager
- NSATSTypesetter is the system-default concrete subclass



# NSTypesetter

- Performs actual line layout operations
- Abstract class declaring interface communicating with layout manager
- NSATSTypesetter is the system-default concrete subclass
- -layoutParagraphAtPoint: drives the layout process

# Typesetting Process

1. Layout manager requests typesetter to produce layout

```
- (NSRange)layoutCharactersInRange:(NSRange)characterRange
                 forLayoutManager:(NSLayoutManager *)layoutManager
    maximumNumberOfLineFragments:(NSUInteger)maxNumLines;
```

# Typesetting Process

1. Layout manager requests typesetter to produce layout
2. Typesetter determines paragraph boundaries

```
- (void)setParagraphGlyphRange:(NSRange)paragraphRange
             separatorGlyphRange:(NSRange)paragraphSeparatorRange;
```

# Typesetting Process

1. Layout manager requests typesetter to produce layout
2. Typesetter determines paragraph boundaries
3. Typesetter performs layout

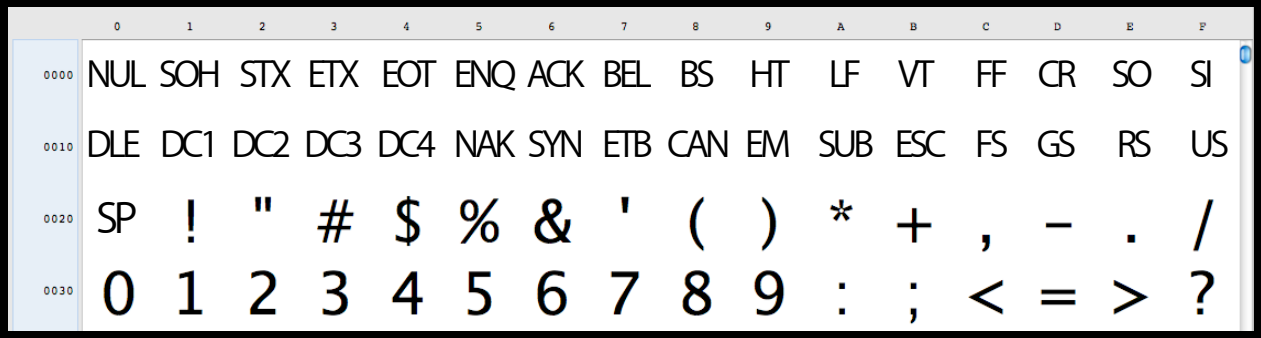
```
- (NSUInteger)layoutParagraphAtPoint:(NSPointPointer)lineFragmentOrigin;
```

# Typesetting Process

1. Layout manager requests typesetter to produce layout
2. Typesetter determines paragraph boundaries
3. Typesetter performs layout
4. Go back to 2.

# Control Character Handling

- Many characters controlling and formatting text contents



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 0010 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 0020 | SP | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 0030 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |

# Control Character Handling

- Many characters controlling and formatting text contents
- Tab, whitespace, line break, paragraph break, and container break

```
enum {
    NSTypesetterZeroAdvancementAction = (1 << 0),
    NSTypesetterWhitespaceAction = (1 << 1),
    NSTypesetterHorizontalTabAction = (1 << 2),
    NSTypesetterLineBreakAction = (1 << 3),
    NSTypesetterParagraphBreakAction = (1 << 4),
    NSTypesetterContainerBreakAction = (1 << 5)
};
typedef NSUInteger NSTypesetterControlCharacterAction;
```

# Control Character Handling

- Many characters controlling and formatting text contents
- Tab, whitespace, line break, paragraph break, and container break
- actionForControlCharacterAtIndex: determines the control action

```
- (NSTypesetterControlCharacterAction)actionForControlCharacterAtIndex:(NSUInteger)index;
```

# Control Character Handling

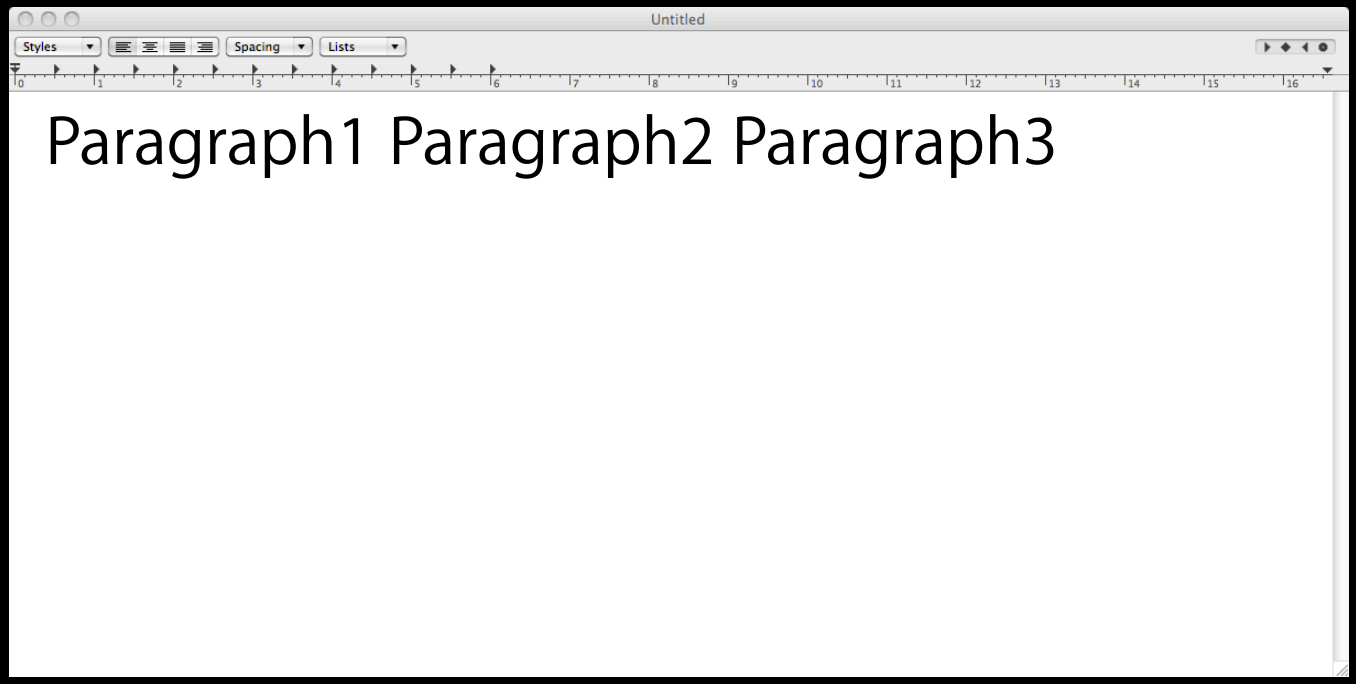
- Many characters controlling and formatting text contents
- Tab, whitespace, line break, paragraph break, and container break
- actionForControlCharacterAtIndex: determines the control action

```
extern NSString *lineFoldingAttributeName;

- (NSTypesetterControlCharacterAction)actionForControlCharacterAtIndex:(NSUInteger)index {
    NSAttributedString *attrString = [self attributedString];
    id value = [attrString attribute:lineFoldingAttributeName
                             atIndex:index effectiveRange:NULL];

    if (value && [value boolValue]) return NSTypesetterZeroAdvancementAction;

    return [super actionForControlCharacterAtIndex:charIndex];
}
```

# Tip: How to Customize Character to Glyph Mapping

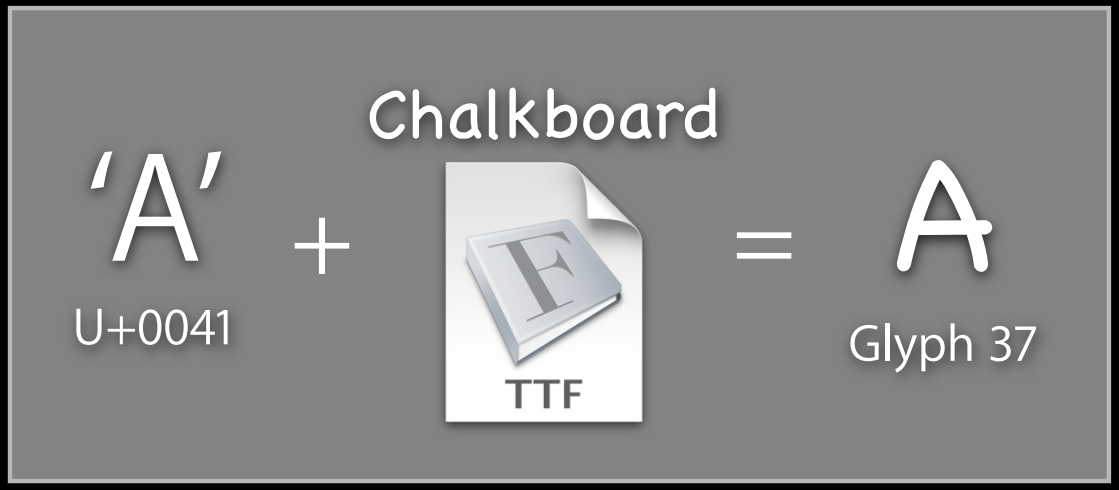
# Hide Text in Folded Range

## Substitute glyphs in a folded text range



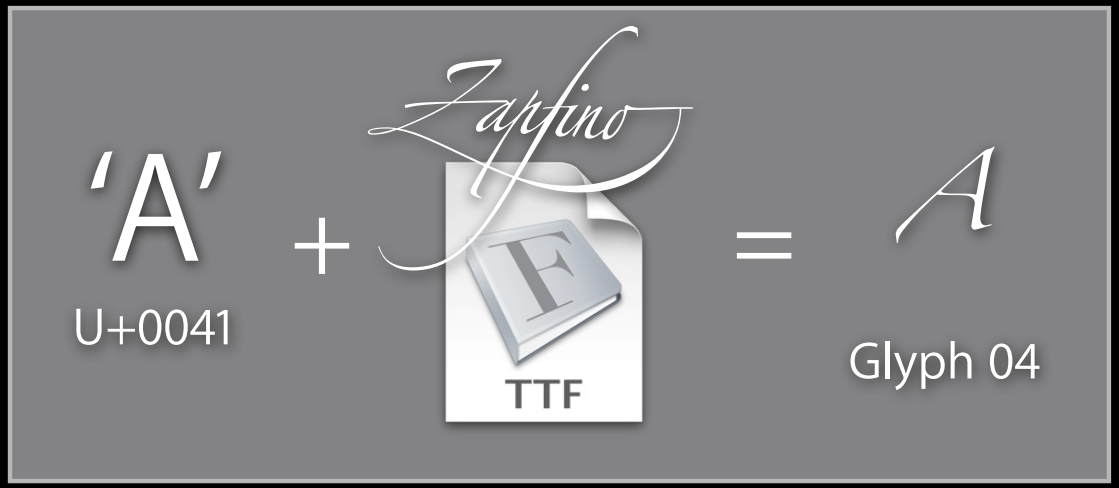
# Glyph Generation

- A glyph is the index for a graphical element in the font



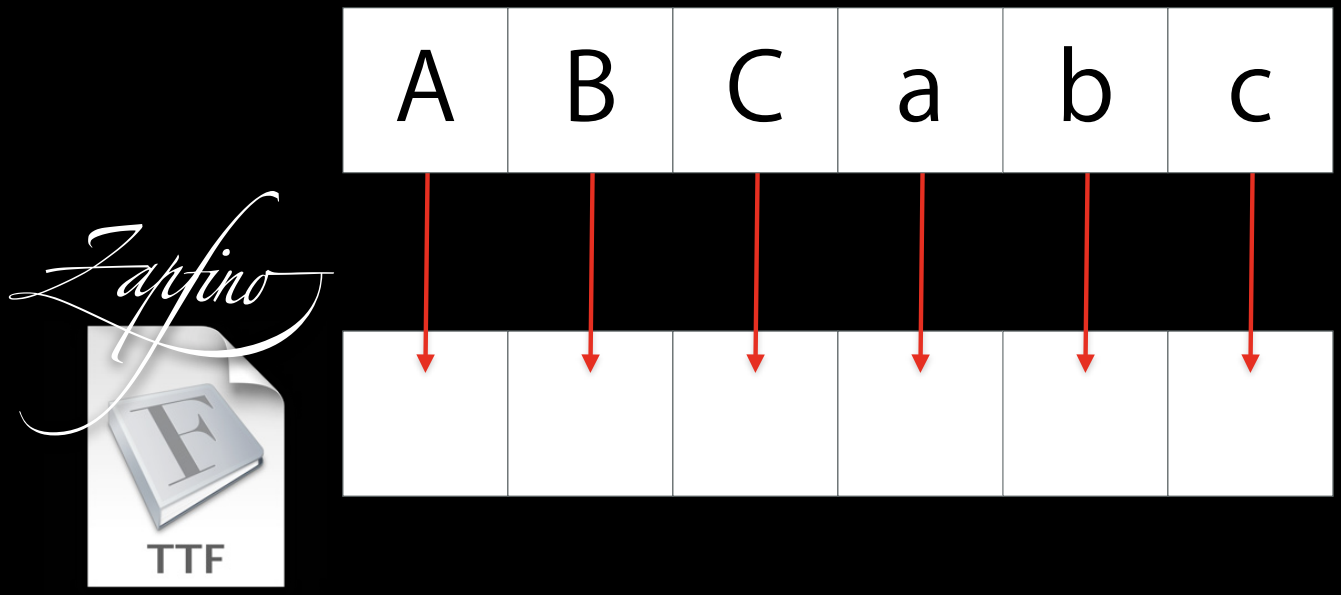
# Glyph Generation

- A glyph is the index for a graphical element in the font



# Glyph Generation

- A glyph is the index for a graphical element in the font
- Process for mapping Unicode characters to glyph IDs in the font



# Glyph Generation

- A glyph is the index for a graphical element in the font
- Process for mapping Unicode characters to glyph IDs in the font
- NSGlyphGenerator performs the glyph generation process

# Subclassing NSGlyphGenerator

- NSGlyphGenerator is an abstract class for a class cluster

# Subclassing NSGlyphGenerator

- NSGlyphGenerator is an abstract class for a class cluster
- Layout manager requests glyph generation

```
- (void)generateGlyphsForGlyphStorage:(id <NSGlyphStorage>)glyphStorage
           desiredNumberOfCharacters:(NSUInteger)nChars
                          glyphIndex:(NSUInteger *)glyphIndex
                      characterIndex:(NSUInteger *)charIndex;
```

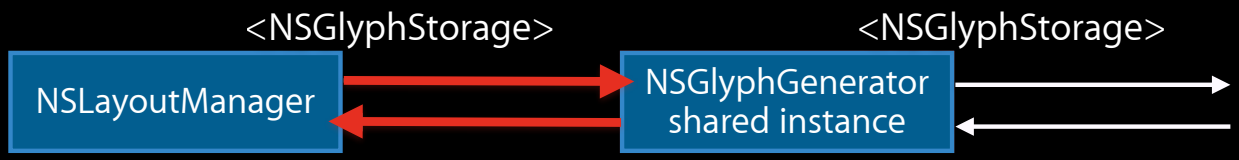
# Subclassing NSGlyphGenerator

- NSGlyphGenerator is an abstract class for a class cluster
- Layout manager requests glyph generation
- Glyph generator communicates results via NSGlyphStorage protocol

```
@protocol NSGlyphStorage
- (void)insertGlyphs:(const NSGlyph *)glyphs
                      length:(NSUInteger)length
        forStartingGlyphAtIndex:(NSUInteger)glyphIndex
                 characterIndex:(NSUInteger)charIndex;

/* rest of the protocol here */
@end
```

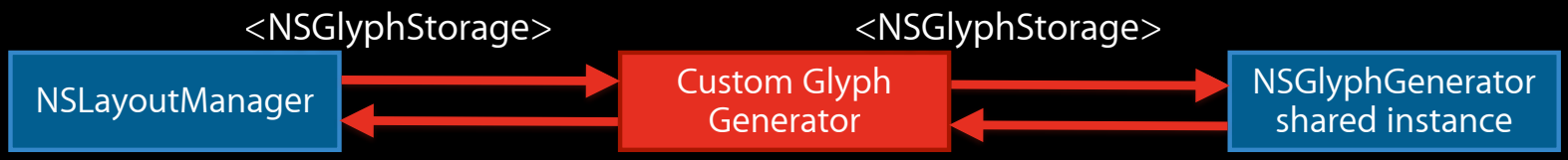
# Subclassing NSGlyphGenerator

- NSGlyphGenerator is an abstract class for a class cluster
- Layout manager requests glyph generation
- Glyph generator communicates results via NSGlyphStorage protocol
- An NSGlyphGenerator subclass delegates to the shared instance



# Subclassing NSGlyphGenerator

- NSGlyphGenerator is an abstract class for a class cluster
- Layout manager requests glyph generation
- Glyph generator communicates results via NSGlyphStorage protocol
- An NSGlyphGenerator subclass delegates to the shared instance



# Subclassing NSGlyphGenerator

```
@interface LineFoldingGlyphGenerator : NSGlyphGenerator <NSGlyphStorage> {
    id <NSGlyphStorage> _destination;
}
@end
```

# Subclassing NSGlyphGenerator

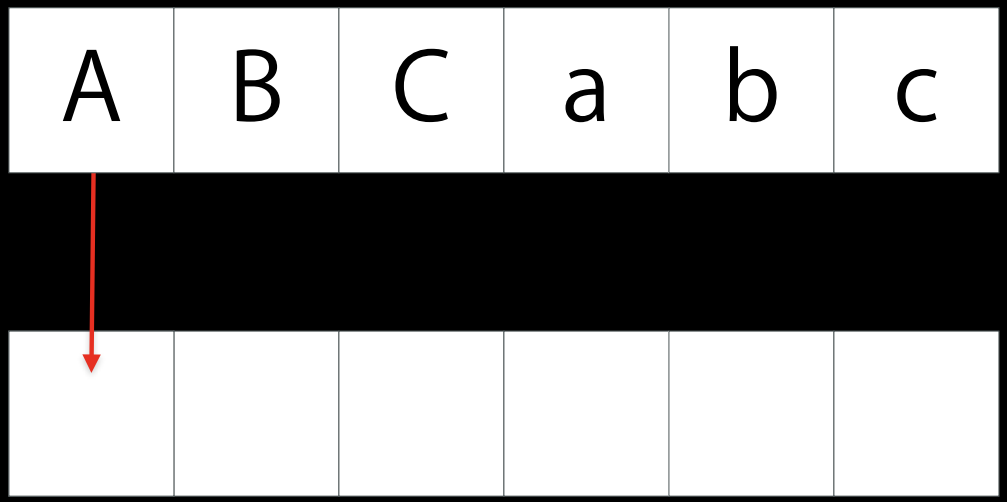
```
@implementation LineFoldingGlyphGenerator
- (void)generateGlyphsForGlyphStorage:(id <NSGlyphStorage>)glyphStorage
           desiredNumberOfCharacters:(NSUInteger)nChars
                          glyphIndex:(NSUInteger *)glyphIndex
                       characterIndex:(NSUInteger *)charIndex {
    NSGlyphGenerator instance = [NSGlyphGenerator sharedGlyphGenerator];

    _destination = glyphStorage;
    [instance generateGlyphsForGlyphStorage:self
                  desiredNumberOfCharacters:nChars
                                 glyphIndex:glyphIndex
                             characterIndex:charIndex];
    _destination = nil;
}
@end
```

# Subclassing NSGlyphGenerator

```
@implementation LineFoldingGlyphGenerator
- (void)insertGlyphs:(const NSGlyph *)glyphs
                      length:(NSUInteger)length
        forStartingGlyphAtIndex:(NSUInteger)glyphIndex
                 characterIndex:(NSUInteger)charIndex {

    // Glyph customization code goes here…

    [_destination insertGlyphs:glyphs
                        length:length
       forStartingGlyphAtIndex:glyphIndex
                characterIndex:charIndex];
}
@end
```

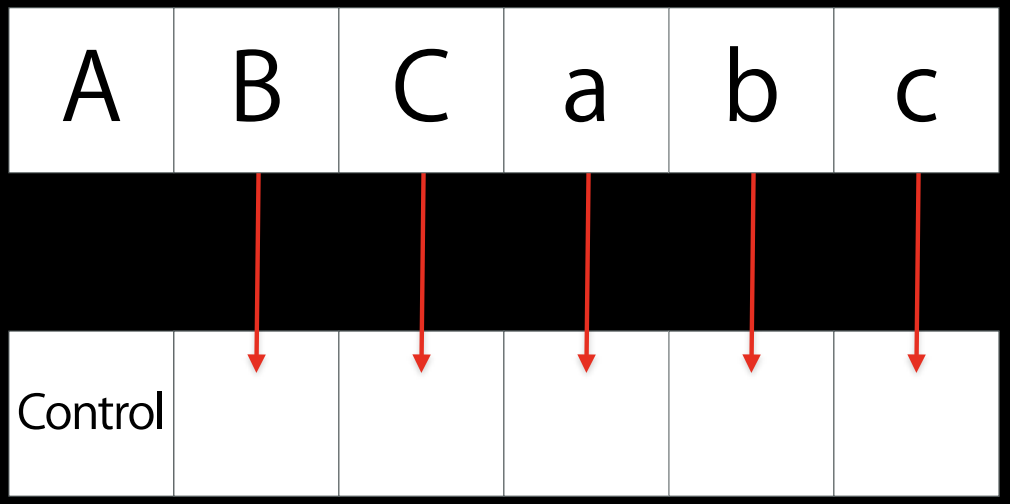
# Custom Glyph Generation for Text Folding

- The first character in a folded text range becomes NSControlGlyph



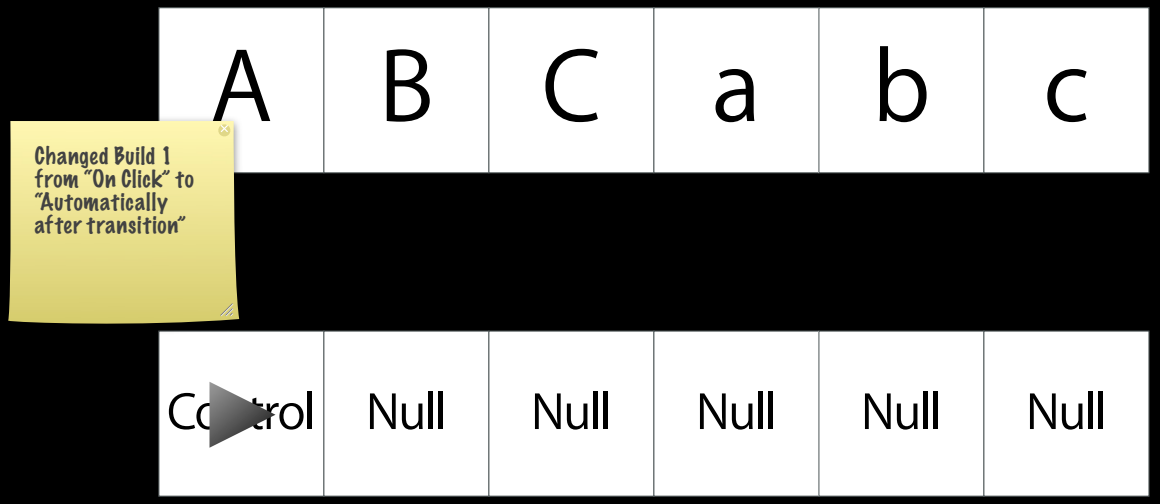
# Custom Glyph Generation for Text Folding

- The first character in a folded text range becomes NSControlGlyph
- The rest are NSNullGlyph



# Custom Glyph Generation for Text Folding

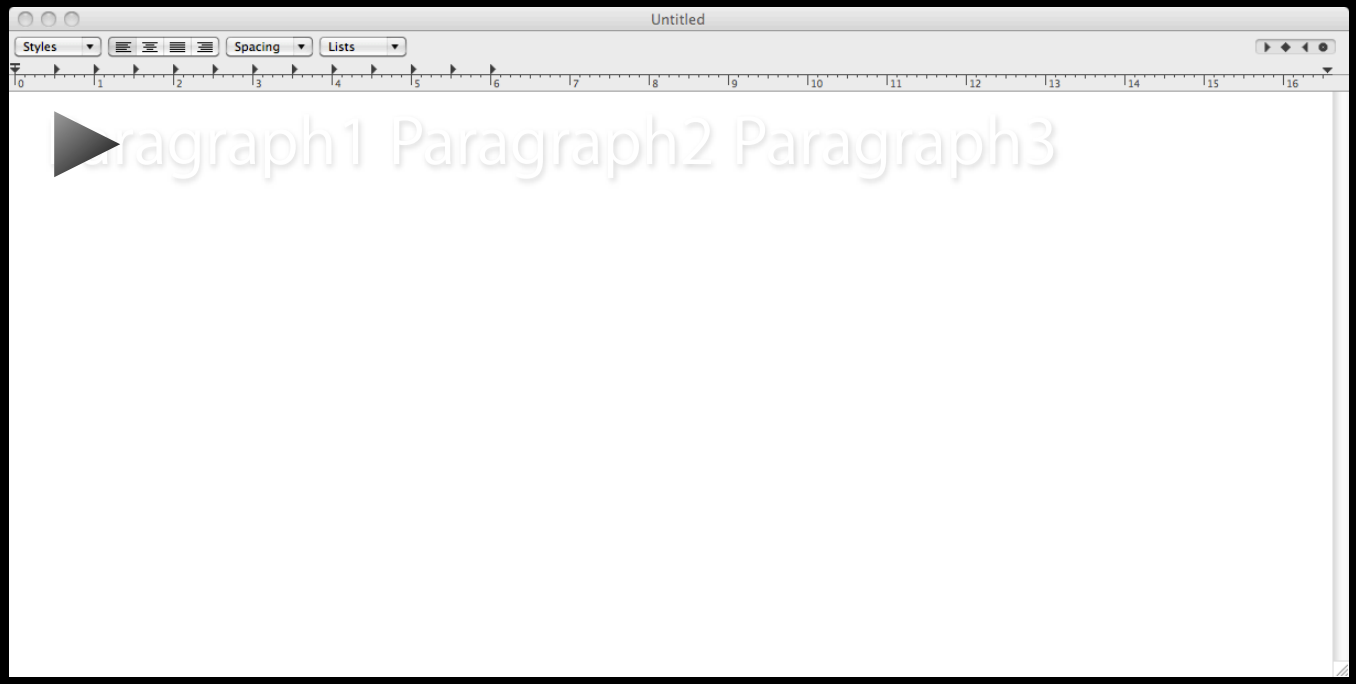
- The first character in a folded text range becomes NSControlGlyph
- The rest are NSNullGlyph



# Tip: How to Add UI Widgets to Text Contents

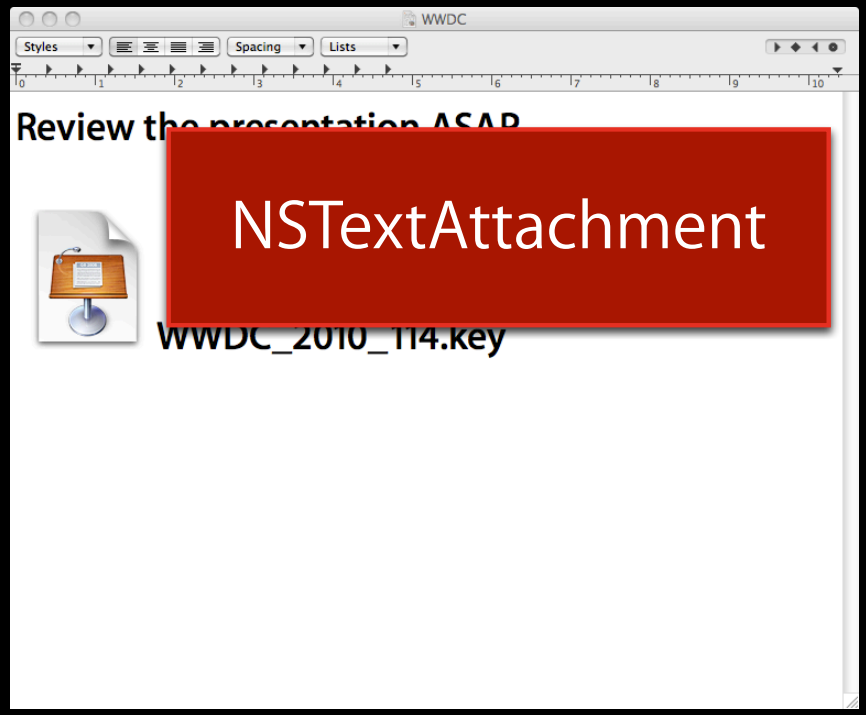
# Showing Disclosure Triangle

## Text attachment acting as an inline UI widget



# Embedding Inline Objects

- Image embedding is built in
- Files can be handled easily
- Designed to handle arbitrary objects
- Beyond just making space for images



# NSTextAttachment

- Encapsulates both storage and icon
  - NSFileWrapper
  - NSTextAttachmentCell
- Complete suite of API for object embedding
- NSTokenField utilizes custom text attachments

# NSTextAttachmentCell

## Fully customizable inline object

### Custom Shape

```
- (void)drawWithFrame:(NSRect)cellFrame
               inView:(NSView *)controlView
       characterIndex:(NSUInteger)charIndex
        layoutManager:(NSLayoutManager *)layoutManager;

- (NSRect)cellFrameForTextContainer:(NSTextContainer *)textContainer
               proposedLineFragment:(NSRect)lineFrag
                      glyphPosition:(NSPoint)position
                     characterIndex:(NSUInteger)charIndex;
```

# NSTextAttachmentCell

## Fully customizable inline object

### Custom Mouse Handling

```
- (BOOL)wantsToTrackMouseForEvent:(NSEvent *)theEvent
                           inRect:(NSRect)cellFrame
                           ofView:(NSView *)controlView
                 atCharacterIndex:(NSUInteger)charIndex;

- (BOOL)trackMouse:(NSEvent *)theEvent
            inRect:(NSRect)cellFrame
            ofView:(NSView *)controlView
  atCharacterIndex:(NSUInteger)charIndex
      untilMouseUp:(BOOL)flag;
```

# Synthesizing Text Attachment Attribute

- Returning a custom text attachment cell

```
@interface LineFoldingTextStorage : NSTextStorage {
    NSTextStorage *_attributedString;
    BOOL _lineFoldingEnabled;
}

@property(getter=isLineFoldingEnabled) BOOL lineFoldingEnabled;
@end
```

# Synthesizing Text Attachment Attribute

- Returning a custom text attachment cell

```
@implementation LineFoldingTextStorage

- (NSDictionary *)attributesAtIndex:(NSUInteger)location
                     effectiveRange:(NSRangePointer)range {
    NSDictionary *attributes = [_attributedString attributesAtIndex:location
                                                     effectiveRange:range];

    if (self.isLineFoldingEnabled) {
        /* Add a custom attachment attribute here */
    }

    return attributes;
}
```

# Synthesizing Text Attachment Attribute

- Returning a custom text attachment cell
- Synthesize the attribute when rendering and laying out
  - -[NSLayoutManager drawGlyphsForGlyphRange:atPoint:]
  - -[NSATSTypesetter layoutParagraphAtPoint:]

```
- (void)drawGlyphsForGlyphRange:(NSRange)glyphsToShow
                        atPoint:(NSPoint)origin {
    NSTextStorage *textStorage = [self textStorage];

    [(LineFoldingTextStorage *)textStorage setLineFoldingEnabled:YES];
    [super drawGlyphsForGlyphRange:glyphsToShow atPoint:origin];
    [(LineFoldingTextStorage *)textStorage setLineFoldingEnabled:NO];
}
```

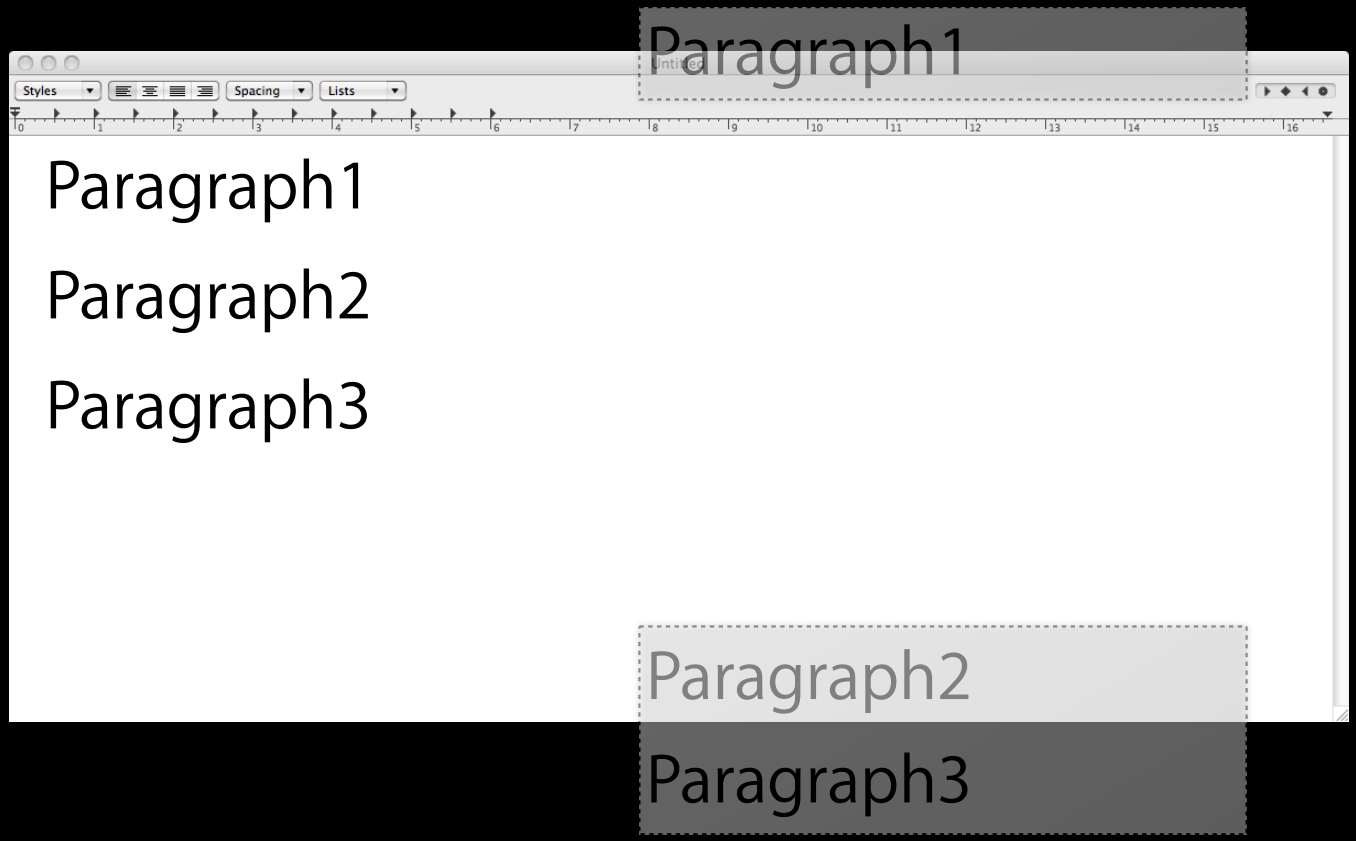
# Tip: How to Animate Text

# Text Animation

- Manipulate like graphic elements
- Choose appropriate animating text unit
- Capture starting and ending state
- Use overlay views

# Attaching Overlay

## Animate overlay views

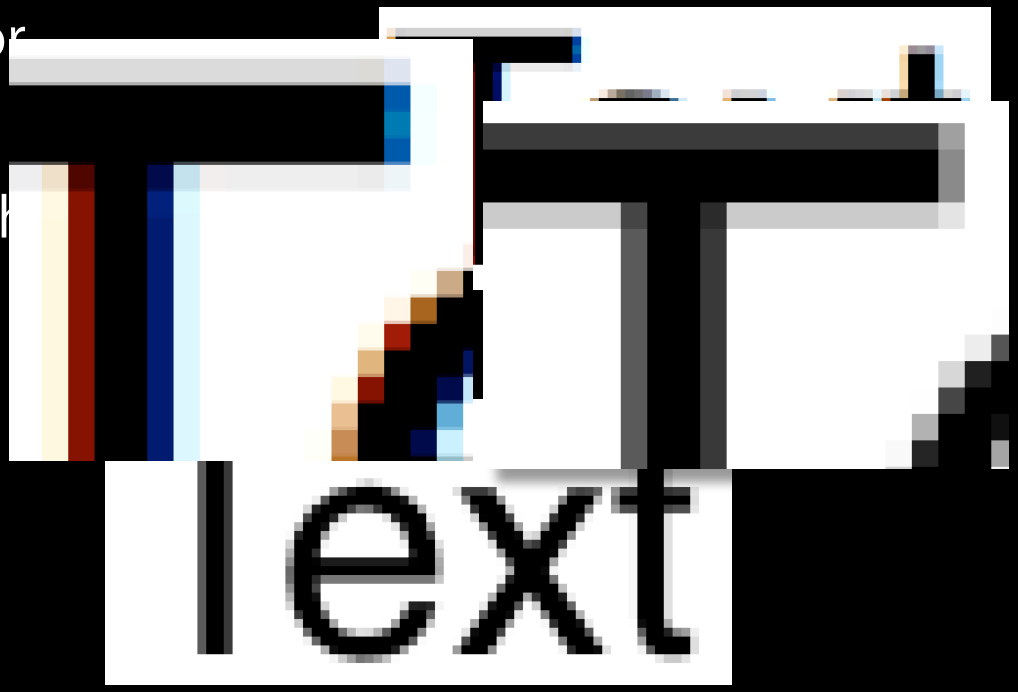


# Overlay Animation

- Can avoid complication
- Easier to manage start and end state capturing
- More efficient by focusing to visible area
- Easier to prevent font smoothing issues

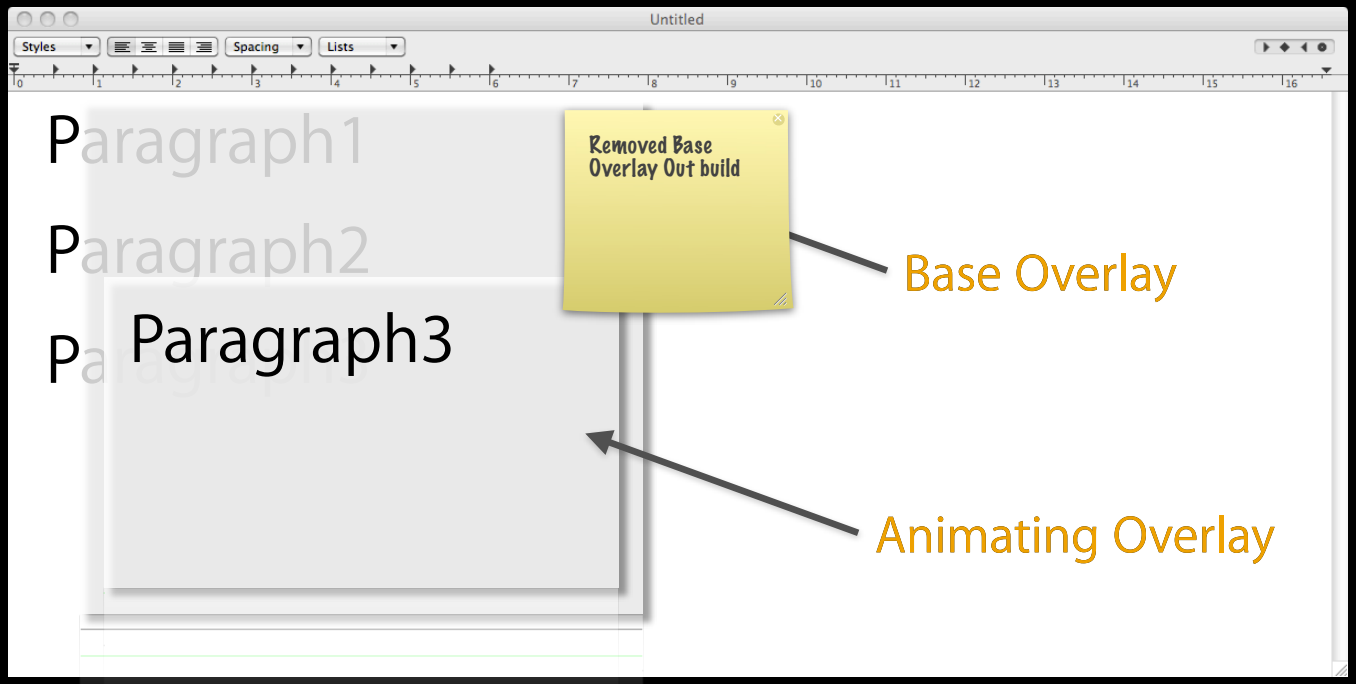
# Font Smoothing vs. Transparent Background

- a.k.a. “subpixel rendering”
- Cannot choose appropriate color
- Use opaque layer background
- OK to disable font smoothing wh



# Animating NSTextView Contents

## Attaching a layer hosting view on the fly



# Animating NSTextView Contents

```
@interface LineFoldingAnimationOverlay : NSView {
    void (^_renderer)(void);
}
@property (copy) void (^renderer)(void);
@end

@implementation LineFoldingAnimationOverlay : NSView
- (void)drawRect:(NSRect)dirtyRect {
    if (_renderer) _renderer();
}
@end
```

# Animating NSTextView Contents

- Cache into CoreAnimation layer

```
bounds = [layoutManager boundingRectForGlyphRange:glyphRange
                                  inTextContainer:textContainer];

[overlay setRenderer:(void (^)(void))^(void) {
    NSPoint point = NSMakePoint(-NSMinX(bounds), -NSMinY(bounds));
    [color set];
    NSRectFill([overlay bounds]);
    [layoutManager drawGlyphsForGlyphRange:glyphRange atPoint:point];
}];

[overlay display];
```

# Animating NSTextView Contents

- Recommend attaching a layer hosting view on the fly
- Controlling the layout process for capturing the “right” moment
  - -beginEditing and -endEditing for delaying layout

```
NSArray *ranges = [self rangesForUserParagraphAttributeChange];

[textStorage beginEditing];
[ranges enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
  NSRange currentRange = [obj rangeValue];

  /* cache pre-animation state */

  /* modify textStorage for currentRange */
}];
[textStorage endEditing];
```

# More Information

**Bill Dudney**
Application Frameworks Evangelist
dudney@apple.com

**Documentation**
Cocoa Text Architecture Guide
http://developer.apple.com/mac/library/documentation/TextFonts/Conceptual/CocoaTextArchitecture

**Apple Developer Forums**
http://devforums.apple.com

# Related Sessions

| | |
|---|---|
| API Design for Cocoa and Cocoa Touch | Marina<br>Thursday 4:30PM |
| Key Event Handling in Cocoa Applications | Russian Hill<br>Friday 11:30AM |

# Labs

| Cocoa Lab | Application Frameworks Lab D<br>Thursday 2:00PM |
| --- | --- |

# Summary

- The Cocoa Text System provides virtually unlimited extensibility
- Its modular design allows customization at the right level for any kind of application needs
- Seek customization points in the primary classes first; then, consider helper class subclassing

# Q&A

WWDC10

The last slide after the logo is intentionally left blank for all