Improving the Full Screen Window Experience

On OS X

Session 221

Corbin Dunn  AppKit Software Engineer
Taylor Kelly  AppKit Software Engineer
Overview

Full Screen
  • Adopting full screen
  • Titlebar Accessory View Controllers
  • Full Screen Tile API

Flexible Layout
  • NSSplitViewController
  • Auto Layout and NSStackView
  • NSCollectionView
Full Screen Mode

User benefits

Focus attention on a single task
Make the most of screen real estate
Why Make Full Screen a System Feature

Per-window model

Focus on a single task

Consistent user experience

• Standard enter and exit user interface
• Standard navigation
A Full Screen Capable Window

The new MacBook
A Full Screen Capable Window

The new MacBook
A Full Screen Capable Window

The new MacBook
A Full Screen Capable Window

The new MacBook
Full Screen Checklist

The Basics
✓ Specify which windows can be made full screen
✓ Add an “Enter Full Screen” menu item

Making Your App Shine
✓ Consider auto-hiding your window’s toolbar
✓ Modify the window’s contents or layout for full screen
✓ Utilize Titlebar Accessory View Controllers
✓ Test full screen tiles
Windows in Full Screen
Primary Window

Can be made the full screen window
Main document window
Windows in Full Screen

Auxiliary Window

Generally not needed

For an app to add a window to another app’s full screen space

See AppKit release notes
Make Windows Fullscreen Capable
In Xcode / Interface Builder
Make Windows Fullscreen Capable
In Xcode / Interface Builder
Make Windows Fullscreen Capable
In Xcode / Interface Builder
Make Windows Full Screen Capable

In code

Primary windows

```javascript
window.collectionBehavior = [window.collectionBehavior, .FullScreenPrimary]
```

For auxiliary windows that can exist in another fullscreen space

```javascript
window.collectionBehavior = [window.collectionBehavior, .FullScreenAuxiliary]
```
Make Windows Full Screen Capable

Checking state

Is this window in full screen?

let inFullscreen = (window.styleMask & NSFullScreenWindowMask) != 0
Automatically Hiding the Toolbar
Default behavior is always visible
Automatically Hiding the Toolbar
Menu bar reveal also shows the titlebar
Automatically Hiding the Toolbar

Toolbar can be autohidden in full screen
Automatically Hiding the Toolbar
Menu bar reveal shows the toolbar and titlebar
func window(window: NSWindow, willUseFullScreenPresentationOptions proposedOptions: NSApplicationPresentationOptions) -> NSApplicationPresentationOptions {
    // Show and hide the toolbar together with menu bar
    return [proposedOptions, .AutoHideToolbar]
}
Custom Full Screen Animations

Custom animations aren’t needed as often
Can be implemented with the following two delegate methods:

```swift
func customWindowsToEnterFullScreenForWindow(NSWindow) -> [NSWindow]? 
func window(NSWindow, 
    startCustomAnimationToEnterFullScreenWithDuration: NSTimeInterval)
```

NOTE: These may not be called when entering a tile with Mission Control
Titlebar Accessory View Controllers
Titlebar Accessory View Controllers
For Full Screen and Normal Windows
Titlebar Accessory View Controllers
For Full Screen and Normal Windows
Titlebar Accessory View Controllers
For Full Screen and Normal Windows
Titlebar Accessory View Controllers
For Full Screen and Normal Windows

Titlebar Accessory View in Full Screen
Titlebar Accessory View Controllers
For Full Screen and Normal Windows
Titlebar Accessory View Controllers
For Full Screen and Normal Windows
Titlebar Accessory View Controllers

Default window content view area
Titlebar Accessory View Controllers

Default window content view area
Titlebar Accessory View Controllers

Full content area under the titlebar and toolbar
Titlebar Accessory View Controllers

Full content area under the titlebar and toolbar

Full content area extends to under the titlebar
Titlebar Accessory View Controllers
Full content area under the titlebar and toolbar

window.styleMask = window.styleMask | NSFullSizeContentViewWindowMask
Titlebar Accessory View Controllers

NSWindow API in 10.10

class NSTitlebarAccessoryViewController : NSViewController {

    var layoutAttribute: NSLayoutAttribute
    var fullScreenMinHeight: CGFloat

}
Titlebar Accessory View Controllers

NSWindow API in 10.10

class NSTitlebarAccessoryViewController : NSViewController {
    var layoutAttribute: NSLayoutAttribute
    var fullScreenMinHeight: CGFloat
}

![Image of a computer window with a titlebar and accessory view controllers]
Titlebar Accessory View Controllers

NSTitlebarAccessoryViewViewController

Automatically has blurring behind it
Automatically contained in an NSVisualEffectView
Automatically works in full screen
Has a semi-managed size
class NSTitlebarAccessoryViewController : NSViewController {

    var layoutAttribute: NSLayoutAttribute
    var fullScreenMinHeight: CGFloat

}
Titlebar Accessory View Controllers

Layout Attribute

accessoryViewController.layoutAttribute = NSLayoutAttribute.Bottom

- Height is the view’s height
- Width is the window’s width and automatically is set with the window
Titlebar Accessory View Controllers

Layout Attribute

accessoryViewController.layoutAttribute = NSLayoutAttribute.Bottom

- Height is the view’s height
- Width is the window’s width and automatically is set with the window
Titlebar Accessory View Controllers

Layout Attribute

accessoryViewController.layoutAttribute = NSLayoutAttribute.Right

• Width is the view’s width
• Height is the height of the complete toolbar area
  - Automatically changes when the toolbar height changes
Titlebar Accessory View Controllers

Layout Attribute

```swift
accessoryViewController.layoutAttribute = NSLayoutAttribute.Right
```

- Width is the view’s width
- Height is the height of the complete toolbar area
  - Automatically changes when the toolbar height changes
Titlebar Accessory View Controllers

Layout Attribute

accessoryViewController.layoutAttribute = NSLayoutAttribute.Left

• Appears to the right of the window buttons
• Width is the view’s width
• Height is the height of the complete toolbar area and automatically is changed as needed
• Overlaps the toolbar
Titlebar Accessory View Controllers

Layout Attribute

accessoryViewController.layoutAttribute = NSLayoutAttribute.Left

- Appears to the right of the window buttons
- Width is the view’s width
- Height is the height of the complete toolbar area and automatically is changed as needed
- Overlaps the toolbar
class NSTitlebarAccessoryViewController : NSViewController {

    var layoutAttribute: NSLayoutAttribute
    var fullScreenMinHeight: CGFloat

}
Titlebar Accessory View Controllers

Full screen minimum height

Only applicable when the layoutAttribute is .Bottom

The minimum height is used when the menu bar is hidden

The accessory view automatically appears under the toolbar when the menu is shown

The default height is 0

• Completely hidden when the menu bar is not visible
Titlebar Accessory View Controllers

Full screen minimum height = 0
Titlebar Accessory View Controllers

Full screen minimum height = 0
Titlebar Accessory View Controllers

Full screen minimum height = 0
accessoryViewController.fullScreenMinHeight replaces the NSToolbar API:

```swift
extension NSToolbar {
    var fullScreenAccessoryView: NSView?
    var fullScreenAccessoryViewMinHeight: CGFloat
    var fullScreenAccessoryViewMaxHeight: CGFloat
}
```
Titlebar Accessory View Controllers
NSWindow API in 10.10

```swift
var titlebarAccessoryViewControllers: [NSTitlebarAccessoryViewController]

func addTitlebarAccessoryViewController(NSTitlebarAccessoryViewController)

func insertTitlebarAccessoryViewController(NSTitlebarAccessoryViewController, atIndex: Int)

func removeTitlebarAccessoryViewControllerAtIndex(Int)
```
Titlebar Accessory View Controllers

NSWindow API in 10.10

var titlebarAccessoryViewControllers: [NSTitlebarAccessoryViewController]

func addTitlebarAccessoryViewController(NSTitlebarAccessoryViewController)

func insertTitlebarAccessoryViewController(NSTitlebarAccessoryViewController, atIndex: Int)

func removeTitlebarAccessoryViewControllerAtIndex(Int)
Titlebar Accessory View Controllers

NSWindow API in 10.10

var titlebarAccessoryViewControllers: [NSTitlebarAccessoryViewController]

func addTitlebarAccessoryViewController(NSTitlebarAccessoryViewController)

func insertTitlebarAccessoryViewController(NSTitlebarAccessoryViewController, atIndex: Int)

func removeTitlebarAccessoryViewControllerAtIndex(Int)
Titlebar Accessory View Controllers

NSWindow API in 10.10

```swift
var titlebarAccessoryViewControllers: [NSTitlebarAccessoryViewController]

func addTitlebarAccessoryViewController(NSTitlebarAccessoryViewController)

func insertTitlebarAccessoryViewController(NSTitlebarAccessoryViewController, atIndex: Int)

func removeTitlebarAccessoryViewControllerAtIndex(Int)
```
Titlebar Accessory View Controllers
NSWindow API in 10.10

var titlebarAccessoryViewControllers: [NSTitlebarAccessoryViewController]

func addTitlebarAccessoryViewController(NSTitlebarAccessoryViewController)

func insertTitlebarAccessoryViewController(NSTitlebarAccessoryViewController, atIndex: Int)

func removeTitlebarAccessoryViewControllerAtIndex(Int)
Titlebar Accessory View Controllers

Basic example

Adding:

```swift
window.addTitlebarAccessoryViewController(accessoryViewController)
```

Removing:

```swift
accessoryViewController.removeFromParentViewController()
```
Full Screen Tiles
Full Screen Tiles

User benefits

Focus attention single task that may involve multiple windows
Still make the most of screen real estate
Full Screen Tiles

User benefits

Focus attention single task that may involve multiple windows
Still make the most of screen real estate
Supporting Full Screen Tiles

Our most personal device yet.

To make the best use of its size and location on your wrist, Apple Watch has all-new interactions and technologies. They let you do familiar things more quickly and conveniently. As well as some things that simply weren't possible before. So using it is a whole new experience. One that's more personal than ever.

Learn more
Supporting Full Screen Tiles

Our most personal device yet.

To make the best use of its size and location on your wrist, Apple Watch has all-new interactions and technologies. They let you do familiar things more quickly and conveniently. As well as some things that simply weren’t possible before. So using it is a whole new experience. One that’s more personal than ever.

Learn more >
Supporting Full Screen Tiles

What is allowed to tile in full screen?

Windows can automatically be full screen tiles when

• The window must be resizable
• The window can not be a panel

Heuristics may change and evolve over time
Supporting Full Screen Tiles
What is allowed to tile in full screen?

Windows can explicitly allow full screen tiling with:

```javascript
window.collectionBehavior =
    [window.collectionBehavior, .FullScreenAllowsTiling]
```

Windows can explicitly disallow full screen tiling with:

```javascript
window.collectionBehavior =
    [window.collectionBehavior, .FullScreenDisallowsTiling]
```
Supporting Full Screen Tiles

If allowed to tile, can it fit?

The minimum and maximum values are

• Automatically determined by auto layout
• Determined by the window’s \texttt{minSize} and \texttt{maxSize}
  - Or \texttt{contentMinSize} / \texttt{contentMaxSize}, depending on which is used
• The values should be the same when in full screen and when not in full screen
Supporting Full Screen Tiles
If allowed to tile, can it fit?

If a full screen window layout is significantly different from non full screen
Supporting Full Screen Tiles

If allowed to tile, can it fit?

If a full screen window layout is significantly different from non full screen
Supporting Full Screen Tiles
If allowed to tile, can it fit?

If a full screen window layout is significantly different from non full screen...
Supporting Full Screen Tiles

If allowed to tile, can it fit?

If a full screen window layout is significantly different from non full screen.
func windowWillEnterFullScreen(notification: NSNotification) {
    tickerTextField.hidden = false
    bottomConstraint.priority = 100
}

func windowWillExitFullScreen(notification: NSNotification) {
    tickerTextField.hidden = true
    bottomConstraint.priority = 750
}
Supporting Full Screen Tiles
Customize the size limits in full screen

Optional: set the NSWindow minimum and maximum full screen tile sizes with

```swift
var minFullScreenContentSize: NSSize
var maxFullScreenContentSize: NSSize
```

ONLY use these when the window’s UI is different in full screen
This must be done as early as possible
• Before the window is in full screen or a tile

Can be updated at anytime
• Including when the window is in full screen or a tile

OS X will only allow tiles that fit together and don’t exceed the screen width
Supporting Full Screen Tiles
Customize the size limits in full screen

windowA.minFullScreenContentSize = (1200, 100)
Supporting Full Screen Tiles

Customize the size limits in full screen

windowB.minFullScreenContentSize = (1300, 100)
Supporting Full Screen Tiles
Not allowed together on a small display

Avoid setting a large minFullScreenContentSize
Flexible Layout

A window’s a window, no matter how small

Taylor Kelly AppKit Software Engineer
Flexible Layout
Flexible Layout
Flexible Layout

```
<table>
<thead>
<tr>
<th>Width</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1280</td>
<td>800</td>
</tr>
</tbody>
</table>
```
Flexible Layout
Flexible Layout
Flexible Layout

Auto Layout Preface
Flexible Layout

Auto Layout Preface

Sidebars with NSSplitViewController
Flexible Layout

Auto Layout Preface
Sidebars with NSSplitViewController
NSStackView
Flexible Layout

Auto Layout Preface
Sidebars with NSSplitViewController
NSStackView
NSCollectionView
Auto Layout
Auto Layout

Powerful, constraint-based layout system
Auto Layout

Powerful, constraint-based layout system

Natural expressions of relationships
Auto Layout

Powerful, constraint-based layout system
Natural expressions of relationships
Priorities establish precedence between constraints
Auto Layout

Powerful, constraint-based layout system
Natural expressions of relationships
Priorities establish precedence between constraints

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mysteries of Auto Layout, Part 1</td>
<td>Presidio</td>
<td>Thursday 11:00AM</td>
</tr>
<tr>
<td>Mysteries of Auto Layout, Part 2</td>
<td>Presidio</td>
<td>Thursday 1:30PM</td>
</tr>
</tbody>
</table>
Auto Layout

Priorities

Wide Button  Centered Label  Thin
Auto Layout

Priorities

Wide Button

Centered Label

Thin
Auto Layout

Priorities
Auto Layout

Priorities

Wide Button >= Centered Label

Thin
Auto Layout

Priorities

Wide Button → Centered Label

Thin
Auto Layout
Priorities

Wide Button >= Centered Label

Thin
Auto Layout

Priorities

Wide Button >= Centered Label

Thin
Auto Layout

Priorities

Wide Button >= Centered Label >= Thin
Auto Layout

Priorities
Auto Layout

Priorities

Wide Button  Centered Label  Thin
Auto Layout

Priorities

class NSLayoutConstraint {
    var priority: NSLayoutPriority
Auto Layout

Priorities

```swift
class NSLayoutConstraint {
    var priority: NSLayoutPriority
```
class NSLayoutConstraint {
    var priority: NSLayoutPriority
}

NSLayoutPriorityRequired = 1000.0
Auto Layout

Priorities

class NSLayoutConstraint {
    var priority: NSLayoutPriority

    NSLayoutPriorityRequired = 1000.0

    NSLayoutPriorityDefaultLow = 250.0
class NSLayoutConstraint {
    var priority: NSLayoutPriority

    NSLayoutPriorityRequired = 1000.0

    NSLayoutPriorityDefaultLow = 250.0
    NSLayoutPriorityDragThatCannotResizeWindow = 490.0
Auto Layout

Priorities

class NSLayoutConstraint {
    var priority: NSLayoutPriority

    NSLayoutPriorityRequired = 1000.0
    NSLayoutPriorityDefaultLow = 250.0
    NSLayoutPriorityDragThatCannotResizeWindow = 490.0
    NSLayoutPriorityWindowSizeStayPut = 500.0
class NSLayoutConstraint {
    var priority: NSLayoutPriority

    NSLayoutPriorityRequired = 1000.0
    NSLayoutPriorityDefaultLow = 250.0
    NSLayoutPriorityDragThatCannotResizeWindow = 490.0
    NSLayoutPriorityWindowSizeStayPut = 500.0
    NSLayoutPriorityDragThatCanResizeWindow = 510.0
Auto Layout

Priorities

class NSLayoutConstraint {
    var priority: NSLayoutPriority
}

NSLayoutPriorityRequired = 1000.0

NSLayoutPriorityDefaultLow = 250.0
NSLayoutPriorityDragThatCannotResizeWindow = 490.0
NSLayoutPriorityWindowSizeStayPut = 500.0
NSLayoutPriorityDragThatCanResizeWindow = 510.0
NSLayoutPriorityDefaultHigh = 750.0
NSSplitViewController
NSSplitViewController

Container NSViewController introduced in 10.10
NSSplitViewController

Container NSViewController introduced in 10.10
Manages an NSSplitView
NSSplitViewController

Container NSViewController introduced in 10.10
Manages an NSSplitView

NSSplitViewItem
• holdingPriority
• collapsed
• Animated collapses
NSSplitViewController

Container NSViewController introduced in 10.10
Manages an NSSplitView

NSSplitViewItem
• holdingPriority
• collapsed
• Animated collapses
NSSplitViewController
New in 10.11
NSSplitViewController

New in 10.11

Sidebars
NSSplitViewController
New in 10.11

Sidebars
Spring Loading
NSSplitViewController
New in 10.11

Sidebars

Spring Loading

Metrics
• Minimum / maximum thickness
• Proportional resizing
• Automatic maximum thickness
• Preferred thickness fraction
The epicenter of change.

WWDC15

Apple Worldwide Developers Conference.
June 8-12, San Francisco.

Apple's renowned developer community will come together at WWDC to learn about the future of iOS and OS X.
The epicenter of change.

WWDC15

Apple Worldwide Developers Conference.
June 8-12, San Francisco.
Apple's renowned developer community will come together at WWDC to learn about the future of iOS and OS X.
Sidebars

class NSSplitViewItem {
    init(sidebarWithViewController: NSViewController)
class NSSplitViewItem {
    init(sidebarWithViewController: NSViewController)
class NSSplitViewItem {
    init(sideBarWithViewController: NSViewController)
}

Material background and vibrant divider
class NSSplitViewItem {
    init(sidebarWithViewController: NSViewController)
}

Material background and vibrant divider
Auto-collapse and -uncollapse

Sidebars
class NSSplitViewItem {
    init(sidebarWithViewController: NSViewController)
}

Material background and vibrant divider
Auto-collapse and -uncollapse
Overlays
class NSSplitViewItem {
    init(sidebarWithViewController: NSViewController)

    Material background and vibrant divider
    Auto-collapse and -uncollapse
    Overlays
    Standard metrics
    minimumThickness, maximumThickness
    preferredThicknessFraction
    springLoaded = true
    canCollapse = true
class NSSplitViewController {
    @IBAction func toggleSidebar(AnyObject?)
}
class NSSplitViewController {
    @IBAction func toggleSidebar(AnyObject?)

class NSSplitViewController {
    @IBAction func toggleSidebar(AnyObject?)
}
Spring Loading
Spring Loading
class NSSplitViewItem {
    var springLoaded: Bool
class NSSplitViewItem {
    var springLoaded: Bool

    Transiently uncollapse during drags
class NSSplitViewItem {
    var springLoaded: Bool

    Transiently uncollapse during drags
    Both sidebars and non-sidebars
class NSSplitViewItem {
    var minimumThickness: CGFloat
    var maximumThickness: CGFloat
    var holdingPriority: NSLayoutPriority
    var preferredThicknessFraction: CGFloat
    var automaticMaximumThickness: CGFloat
}

class NSSplitViewController {
    var minimumThicknessForInlineSidebars: CGFloat
}
class NSSplitViewItem {
    var minimumThickness: CGFloat
    var maximumThickness: CGFloat
}
class NSSplitViewItem {
    var minimumThickness: CGFloat
    var maximumThickness: CGFloat
class NSSplitViewItem {
    var minimumThickness: CGFloat
    var maximumThickness: CGFloat
}

Metrics

≥ 140
≥ 390
≤ 300

MacBook
Light. Years ahead.
class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority
}

(class constraint priority)
class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority

    (constraint priority)
Metrics

class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority

    (constraint priority)
}
Metrics

class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority

    (constraint priority)

--220 @260  --740 @250

MacBook
Light. Years ahead.
class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority

    (constraint priority)
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority

(constraint priority)
class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority

    (constraint priority)
class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority

    (constraint priority)

MacBook
Light. Years ahead.

==270 @250
==910 @250
class NSSplitViewItem {
    var holdingPriority: NSLayoutPriority

    // constraint priority
}

Metrics

MacBook
Light. Years ahead.

==220 @250
==740 @250
class NSSplitViewItem {
    var preferredThicknessFraction: CGFloat
    var automaticMaximumThickness: CGFloat
    (percentage)
    (points)
class NSSplitViewItem {
    var preferredThicknessFraction: CGFloat
    var automaticMaximumThickness: CGFloat
}

Metrics

Apple MacBook
Light. Years ahead.

15%
class NSSplitViewItem {
    var preferredThicknessFraction: CGFloat
    var automaticMaximumThickness: CGFloat
    
    (percentage)
    (points)
class NSSplitViewItem {
    var preferredThicknessFraction: CGFloat
    var automaticMaximumThickness: CGFloat

    Metrics

    15% ≤ 280

    MacBook
    Light. Years ahead.
class NSSplitViewController {
    var minimumThicknessForInlineSidebars: CGFloat

    Metrics
class NSSplitViewController {
    var minimumThicknessForInlineSidebars: CGFloat
}
class NSSplitViewController {
    var minimumThicknessForInlineSidebars: CGFloat

    (points)
class NSSplitViewController {
    var minimumThicknessForInlineSidebars: CGFloat
}
class NSSplitViewController {
    var minimumThicknessForInlineSidebars: CGFloat
}
class NSSplitViewController {
    var minimumThicknessForInlineSidebars: CGFloat
}

MacBook
Light. Years ahead.
NSSplitView
Arranged subviews
NSSplitView
Arranged subviews

Pre-10.11, all subviews were treated as split panes
NSSplitView

Arranged subviews

Pre-10.11, all subviews were treated as split panes

class NSView {
    var subviews: [NSView]
    func addSubview(NSView)
    func addSubview(NSView, positioned: NSWindowOrderingMode, relativeTo: NSView?)
    func removeFromSuperview()
    func removeFromSuperview()
NSSplitView

Arranged subviews

Pre-10.11, all subviews were treated as split panes

class NSView {
    var subviews: [NSView]
    func addSubview(NSView)
    func addSubview(NSView, positioned: NSWindowOrderingMode, relativeTo: NSView?)
    func removeFromSuperview()

    Cannot add subviews that are not split panes
NSSplitView

Arranged subviews

Pre-10.11, all subviews were treated as split panes

```swift
class NSView {
    var subviews: [NSView]
    func addSubview(NSView)
    func addSubview(NSView, positioned: NSWindowOrderingMode, relativeTo: NSView?)
    func removeFromSuperview()
}
```

Cannot add subviews that are not split panes — dividers
NSSplitView
Arranged subviews

Pre-10.11, all subviews were treated as split panes

class NSView {
    var subviews: [NSView]
    func addSubview(NSView)
    func addSubview(NSView, positioned: NSWindowOrderingMode, relativeTo: NSView?)
    func removeFromSuperview()
}

Cannot add subviews that are not split panes — dividers
z-order vs arranged order
NSSplitView
Arranged subviews
NSSplitView

Arranged subviews

With 10.11, subviews are designated as arranged
NSSplitView
Arranged subviews

With 10.11, subviews are designated as arranged

class NSSplitView {
    var arrangedSubviews: [NSView]
    func addArrangedSubview(NSView)
    func insertArrangedSubview(NSView, atIndex: NSInteger)
    func removeArrangedSubview(NSView)
    var arrangesAllSubviews: Bool
NSSplitView

Arranged subviews

With 10.11, subviews are designated as arranged

class NSSplitView {
    var arrangedSubviews: [NSView]
    func addArrangedSubview(NSView)
    func insertArrangedSubview(NSView, atIndex: NSInteger)
    func removeArrangedSubview(NSView)

    var arrangesAllSubviews: Bool

    arrangesAllSubviews defaults to true, matching legacy behavior
    subviews == arrangedSubviews
With 10.11, subviews are designated as arranged

```swift
class NSSplitView {
    var arrangedSubviews: [NSView]
    func addArrangedSubview(NSView)
    func insertArrangedSubview(NSView, atIndex: NSInteger)
    func removeArrangedSubview(NSView)

    var arrangesAllSubviews: Bool

    arrangesAllSubviews defaults to true, matching legacy behavior
    subviews == arrangedSubviews

    arrangedSubviews is always a subset of subviews
}
NSSplitView

Arranged subviews

With 10.11, subviews are designated as arranged

class NSSplitView {
    var arrangedSubviews: [NSView]
    func addArrangedSubview(NSView)
    func insertArrangedSubview(NSView, atIndex: NSInteger)
    func removeArrangedSubview(NSView)

    var arrangesAllSubviews: Bool

    arrangesAllSubviews defaults to true, matching legacy behavior
    subviews == arrangedSubviews

    arrangedSubviews is always a subset of subviews

    Setting arrangesAllSubviews to false allows NSSplitView to use divider views
** NSSplitView  
Arranged subviews 

`arrangesAllSubviews` defaults to `true`, matching legacy behavior

subviews == arrangedSubviews 

`arrangedSubviews` is always a subset of `subviews`

Setting `arrangesAllSubviews` to `false` allows NSSplitView to use divider views
**NSSplitView**

**Arranged subviews**

arrangesAllSubviews defaults to `true`, matching legacy behavior

```
subviews == arrangedSubviews
```

arrangedSubviews is always a subset of subviews

Setting `arrangesAllSubviews` to `false` allows NSSplitView to use divider views
NSSplitView Debugging

(lldb) po splitView
NSSplitView

Debugging

(lldb) po splitView

<NSSplitView:0xea210 delegate="(CalUISplitViewController)0xeeb00" layout="constraints", dividers="views", arrangesAllSubviews="no">
NSSplitView

Debugging

(lldb) po splitView

<NSSplitView:0xea210 delegate="(CalUISplitViewController)0xeerb00" layout="constraints", dividers="views", arrangesAllSubviews="no">

layout =
- constraints
- resizeSubviews
- resizeSubviews-autoResizingConstraints
NSSplitView

Debugging

(lldb) po splitView

<NSSplitView:0xea210 delegate="(CalUISplitViewController)0xeeb00" layout="constraints", dividers="views", arrangesAllSubviews="no">

layout =
  • constraints
  • resizeSubviews
  • resizeSubviews-autoResizingConstraints

dividers =
  • views
  • layers
  • drawRect
  • notDrawn
NSSplitView

Debugging

(lldb) po splitView.debugReasonForLayoutMode
NSSplitView

Debugging

(lldb) po splitView.debugReasonForLayoutMode

The split view delegate overrides the following method(s) which prevent using constraints:

...
NSSplitView

Debugging

(lldb) po splitView.debugReasonForLayoutMode

The split view delegate overrides the following method(s) which prevent using constraints:

... 

The split view does not have an autolayout engine (yet) -- it does not use constraints
NSSplitView

Debugging

(lldb) po splitView.debugReasonForLayoutMode

The split view delegate overrides the following method(s) which prevent using constraints:
   ...

The split view does not have an autolayout engine (yet) -- it does not use constraints

The split view has a layout engine and is otherwise not prevented from using constraints -- it uses constraints
NSSplitView

Debugging

(lldb) po splitView.debugReasonForLayoutMode

The split view delegate overrides the following method(s) which prevent using constraints:

... 

The split view does not have an autolayout engine (yet) -- it does not use constraints

The split view has a layout engine and is otherwise not prevented from using constraints -- it uses constraints

The split view is owned by an NSSplitViewController -- it uses constraints
NSSplitView

Debugging

(lldb) po splitView.constraints
NSSplitView

Debugging

(lldb) po splitView.constraints

<NSLayoutConstraint H:[splitView(>=639)]>
<NSLayoutConstraint V:[splitView(>=503)]>
<NSLayoutConstraint H:|-(0)-[sidebar]>
<NSLayoutConstraint H:[content]-(0)-|>
<NSLayoutConstraint V:-(0)-[sidebar]>
<NSLayoutConstraint sidebar.bottom == splitView.bottom>
<NSLayoutConstraint V:-(0)-[content]>
<NSLayoutConstraint content.bottom == splitView.bottom>
<NSLayoutConstraint H:[sidebar]-(0)-[vibrantDivider1]>
<NSLayoutConstraint H:[vibrantDivider1]-(0)-[content]>
<NSLayoutConstraint V:-(0)-[vibrantDivider1]>
<NSLayoutConstraint vibrantDivider1.bottom == splitView.bottom>
<NSLayoutConstraint V:-(0)-[divider2]>
<NSLayoutConstraint divider2.bottom == splitView.bottom>
<NSLayoutConstraint H:-(0)-[searchbar]>
<NSLayoutConstraint V:-(0)-[searchbar]>
<NSLayoutConstraint H:[searchbar(205@999.99)] priority:999.99>
<NSLayoutConstraint V:[searchbar(0@999.99)] priority:999.99>
<NSLayoutConstraint H:[sidebar(158@250)] priority:250>
<NSLayoutConstraint H:[content(776@251)] priority:251>
<NSLayoutConstraint H:[searchbar(205@250)] priority:250>
NSSplitView

Debugging

(lldb) po splitView.constraints

<(LayoutConstraint H:[splitView(>=639)])>
<LayoutConstraint V:[splitView(>=503)]>
LayoutConstraint 'NSSplitView.Edge.Leading' H:|-(@0)-[sidebar]>
LayoutConstraint 'NSSplitView.Edge.Trailing' H:[content]-(@0)-|>
LayoutConstraint 'NSSplitView.Edge.Top.0' V:|-(@0)-[sidebar]>
LayoutConstraint 'NSSplitView.Edge.Bottom.0' sidebar.bottom == splitView.bottom>
LayoutConstraint 'NSSplitView.Edge.Top.1' V:|-(@0)-[content]>
LayoutConstraint 'NSSplitView.Edge.Bottom.1' content.bottom == splitView.bottom>
LayoutConstraint 'NSSplitView.Stack.0-d0' H:[sidebar]-(@0)-[vibrantDivider1]>
LayoutConstraint 'NSSplitView.Stack.d0-1' H:[vibrantDivider1]-(@0)-[content]>
LayoutConstraint 'NSSplitView.Divider.Edge.Top.1' V:|-(@0)-[vibrantDivider1]>
LayoutConstraint 'NSSplitView.Divider.Edge.Bottom.1' vibrantDivider1.bottom == splitView.bottom>
LayoutConstraint 'NSSplitView.Divider.Edge.Top.2' V:|-(@0)-[divider2]>
LayoutConstraint 'NSSplitView.Divider.Edge.Bottom.2' divider2.bottom == splitView.bottom>
LayoutConstraint 'NSSplitView.Collapsed.2.MinX' H:|-(@0)-[searchbar]>
LayoutConstraint 'NSSplitView.Collapsed.2.MinY' V:|-(@0)-[searchbar]>
LayoutConstraint 'NSSplitView.Collapsed.2.Width' H:[searchbar(205@999.99)] priority:999.99>
LayoutConstraint 'NSSplitView.Collapsed.2.Height' V:[searchbar(0@999.99)] priority:999.99>
LayoutConstraint 'NSSplitView.PreferredSize.0' H:[sidebar(158@250)] priority:250>
LayoutConstraint 'NSSplitView.PreferredSize.1' H:[content(776@251)] priority:251>
LayoutConstraint 'NSSplitView.PreferredSize.2' H:[searchbar(205@250)] priority:250>
NSSplitView

Debugging

(lldb) po splitView.constraints

<NSLayoutConstraint H:[splitView(>=639)]>
<NSLayoutConstraint V:[splitView(>=503)]>
<NSLayoutConstraint 'NSSplitView.Edge.Leading' H:|-0-[sidebar]>
<NSLayoutConstraint 'NSSplitView.Edge.Trailing' H:[content]-(0)-|>
<NSLayoutConstraint 'NSSplitView.Edge.Top.0' V:|-0-[sidebar]>
<NSLayoutConstraint 'NSSplitView.Edge.Bottom.0' sidebar.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Edge.Top.1' V:|-0-[content]>
<NSLayoutConstraint 'NSSplitView.Edge.Bottom.1' content.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Stack.0-d0' H:[sidebar]-(0)-[vibrantDivider1]>
<NSLayoutConstraint 'NSSplitView.Stack.d0-1' H:[vibrantDivider1]-[0]-[content]>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Top.1' V:|-0-[vibrantDivider1]>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Bottom.1' vibrantDivider1.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Top.2' V:|-0-[divider2]>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Bottom.2' divider2.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.MinX' H:|-0-[searchbar]>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.MinY' V:|-0-[searchbar]>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.Width' H:[searchbar(205@999.99)] priority:999.99>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.Height' V:[searchbar(0@999.99)] priority:999.99>
<NSLayoutConstraint 'NSSplitView.PreferredSize.0' H:[sidebar(158@250)] priority:250>
<NSLayoutConstraint 'NSSplitView.PreferredSize.1' H:[content(776@251)] priority:251>
<NSLayoutConstraint 'NSSplitView.PreferredSize.2' H:[searchbar(205@250)] priority:250>
NSSplitView
Debugging

(lldb) po splitView.constraints

<NSLayoutConstraint H:[splitView(>=639)]>
<NSLayoutConstraint V:[splitView(>=503)]>
<NSLayoutConstraint 'NSSplitView.Edge.Leading' H:|-(0)-[sidebar]>
<NSLayoutConstraint 'NSSplitView.Edge.Trailing' H:[content]-(0)-|>
<NSLayoutConstraint 'NSSplitView.Edge.Top.0' V:|-(0)-[sidebar]>
<NSLayoutConstraint 'NSSplitView.Edge.Bottom.0' sidebar.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Edge.Top.1' V:|-(0)-[content]>
<NSLayoutConstraint 'NSSplitView.Edge.Bottom.1' content.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Stack.0-d0' H:[sidebar]-(0)-[vibrantDivider1]>
<NSLayoutConstraint 'NSSplitView.Stack.d0-1' H:[vibrantDivider1]-(0)-[content]>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Top.1' V:|-(0)-[vibrantDivider1]>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Bottom.1' vibrantDivider1.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Top.2' V:|-(0)-[divider2]>
<NSLayoutConstraint 'NSSplitView.Divider.Edge.Bottom.2' divider2.bottom == splitView.bottom>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.MinX' H:|-(0)-[searchbar]>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.MinY' V:|-(0)-[searchbar]>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.Width' H:[searchbar(205@999.99)] priority:999.99>
<NSLayoutConstraint 'NSSplitView.Collapsed.2.Height' V:[searchbar(0@999.99)] priority:999.99>
<NSLayoutConstraint 'NSSplitView.PreferredSize.0' H:[sidebar(158@250)] priority:250>
<NSLayoutConstraint 'NSSplitView.PreferredSize.1' H:[content(776@251)] priority:251>
<NSLayoutConstraint 'NSSplitView.PreferredSize.2' H:[searchbar(205@250)] priority:250>
NSStackView
NSStackView

Horizontal / vertical stacks of views
NSStackView

Horizontal / vertical stacks of views
Built with constraints
NSStackView

Horizontal / vertical stacks of views
Built with constraints
Alignment and distribution
NSStackView

Horizontal / vertical stacks of views
Built with constraints
Alignment and distribution
Clipping and detaching behavior
NSStackView

Horizontal / vertical stacks of views
Built with constraints
Alignment and distribution
Clipping and detaching behavior
Performance improvements in 10.11
NSStackView

Horizontal / vertical stacks of views
Built with constraints
Alignment and distribution
Clipping and detaching behavior
Performance improvements in 10.11
NSStackView
Distributions
enum NSStackViewDistribution : Int {
    case GravityAreas
    case Fill
    case FillEqually
    case FillProportionally
    case EqualSpacing
    case EqualCentering
}
enum NSStackViewDistribution : Int {
    case GravityAreas
    case Fill
    case FillEqually
    case FillProportionally
    case EqualSpacing
    case EqualCentering
}

class NSStackView {
    var distribution: NSStackViewDistribution
NSStackView
.GravityAreas

Leading  Leading  Center  Trailing
NSStackView
.GravityAreas

Leading  Leading  Center  Trailing
NSStackView
.Fill

Big Rect  Even Bigger Rect  1  Tiny
NSStackView
.Fill

Big Rect | Even Bigger Rect | 1 | Tiny
NSStackView
.FillEqually

Big Rect  Even Bigger Rect  1  Tiny
NSStackView
.FillEqually

Big Rect    Even Bigger Rect    1    Tiny
NSStackView
.FillProportionally
NSStackView
.FillProportionally
NSStackView
.EqualSpacing

Big Rect  Even Bigger Rect  1  Tiny
NSStackView

.EqualSpacing
NSStackView
.EqualSpacing
NSStackView
.EqualCentering

Big Rect  Even Bigger Rect  1  Tiny
NSStackView
.EqualCentering
NSStackView
.EqualCentering

Big Rect  Even Bigger Rect  1  Tiny
NSStackView
Visibility Priorities
NSStackView

Visibility Priorities

1 2 3 4 5 6 7 8
NSStackView
Visibility Priorities

stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
NSStackView
Visibility Priorities

```
stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
```
NSStackView
Visibility Priorities

stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
stackView.setVisibilityPriority(priority, forView: aView)
NSStackView
Visibility Priorities

stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
stackView.setVisibilityPriority(priority, forView: aView)
stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
stackView.setVisibilityPriority(priority, forView: aView)
NSStackView
Visibility Priorities

1 2 3 4 5

stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
stackView.setVisibilityPriority(priority, forView: aView)
NSStackView
Visibility Priorities

stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
stackView.setVisibilityPriority(priority, forView: aView)
NSStackView
Visibility Priorities

stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
stackView.setVisibilityPriority(priority, forView: aView)

func stackView(NSStackView, willDetachViews: [NSView])
func stackView(NSStackView, didReattachViews: [NSView])
NSCollectionView
NSCollectionView

Efficient and scalable presentation of collections
NSCollectionView

Efficient and scalable presentation of collections
Powerful layout support
NSCollectionView

Efficient and scalable presentation of collections

Powerful layout support

• NSCollectionViewGridLayout
NSCollectionView

Efficient and scalable presentation of collections
Powerful layout support
  • NSCollectionViewGridLayout
  • NSCollectionViewFlowLayout
NSCollectionView

Efficient and scalable presentation of collections

Powerful layout support

• NSCollectionViewGridLayout
• NSCollectionViewFlowLayout
• Custom Layouts
NSCollectionView

Efficient and scalable presentation of collections
Powerful layout support
• NSCollectionViewGridLayout
• NSCollectionViewFlowLayout
• Custom Layouts
Sample Code
Exhibition — An Adaptive OS X App

Exhibition
Exhibition

Full Screen API
Exhibition

Full Screen API
Exhibition

Full Screen API
Exhibition

Full Screen API

NSSplitViewController
Exhibition

Full Screen API

NSSplitViewController
Exhibition

Full Screen API

NSSplitViewController
Exhibition

Full Screen API

NSSplitViewController

 NSScrollView

Exhibition

Full Screen API

NSSplitViewController

NSScrollView

Antelope Canyon

<table>
<thead>
<tr>
<th>Edit</th>
<th>Print</th>
<th>Share</th>
<th>Enhance</th>
<th>Colorize</th>
<th>Maximize</th>
<th>Miniaturize</th>
<th>Tranquilize</th>
</tr>
</thead>
</table>

Table showing various options for Antelope Canyon.
Exhibition

Full Screen API

NSSplitViewController

NSScrollView
Exhibition

Full Screen API

NSSplitViewController

NSScrollView
Exhibition

Full Screen API

NSSplitViewController

NSStackView

NSCollectionView
Exhibition

Full Screen API

NSSplitViewController

NSStackView

NSCollectionView
Summary
Summary
Summary

Full Screen

• Adopting full screen
Summary

Full Screen

• Adopting full screen

• Titlebar Accessory View Controllers
Summary

Full Screen

- Adopting full screen
- Titlebar Accessory View Controllers
- Full Screen Tile API
Summary

Full Screen
- Adopting full screen
- Titlebar Accessory View Controllers
- Full Screen Tile API

Flexible Layout
Summary

Full Screen
  • Adopting full screen
  • Titlebar Accessory View Controllers
  • Full Screen Tile API

Flexible Layout
  • NSSplitViewController
Summary

Full Screen
• Adopting full screen
• Titlebar Accessory View Controllers
• Full Screen Tile API

Flexible Layout
• NSSplitViewController
• Auto Layout and NSStackView
Summary

Full Screen
- Adopting full screen
- Titlebar Accessory View Controllers
- Full Screen Tile API

Flexible Layout
- NSSplitViewController
- Auto Layout and NSStackView
- NSCollectionView
More Information

Documentation
Cocoa Documentation

Technical Support
Apple Developer Forums
http://developer.apple.com/forums

General Inquiries
Paul Marcos, App Frameworks Evangelist
pmarcos@apple.com
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mysteries of Auto Layout, Part 1</td>
<td>Presidio</td>
<td>Thursday 1:00AM</td>
</tr>
<tr>
<td>Mysteries of Auto Layout, Part 2</td>
<td>Presidio</td>
<td>Thursday 1:30PM</td>
</tr>
<tr>
<td>What's New in NSCollectionView</td>
<td>Mission</td>
<td>Thursday 4:30PM</td>
</tr>
</tbody>
</table>
## Labs

<table>
<thead>
<tr>
<th>Lab</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Builder and Auto Layout Lab</td>
<td>Developer Tools Lab C</td>
<td>Thursday 2:30PM</td>
</tr>
<tr>
<td>Cocoa and Full Screen Support Lab</td>
<td>Frameworks Lab D</td>
<td>Thursday 3:30PM</td>
</tr>
<tr>
<td>Cocoa and NSCollectionView Lab</td>
<td>Frameworks Lab B</td>
<td>Friday 9:00AM</td>
</tr>
</tbody>
</table>