TextKit Best Practices

Session 221

Donna Tom, TextKit
Emily Van Haren, Authoring Tools
Key concepts
Examples
Best practices
Key concepts
Examples
Best practices
What is TextKit?
import TextKit
import TextKit

No such module 'TextKit'
Choosing the Right Control
Choosing the Right Control

Start

Text input?

Yes Yes Yes

Secure entry?

Yes

One line?

Yes

UITextField

No

No

UITextView

Selection or scrolling?

Yes

No

No

UILabel
Choosing the Right Control

Start

Text input?

Yes: Secure entry?

Yes: One line?

Yes: UITextField

No:Selection or scrolling?

Yes: UITextView

No: UILabel
Choosing the Right Control

Start

Text input?
- Yes
  - Secure entry?
    - Yes
      - One line?
        - Yes
          - UITextField
        - No
          - UITextView
    - No
      - Selection or scrolling?
        - Yes
          - UILabel
        - No
          - UITextView
Choosing the Right Control

Start

Text input?

Yes

Secure entry?

Yes

UITextField

No

No

No

No

No

No

No

Selection or scrolling?

Yes

UITextView

No

One line?

No

No

No

No

UILabel
Choosing the Right Control

Start

Text input?

Yes

Secure entry?

Yes

UITextField

Yes

One line?

Yes

UITextView

Selection or scrolling?

No

UILabel

No

No

No

No

No

No

No

No

No

No

No
Choosing the Right Control

Start

Text input?

Yes

Secure entry?

Yes

One line?

Yes

UITextField

No

No

No

No

Selection or scrolling?

Yes

UITextView

No

No

No

UILabel
Choosing the Right Control

Start

Yes

Text input?

Yes

Secure entry?

Yes

UITextField

No

One line?

Yes

UITextView

No

No

No

No

No

No

Secure entry?

Selection or scrolling?

Yes

Selection or scrolling?

No

UILabel
Choosing the Right Control

Start

Yes: Text input?

Yes: Secure entry?

Yes: One line?

UITextField

No: Selection or scrolling?

Yes: Secure entry?

Yes: One line?

UITextView

No: One line?

UITextView

No: Secure entry?

UILabel

No: Selection or scrolling?

UILabel
Choosing the Right Control

Start

Text input?  

Yes  

Secure entry?  

Yes  

One line?  

Yes  

UITextField  

No  

Selection or scrolling?  

Yes  

UITextView  

No  

UITextView  

No  

UILabel
Choosing the Right Control

Start

Text input?

Yes

Secure entry?

Yes

NSSecureTextField

No

Lots of text?

Yes

NSTextView

No

NSTextField

Lots of text?

NSSecureTextField

NSTextField
Choosing the Right Control

Start

Text input?

Yes → Secure entry?

Yes → NSSecureTextField

No → NSTextView

Lots of text?

Yes → NSSecureTextField

No → NSTextField
Choosing the Right Control

Start

Text input?

Secure entry?

Lots of text?

Yes

Yes

Yes

Yes

 NSSecureTextField

NSSecureTextField

Yes

No

No

No

No

No

NSTextView

NSTextView

Lots of text?

NSTextField

NSTextField
Choosing the Right Control

1. **Start**
   - **Text input?**
     - Yes: **Secure entry?**
       - Yes: **NSSecureTextField**
       - No: **NSTextView**
     - No: **NSTextField**
   - No: **NSTextView**
Choosing the Right Control

Start

Text input?

Secure entry?

Lots of text?

NSSecureTextField

NSTextView

NSTextField
Choosing the Right Control

Start

- Text input?
  - Yes: Secure entry?
    - Yes: NSSecureTextField
    - No: NSTextView
  - No: NSTextField

- No: Lots of text?
  - Yes: NSSecureTextField
  - No: NSTextView
  - No: NSTextField

AppKit
Choosing the Right Control

Start

Text input?

Secure entry?

Lots of text?

Yes

Yes

Yes

Yes

NSSecureTextField

NSTextView

NSTextView

NSSecureTextField

NSTextFormField
What about string drawing?
What about string drawing?

```swift
func draw(at: CGPoint)

func draw(in: CGRect)

func draw(with: CGRect, 
    options: NSStringDrawingOptions = [], 
    context: NSStringDrawingContext?)
```
Choosing the Right Control

String drawing

Use for small amounts of text

Limit drawing frequency

Limit number of custom attributes
### Choosing the Right Control

<table>
<thead>
<tr>
<th>Feature</th>
<th>Label</th>
<th>Text Field</th>
<th>Text View</th>
<th>String Drawing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display text on screen</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Respond to touch and/or mouse events</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Accessibility support</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Vibrancy</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Respond to focus changes</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Selection and editing</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Secure text entry</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Choosing the Right Customization Point
Controller
Layout

Model
Storage

View
Display
Choosing the Right Customization Point

NSTextStorage

Holds string data and attributes

Waffles are very delicious.
Choosing the Right Customization Point
NSTextContainer

Defines geometric area for text layout

The first requisite in the preparation of good sandwiches is to have perfect bread in suitable condition. Either white, brown, rye, or entire wheat bread may be used, but it should be of close, even texture and at least one day old. Cream the butter with a wooden spoon and spread smoothly on the bread before it is avoid spreading the butter to serve, cut the triangular, long, narrow, cut; after cutting remove the crust and or filling over the edge. When ready sandwiches either square, round, or crescent shaped.

In making rolled bread crust of a loaf of fresh layer of butter on one buttered end in as thin spread with the slice and lay on a firmly around the rolled place until ready to serve, sandwiches with baby ribbon sandwiches, cut off the bread and spread a thin end of the loaf; cut off this a slice as possible and sandwich filling; roll up this napkin draw the napkin bread and pin it. Put in a cool then remove napkin and tie the or fasten with a toothpick.

French rolls may be used for picnics and out-of-door luncheons. Remove from
Text System Storage Layer Overview

Attributed String Programming Guide

Text Attribute Programming Topics

<table>
<thead>
<tr>
<th>Introduction to Attributed Strings for iOS</th>
<th>WWDC 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Attributed Strings for iOS</td>
<td>WWDC 2012</td>
</tr>
</tbody>
</table>
Storage
- NSTextStorage
- NSTextContainer

Layout

Display
Documentation

Text Programming Guide for iOS

Text System User Interface Layer Programming Guide
Diagram showing a central 'Layout' component connected to 'Storage' and 'Display' through 'NSLayoutManager'.
Choosing the Right Customization Point
NSLayoutManager

Coordinates between storage and display
Controls layout process
Choosing the Right Customization Point

Layout process

Attribute Fixing → Glyph Generation → Glyph Layout → Display

Before:

Tempura (天麩羅) is a tasty Japanese food. 🍤

Times New Roman
Choosing the Right Customization Point

Layout process

Attribute Fixing → Glyph Generation → Glyph Layout → Display

After:

Tempura (天麩羅) is a tasty Japanese food. 🍗
Choosing the Right Customization Point
Layout process

- Attribute Fixing
- Glyph Generation
- Glyph Layout
- Display
Choosing the Right Customization Point

Layout process

Attribute Fixing → Glyph Generation → Glyph Layout → Display
Choosing the Right Customization Point

Glyphs

multiple characters

single glyph
Choosing the Right Customization Point

Glyphs

single character

multiple glyphs
Choosing the Right Customization Point

Layout process

Attribute Fixing → Glyph Generation → Glyph Layout → Display
Choosing the Right Customization Point

Layout process

- Attribute Fixing
- Glyph Generation
- Glyph Layout
- Display

characters to glyphs
Choosing the Right Customization Point

Layout process

Attribute Fixing → Glyph Generation → Glyph Layout → Display

- characters to glyphs
- position glyphs
Text Layout Programming Guide
Layout

NSLayoutManager

Storage

Display
Choosing the Right Configuration
This is the standard configuration. The text container holds a weak reference to the text view, and the text view retains the whole tree by retaining the root text storage.
As text is added, it fills the region defined by the first text container. The text is displayed in the text view that is paired with the text container.

When there’s no more room, another text container (and associated text view) is added, and the text flows onto the second page.
Multiple layout managers allow you to have multiple presentations of the same text. The text within each view can have separate layout and selections.
Related Previous Sessions

Advanced Cocoa Text Tips and Tricks

WWDC 2010
Choosing the Right Customization Approach
Choose the right tool for the job
Choosing the Right Customization Approach

- Delegation
- Notifications
- Subclassing
Choosing the Right Customization Approach

- Delegation
- Notifications
- Subclassing
Choosing the Right Customization Approach

Delegation
Notifications
Subclassing
TextKit is everywhere.
Apple News on iOS
Students succeed with iPad and Mac

When you walk into Wilder Elementary School, just outside of Boise, Idaho, it’s what
Students succeed with iPad and Mac

When you walk into Wilder Elementary School, just outside of Boise, Idaho, it’s what
Spotlight
WEEKEND PURSUITS

From the Apple News Editors
Choosing the right control.
Label - A variably sized amount of static text.
@IBOutlet weak var spotlightLabel: UILabel!

override func viewDidLoad() {
    super.viewDidLoad()
    spotlightLabel.text = "Spotlight"
    spotlightLabel.textColor = .darkText
    spotlightLabel.font = UIFont.preferredFont(forTextStyle: .body)
    spotlightLabel.adjustsFontForContentSizeCategory = true
}
From the Apple News Editors
From the Apple News Editors
From the Apple News Editors
From the Apple News Editors

Font: .preferredFont(forTextStyle: .body)

Color: .darkText .white
@IBOutlet weak var label: UILabel!

override func viewDidLoad() {
    super.viewDidLoad()

    let mutableAttrString = NSMutableAttributedString(string: label.text)

    let sourceRange = // Range of "Apple News Editors"
    mutableAttrString.addAttribute(.foregroundColor, value: UIColor.white, range: sourceRange)

    label.attributedText = mutableAttrString
}
@IBOutlet weak var label: UILabel!

override func viewDidLoad() {
    super.viewDidLoad()
    let mutableAttrString = NSMutableAttributedString(string: label.text)

    let sourceRange = // Range of “Apple News Editors”
    mutableAttrString.addAttribute(.foregroundColor, value: UIColor.white, range: sourceRange)

    label.attributedText = mutableAttrString
}
@IBOutlet weak var label: UILabel!

override func viewDidLoad() {
    super.viewDidLoad()

    let mutableAttrString = NSMutableAttributedString(string: label.text)

    let sourceRange = // Range of “Apple News Editors”
    mutableAttrString.addAttribute(.foregroundColor, value: UIColor.white, range: sourceRange)

    label.attributedText = mutableAttrString
}
Spotlight
WEEKEND PURSUITS From the Apple News Editors
Students succeed with iPad and Mac

When you walk into Wilder Elementary School, just outside of Boise, Idaho, it’s what
Students succeed with iPad and Mac

When you walk into Wilder Elementary School, just outside of Boise, Idaho, it’s what
Students succeed with iPad and Mac

When you walk into Wilder Elementary School, just outside of Boise, Idaho, it’s what
TextView

- Displays multiple lines of editable text and sends an action message to a target object when Return is tapped.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Nibh erat consequntur for ture non ipsum dolari odioque cillum.

Spotlight
WEEKEND PURSUITS
From the Apple News Editors
Spotlight
WEEKEND PURSUITS
From the Apple News Editors

students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac
Students succeed with iPad and Mac

Spotlight
WEEKEND PURSUITS
From the Apple news Editors
Students succeed with iPad and Mac
@IBOutlet weak var headlineTextView: UITextView!

override func viewDidLoad() {
    super.viewDidLoad()

    headlineTextView.text = "Students succeed with iPad and Mac"
    headlineTextView.textColor = .darkText
    headlineTextView.font = UIFont(name: "Georgia", size: 37)
    headlineTextView.backgroundColor = nil
    headlineTextView.isEditable = false
    headlineTextView.isScrollEnabled = false
}

@IBOutlet weak var headlineTextView: UITextView!

override func viewDidLoad() {
    super.viewDidLoad()
    headlineTextView.text = "Students succeed with iPad and Mac"
    headlineTextView.textColor = .darkText
    headlineTextView.font = UIFont(name: "Georgia", size: 37)
    headlineTextView.backgroundColor = nil
    headlineTextView.isEditable = false
    headlineTextView.isScrollEnabled = false
}

UIKit
Choosing the right configuration.
TextEdit on macOS
Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.
Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.
Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.
Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or villify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.
Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.
Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.
Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.
Here’s to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They’re not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can’t do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.


Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.


Vivamus non efficitur neque. Sed tristique orutum auctor. Ut imperdiet sem id lorem

vel molestie eu, dignissim nec erat. Morbi finibus luctus feugiat.

suscipit augue, et semper mauris.

Here's to the crazy ones. The misfits. The rebels. The trouble-makers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules, and they have no respect for the status-quo. You can quote them, disagree with them, glorify, or vilify them. But the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see genius. Because the people who are crazy enough to think they can change the world, are the ones who do.


Vivamus nec efficitur nec. Sed tristique erat, auctor ut. Ut imperdiet sem id lorem.
Choosing the right customization approach.
Our Journal App on macOS
Wednesday, June 6, 2018
Wednesday, June 6, 2018
Wednesday, June 6, 2018
Wednesday, June 6, 2018
Wednesday, June 6, 2018
Delegation

`func textStorage(_:didProcessEditing:range:changeInLength:)`

Notification

`NSTextStorage`

`didProcessEditingNotification`

Subclassing

`NSTextView`

`override func textDidChange()`
Delegation

NSTextStorageDelegate
func textStorage(_:didProcessEditing:range:changeInLength:)

Notification

NSTextStorage
didProcessEditingNotification

Subclassing

NSTextView
override func textDidChange()
func registerObservers() {
    NotificationCenter.default.addObserver(self,
        selector: #selector(updateWordCount),
        name: NSTextStorage.didProcessEditingNotification,
        object: nil)
}

@objc func updateWordCount() {
    guard let textStorage = textView.textStorage else { fatalError() }
    let wordCount = textStorage.words.count
    wordCountLabel.stringValue = String(wordCount)
}
func registerObservers() {
    NotificationCenter.default.addObserver(self,
        selector: #selector(updateWordCount),
        name: NSTextStorage.didProcessEditingNotification,
        object: nil)
}

@objc func updateWordCount() {
    guard let textStorage = textView.textStorage else { fatalError() }

    let wordCount = textStorage.words.count
    wordCountLabel.stringValue = String(wordCount)
}
```swift
func registerObservers() {
    NotificationCenter.default.addObserver(self,
        selector: #selector(updateWordCount),
        name: NSTextStorage.didProcessEditingNotification,
        object: nil)
}

@objc func updateWordCount() {
    guard let textStorage = textView.textStorage else { fatalError() }

    let wordCount = textStorage.words.count
    wordCountLabel.stringValue = String(wordCount)
}
```
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. The rebels. The troublemakers.
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. The rebels. The troublemakers.
Today I celebrated the crazy ones. The misfits. The rebels. The troublemakers.
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. The rebels. The troublemakers.
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.
Delegation

NSTextStorageDelegate
func textStorage(_:didProcessEditing:range:changeInLength:)

Notification

NSTextStorage
didProcessEditingNotification

Subclassing

NSTextStorage
override func replaceCharacters(in:with:)
NSTextStorageDelegate
func textStorage(_:didProcessEditing:range:changeInLength:)

NSTextStorage
didProcessEditingNotification

NSTextStorage
override func replaceCharacters(in:with:)
func textStorage(_ textStorage: NSTextStorage, 
   didProcessEditing editedMask: NSTextStorageEditActions, 
   range editedRange: NSRange, 
   changeInLength delta: Int) {

    let boldRange = // Range including start and end **
    let start = boldRange.location

    guard let font = textStorage.attribute(.font, at: start, effectiveRange: nil) as? NSFont 
    else { fatalError() }
    let boldFont = NSFontManager.shared.convert(font, toHaveTrait: .boldFontMask)

    textStorage.addAttribute(.font, value: boldFont, range: boldRange)
}
func textStorage(_ textStorage: NSTextStorage, didProcessEditing editedMask: NSTextStorageEditActions, range editedRange: NSRange, changeInLength delta: Int) {

    let boldRange = // Range including start and end **
    let start = boldRange.location

    guard let font = textStorage.attribute(.font, at: start, effectiveRange: nil) as? NSFont
    else { fatalError() }

    let boldFont = NSFontManager.shared.convert(font, toHaveTrait: .boldFontMask)

    textStorage.addAttribute(.font, value: boldFont, range: boldRange)
}
func textStorage(_ textStorage: NSTextStorage, 
    didProcessEditing editedMask: NSTextStorageEditActions, 
    range editedRange: NSRange, 
    changeInLength delta: Int) {

    let boldRange = // Range including start and end **
    let start = boldRange.location

    guard let font = textStorage.attribute(.font, at: start, effectiveRange: nil) as? NSFont 
        else { fatalError() } 
    let boldFont = NSFontManager.shared.convert(font, toHaveTrait: .boldFontMask)

    textStorage.addAttribute(.font, value: boldFont, range: boldRange)
}
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.

```swift
let string = "Hello World!"
```

Word Count: 17
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. "The rebels." The troublemakers.

Swift Code

```swift
let string = "Hello World!"
```

Word Count: 17
NSTextStorageDelegate
func textStorage(_ didProcessEditing:range:changeInLength:)

Notification
NSTextStorage
didProcessEditingNotification

Subclassing
NSTextStorage
override func replaceCharacters(in:with:)
NSTextStorageDelegate

func textStorage(_:didProcessEditing:range:changeInLength:)

NSTextStorage

didProcessEditingNotification

NSTextStorage

override func replaceCharacters(in:with:)

Delegation

Notification

Subclassing
class CustomTextStorage: NSTextStorage {
    override var string: String {
        // Return string
    }

        // Return attributes
    }

    override func replaceCharacters(in range: NSRange, with str: String) {
        // Replace characters
    }

    override func setAttributes(_: [NSAttributedString.Key : Any]?, range: NSRange) {
        // Set attributes
    }
}
class CustomTextStorage: NSTextStorage {

    override func replaceCharacters(in range: NSRange, with str: String) {
        // Replace characters

        let codeBlockRange = // Range including start and end ``
        let paragraphStyle = // Get existing or new paragraph style

        let mutableStyle = NSMutableParagraphStyle()
        mutableStyle.setParagraphStyle(paragraphStyle)
        mutableStyle.textBlocks = [NSTextBlock()]

        addAttribute(.paragraphStyle, value: mutableStyle, range: codeBlockRange)
    }
}
class CustomTextStorage: NSTextStorage {

    override func replaceCharacters(in range: NSRange, with str: String) {
        // Replace characters

        let codeBlockRange = // Range including start and end ````
        let paragraphStyle = // Get existing or new paragraph style

        let mutableStyle = NS MUT A T L EParagraphStyle()
        mutableStyle.setParagraphStyle(paragraphStyle)
        mutableStyle.textBlocks = [NSTextBlock()]

        addAttribute(.paragraphStyle, value: mutableStyle, range: codeBlockRange)
    }
}
class CustomTextStorage: NSTextStorage {

    override func replaceCharacters(in range: NSRange, with str: String) {
        // Replace characters

        let codeBlockRange = // Range including start and end ```
        let paragraphStyle = // Get existing or new paragraph style

        let mutableStyle = NSMakeRangeMutableParagraphStyle()
        mutableStyle.setParagraphStyle(paragraphStyle)
        mutableStyle.textBlocks = [NSTextBlock()]

        addAttribute(.paragraphStyle, value: mutableStyle, range: codeBlockRange)
    }
}
class CodeBlock: NSTextBlock {

    override init() {
        super.init()

        setWidth(15.0, type: .absoluteValueType, for: .padding)
        setWidth(45.0, type: .absoluteValueType, for: .padding, edge: .minY)

        backgroundColor = NSColor(white: 0.95, alpha: 1.0)
    }
}
class CodeBlock: NSTextBlock {

    override init() {
        super.init()

        setWidth(15.0, type: .absoluteValueType, for: .padding)
        setWidth(45.0, type: .absoluteValueType, for: .padding, edge: .minY)

        backgroundColor = NSColor(white: 0.95, alpha: 1.0)
    }
}
class CodeBlock: NSTextBlock {

    override func drawBackground(withFrame frameRect: NSRect, in controlView: NSView, characterRange charRange: NSRange, layoutManager: NSLayoutManager) {

        let adjustedFrame: NSRect = // Padded rect inside frameRect
        super.drawBackground(withFrame: adjustedFrame, in: controlView, characterRange: charRange, layoutManager: layoutManager)

        let drawPoint: NSPoint = // Point inset from adjustedFrame origin
        let drawString = NSString(string: "Swift Code")
        let attributes = [NSAttributedString.Key.font: font, .foregroundColor: .blue]
        drawString.draw(at: drawPoint, withAttributes: attributes)
    }
}
class CodeBlock: NSTextBlock {

    override func drawBackground(withFrame frameRect: NSRect, in controlView:NSView, characterRange charRange: NSRange, layoutManager: NSLayoutManager) {

        let adjustedFrame: NSRect = // Padded rect inside frameRect
        super.drawBackground(withFrame: adjustedFrame, in: controlView, characterRange: charRange, layoutManager: layoutManager)

        let drawPoint: NSPoint = // Point inset from adjustedFrame origin
        let drawString = NSString(string: "Swift Code")
        let attributes = [NSAttributedString.Key.font: font, .foregroundColor: .blue]
        drawString.draw(at: drawPoint, withAttributes: attributes)
    }
}
class CustomTextStorage: NSTextStorage {

    override func replaceCharacters(in range: NSRange, with str: String) {
        // Replace characters

        let codeBlockRange = // Range including start and end ````
        let paragraphStyle = // Get existing or new paragraph style

        let mutableStyle = NSMutableParagraphStyle()
        mutableStyle.setParagraphStyle(paragraphStyle)
        mutableStyle.textBlocks = [CodeBlock()]

        addAttribute(.paragraphStyle, value: mutableStyle, range: codeBlockRange)
    }
}
class CustomTextStorage: NSTextStorage {

    override func replaceCharacters(in range: NSRange, with str: String) {
        // Replace characters

        let codeBlockRange = // Range including start and end ````
        let paragraphStyle = // Get existing or new paragraph style

        let mutableStyle = NSMutableParagraphStyle()
        mutableStyle.setParagraphStyle(paragraphStyle)
        mutableStyle.textBlocks = [CodeBlock()]

        addAttribute(.paragraphStyle, value: mutableStyle, range: codeBlockRange)
    }
}
class ViewController: NSViewController {

    @IBOutlet var textView: NSTextView!

    override func viewDidLoad() {
        super.viewDidLoad()

        guard let layoutManager = textView.layoutManager else { fatalError() }

        let customTextStorage = CustomTextStorage()
        layoutManager.replaceTextStorage(customTextStorage)
    }
}
class ViewController: NSViewController {

    @IBOutlet var textView: NSTextView!

    override func viewDidLoad() {
        super.viewDidLoad()
        guard let layoutManager = textView.layoutManager else { fatalError() }

        let customTextStorage = CustomTextStorage()
        layoutManager.replaceTextStorage(customTextStorage)
    }
}
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.

Swift Code

```swift
let string = "Hello World!"
```
Wednesday, June 6, 2018

Word Count: 0
class ViewController: NSViewController {

    @IBOutlet var leftTextView: NSTextView!
    @IBOutlet var rightTextView: NSTextView!

    override func viewDidLoad() {
        super.viewDidLoad()

        guard let leftTextStorage = leftTextView.textStorage, 
            let rightLayoutManager = rightTextView.layoutManager else { fatalError() }

        rightLayoutManager.replaceTextStorage(leftTextStorage)
    }
}
class ViewController: NSViewController {

@IBOutlet var leftTextView: NSTextView!
@IBOutlet var rightTextView: NSTextView!

override func viewDidLoad() {
    super.viewDidLoad()

    guard let leftTextStorage = leftTextView.textStorage,
        let rightLayoutManager = rightTextView.layoutManager else { fatalError() }

    rightLayoutManager.replaceTextStorage(leftTextStorage)
}
}
Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.

```swift
let string = "Hello World!"
```
Delegation

func layoutManager(_:shouldGenerateGlyphs:properties:characterIndexes:font:forGlyphRange:)

Notification

Subclassing
func layoutManager(_: layoutManager: NSLayoutManager,
shouldGenerateGlyphs glyphs: UnsafePointer<CGGlyph>,
properties props: UnsafePointer<NSLayoutManager.GlyphProperty>,
characterIndexes charIndexes: UnsafePointer<Int>,
font aFont: NSFont, forGlyphRange glyphRange: NSRange) -> Int {

}
func layoutManager(_: layoutManager: NSLayoutManager,
    shouldGenerateGlyphs glyphs: UnsafePointer<CGGlyph>,
    properties props: UnsafePointer<NSLayoutManager.GlyphProperty>,
    characterIndexes charIndexes: UnsafePointer<Int>,
    font aFont: NSFont, forGlyphRange glyphRange: NSRange) -> Int {
    var controlCharProps: UnsafeMutablePointer<NSLayoutManager.GlyphProperty>? = nil
    for index in 0..<glyphRange.length {
        if charIsMarkdownCharacter { // Determine if charIndexes[index] is markdown character
            controlCharProps?[index] = .null
        }
    }
}
func layoutManager(_ layoutManager: NSLayoutManager,
    shouldGenerateGlyphs glyphs: UnsafePointer<CGGlyph>,
    properties props: UnsafePointer<NSLayoutManager.GlyphProperty>,
    characterIndexes charIndexes: UnsafePointer<Int>,
    font aFont: NSFont, forGlyphRange glyphRange: NSRange) -> Int {
    var controlCharProps: UnsafeMutablePointer<NSLayoutManager.GlyphProperty>? = nil
    for index in 0..<glyphRange.length {
        if charIsMarkdownCharacter { // Determine if charIndexes[index] is markdown character
            controlCharProps?[index] = .null
        }
    }
    }
func layoutManager(_ layoutManager: NSLayoutManager, 
    shouldGenerateGlyphs glyphs: UnsafePointer<CGGlyph>, 
    properties props: UnsafePointer<NSLayoutManager.GlyphProperty>, 
    characterIndexes charIndexes: UnsafePointer<Int>, 
    font aFont: NSFont, forGlyphRange glyphRange: NSRange) -> Int {
    var controlCharProps: UnsafeMutablePointer<NSLayoutManager.GlyphProperty>? = nil
    for index in 0..<glyphRange.length {
        if charIsMarkdownCharacter { // Determine if charIndexes[index] is markdown character
            controlCharProps?[index] = .null
        }
    }
    if let newProps = controlCharProps {
        layoutManager.setGlyphs(glyphs, properties: newProps, characterIndexes: charIndexes, 
                                 font: aFont, forGlyphRange: glyphRange)
        return glyphRange.length
    } else { return 0 }
}
func layoutManager(_ layoutManager: NSLayoutManager, 
    shouldGenerateGlyphs glyphs: UnsafePointer<CGGlyph>, 
    properties props: UnsafePointer<NSLayoutManager.GlyphProperty>, 
    characterIndexes charIndexes: UnsafePointer<Int>, 
    font aFont: NSFont, forGlyphRange glyphRange: NSRange) -> Int {
    var controlCharProps: UnsafeMutablePointer<NSLayoutManager.GlyphProperty>? = nil
    for index in 0..<glyphRange.length {
        if charIsMarkdownCharacter { // Determine if charIndexes[index] is markdown character
            controlCharProps?[index] = .null
        }
    }
    if let newProps = controlCharProps {
        layoutManager.setGlyphs(glyphs, properties: newProps, characterIndexes: charIndexes, 
            font: aFont, forGlyphRange: glyphRange)
        return glyphRange.length
    } else { return 0 }
}
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits. **The rebels.** The troublemakers.

```
Swift Code
```
```
"swift
let string = "Hello World!"
```

Word Count: 17
Wednesday, June 6, 2018

Today I celebrated the crazy ones. The misfits.
**The rebels.** The troublemakers.

Swift Code
```
some
let string = "Hello World!"
```

Word Count: 17
Documentation

Text Programming Guide for iOS

Cocoa Text Architecture Guide

Introduction to Text Layout Programming Guide
Correctness:

Beware of default attributes
Don’t hate.
Don’t hate.
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
// Correctness: Beware of Default Attributes

```swift
    guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

    let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
    let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

    let newText = NSMutableAttributedString(string: myTextView.text)
    newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
    myTextView.attributedText = newText
```
// Correctness: Beware of Default Attributes

```swift
guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
```
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText

Don't hate.
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText

Don't hate. 😃
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText

Don't hate.
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
// Correctness: Beware of Default Attributes

guard let originalFont = UIFont(name: "Comic Sans MS", size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
Don’t hate.
Don’t hate.
Don't hate.
Don’t hate.
Don’t hate.

apply bold font

Helvatica 12
Don’t hate.
Don’t hate.
Don’t hate.
Don’t hate.
Don't hate.
// Correctness
// Approach #1: Avoid mixing plain and attributed text

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(attributedString: myTextView.attributedText)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
// Correctness
// Approach #1: Avoid mixing plain and attributed text

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(attributedString: myTextView.attributedText)
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
// Correctness
// Approach #2: Specify attributes on creation

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text,
                                      attributes: [.font : originalFont])
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
// Correctness
// Approach #2: Specify attributes on creation

guard let originalFont = UIFont(name: "Comic Sans MS" size: 24.0) else { return }

let boldFontDescriptor = originalFont.fontDescriptor.withSymbolicTraits(.traitBold)
let boldFont = UIFont(descriptor: boldFontDescriptor!, size: (originalFont.pointSize))

let newText = NSMutableAttributedString(string: myTextView.text, attributes: [.font : originalFont])
newText.addAttribute(.font, value: boldFont, range: NSRange(location: 0, length: 5))
myTextView.attributedText = newText
## Correctness

**Attributes with default values**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachment</td>
<td>None (no attachment)</td>
</tr>
<tr>
<td>Background color</td>
<td>None (no background)</td>
</tr>
<tr>
<td>Baseline offset</td>
<td>0.0</td>
</tr>
<tr>
<td>Font</td>
<td>Helvetica 12-point</td>
</tr>
<tr>
<td>Foreground color</td>
<td>Black</td>
</tr>
<tr>
<td>Kern</td>
<td>None (use font’s default kerning)</td>
</tr>
<tr>
<td>Ligature</td>
<td>None (use font’s standard ligatures)</td>
</tr>
<tr>
<td>Link</td>
<td>None (no link)</td>
</tr>
<tr>
<td>Paragraph style</td>
<td>Return value of NSParagraphStyle's defaultParagraphStyle method</td>
</tr>
<tr>
<td>Superscript</td>
<td>0</td>
</tr>
<tr>
<td>Underline style</td>
<td>None (no underline)</td>
</tr>
</tbody>
</table>
## Correctness

Attributes with default values

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachment</td>
<td>None (no attachment)</td>
</tr>
<tr>
<td>Background color</td>
<td>None (no background)</td>
</tr>
<tr>
<td>Baseline offset</td>
<td>0.0</td>
</tr>
<tr>
<td>Font</td>
<td>Helvetica 12-point</td>
</tr>
<tr>
<td>Foreground color</td>
<td>Black</td>
</tr>
<tr>
<td>Kern</td>
<td>None (use font’s default kerning)</td>
</tr>
<tr>
<td>Ligature</td>
<td>None (use font’s standard ligatures)</td>
</tr>
<tr>
<td>Link</td>
<td>None (no link)</td>
</tr>
<tr>
<td>Paragraph style</td>
<td>Return value of NSParagraphStyle’s defaultParagraphStyle method</td>
</tr>
<tr>
<td>Superscript</td>
<td>0</td>
</tr>
<tr>
<td>Underline style</td>
<td>None (no underline)</td>
</tr>
</tbody>
</table>
// Correctness : Beware of Default Attributes
// Modifying paragraph style

// Use tail truncation instead of the default word wrapping
let myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.lineBreakMode = .byTruncatingTail

// Apply paragraph style to last word
let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.paragraphStyle, value: myParagraphStyle, range: NSRange(location: 6, length: 5))
myTextView.attributedText = newText
// Correctness: Beware of Default Attributes
// Modifying paragraph style

// Use tail truncation instead of the default word wrapping
let myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.lineBreakMode = .byTruncatingTail

// Apply paragraph style to last word
let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.paragraphStyle, value: myParagraphStyle, range: NSRange(location: 6, length: 5))
myTextView.attributedText = newText
// Correctness : Beware of Default Attributes
// Modifying paragraph style

// Use tail truncation instead of the default word wrapping
let myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.lineBreakMode = .byTruncatingTail

// Apply paragraph style to last word
let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.paragraphStyle, value: myParagraphStyle,
                     range: NSRange(location: 6, length: 5))
myTextView.attributedText = newText

Don’t…
// Correctness: Beware of Default Attributes
// Modifying paragraph style

// Use tail truncation instead of the default word wrapping
let myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.lineBreakMode = .byTruncatingTail

// Apply paragraph style to last word
let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.paragraphStyle, value: myParagraphStyle,
range: NSRange(location: 6, length: 5))
myTextView.attributedText = newText

Don’t hate.
// Correctness : Beware of Default Attributes
// Modifying paragraph style

// Use tail truncation instead of the default word wrapping
let myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.lineBreakMode = .byTruncatingTail

// Apply paragraph style to last word
let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.paragraphStyle, value: myParagraphStyle, range: NSRange(location: 6, length: 5))
myTextView.attributedText = newText

Don’t hate.
Correctness: Beware of Default Attributes
Modifying paragraph style

Use tail truncation instead of the default word wrapping
let myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.lineBreakMode = .byTruncatingTail

Apply paragraph style to last word
let newText = NSMutableAttributedString(string: myTextView.text)
newText.addAttribute(.paragraphStyle, value: myParagraphStyle,
                     range: NSRange(location: 6, length: 5))
myTextView.attributedText = newText

Don’t hate.
Don’t hate.

default paragraph style  


custom paragraph style
Don’t hate.
Correctness

Best practice

Be explicit with attributes

• Avoids “resetting” all or part of string to default attributes
• Ensure correct string coloring for Dark Mode
Performance:

Use noncontiguous layout
Performance

Layout process review

- Attribute Fixing
- Glyph Generation
- Glyph Layout
- Display

Characters to glyphs
Position glyphs
The first requisite in the preparation of good sandwiches is to have perfect bread in suitable condition. Either white, brown, rye, or entire wheat bread may be used, but it should be of close, even texture and at least one day old. Cream the butter with a wooden spoon and spread smoothly on the bread before it is cut; after cutting remove the crust and avoid spreading the butter or filling over the edge. When ready to serve, cut the sandwiches either square, triangular, long, narrow, round, or crescent shaped.

In making rolled bread sandwiches, cut off the crust of a loaf of fresh bread and spread a thin layer of butter on one end of the loaf; cut off this buttered end in as thin a slice as possible and spread with the sandwich filling; roll up this slice and lay on a napkin draw the napkin firmly around the rolled bread and pin it. Put in a cool place until ready to serve, then remove napkin and tie the sandwiches with baby ribbon or fasten with a toothpick.

French rolls may be used for picnics and out-of-door luncheons. Remove from the top of each roll a piece of the crust the size of a silver dollar, and with a spoon take out the centre. Fill the space with highly seasoned.
sandwiches either square, triangular, long, narrow, round, or crescent shaped.

In making rolled bread sandwiches, cut off the crust of a loaf of fresh bread and spread a thin layer of butter on one end of the loaf; cut off this buttered end in as thin a slice as possible and spread with the sandwich filling; roll up this slice and lay on a napkin draw the napkin firmly around the rolled bread and pin it. Put in a cool place until ready to serve, then remove napkin and tie the sandwiches with baby ribbon or fasten with a toothpick.

French rolls may be used for picnics and out-of-door luncheons. Remove from the top of each roll a piece of the crust the size of a silver dollar, and with a spoon take out the centre. Fill the space with highly seasoned chopped meat, fish, lobster, or crab, replace lid, wrap in tissue paper, and serve with pickles or olives.

For very small, dainty sandwiches to be served at afternoon teas or luncheons, the bread may be baked at home in pound baking powder cans. These should be only half filled, and then allowed to rise before baking. You then have a round slice without crust.

A garnish such as the following may be used: For meat sandwiches, use pickles, olives, lettuce, watercress, parsley, and mint. For fish sandwiches, use pickles, olives, cress, parsley, slices of lemon, and hard-boiled egg. For cheese sandwiches, use pickles and olives. For sweet sandwiches, use lettuce, maiden hair fern, smilax, berries, flowers, and candied fruit.

To keep sandwiches fresh, if prepared an hour or two before serving, wring out a napkin in cold water and cover the tray and keep in a cool place or viewport.
sandwiches either square, triangular, long, narrow, round, or crescent shaped.

In making rolled bread sandwiches, cut off the crust of a loaf of fresh bread and spread a thin layer of butter on one end of the loaf; cut off this buttered end in as thin a slice as possible and spread with the sandwich filling; roll up this slice and lay on a napkin draw the napkin firmly around the rolled bread and pin it. Put in a cool place until ready to serve, then remove napkin and tie the sandwiches with baby ribbon or fasten with a toothpick.

French rolls may be used for picnics and out-of-door luncheons. Remove from the top of each roll a piece of the crust the size of a silver dollar, and with a spoon take out the centre. Fill the space with highly seasoned chopped meat, fish, lobster, or crab, replace lid, wrap in tissue paper, and serve with pickles or olives.

For very small, dainty sandwiches to be served at afternoon teas or luncheons, the bread may be baked at home in pound baking powder cans. These should be only half filled, and then allowed to rise before baking. You then have a round slice without crust.

A garnish such as the following may be used: For meat sandwiches, use pickles, olives, lettuce, watercress, parsley, and mint. For fish sandwiches, use pickles, olives, cress, parsley, slices of lemon, and hardboiled egg. For cheese sandwiches, use pickles and olives. For sweet sandwiches, use lettuce, maiden hair fern, smilax, berries, flowers, and candied fruit.

To keep sandwiches fresh, if prepared an hour or two before serving, wring out a napkin in cold water and cover the tray and keep in a cool place or

contiguous
glyph generation
and layout
sandwiches either square, triangular, long, narrow, round, or crescent shaped.

In making rolled bread sandwiches, cut off the crust of a loaf of fresh bread and spread a thin layer of butter on one end of the loaf; cut off this buttered end in as thin a slice as possible and spread with the sandwich filling; roll up this slice and lay on a napkin draw the napkin firmly around the rolled bread and pin it. Put in a cool place until ready to serve, then remove napkin and tie the sandwiches with baby ribbon or fasten with a toothpick.

French rolls may be used for picnics and out-of-door luncheons. Remove from the top of each roll a piece of the crust the size of a silver dollar, and with a spoon take out the centre. Fill the space with highly seasoned chopped meat, fish, lobster, or crab, replace lid, wrap in tissue paper, and serve with pickles or olives.

For very small, dainty sandwiches to be served at afternoon teas or luncheons, the bread may be baked at home in pound baking powder cans. These should be only half filled, and then allowed to rise before baking. You then have a round slice without crust.

A garnish such as the following may be used: For meat sandwiches, use pickles, olives, lettuce, watercress, parsley, and mint. For fish sandwiches, use pickles, olives, cress, parsley, slices of lemon, and hard-boiled egg. For cheese sandwiches, use pickles and olives. For sweet sandwiches, use lettuce, maiden hair fern, smilax, berries, flowers, and candied fruit.

To keep sandwiches fresh, if prepared an hour or two before serving, wring out a napkin in cold water and cover the tray and keep in a cool place or contiguous.
In making rolled bread sandwiches, cut off the crust of a loaf of fresh bread and spread a thin layer of butter on one end of the loaf; cut off this buttered end in as thin a slice as possible and spread with the sandwich filling; roll up this slice and lay on a napkin draw the napkin firmly around the rolled bread and pin it. Put in a cool place until ready to serve, then remove napkin and tie the sandwiches with baby ribbon or fasten with a tooth pick.

French rolls may be used for picnics and out-of-door luncheons. Remove from the top of each roll a piece of the crust the size of a silver dollar, and with a spoon take out the centre. Fill the space with highly seasoned chopped meat, fish, lobster, or crab, replace lid, wrap in tissue paper, and serve with pickles or olives.

For very small, dainty sandwiches to be served at afternoon teas or luncheons, the bread may be baked at home in pound baking powder cans. These should be only half filled, and then allowed to rise before baking. You then have a round slice without crust.

A garnish such as the following may be used: For meat sandwiches, use pickles, olives, lettuce, water-cress, parsley, and mint. For fish sandwiches, use pickles, olives, cress, parsley, slices of lemon, and hard-boiled egg. For cheese sandwiches, use pickles and olives. For sweet sandwiches, use lettuce, maiden hair fern, smilax, berries, flowers, and candied fruit.

To keep sandwiches fresh, if prepared an hour or two before serving, wring out a napkin in cold water and cover the tray and keep in a cool place or.
The first requisite in the preparation of good sandwiches is to have perfect bread in suitable condition. Either white, brown, rye, or entire wheat bread may be used, but it should be of close, even texture and at least one day old. Cream the butter with a wooden spoon and spread smoothly on the bread before it is cut; after cutting remove the crust and avoid spreading the butter or filling over the edge. When ready to serve, cut the sandwiches either square, triangular, long, narrow, round, or crescent shaped.

In making rolled bread sandwiches, cut off the crust of a loaf of fresh bread and spread a thin layer of butter on one end of the loaf; cut off this buttered end in as thin a slice as possible and spread with the sandwich filling; roll up this slice and lay on a napkin draw the napkin firmly around the rolled bread and pin it. Put in a cool place until ready to serve, then remove napkin and tie the sandwiches with baby ribbon or fasten with a tooth pick.

French rolls may be used for picnics and out-of-door luncheons. Remove from the top of each roll a piece of the crust the size of a silver dollar, and with a spoon take out the centre. Fill the space with highly seasoned
In making rolled bread sandwiches, cut off the crust of a loaf of fresh bread and spread a thin layer of butter on one end of the loaf; cut off this buttered end in as thin a slice as possible and spread with the sandwich filling; roll up this slice and lay on a napkin draw the napkin firmly around the rolled bread and pin it. Put in a cool place until ready to serve, then remove napkin and tie the sandwiches with baby ribbon or fasten with a toothpick.

French rolls may be used for picnics and out-of-door luncheons. Remove from the top of each roll a piece of the crust the size of a silver dollar, and with a spoon take out the centre. Fill the space with highly seasoned chopped meat, fish, lobster, or crab, replace lid, wrap in tissue paper, and serve with pickles or olives.

For very small, dainty sandwiches to be served at afternoon teas or luncheons, the bread may be baked at home in pound baking powder cans. These should be only half filled, and then allowed to rise before baking. You then have a round slice without crust.

A garnish such as the following may be used: For meat sandwiches, use pickles, olives, lettuce, watercress, parsley, and mint. For fish sandwiches, use pickles, olives, cress, parsley, slices of lemon, and hard-boiled egg. For cheese sandwiches, use pickles and olives. For sweet sandwiches, use lettuce, maiden hair fern, smilax, berries, flowers, and candied fruit.

To keep sandwiches fresh, if prepared an hour or two before serving, wring out a napkin in cold water and cover the tray and keep in a cool place or wrap in wax paper.
Performance
Turning on noncontiguous layout

NSTextView
Set `allowsNonContiguousLayout` property on layout manager
Performance
Turning on noncontiguous layout

UITextView
Turned on by default
Controlled by layout manager property `allowsNonContiguousLayout`
Requires scrolling enabled
Performance

Best practices

With noncontiguous layout, avoid requesting layout for the entire text at once

// With only one container, this effectively requests layout for all the text
func ensureLayout(for: NSTextContainer)

// Avoid asking for a large range that includes the end of the text
func ensureLayout(forCharacterRange: NSRange)

// Avoid asking for a large range that includes the end of the text
func boundingRect(forGlyphRange: NSRange, in: NSTextContainer)
Related Previous Sessions

Efficient Interactions with Frameworks
Security:
Defense in depth
Your app or framework + iOS, macOS, watchOS, tvOS = A stronger defense
Security:
Consider setting limits on text input
All text input is potentially untrusted
3,142,582 characters?
3,142,582 characters?
Security

Setting limits on text input

Validate text field input with:

- UITextFieldDelegate
- NSFormatter

UIKit AppKit
Security enhancements are coming
Summary

Choose the Right...
- Control
- Customization point
- Customization approach

Best Practices
- Correctness
- Security
- Performance
## More Information

https://developer.apple.com/wwdc18/221

<table>
<thead>
<tr>
<th>Lab</th>
<th>Technology Lab</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text and Fonts Lab</td>
<td>Technology Lab 11</td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>UIKit Layout and Lab</td>
<td>Technology Lab 2</td>
<td>Thursday 11:00AM</td>
</tr>
<tr>
<td>Cocoa Lab</td>
<td>Technology Lab 7</td>
<td>Friday 11:00AM</td>
</tr>
</tbody>
</table>