App Frameworks #WWDC15

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

A Full Screen Capable Window

A Full Screen Capable Window

A Full Screen Capable Window

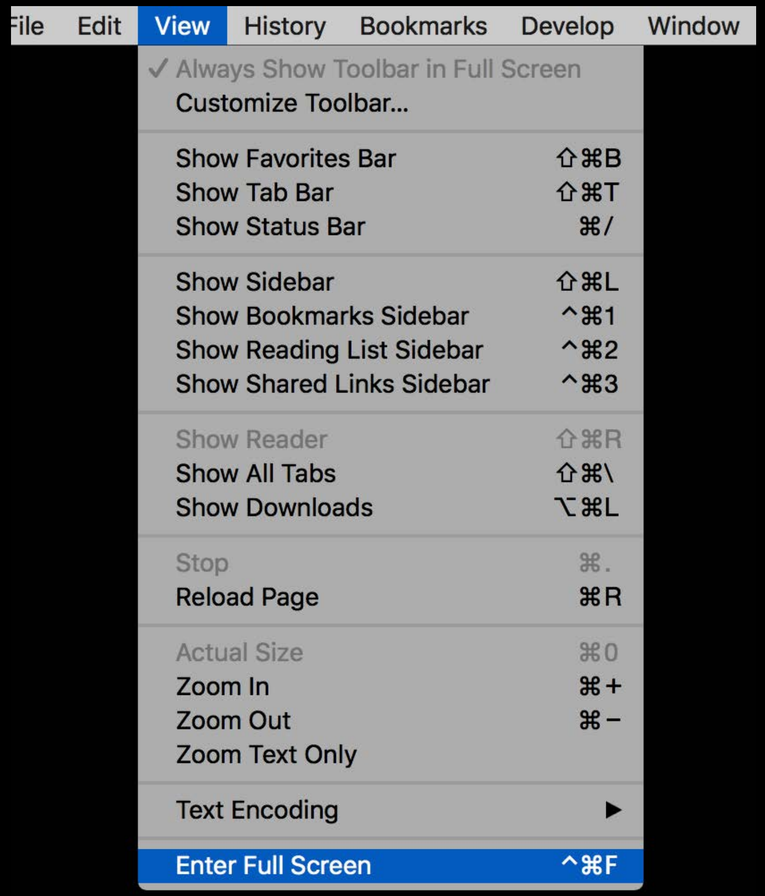
Full Screen Checklist

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

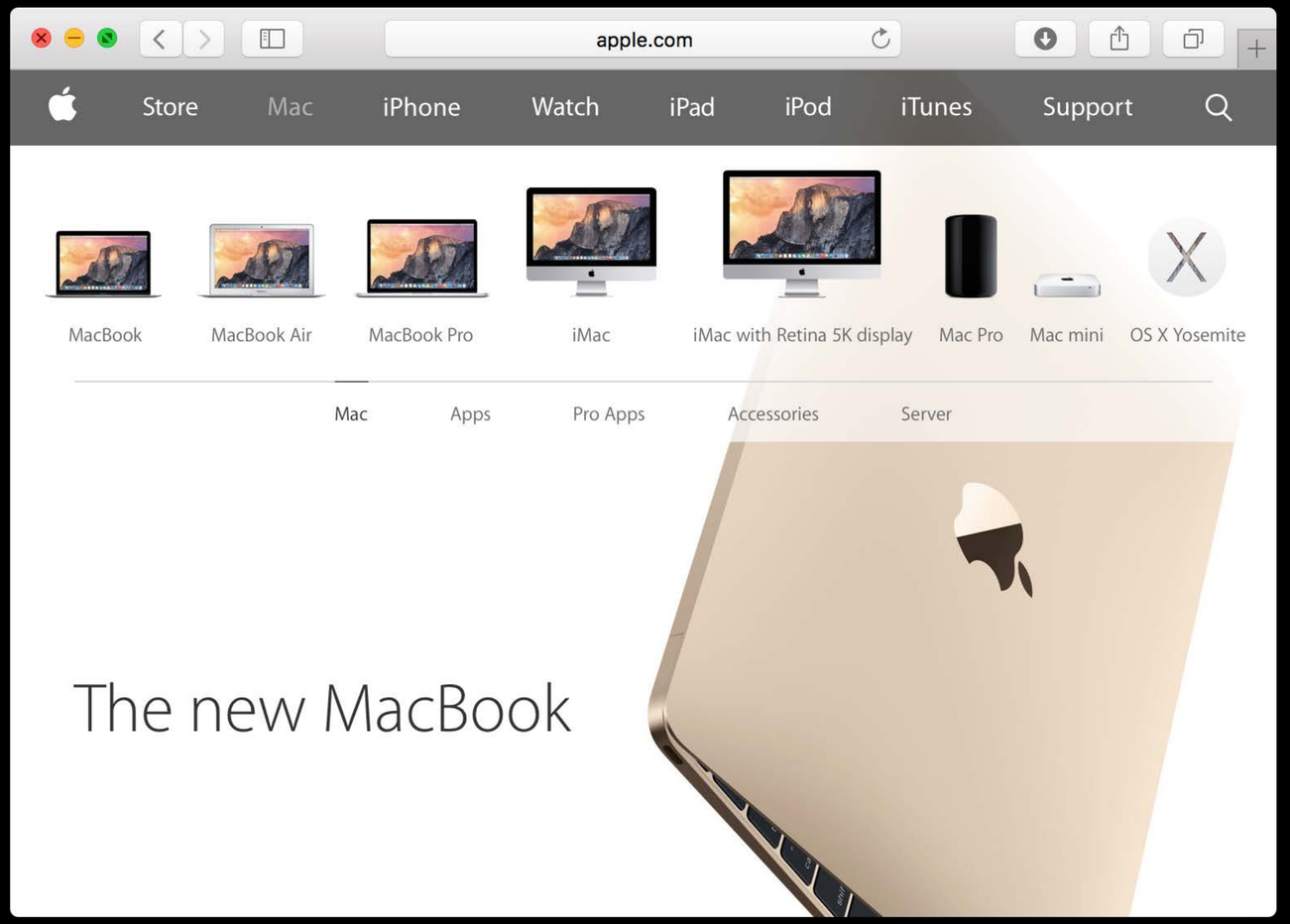


Windows in Full Screen

Primary Window

Can be made the full screen window

Main document window



Windows in Full Screen

Auxiliary Window

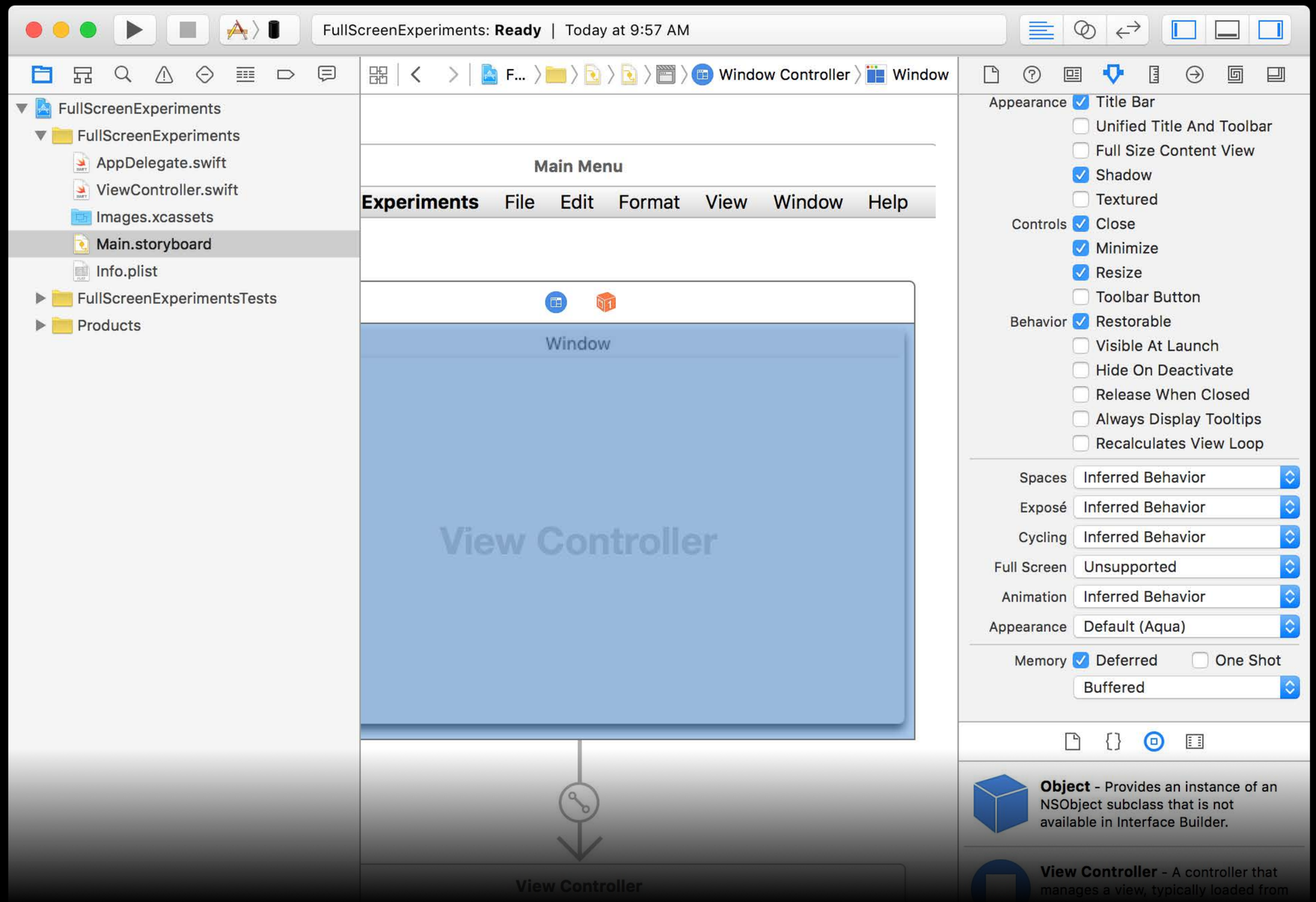
Generally not needed

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

See AppKit release notes

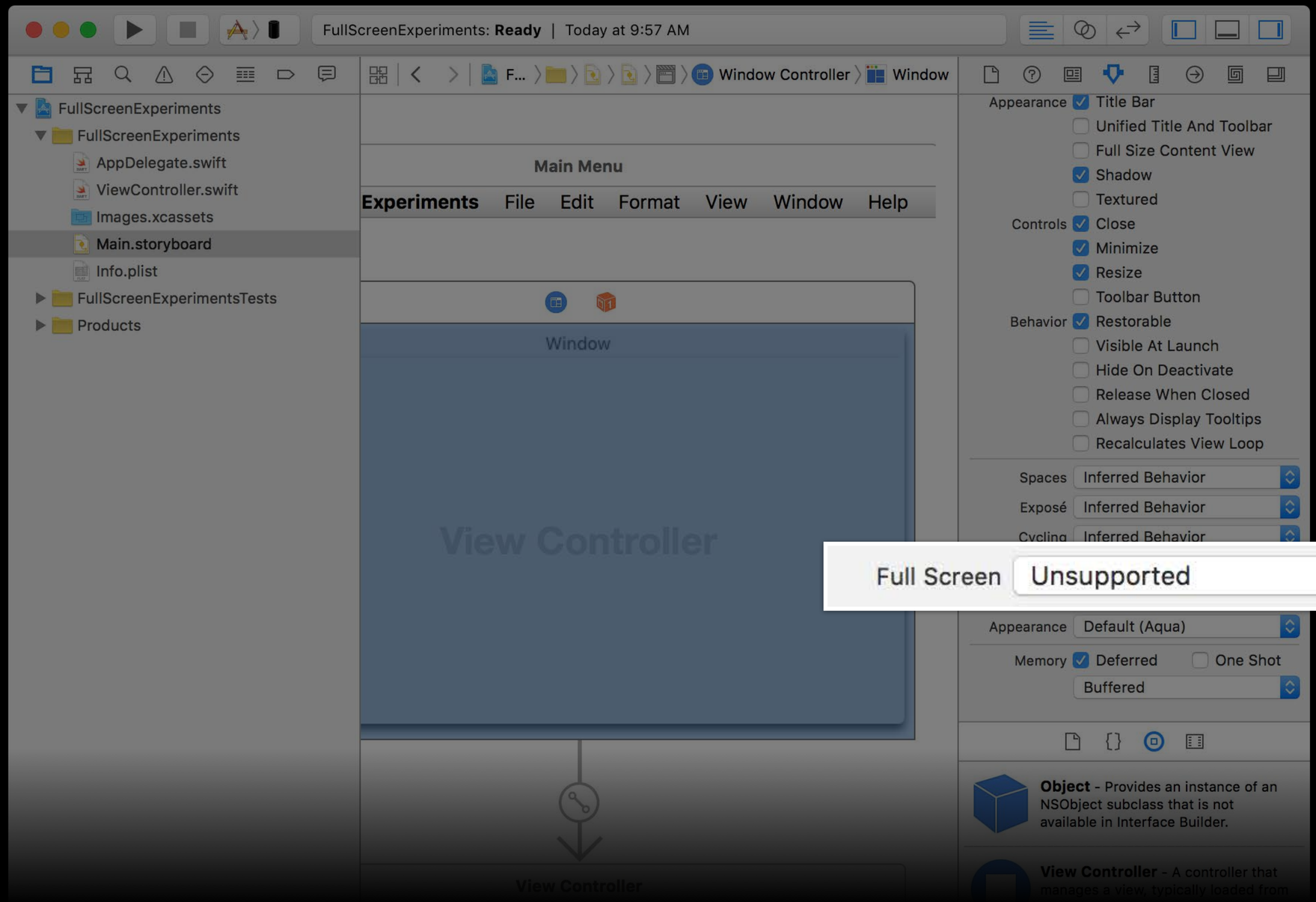
Make Windows Fullscreen Capable

In Xcode / Interface Builder



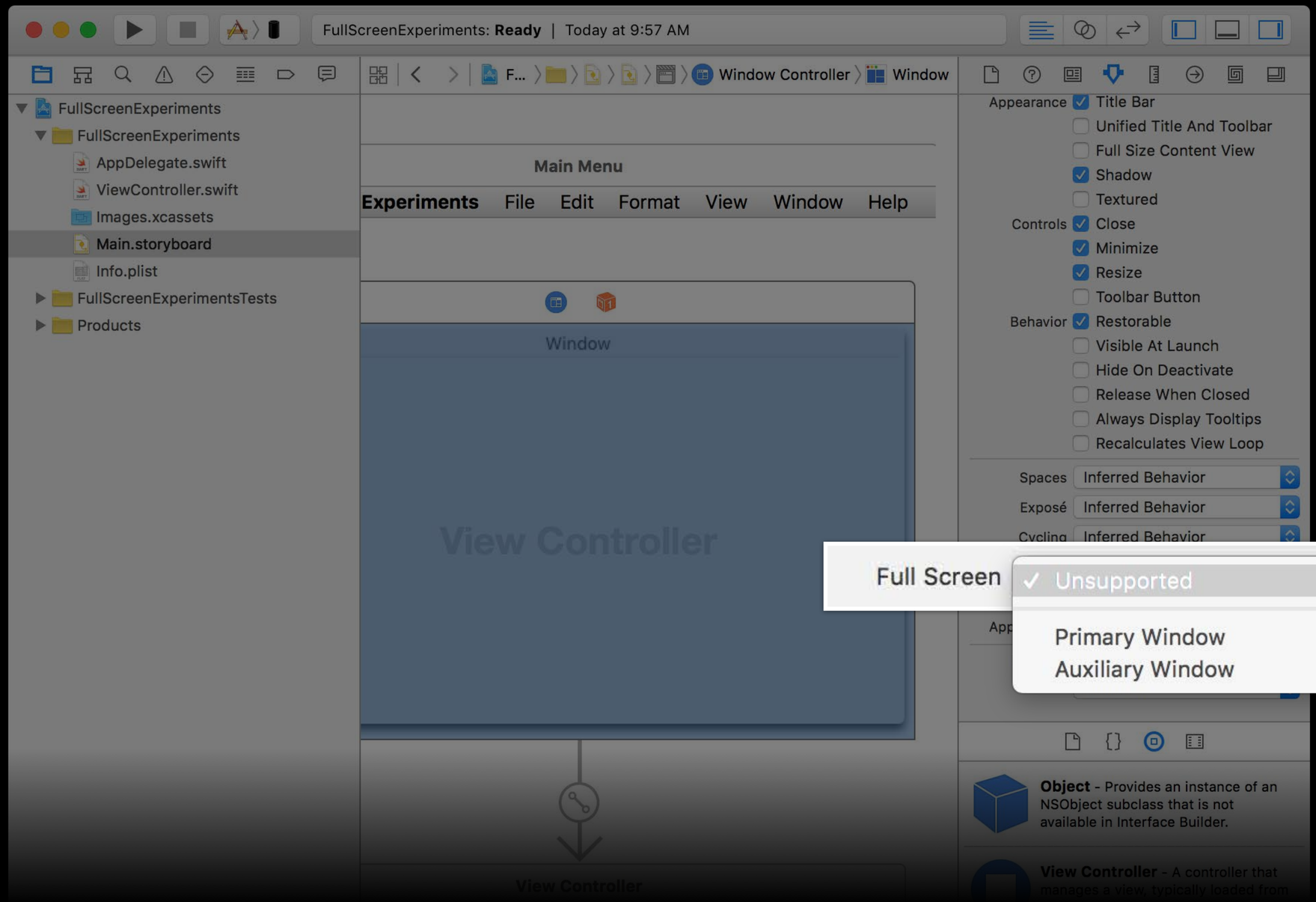
Make Windows Fullscreen Capable

In Xcode / Interface Builder



Make Windows Fullscreen Capable

In Xcode / Interface Builder



Make Windows Full Screen Capable

In code

Primary windows

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

For auxiliary windows that can exist in another fullscreen space

```
window.collectionBehavior = [window.collectionBehavior, .FullScreenAuxiliary]
```

Make Windows Full Screen Capable

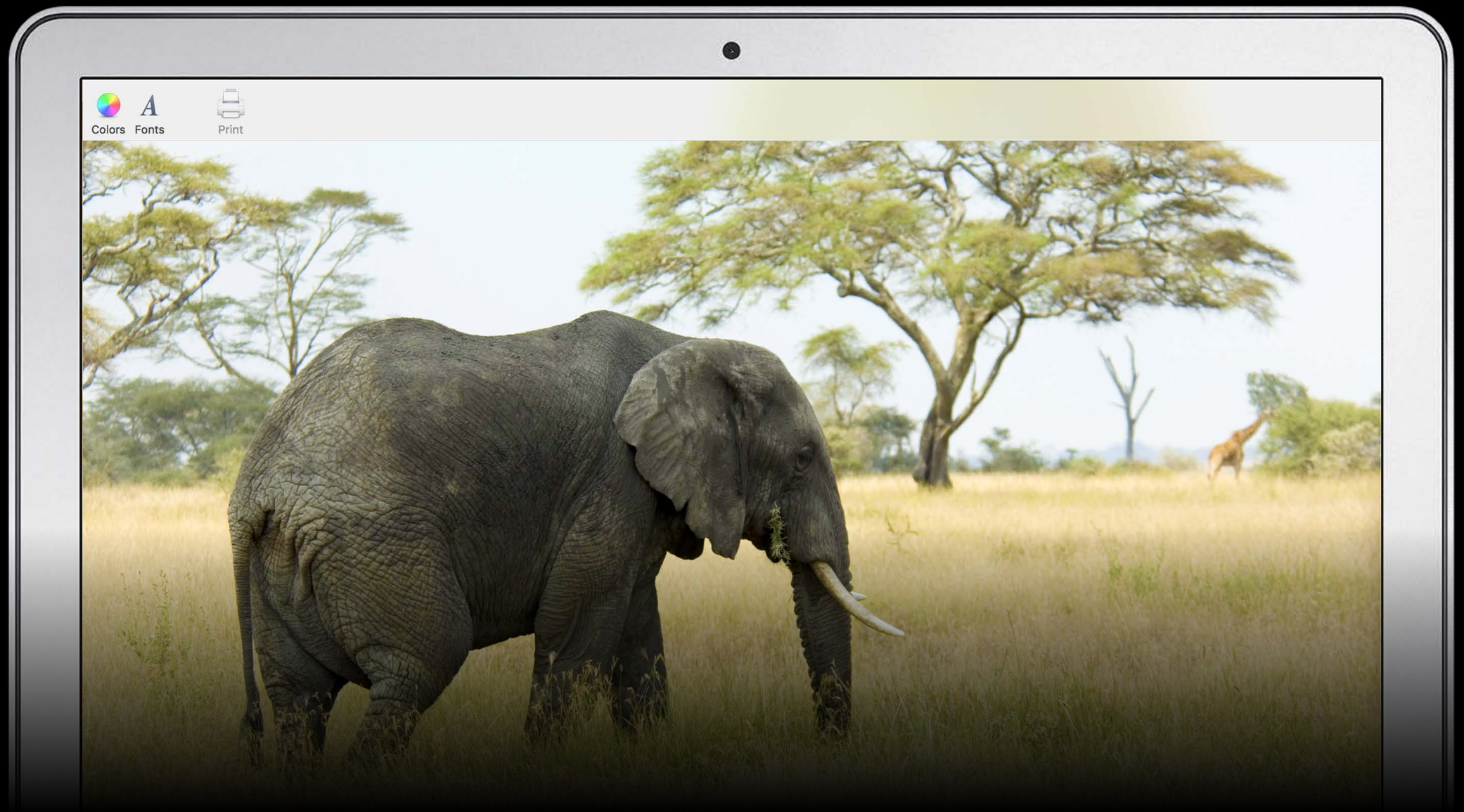
Checking state

Is this window in full screen?

```
let inFullscreen = (window.styleMask & NSFullScreenWindowMask) != 0
```

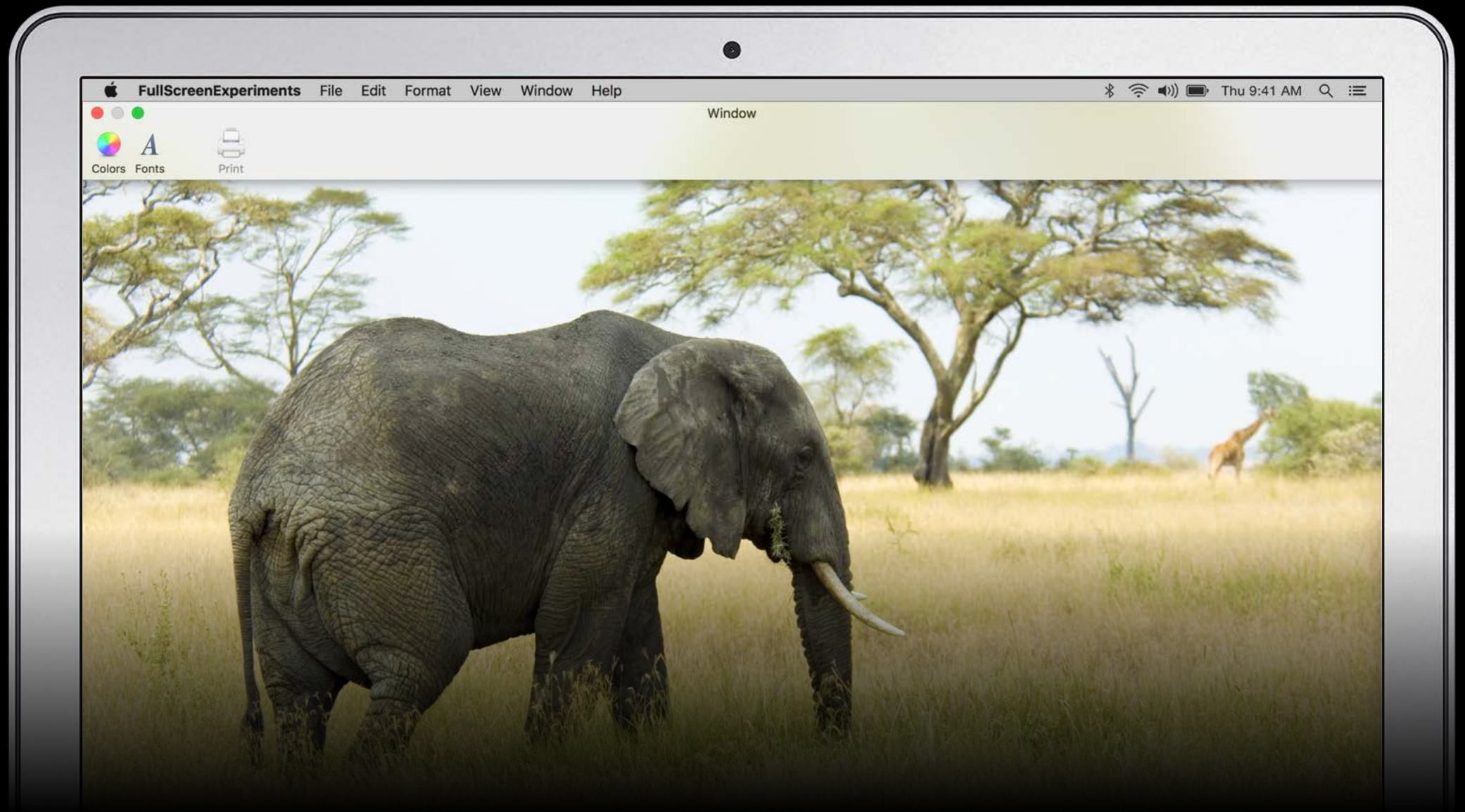
Automatically Hiding the Toolbar

Default behavior is always visible



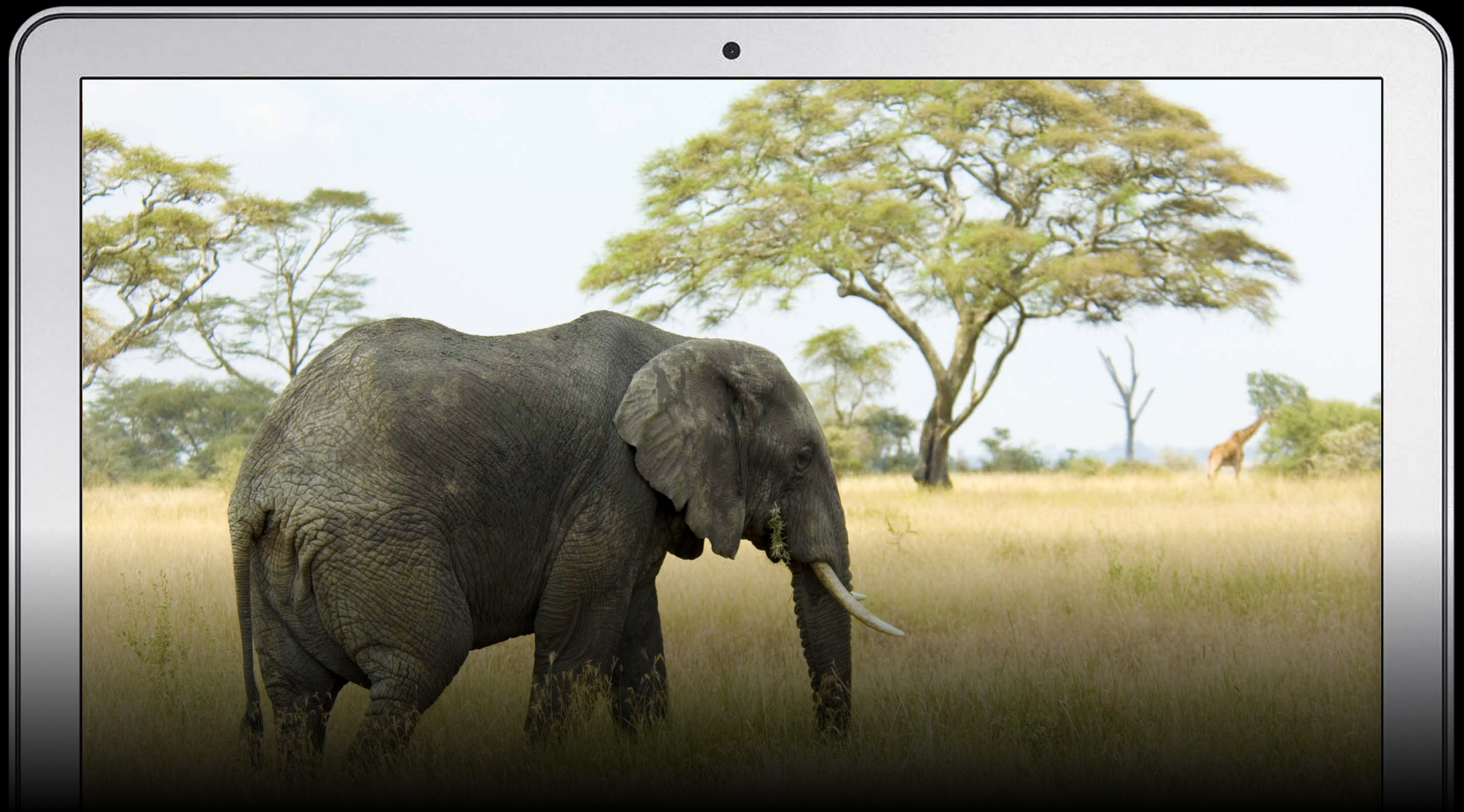
Automatically Hiding the Toolbar

Menu bar reveal also shows the titlebar



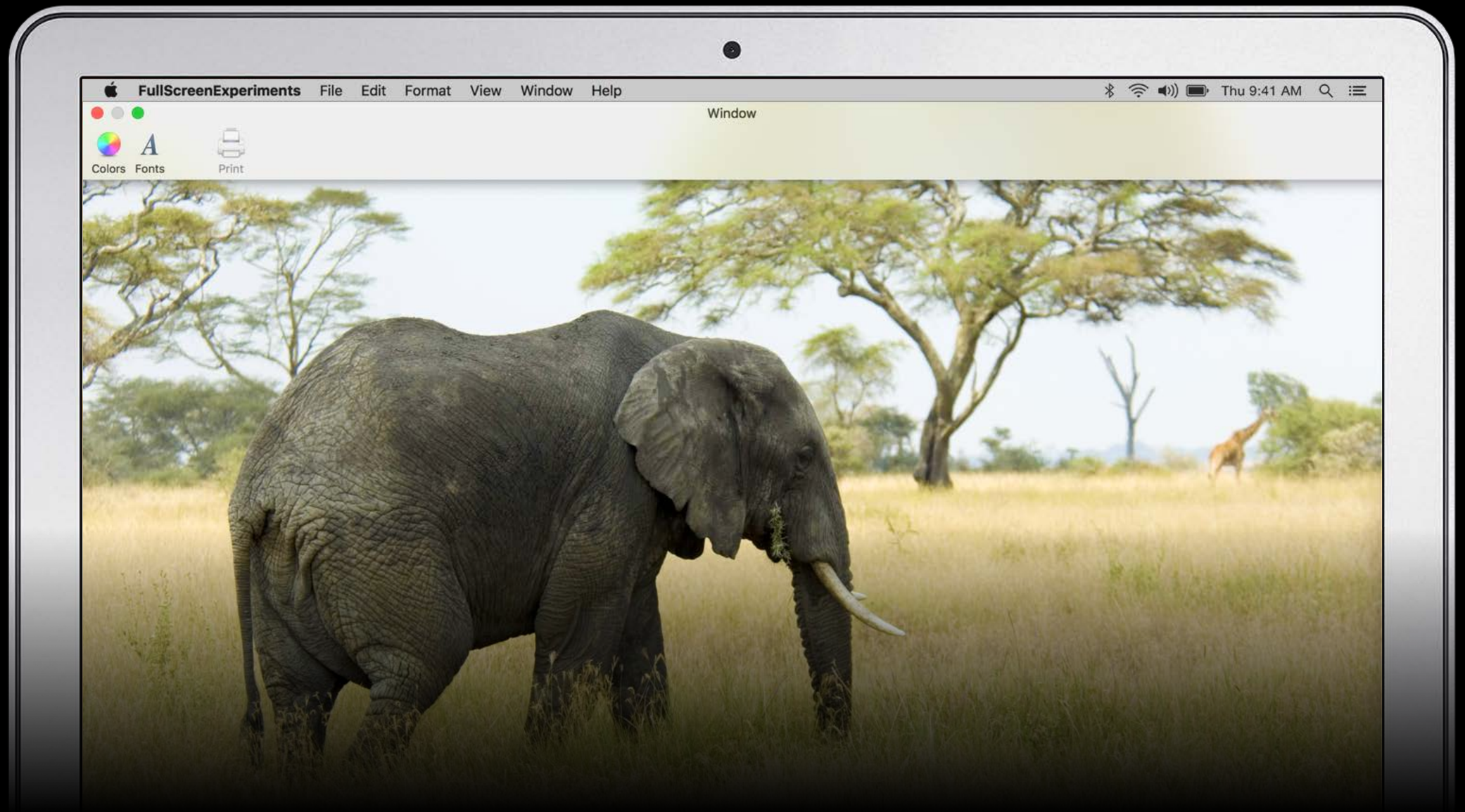
Automatically Hiding the Toolbar

Toolbar can be autohidden in full screen



Automatically Hiding the Toolbar

Menu bar reveal shows the toolbar and titlebar



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:

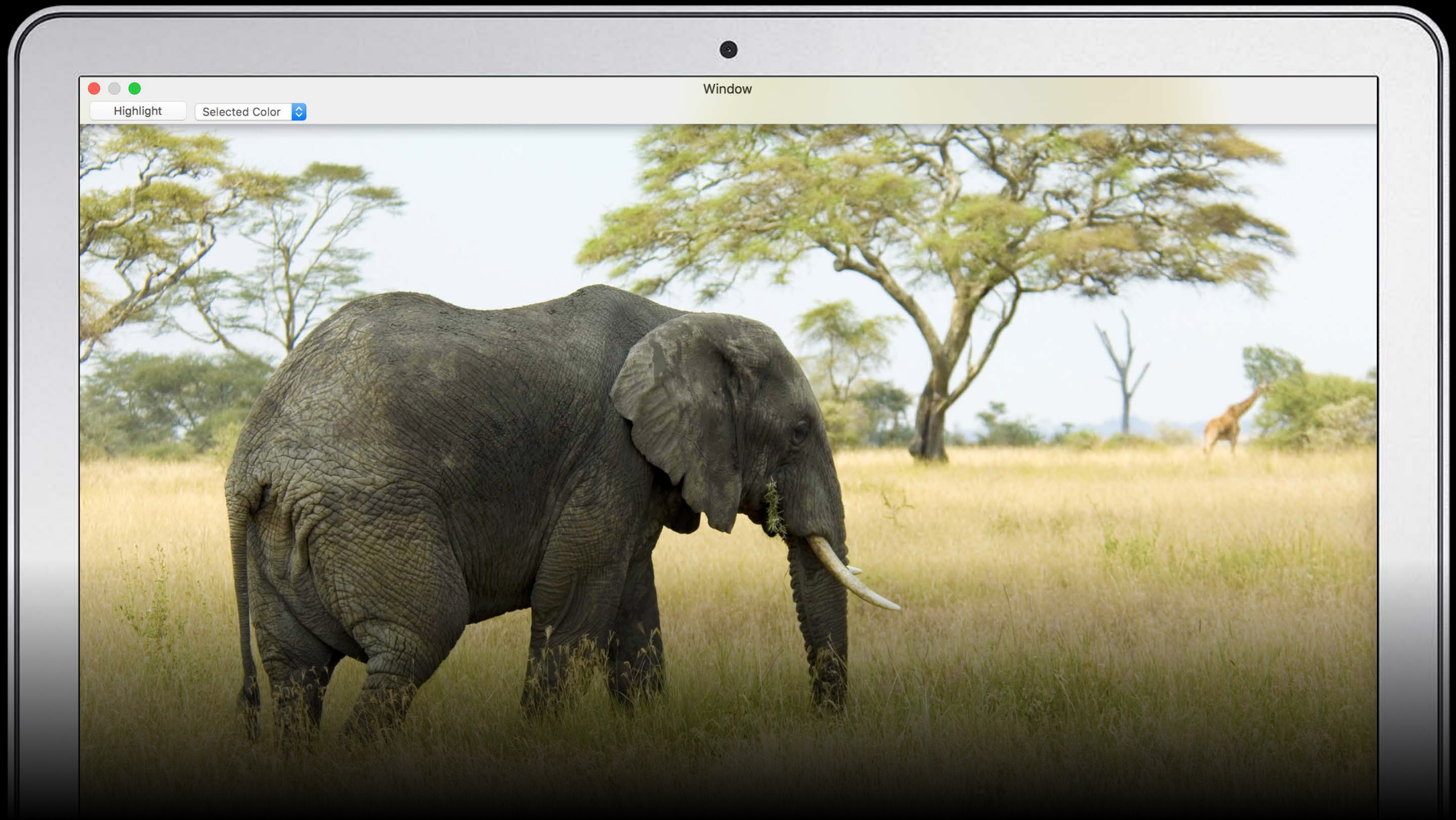
```
func customWindowsToEnterFullScreenForWindow(NSWindow) -> [NSWindow]?
func window(NSWindow,
            startCustomAnimationToEnterFullScreenWithDuration: NSTimeInterval)
```

NOTE: These may not be called when entering a tile with Mission Control

Titlebar Accessory View Controllers

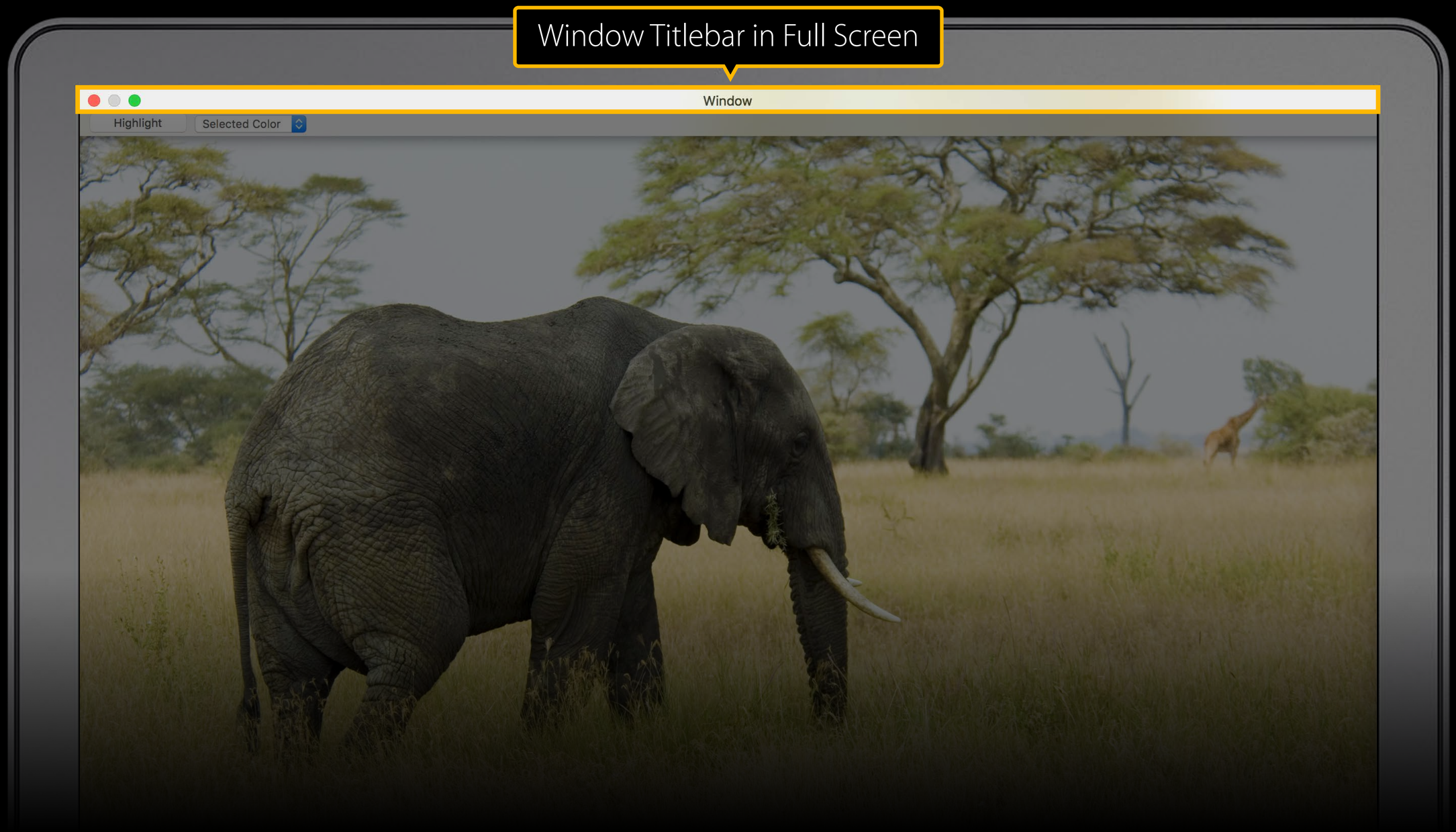
Titlebar Accessory View Controllers

For Full Screen and Normal Windows



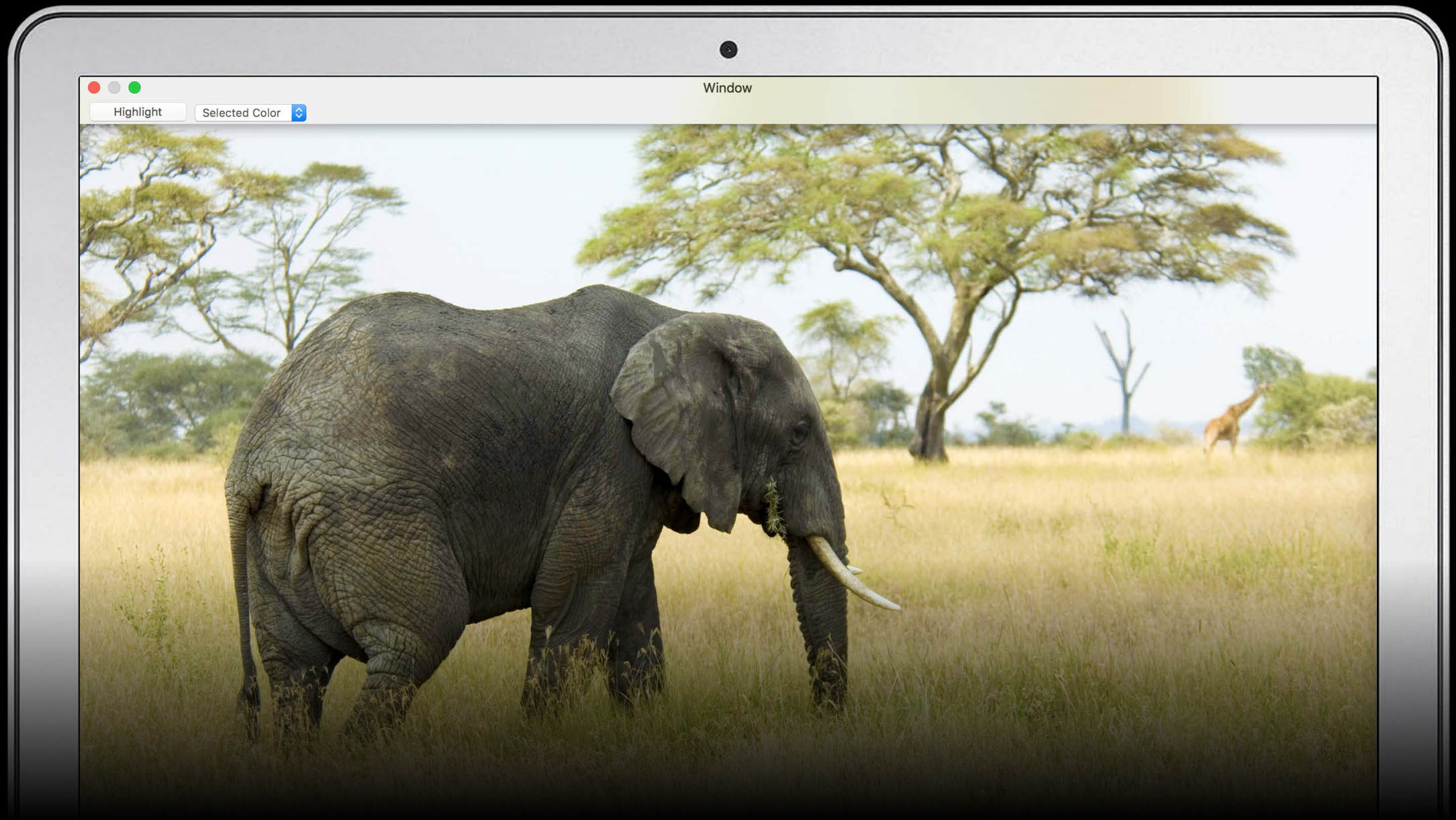
Titlebar Accessory View Controllers

For Full Screen and Normal Windows



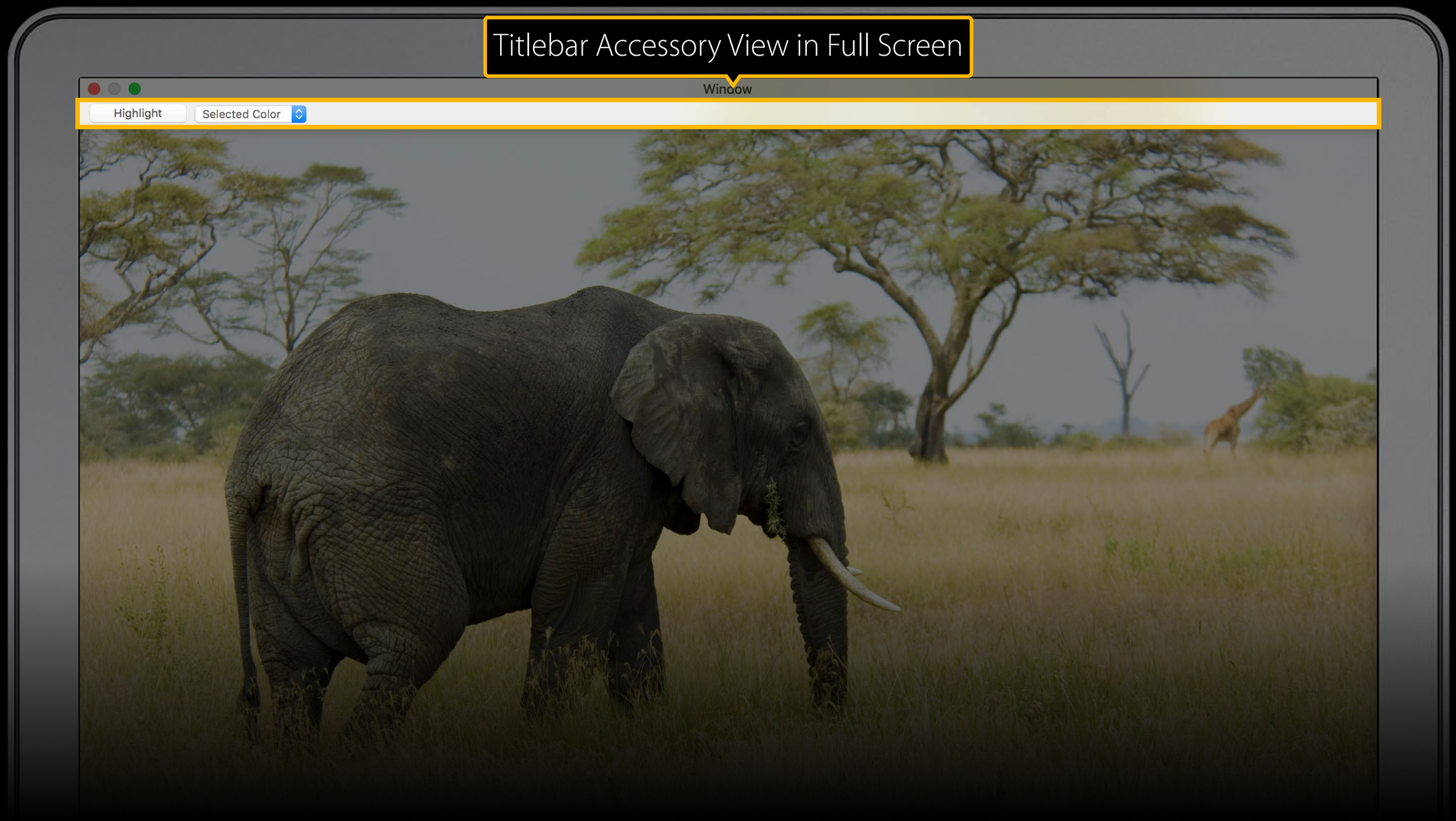
Titlebar Accessory View Controllers

For Full Screen and Normal Windows



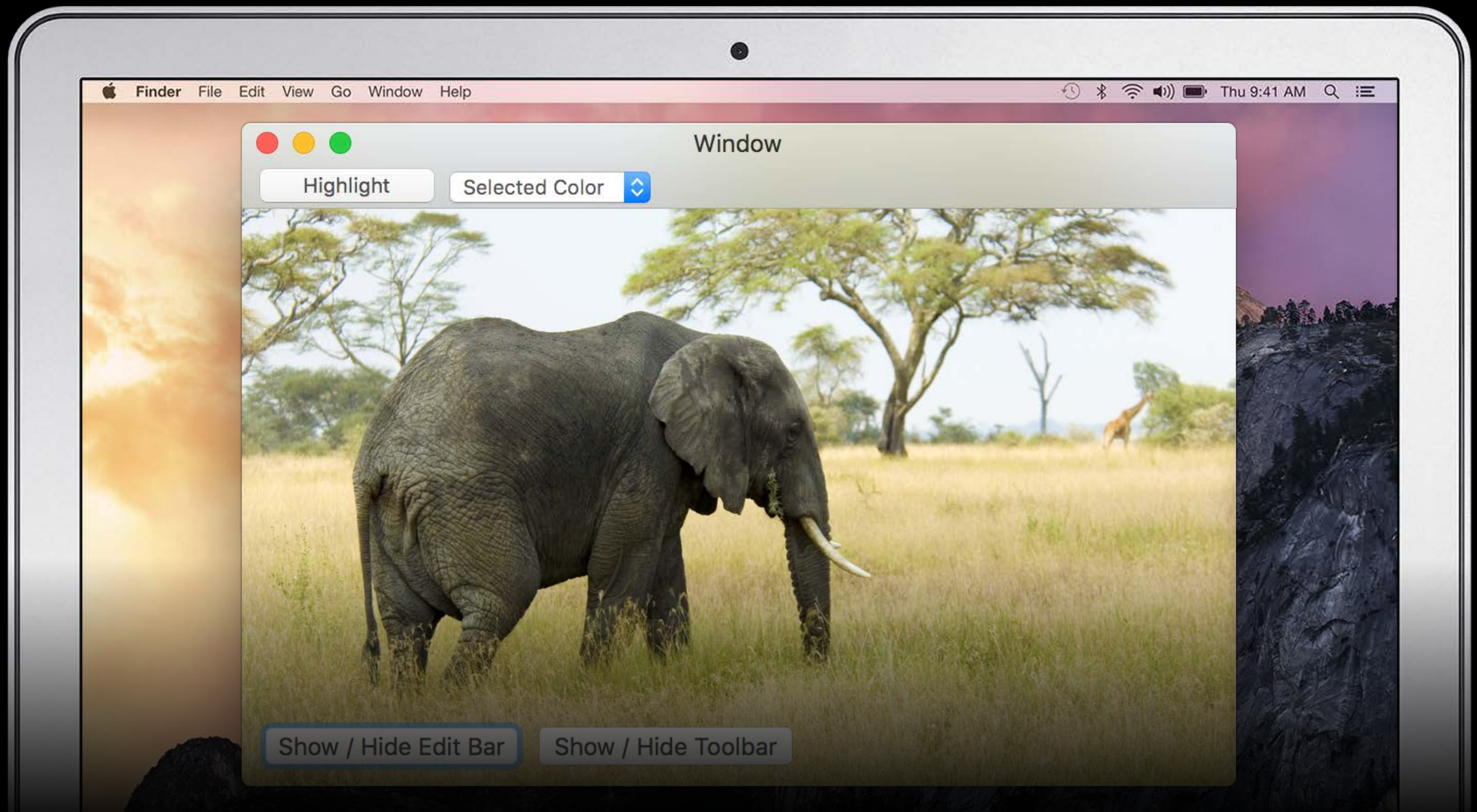
Titlebar Accessory View Controllers

For Full Screen and Normal Windows



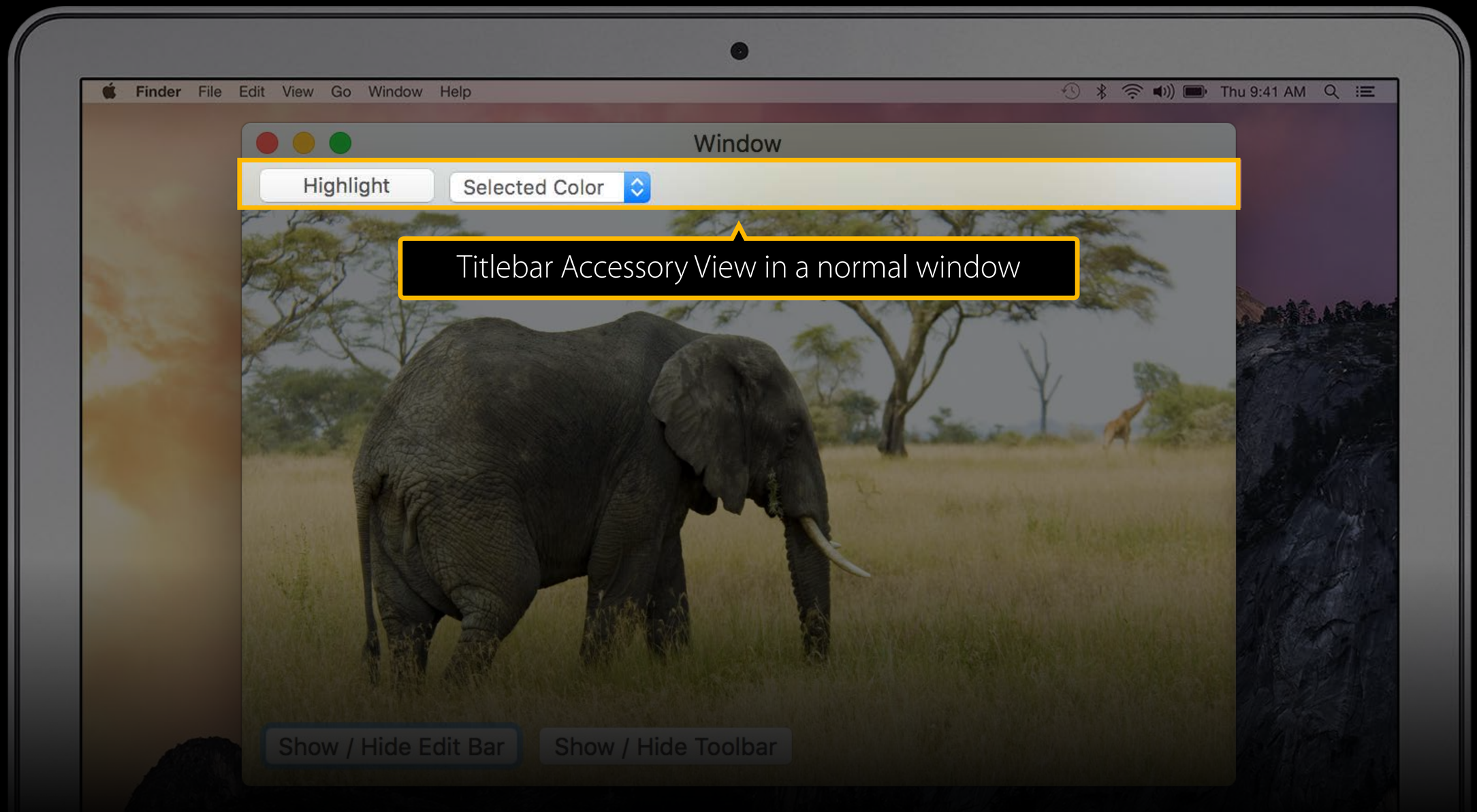
Titlebar Accessory View Controllers

For Full Screen and Normal Windows



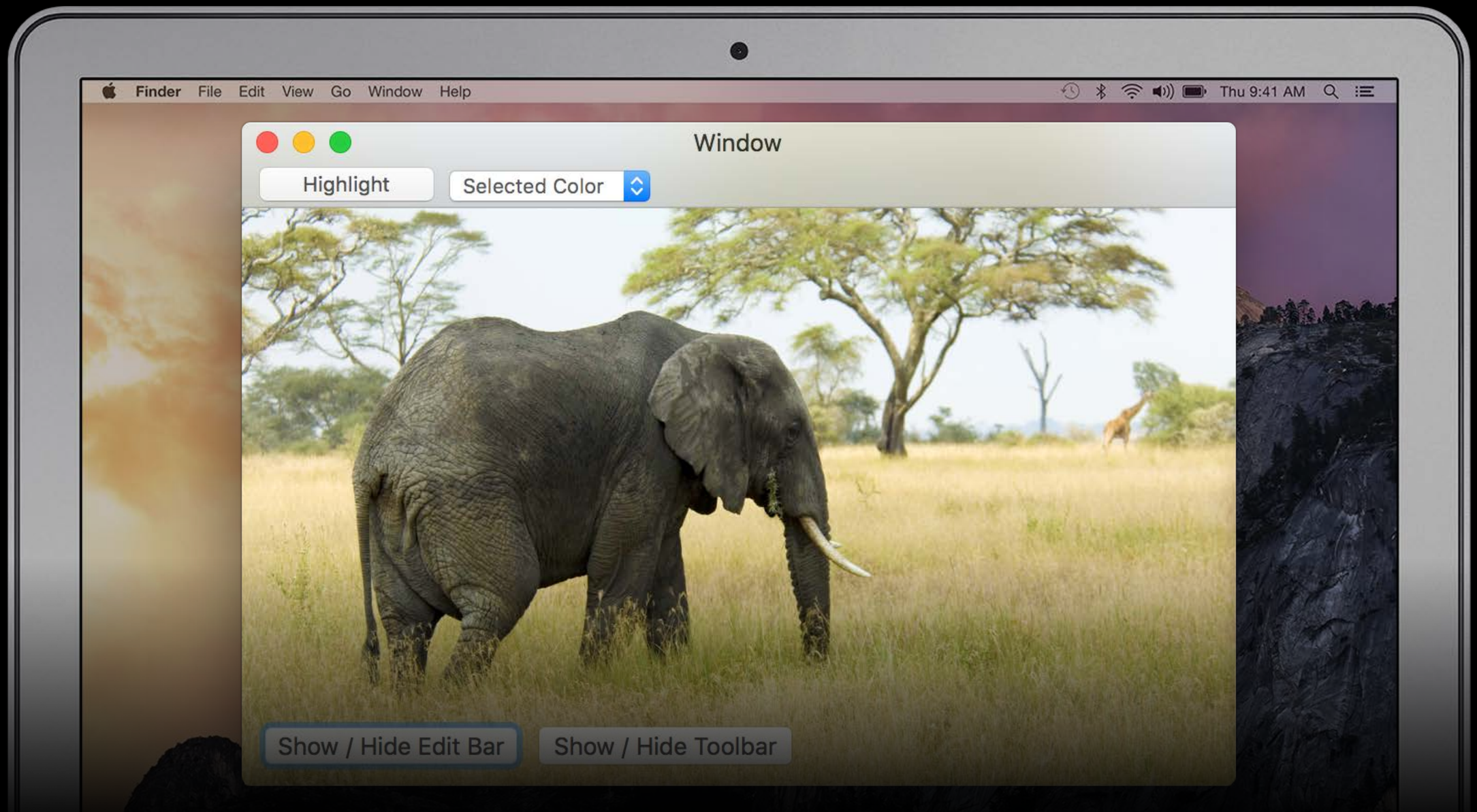
Titlebar Accessory View Controllers

For Full Screen and Normal Windows



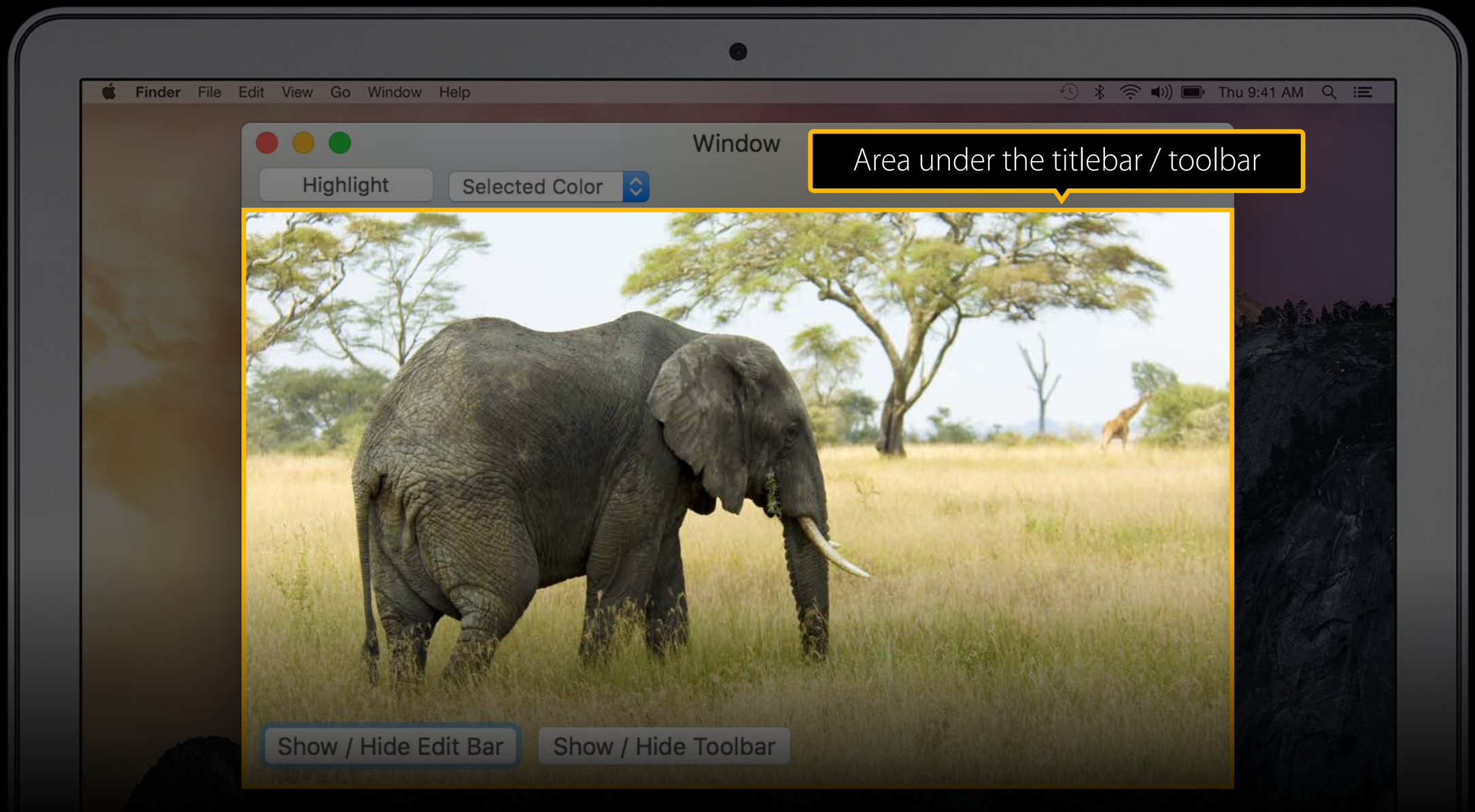
Titlebar Accessory View Controllers

Default window content view area



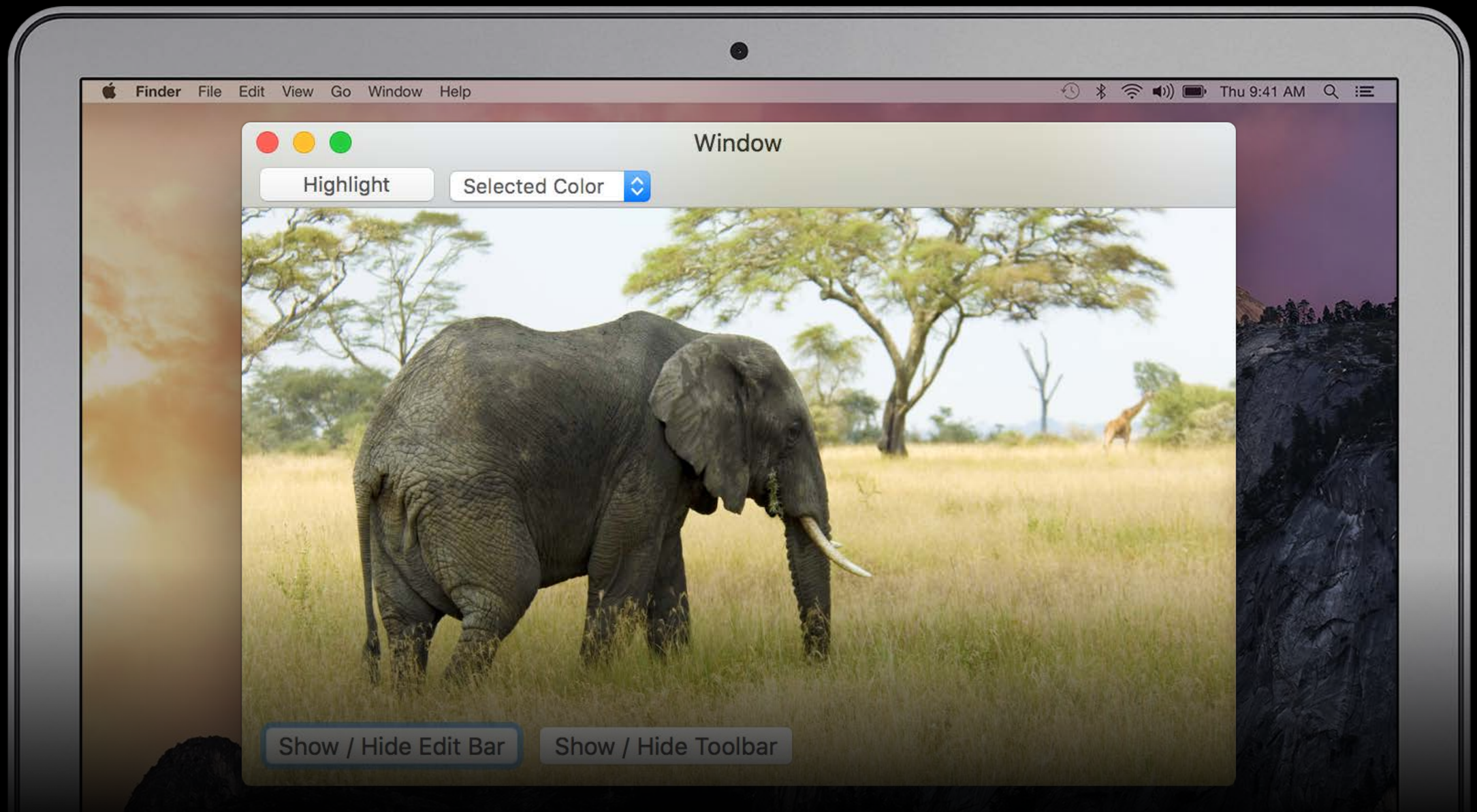
Titlebar Accessory View Controllers

Default window content view area



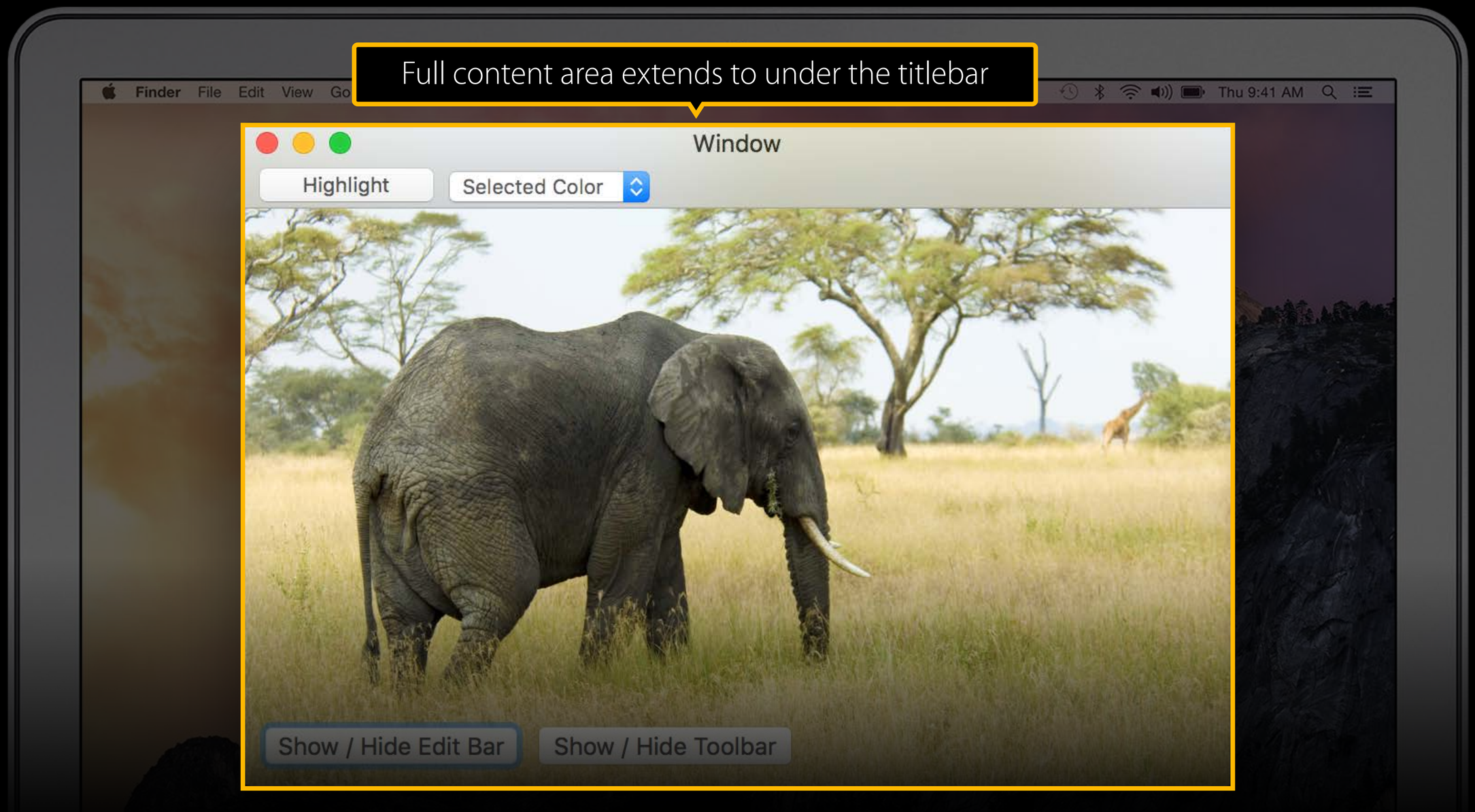
Titlebar Accessory View Controllers

Full content area under the titlebar and toolbar



Titlebar Accessory View Controllers

Full content area under the titlebar and toolbar



Titlebar Accessory View Controllers

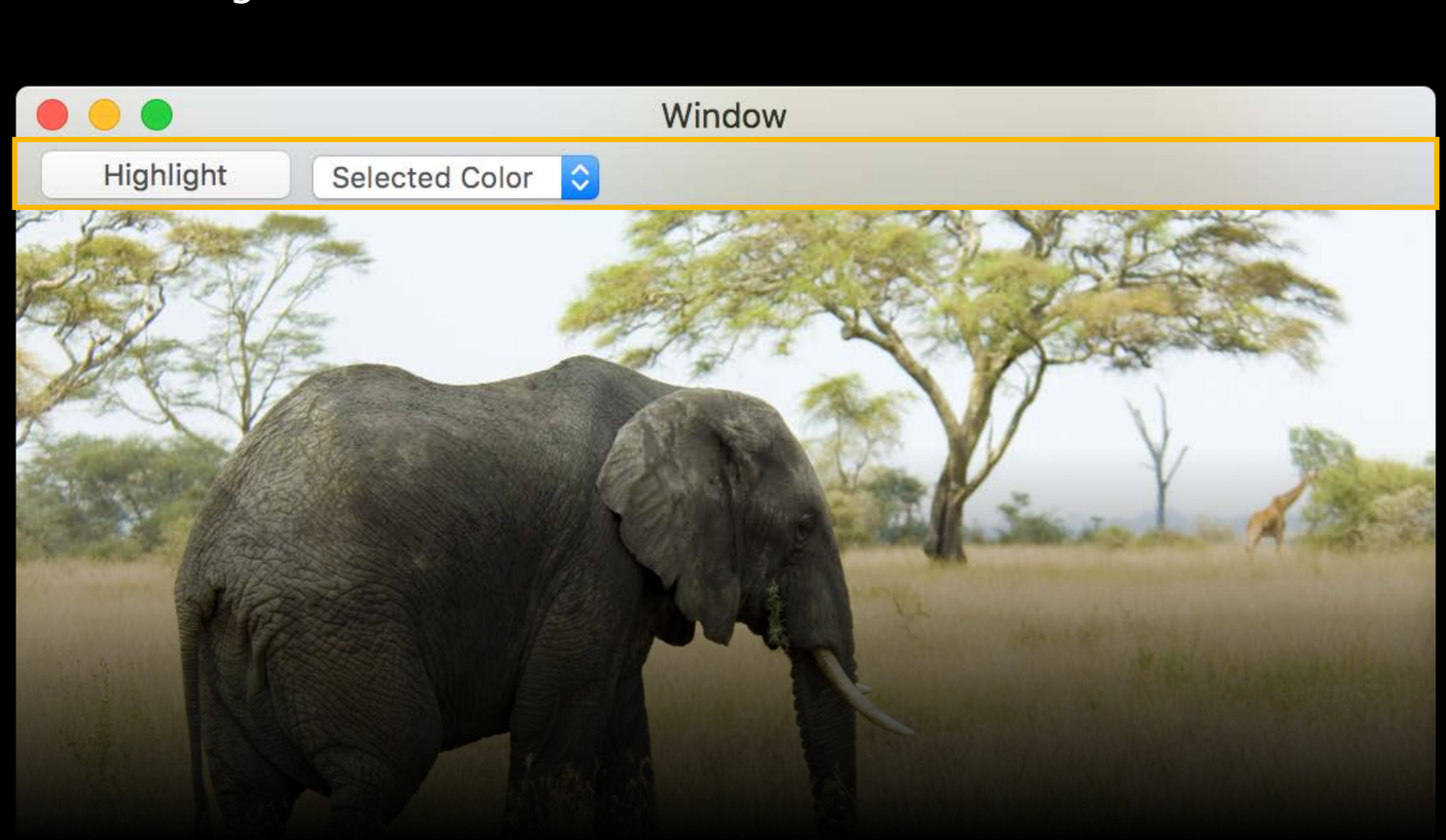
Full content area under the titlebar and toolbar

```
window.styleMask = window.styleMask | NSFullSizeContentViewWindowMask
```

Titlebar Accessory View Controllers

NSWindow API in 10.10

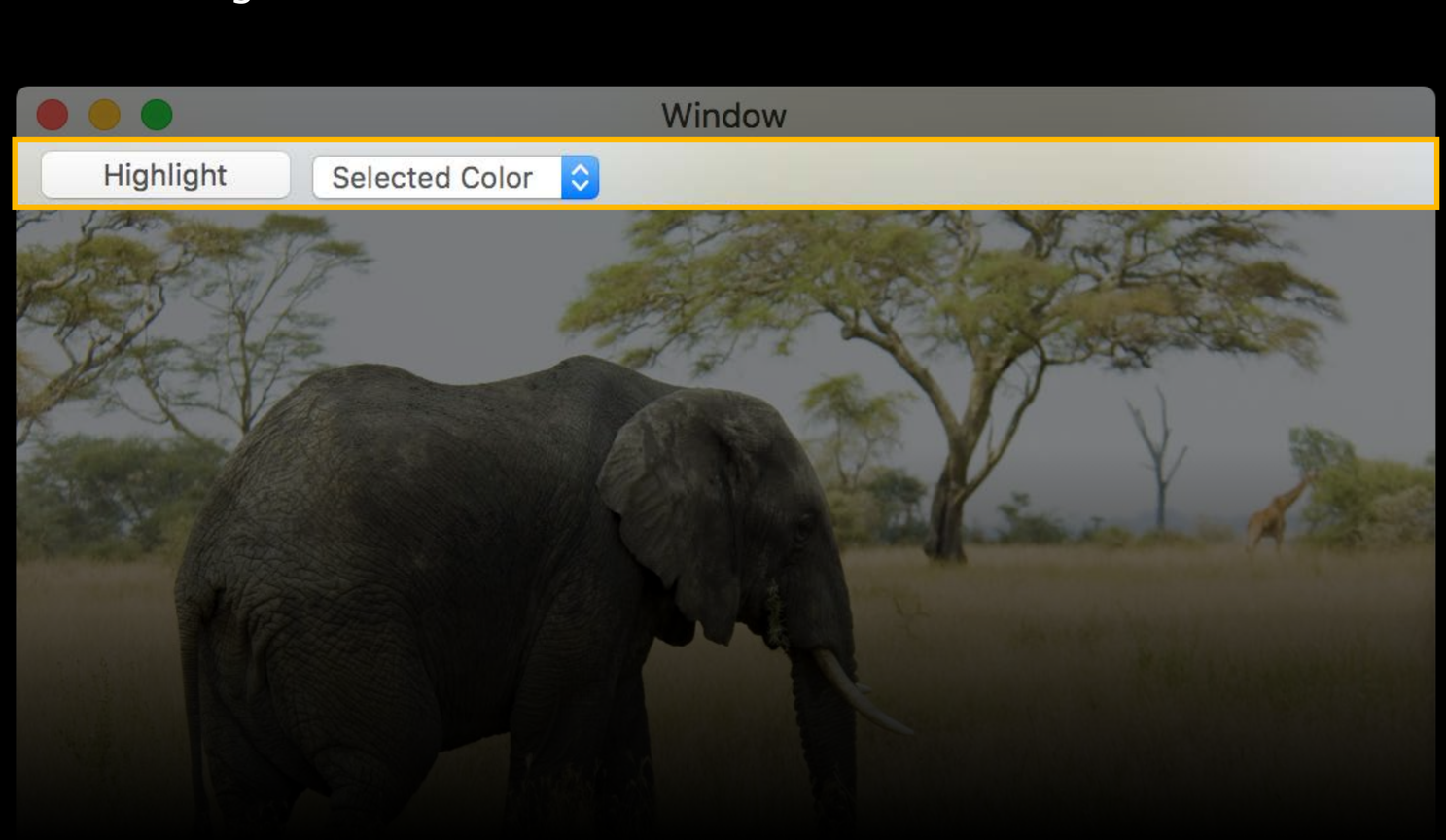
```
class NSTitlebarAccessoryViewController : NSViewController {

    var layoutAttribute: NSLayoutAttribute
    var fullScreenMinHeight: CGFloat

}
```



Titlebar Accessory View Controllers

NSWindow API in 10.10

```
class NSTitlebarAccessoryViewController : NSViewController {

    var layoutAttribute: NSLayoutAttribute
    var fullScreenMinHeight: CGFloat

}
```



Titlebar Accessory View Controllers

NSTitlebarAccessoryViewController

Automatically has blurring behind it

Automatically contained in an NSVisualEffectView

Automatically works in full screen

Has a semi-managed size

Titlebar Accessory View Controllers

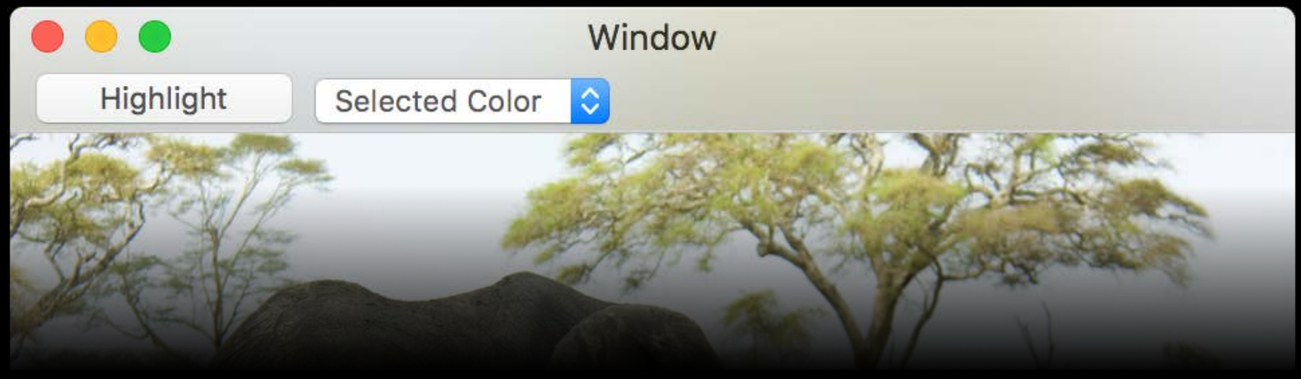
Layout Attribute

```
class NSTitlebarAccessoryViewController : NSViewController {

    var layoutAttribute: NSLayoutAttribute
    var fullScreenMinHeight: CGFloat

}
```

Titlebar Accessory View Controllers

Layout Attribute

```
accessoryViewController.layoutAttribute = NSLayoutAttribute.Bottom
```

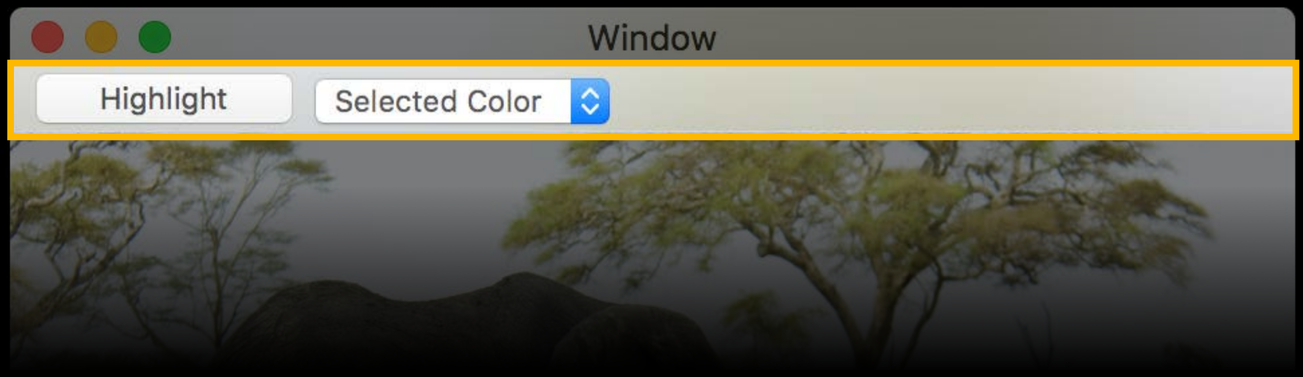


- Height is the view's height
- Width is the window's width and automatically is set with the window

Titlebar Accessory View Controllers

Layout Attribute

```
accessoryViewController.layoutAttribute = NSLayoutAttribute.Bottom
```

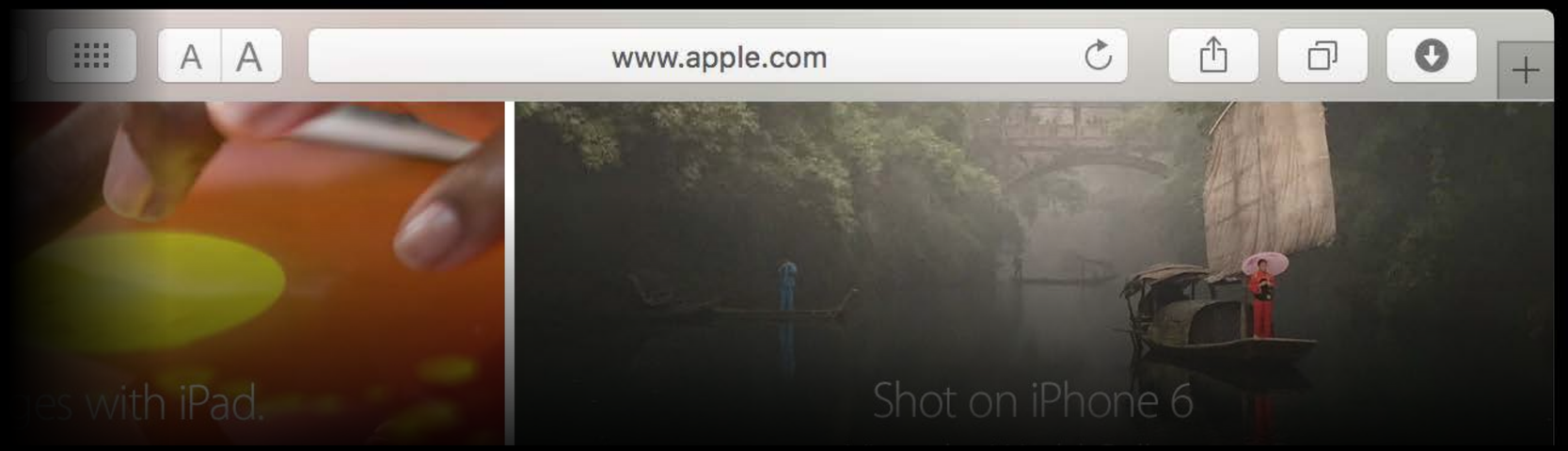


- Height is the view's height
- Width is the window's width and automatically is set with the window

Titlebar Accessory View Controllers

Layout Attribute

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

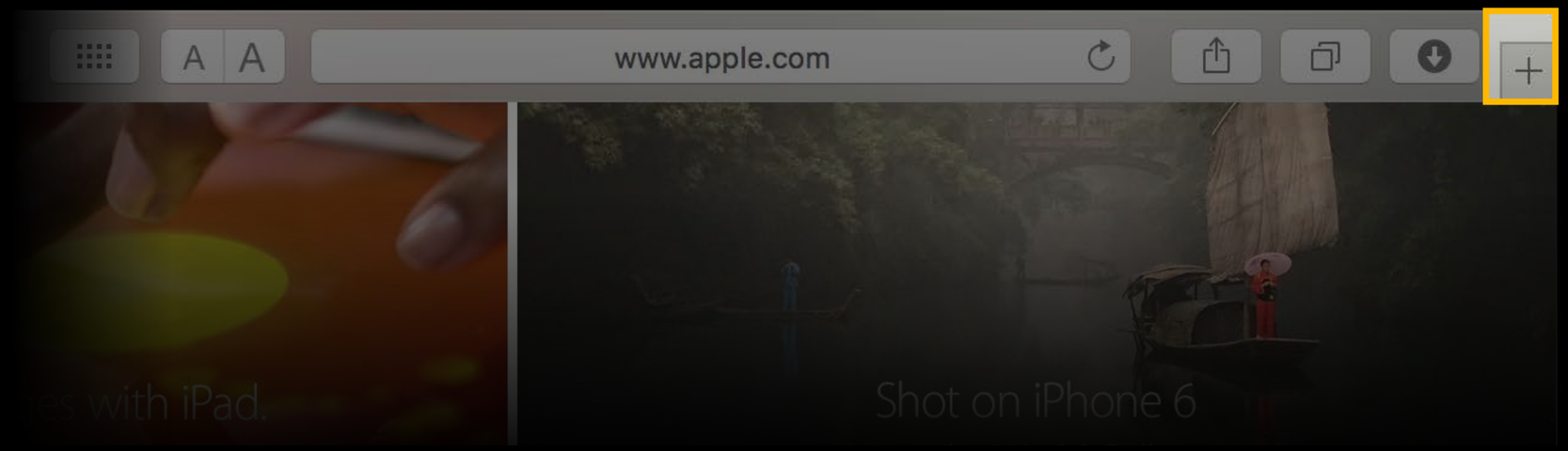


- Width is the view's width
- Height is the height of the complete toolbar area
  - Automatically changes when the toolbar height changes

Titlebar Accessory View Controllers

Layout Attribute

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



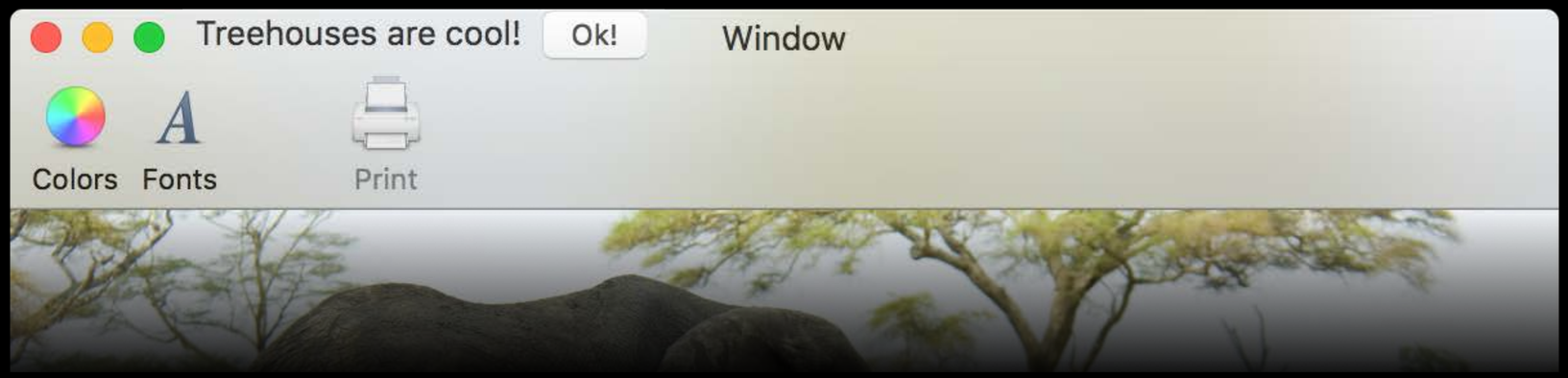
- Width is the view's width
- Height is the height of the complete toolbar area
  - Automatically changes when the toolbar height changes

Titlebar Accessory View Controllers

NEW

Layout Attribute

```
accessoryViewController.layoutAttribute = NSLayoutAttribute.Left
```



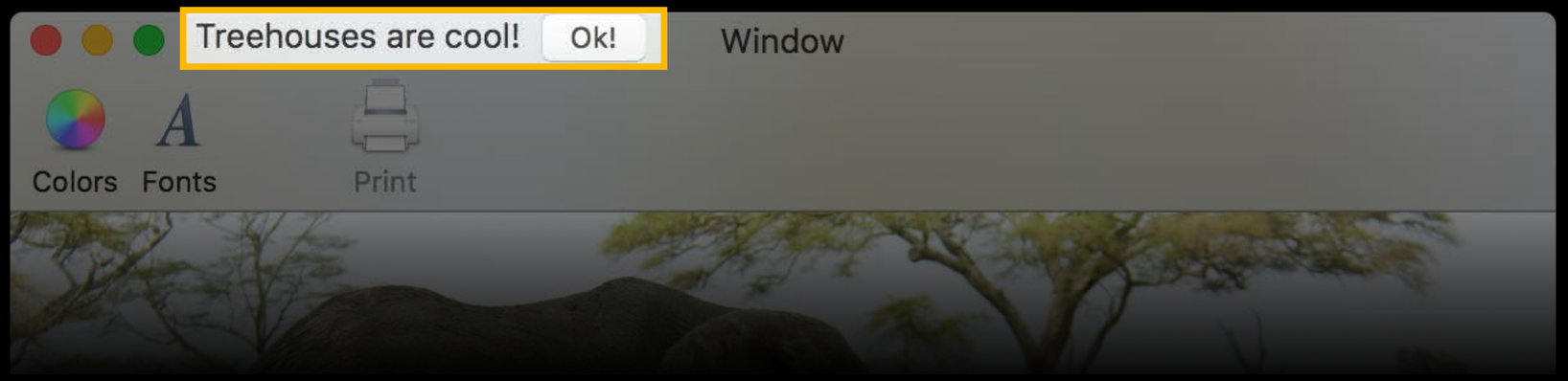
- Appears to the right of the window buttons
- Width is the view's width
- Height is the height of the complete toolbar area and automatically is changed as needed
- Overlaps the toolbar

Titlebar Accessory View Controllers

Layout Attribute

NEW

```
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

Full screen minimum height

```
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

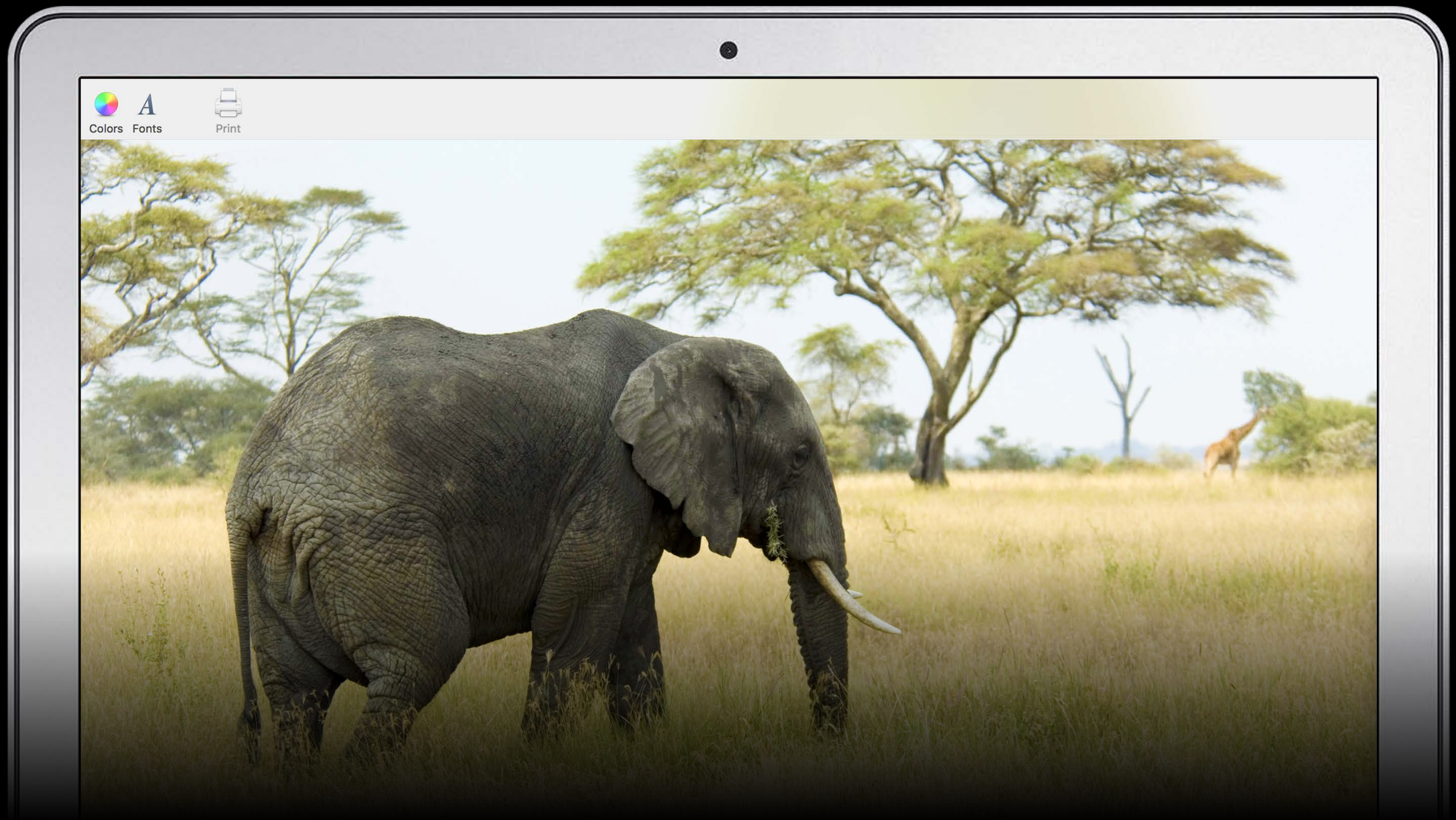
The accessory view automatically appears under the toolbar when the menu is shown

The default height is 0

- Completely hidden when the menu bar is not visible

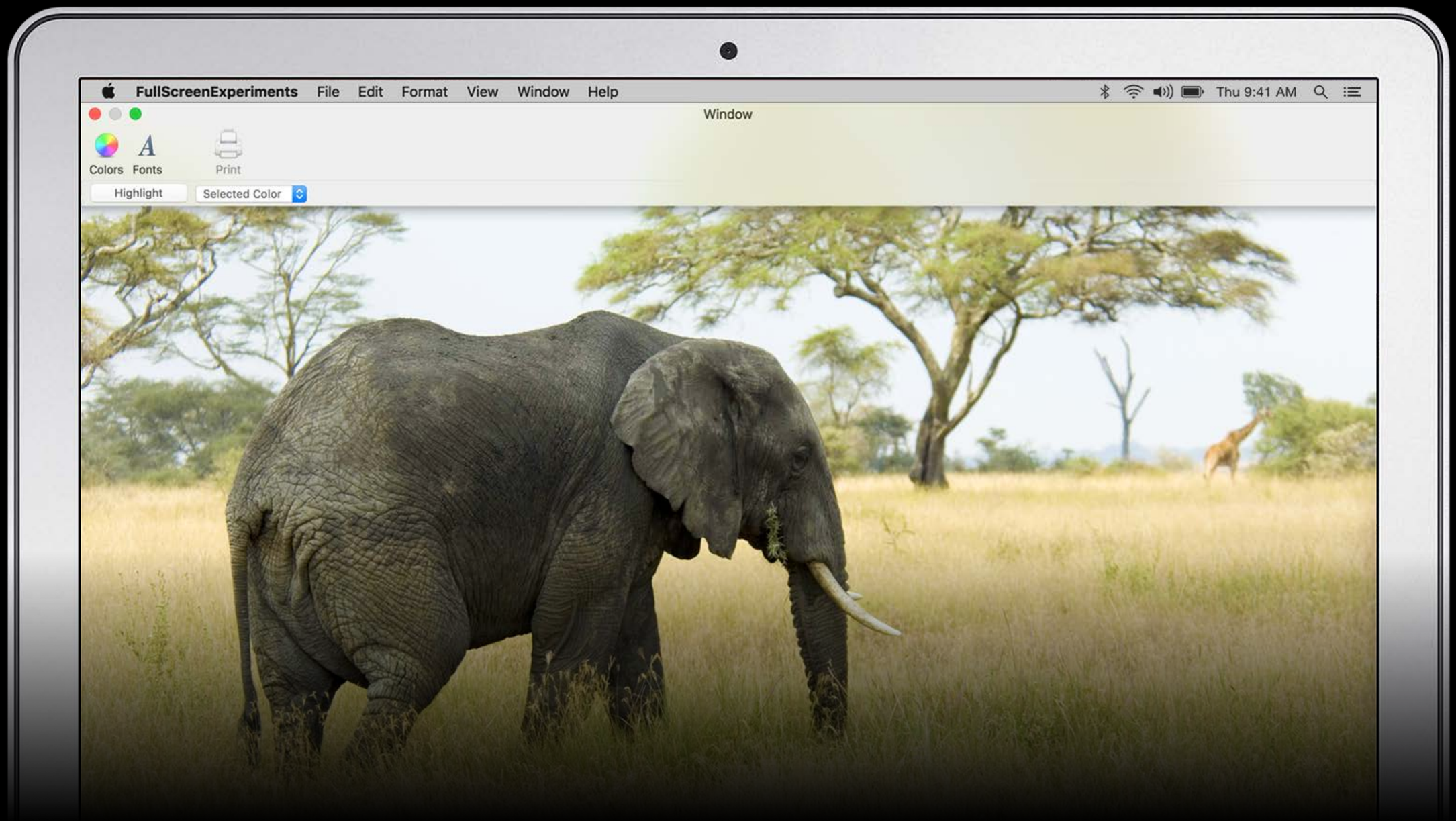
Titlebar Accessory View Controllers

Full screen minimum height = 0



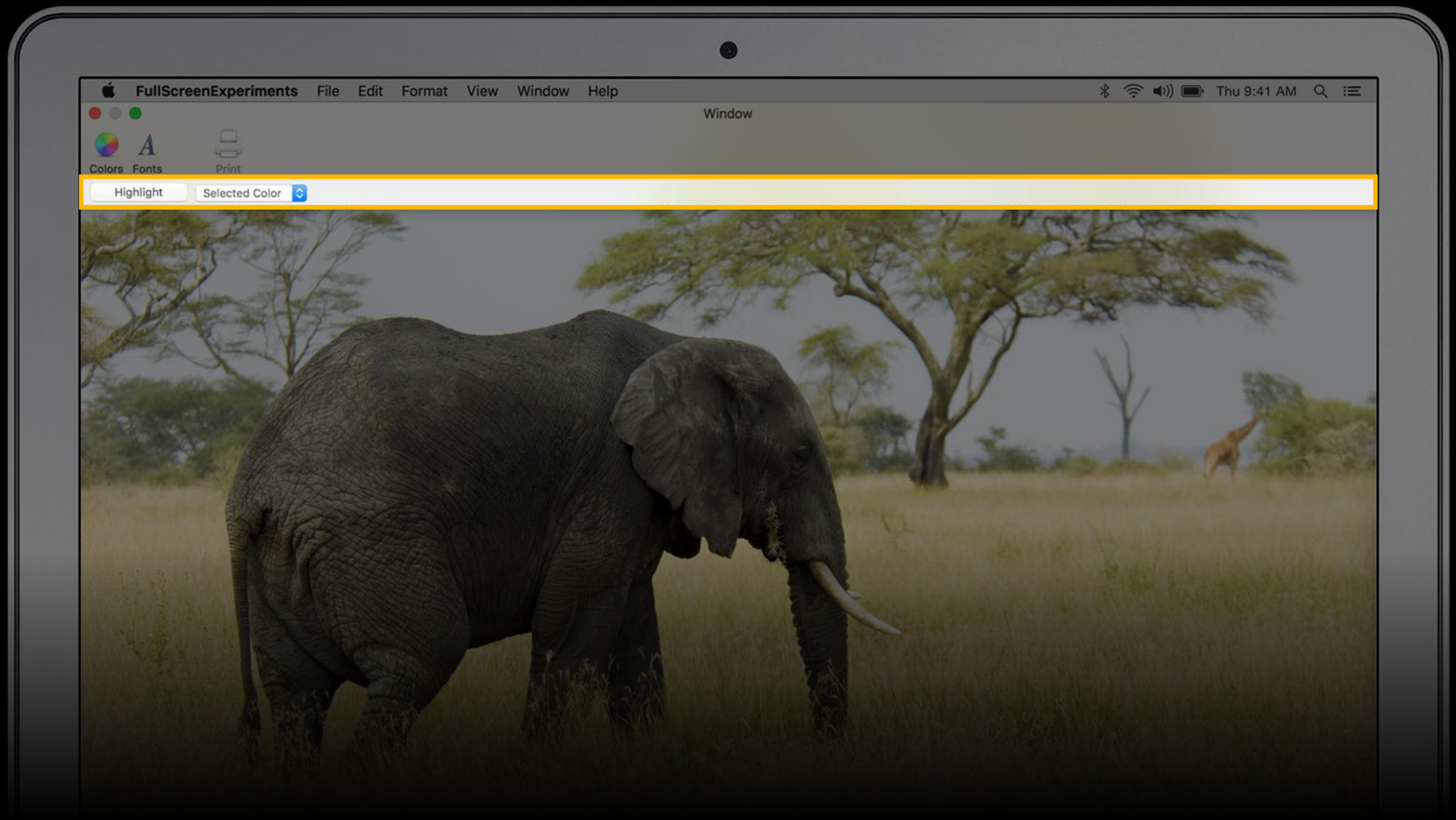
Titlebar Accessory View Controllers

Full screen minimum height = 0



Titlebar Accessory View Controllers

Full screen minimum height = 0



Titlebar Accessory View Controllers

NSToolbar

accessoryViewController.fullScreenMinHeight replaces the NSToolbar API:

```
extension NSToolbar {
    var fullScreenAccessoryView: NSView?
    var fullScreenAccessoryViewMinHeight: CGFloat
    var fullScreenAccessoryViewMaxHeight: CGFloat
}
```

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

```
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

Basic example

Adding:

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

Removing:

```
accessoryViewController.removeFromParentViewController()
```

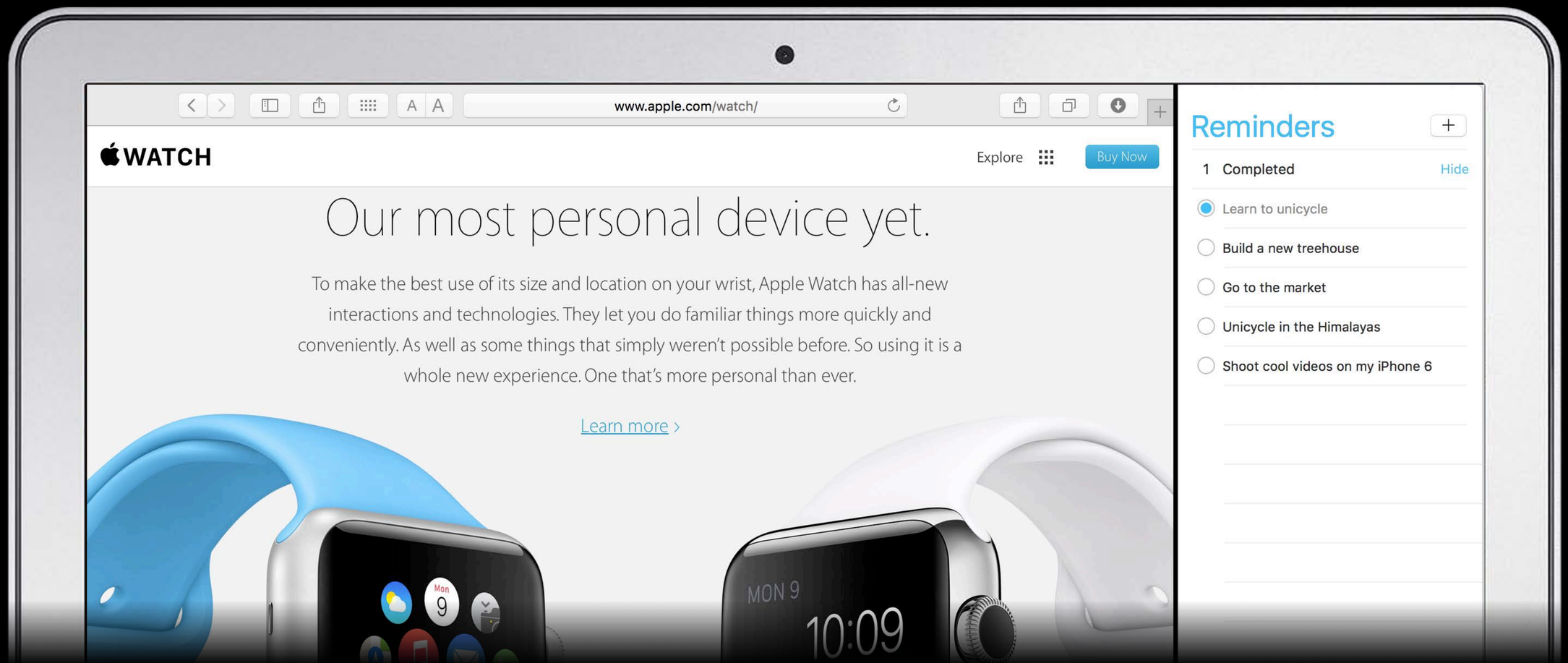
Full Screen Tiles

Full Screen Tiles

User benefits

Focus attention single task that may involve multiple windows

Still make the most of screen real estate

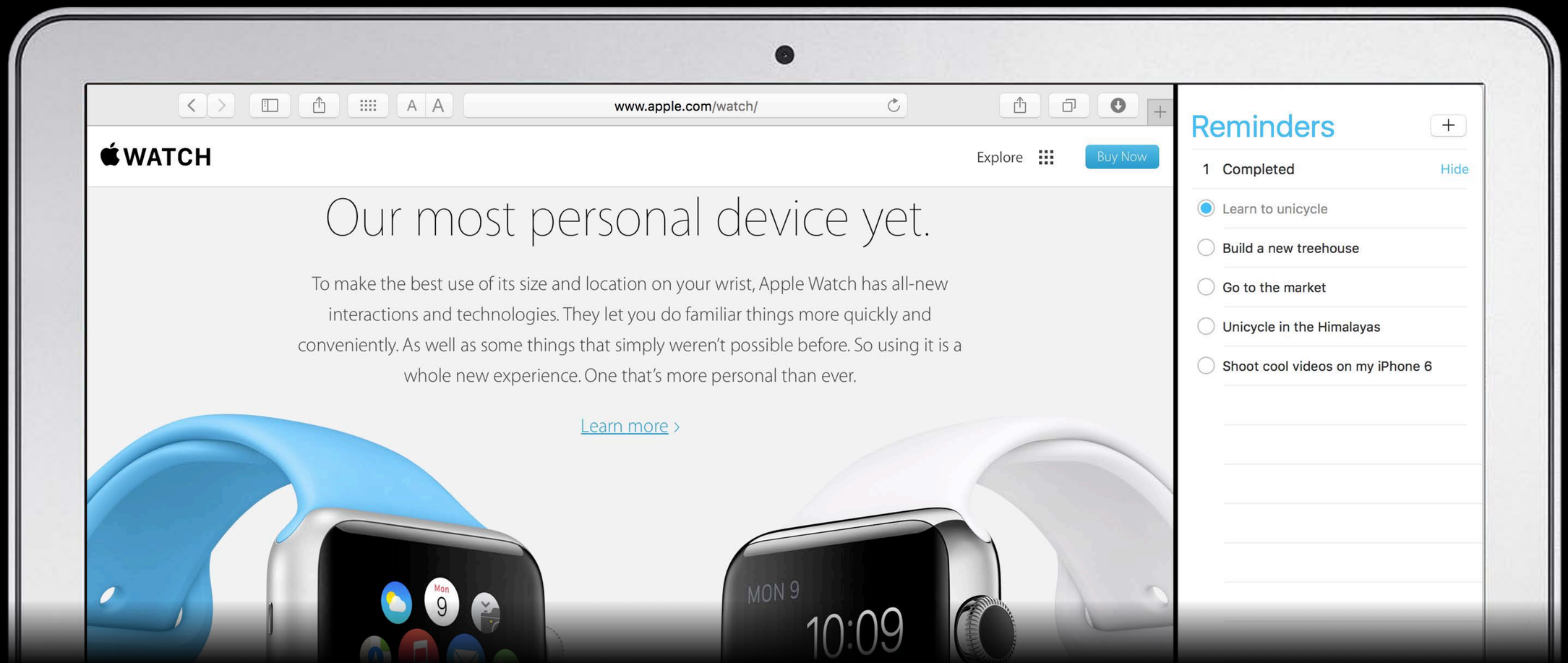


Full Screen Tiles

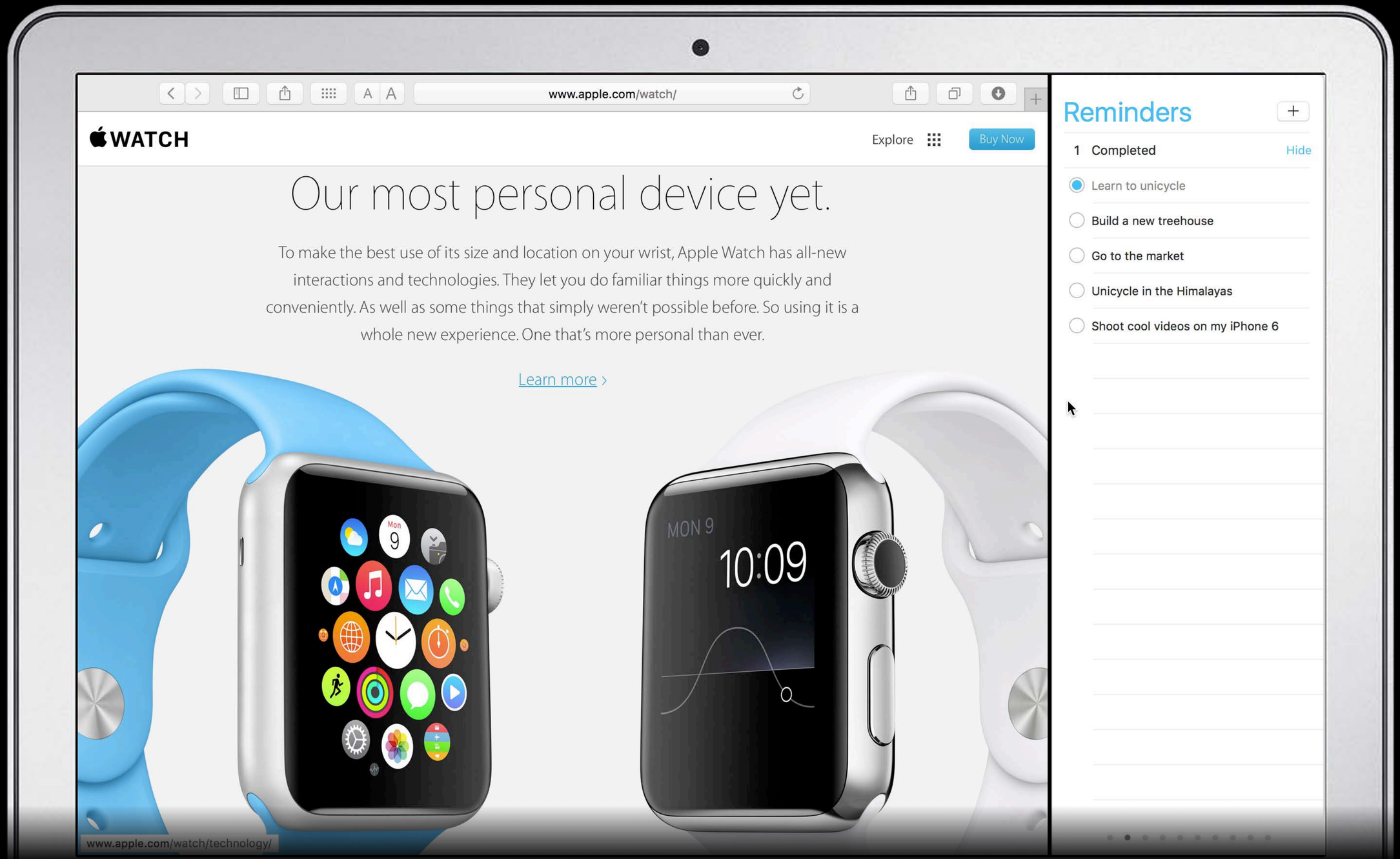
User benefits

Focus attention single task that may involve multiple windows

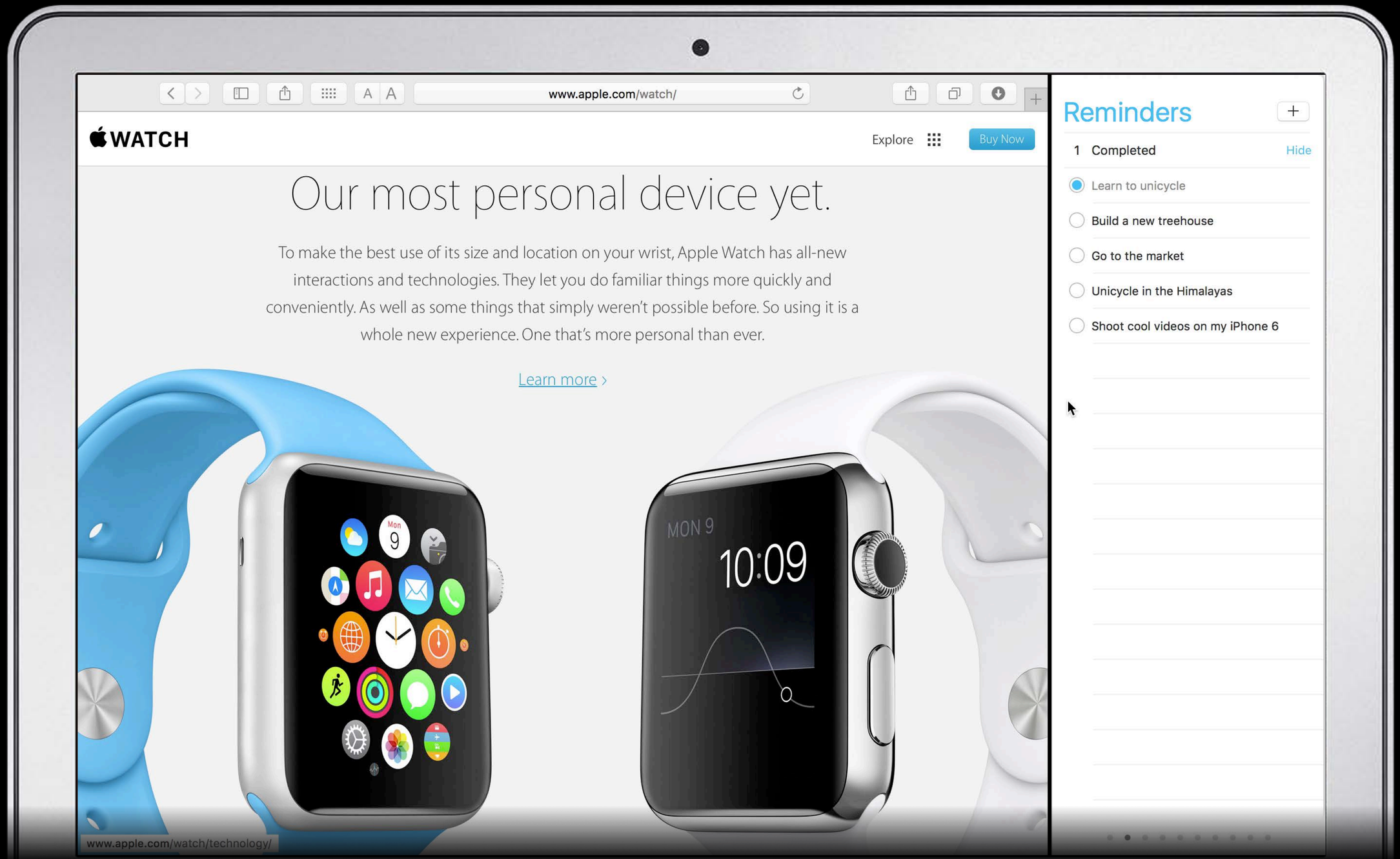
Still make the most of screen real estate



Supporting Full Screen Tiles



Supporting Full Screen Tiles



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:

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

Windows can explicitly disallow full screen tiling with:

```
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 `minSize` and `maxSize`
  - Or `contentMinSize / 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

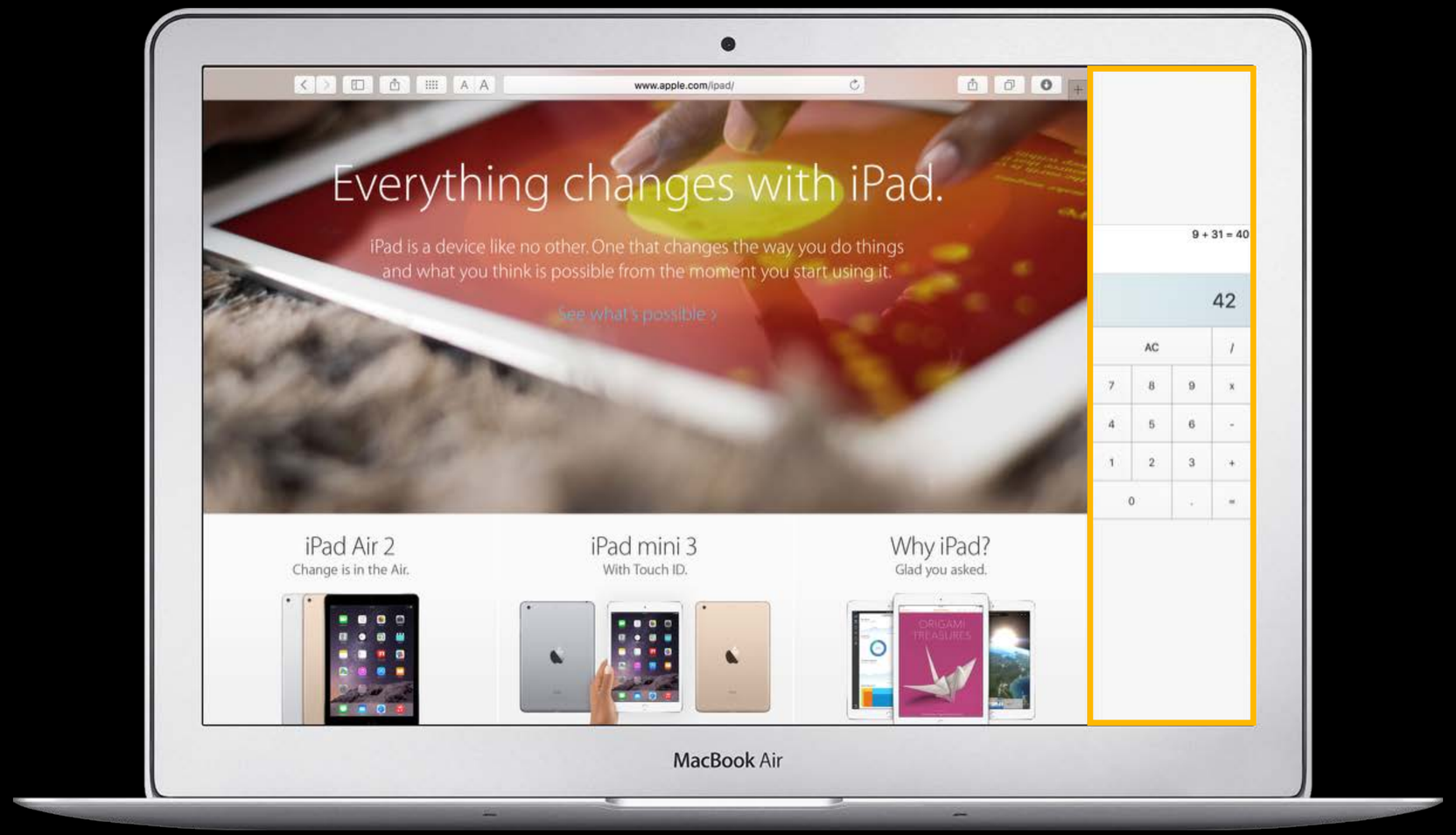
If allowed to tile, can it fit?

If a full screen window layout is significantly different from non full screen

Supporting Full Screen Tiles

If allowed to tile, can it fit?

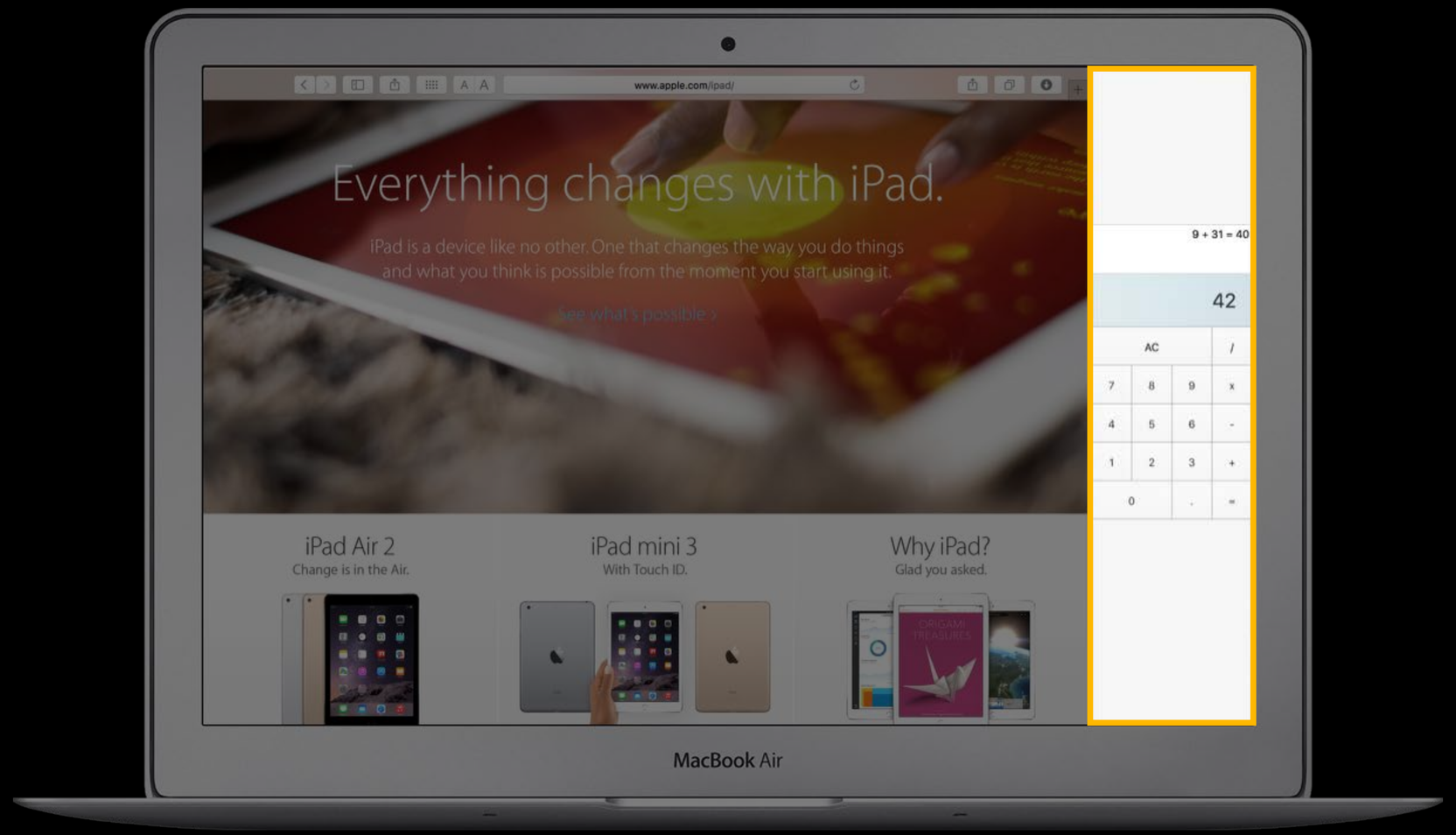
If a full screen window layout is significantly different from non full screen



Supporting Full Screen Tiles

If allowed to tile, can it fit?

If a full screen window layout is significantly different from non full screen



Supporting Full Screen Tiles

Dynamically changing the UI

```
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

```
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 when 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

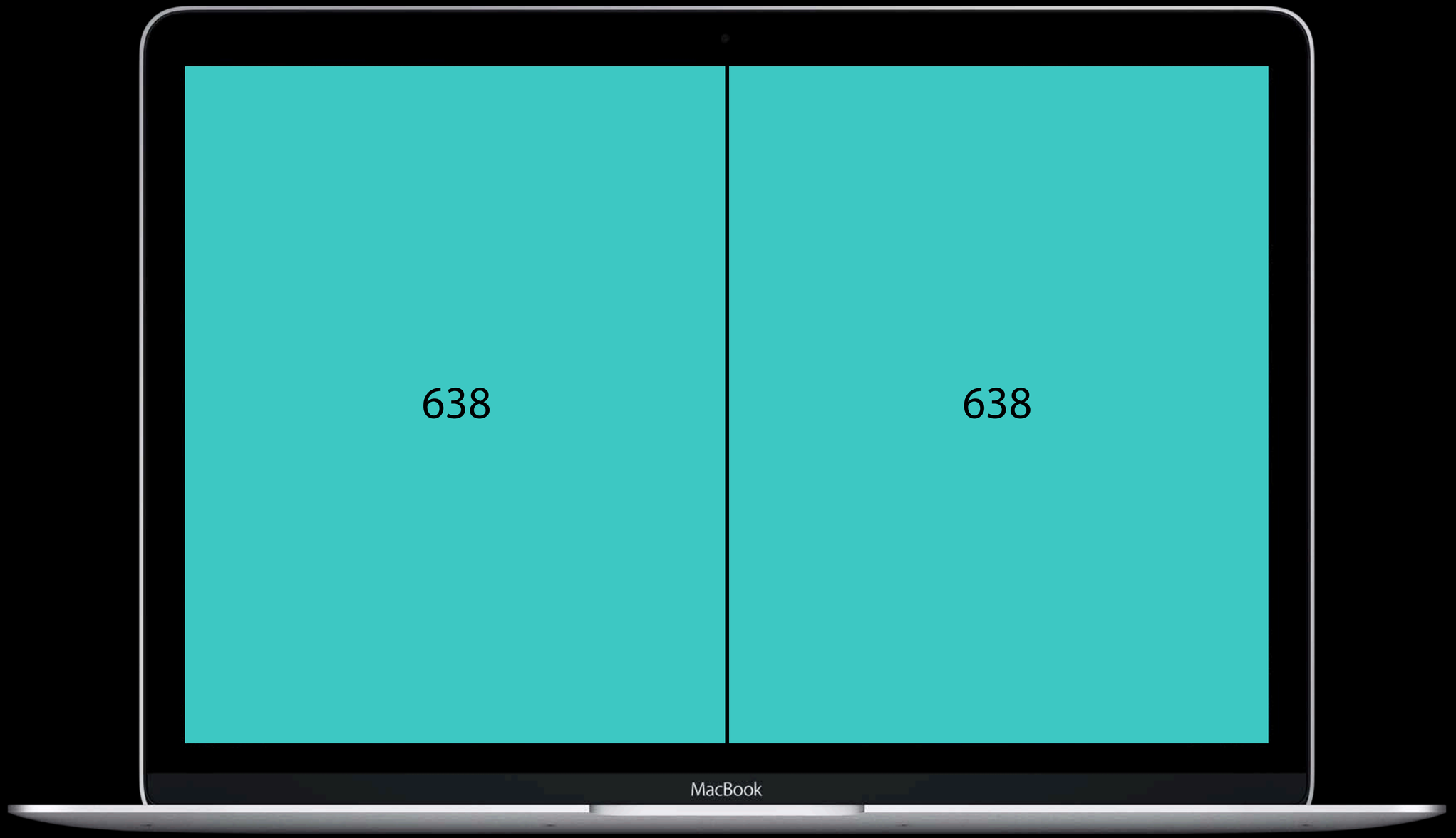
Taylor Kelly AppKit Software Engineer

Flexible Layout

Flexible Layout

Flexible Layout

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

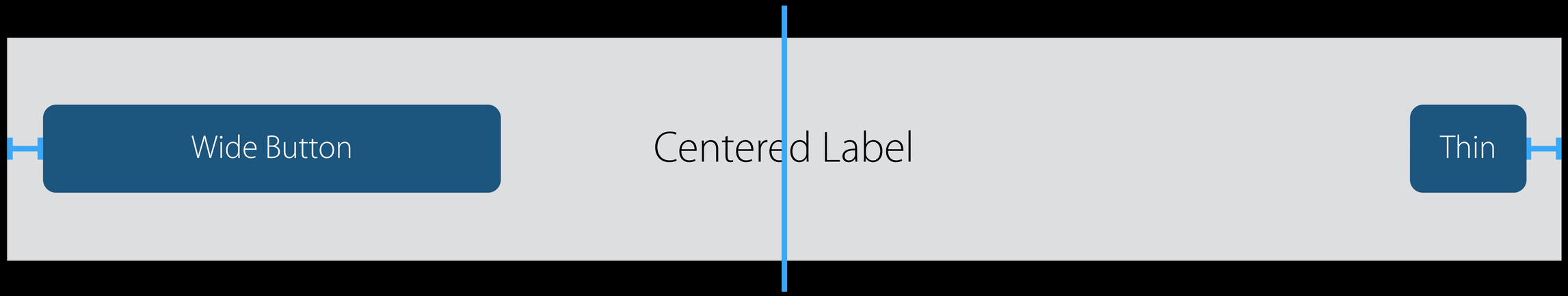
Mysteries of Auto Layout, Part 1	Presidio	Thursday 11:00AM
Mysteries of Auto Layout, Part 2	Presidio	Thursday 1:30PM

Auto Layout

Priorities

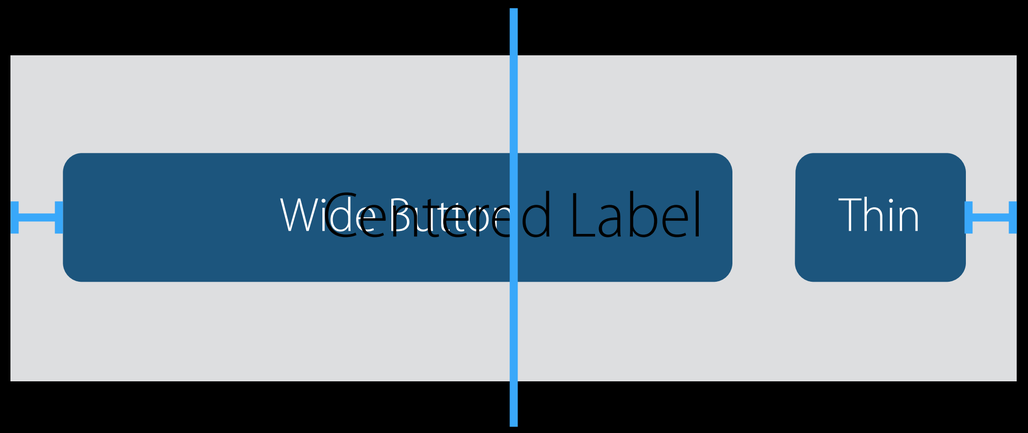
Auto Layout

Priorities



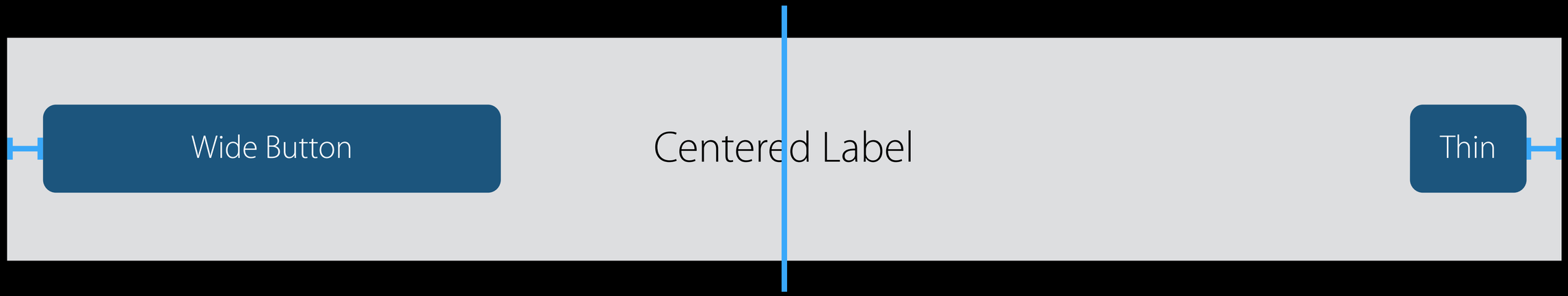
Auto Layout

Priorities



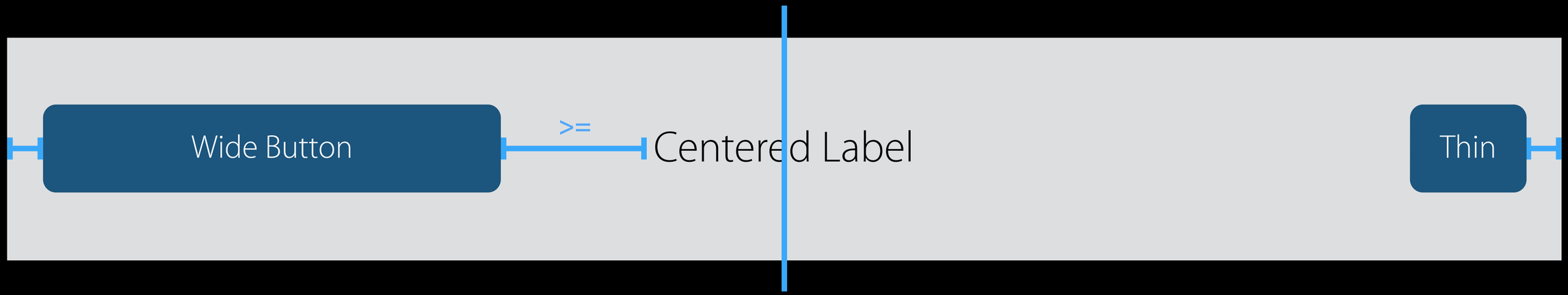
Auto Layout

Priorities



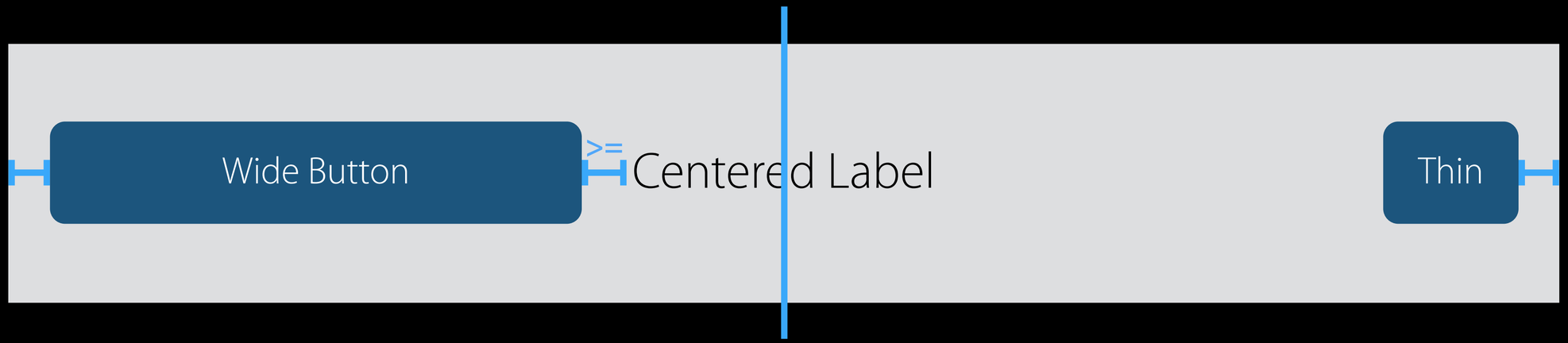
Auto Layout

Priorities



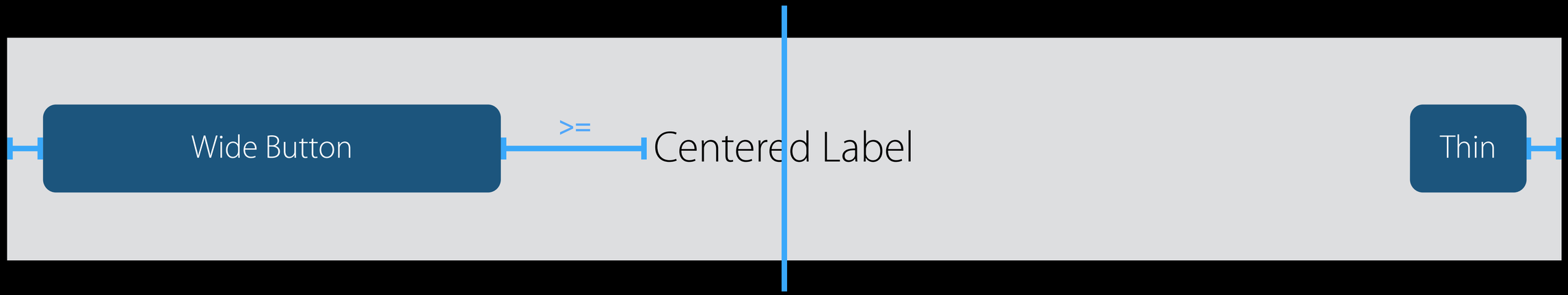
Auto Layout

Priorities



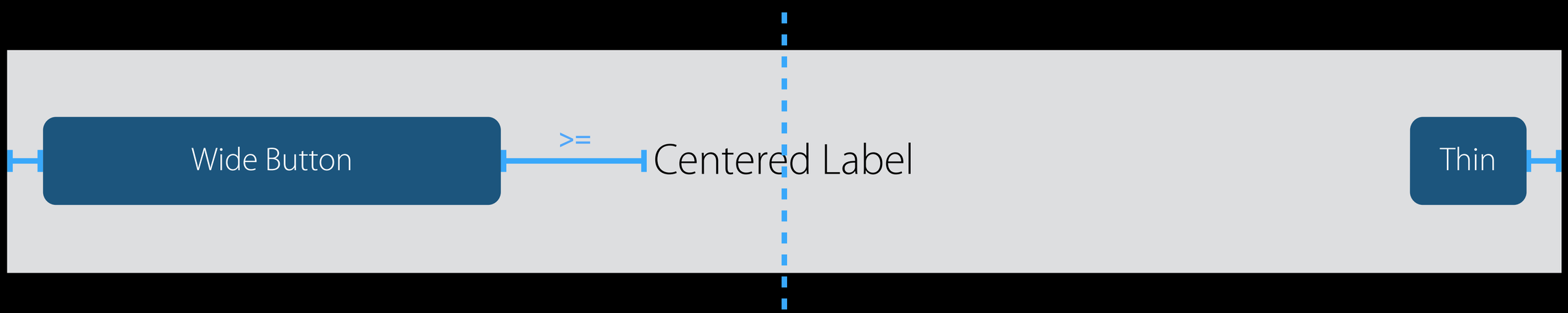
Auto Layout

Priorities



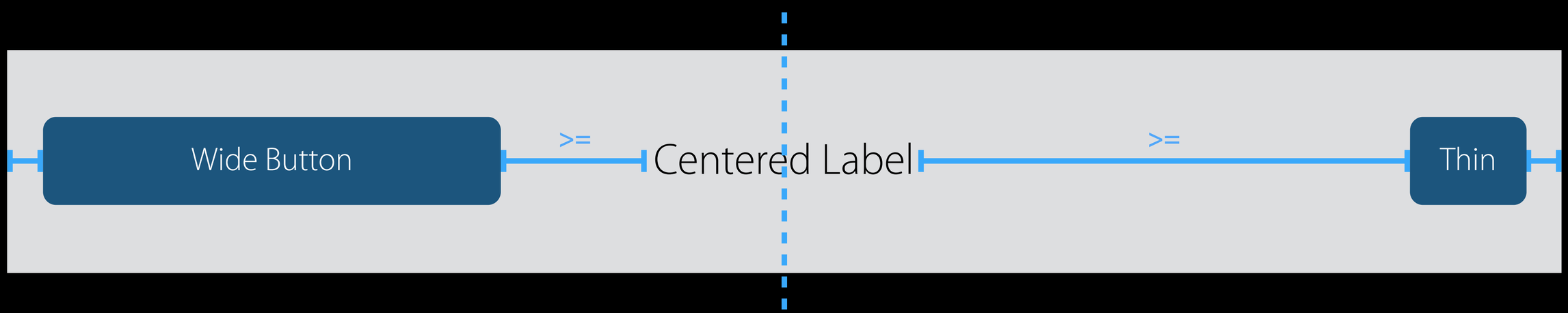
Auto Layout

Priorities



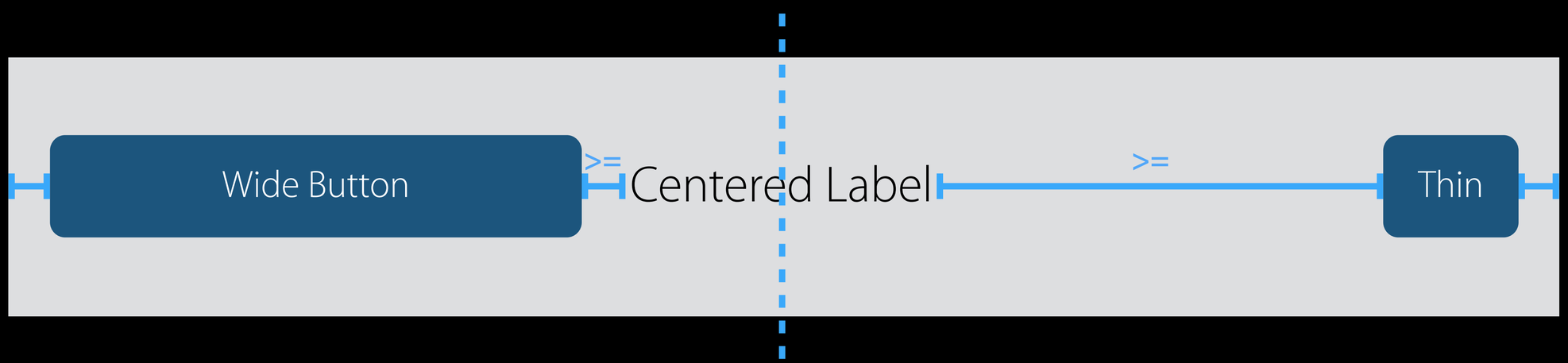
Auto Layout

Priorities



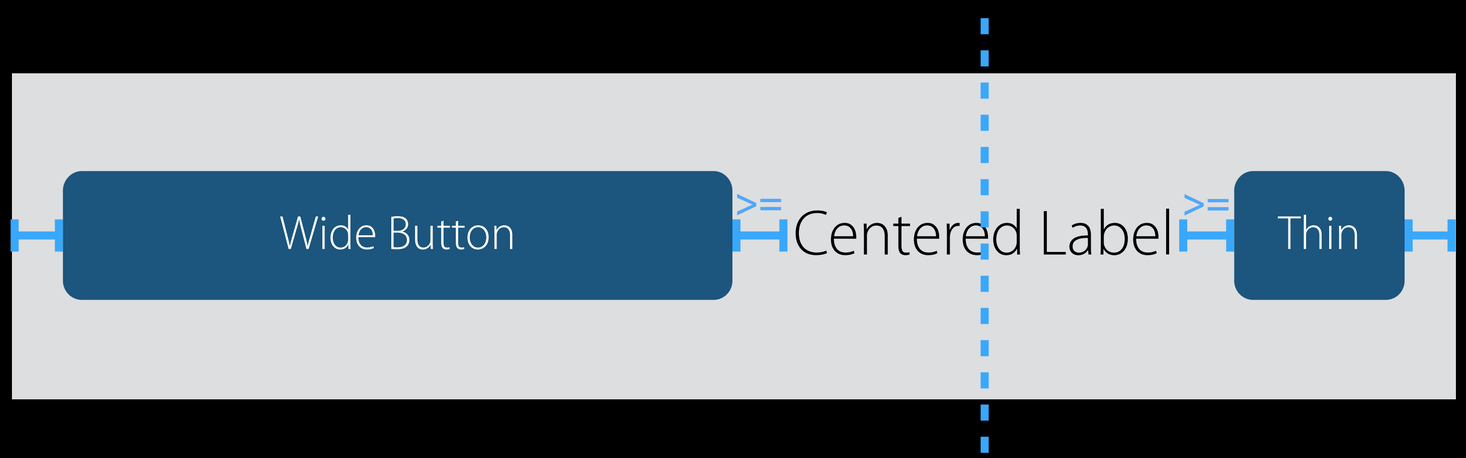
Auto Layout

Priorities



Auto Layout

Priorities



Auto Layout

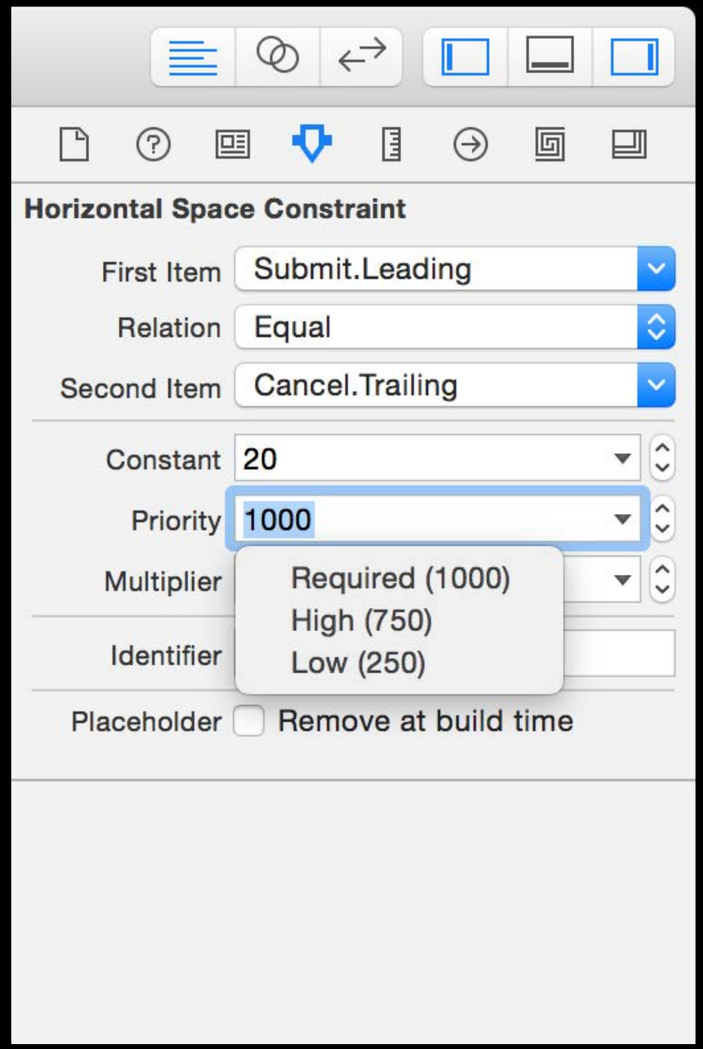
Priorities

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

Auto Layout

Priorities

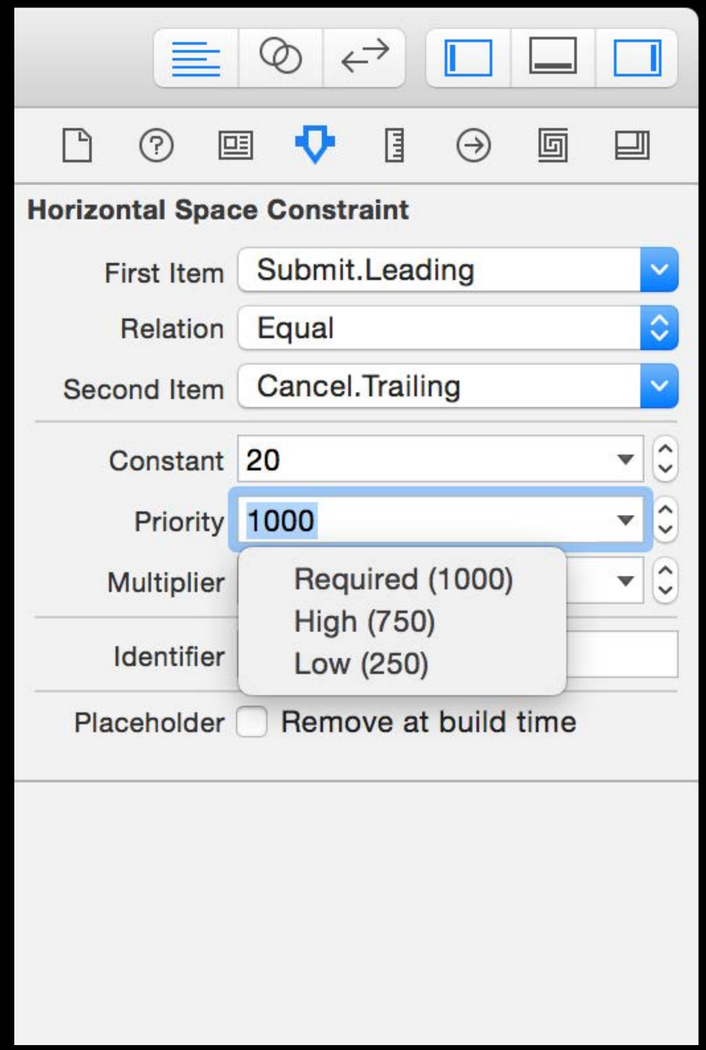
```
class NSLayoutConstraint {
  var priority: NSLayoutPriority
```



Auto Layout

Priorities

```
class NSLayoutConstraint {
  var priority: NSLayoutPriority


NSLayoutPriorityRequired = 1000.0
```



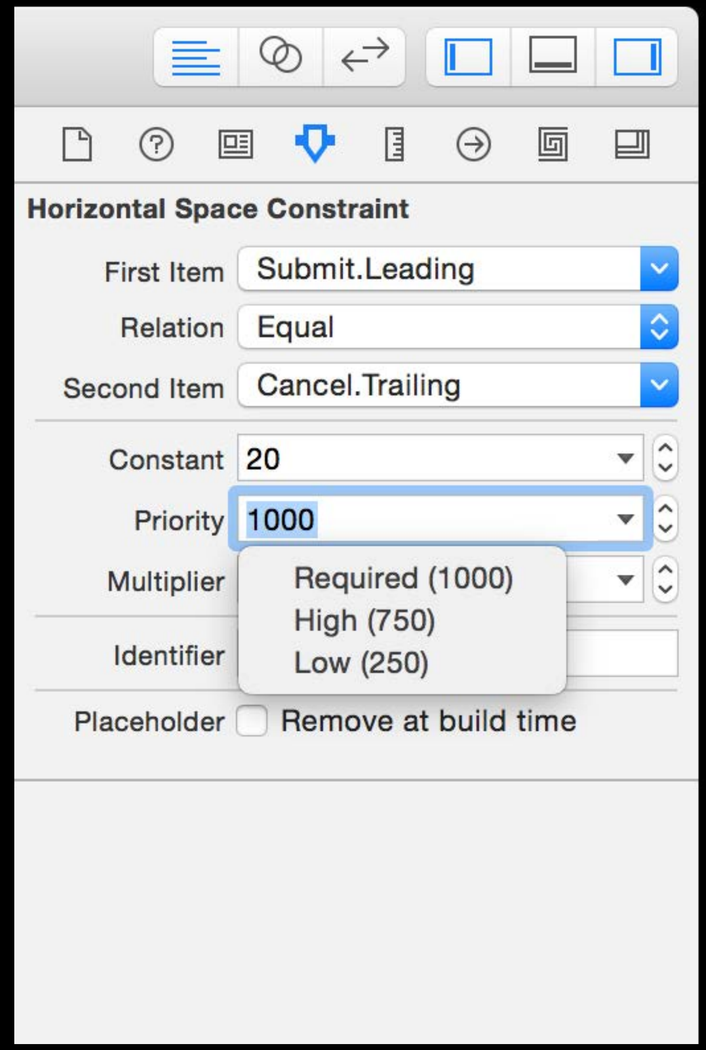
Auto Layout

Priorities

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

```
NSLayoutPriorityRequired = 1000.0
```

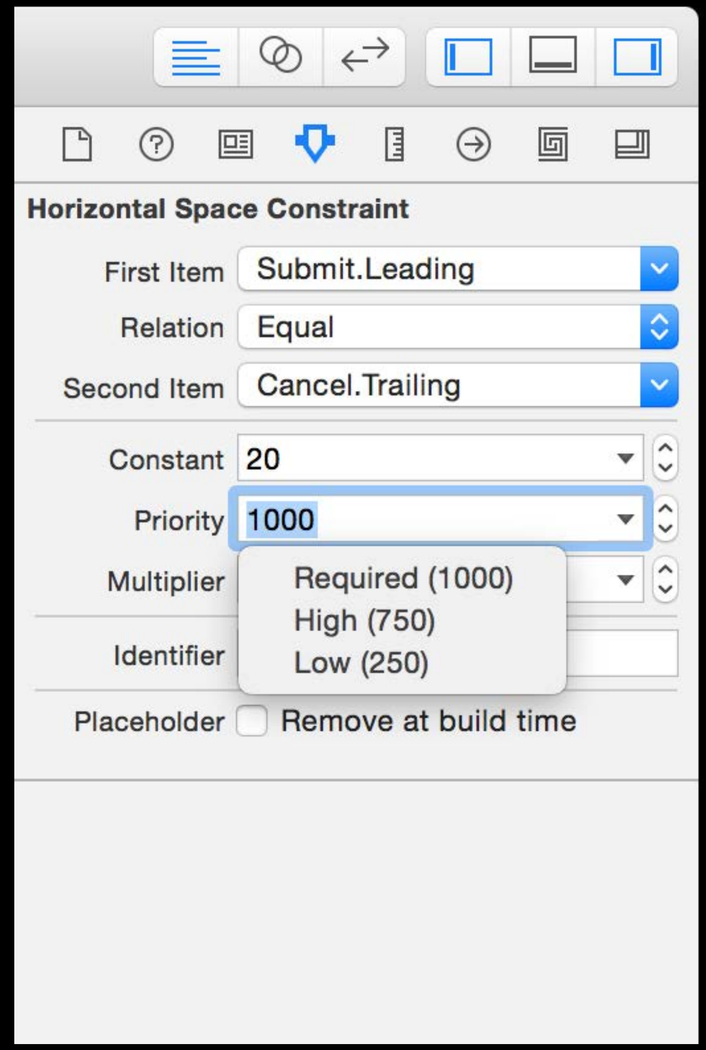
```
NSLayoutPriorityDefaultLow = 250.0
```



Auto Layout

Priorities

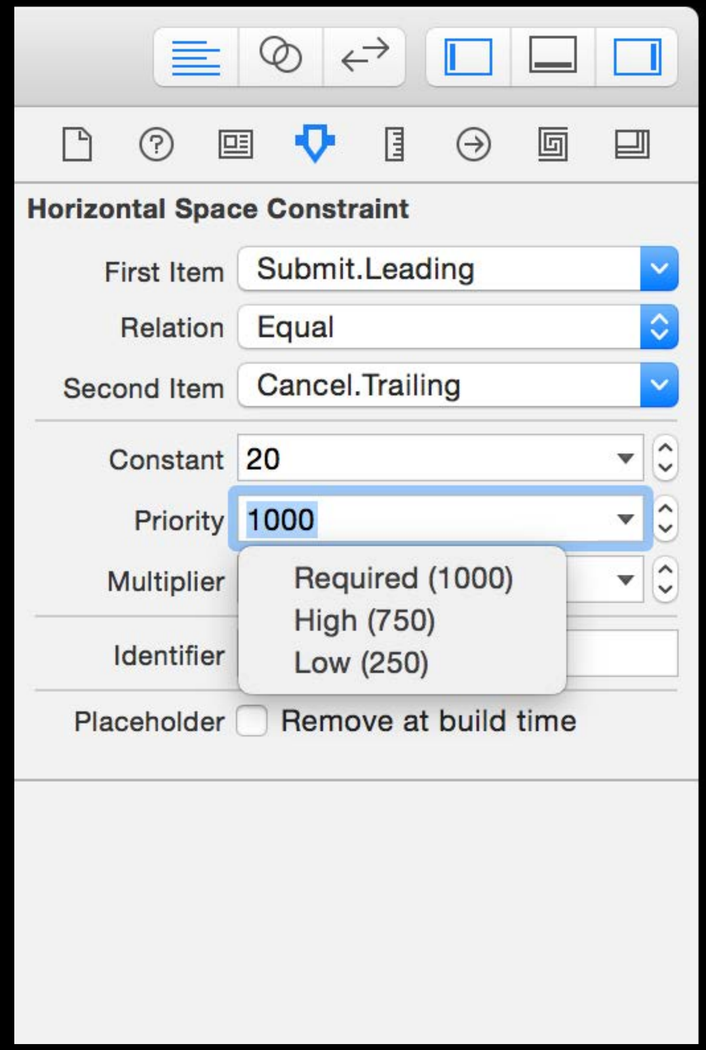
```
class NSLayoutConstraint {
  var priority: NSLayoutPriority


NSLayoutPriorityRequired = 1000.0

NSLayoutPriorityDefaultLow = 250.0
NSLayoutPriorityDragThatCannotResizeWindow = 490.0
```



Auto Layout

Priorities

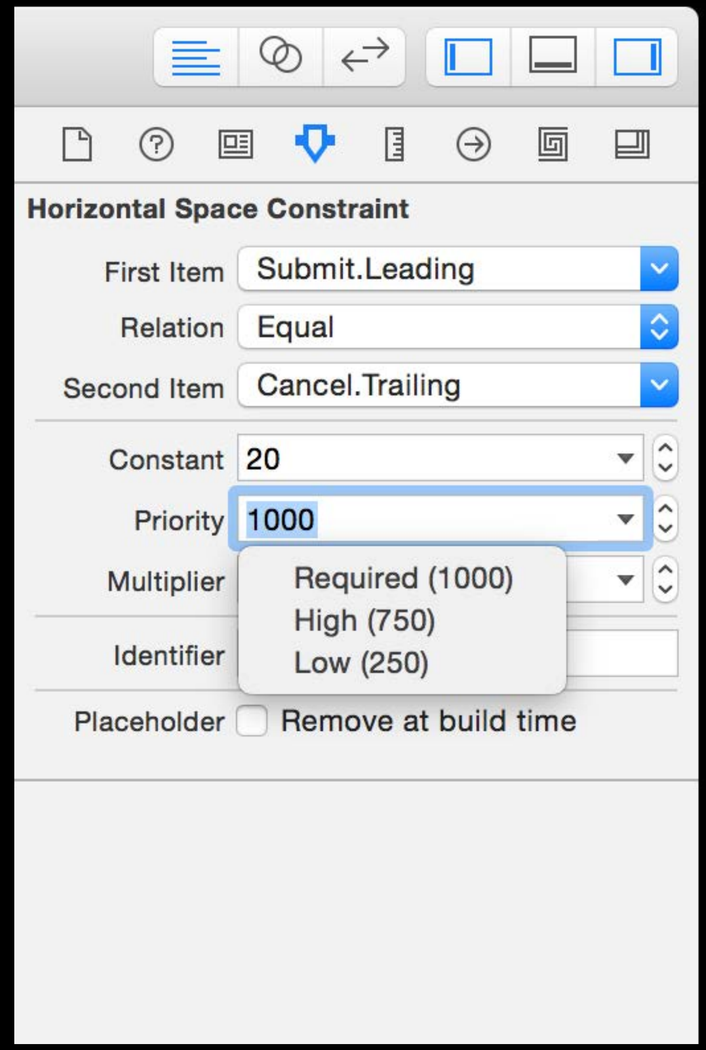
```
class NSLayoutConstraint {
  var priority: NSLayoutPriority


NSLayoutPriorityRequired = 1000.0


  NSLayoutPriorityDefaultLow = 250.0
  NSLayoutPriorityDragThatCannotResizeWindow = 490.0
  NSLayoutPriorityWindowSizeStayPut = 500.0
```



Auto Layout

Priorities

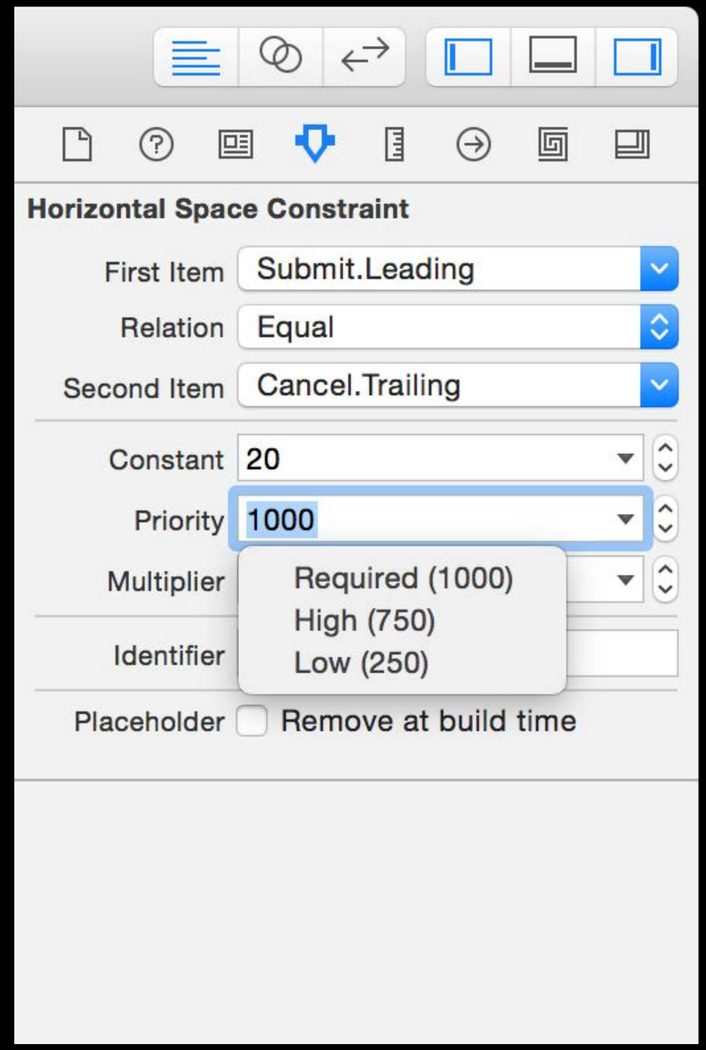
```
class NSLayoutConstraint {
  var priority: NSLayoutPriority


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

NEW

Sidebars

NSSplitViewController

New in 10.11

NEW

Sidebars

Spring Loading

NSSplitViewController

NEW

New in 10.11

Sidebars

Spring Loading

Metrics

- Minimum / maximum thickness
- Proportional resizing
- Automatic maximum thickness
- Preferred thickness fraction

Apple Inc.

Favorites

WWDC15

News Schedule Sessions Labs Events More

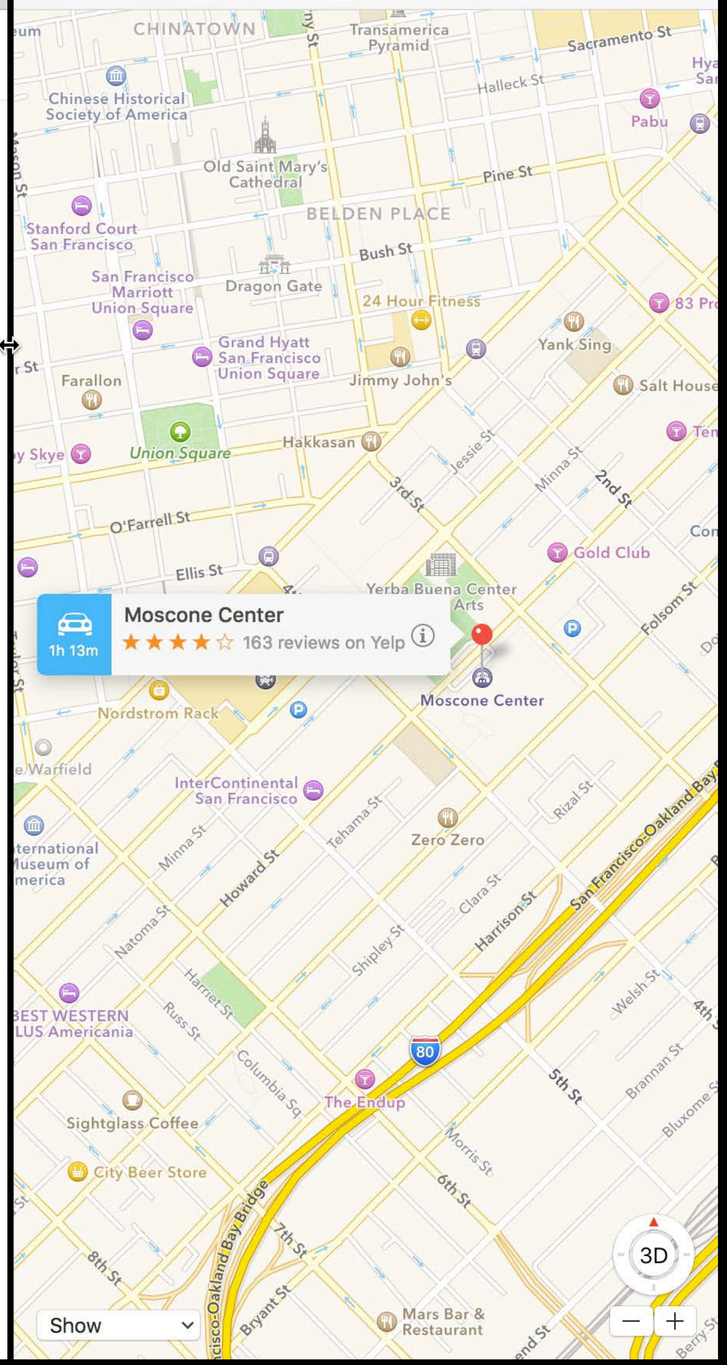
Apple Worldwide Developers Conference.
June 8-12, San Francisco.

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

Edit

Directions

Moscone Center



Apple Inc.

Favorites

WWDC15

News Schedule Sessions Labs Events More

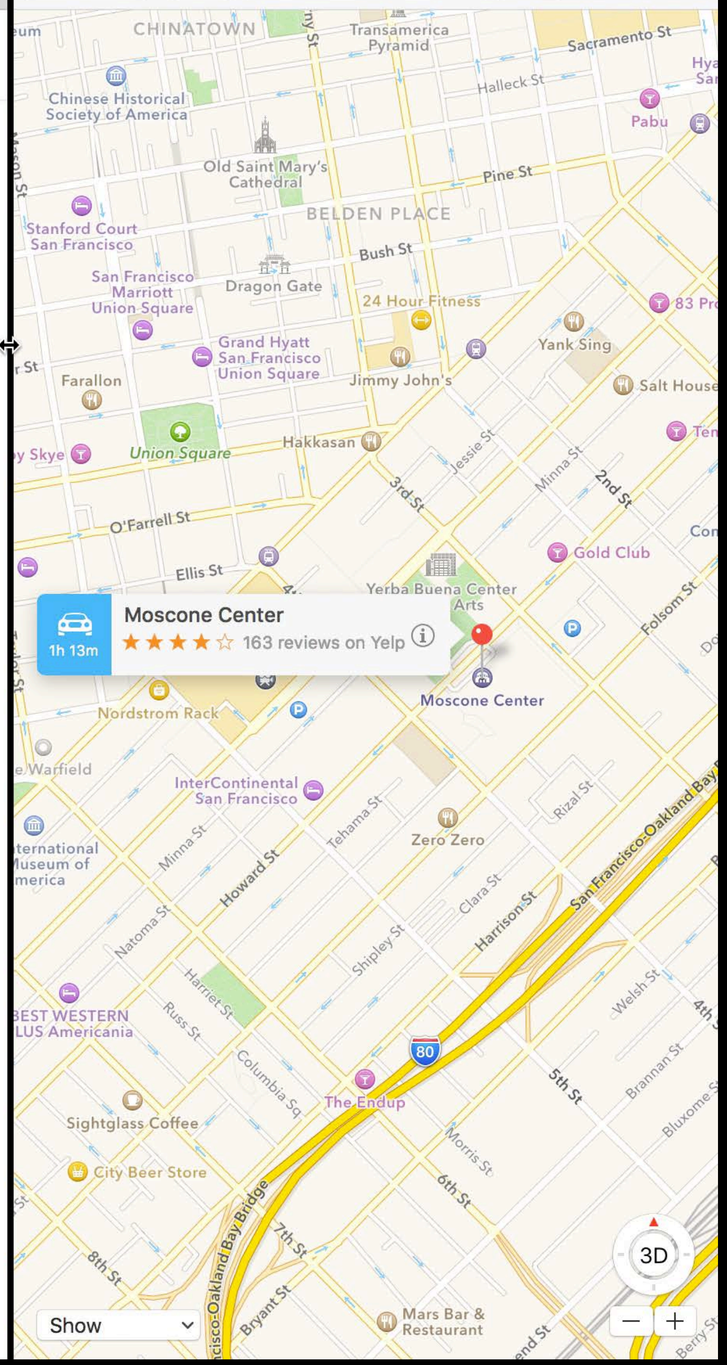
Apple Worldwide Developers Conference.
June 8-12, San Francisco.

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

Edit

Directions

Moscone Center

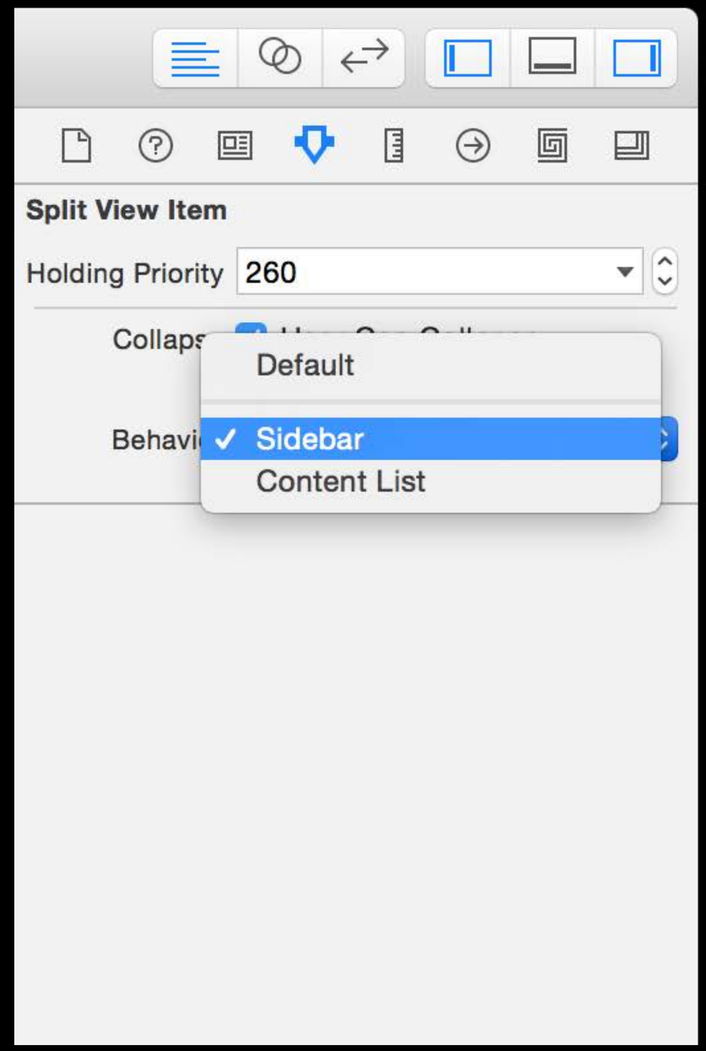


Sidebars

```
class NSSplitViewItem {
  init(sidebarWithViewController: NSViewController)
```

Sidebars

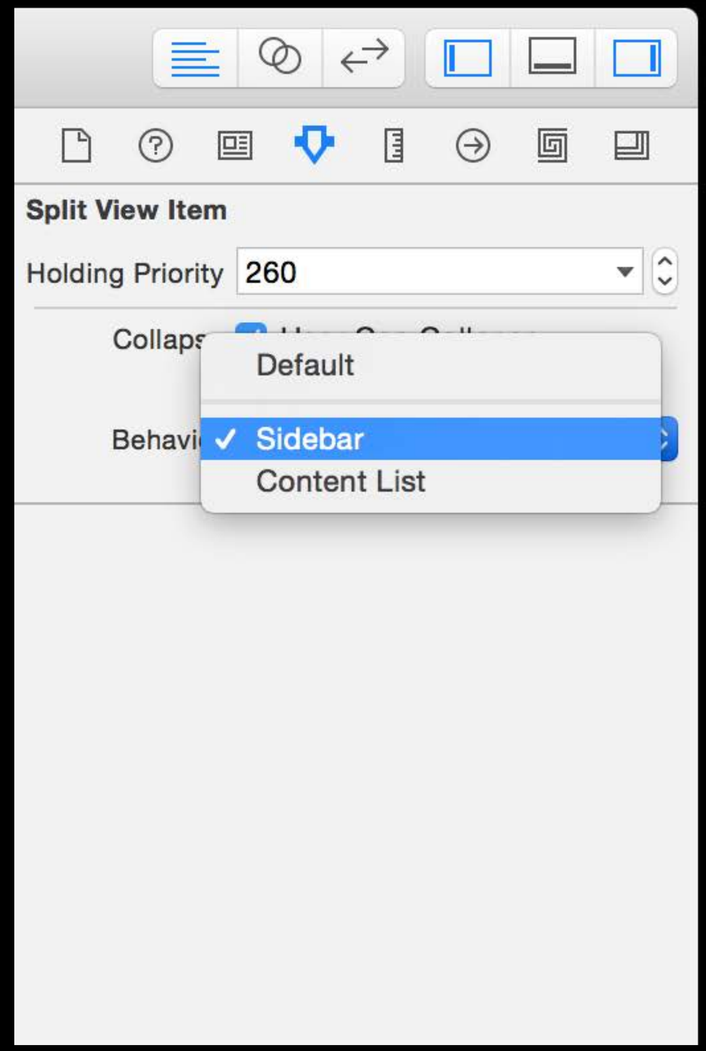
```
class NSSplitViewItem {
  init(sidebarWithViewController: NSViewController)
```



Sidebars

```
class NSSplitViewItem {
  init(sidebarWithViewController: NSViewController)
```

Material background and vibrant divider

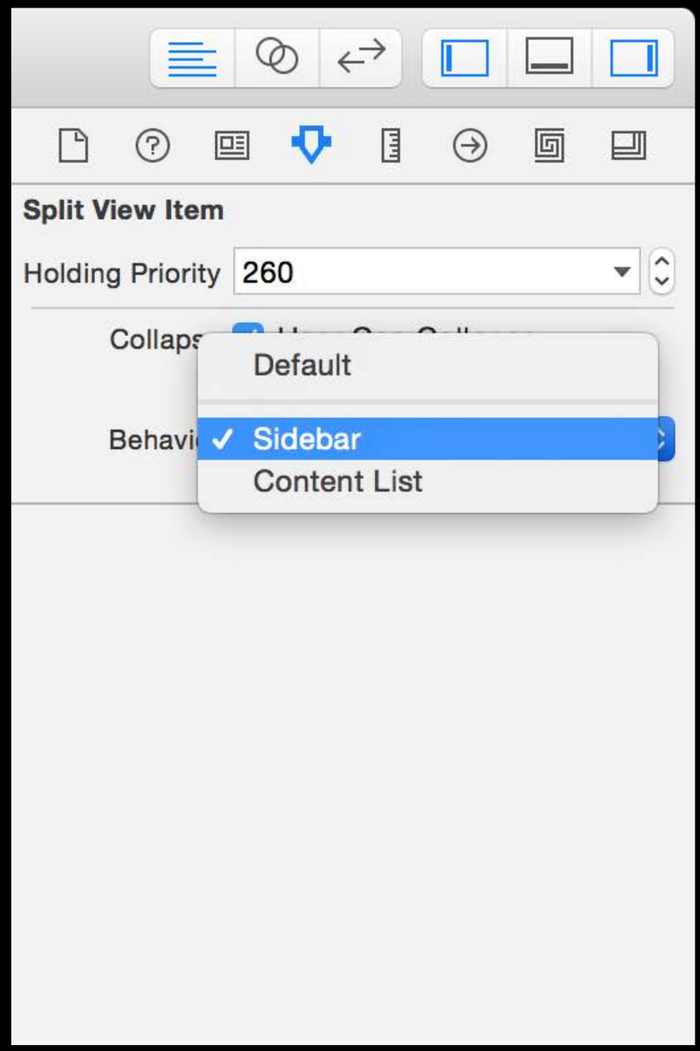


Sidebars

```
class NSSplitViewItem {
  init(sidebarWithViewController: NSViewController)
```

Material background and vibrant divider

Auto-collapse and -uncollapse



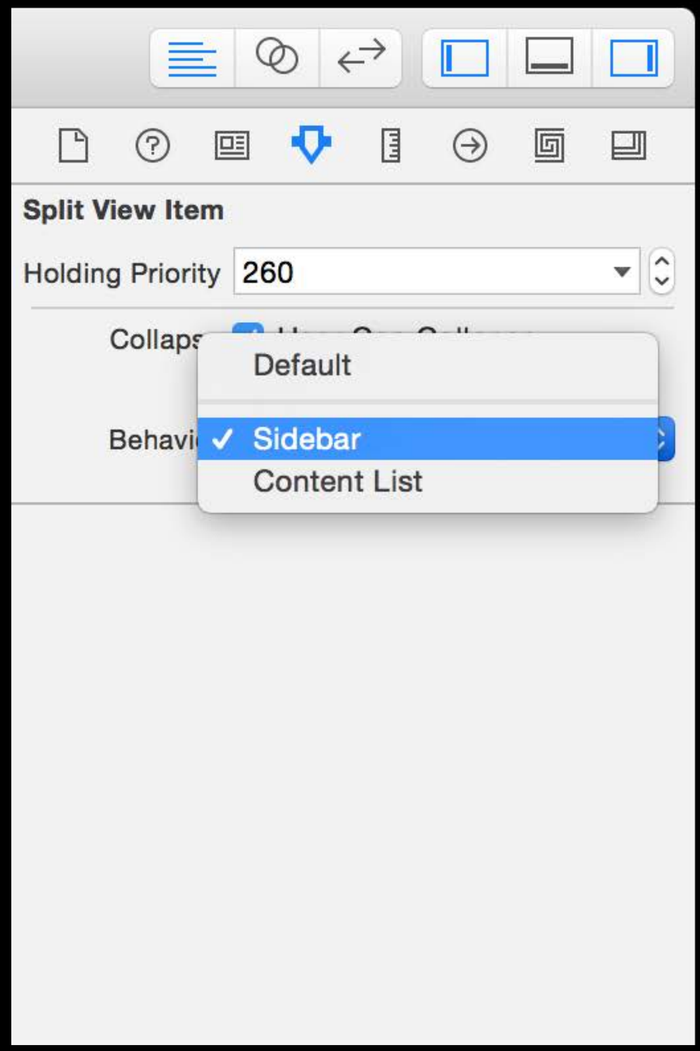
Sidebars

```
class NSSplitViewItem {
  init(sidebarWithViewController: NSViewController)
```

Material background and vibrant divider

Auto-collapse and -uncollapse

Overlays



Sidebars

```
class NSSplitViewItem {
  init(sidebarWithViewController: NSViewController)
```

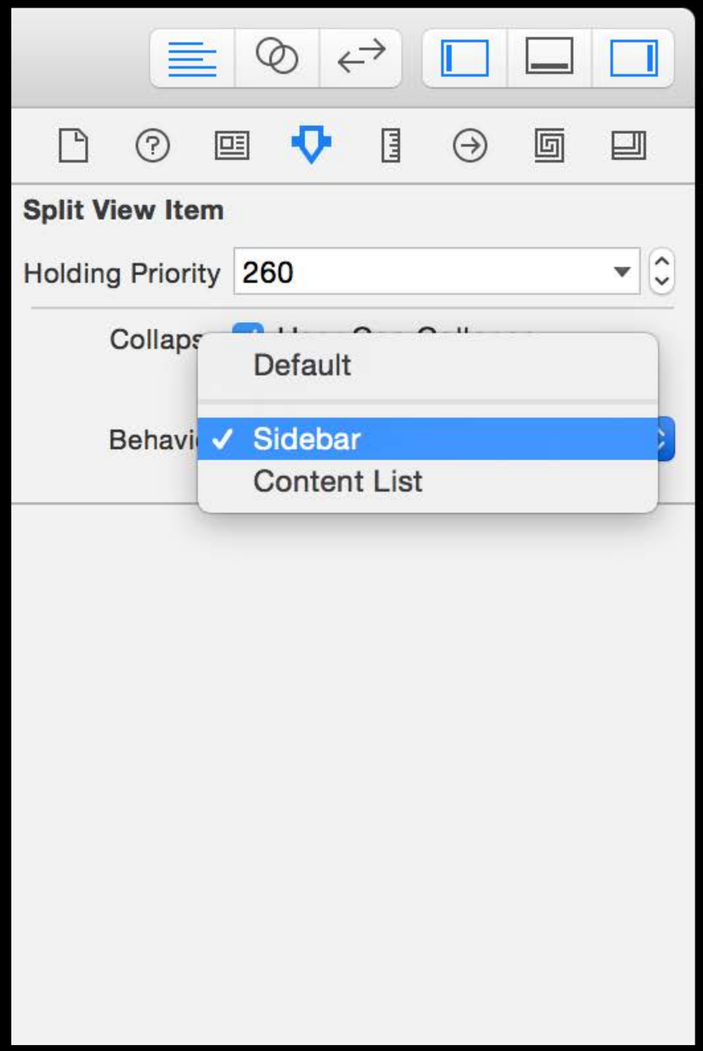
Material background and vibrant divider

Auto-collapse and -uncollapse

Overlays

Standard metrics

```
  minimumThickness, maximumThickness
  preferredThicknessFraction
  springLoaded = true
  canCollapse = true
```



Sidebars

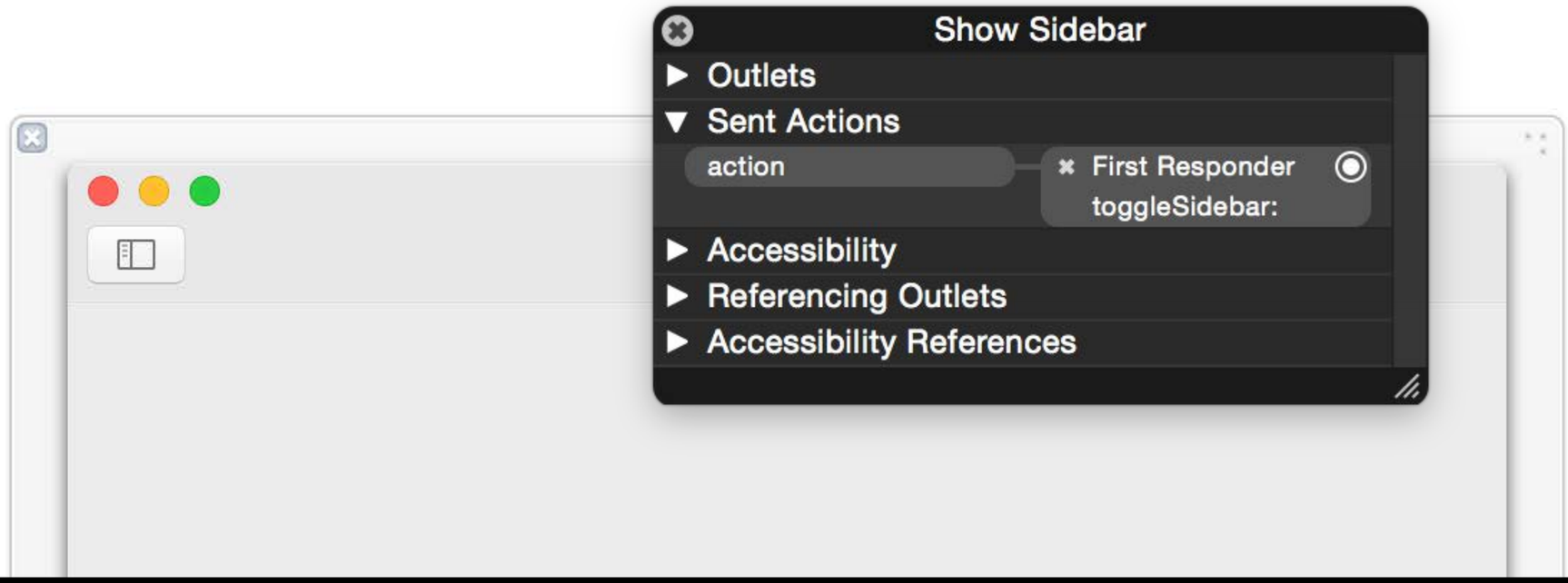
```
class NSSplitViewController {
  @IBAction func toggleSidebar(AnyObject?)
```

Sidebars

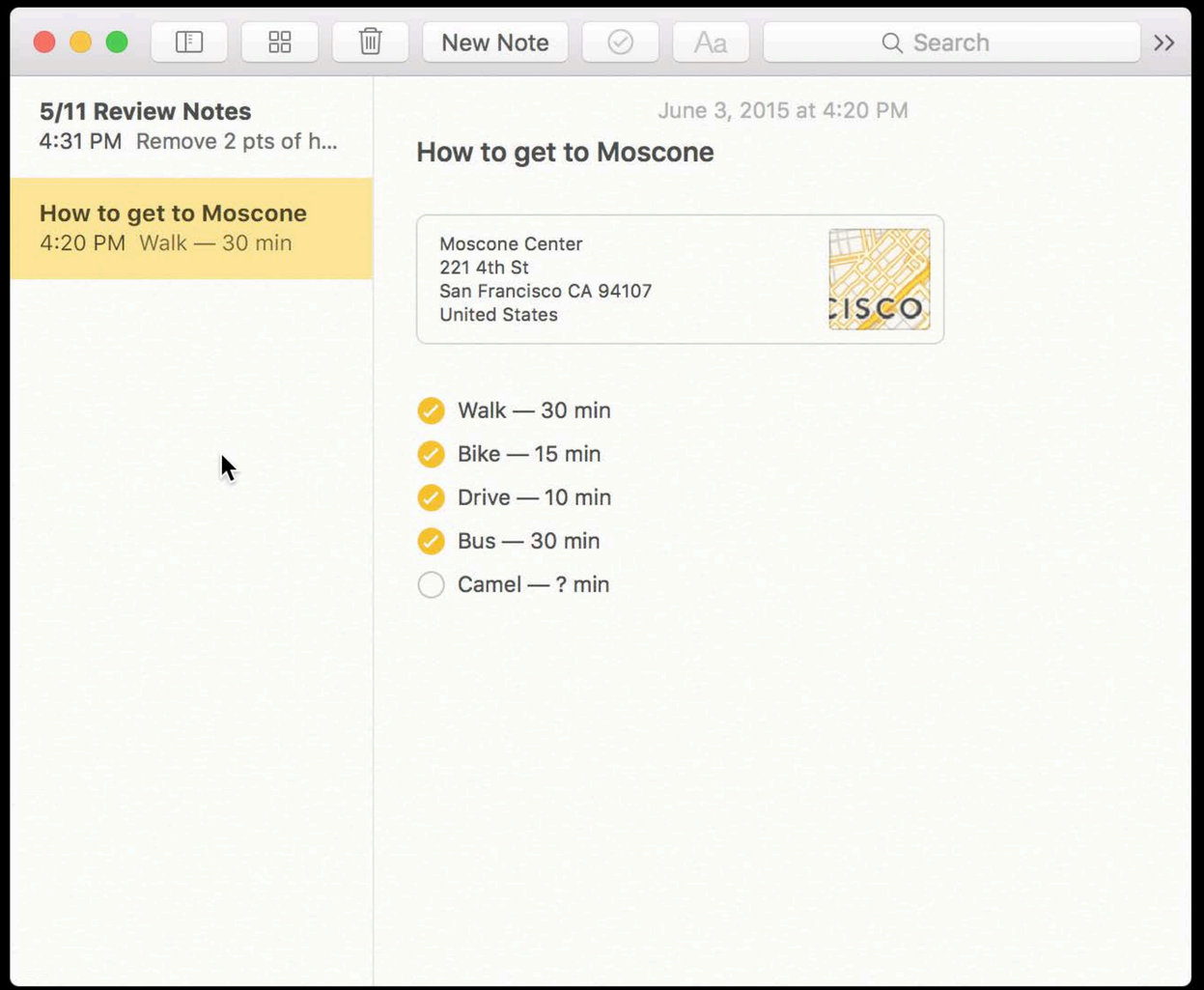
```
class NSSplitViewController {
  @IBAction func toggleSidebar(AnyObject?)
```

Sidebars

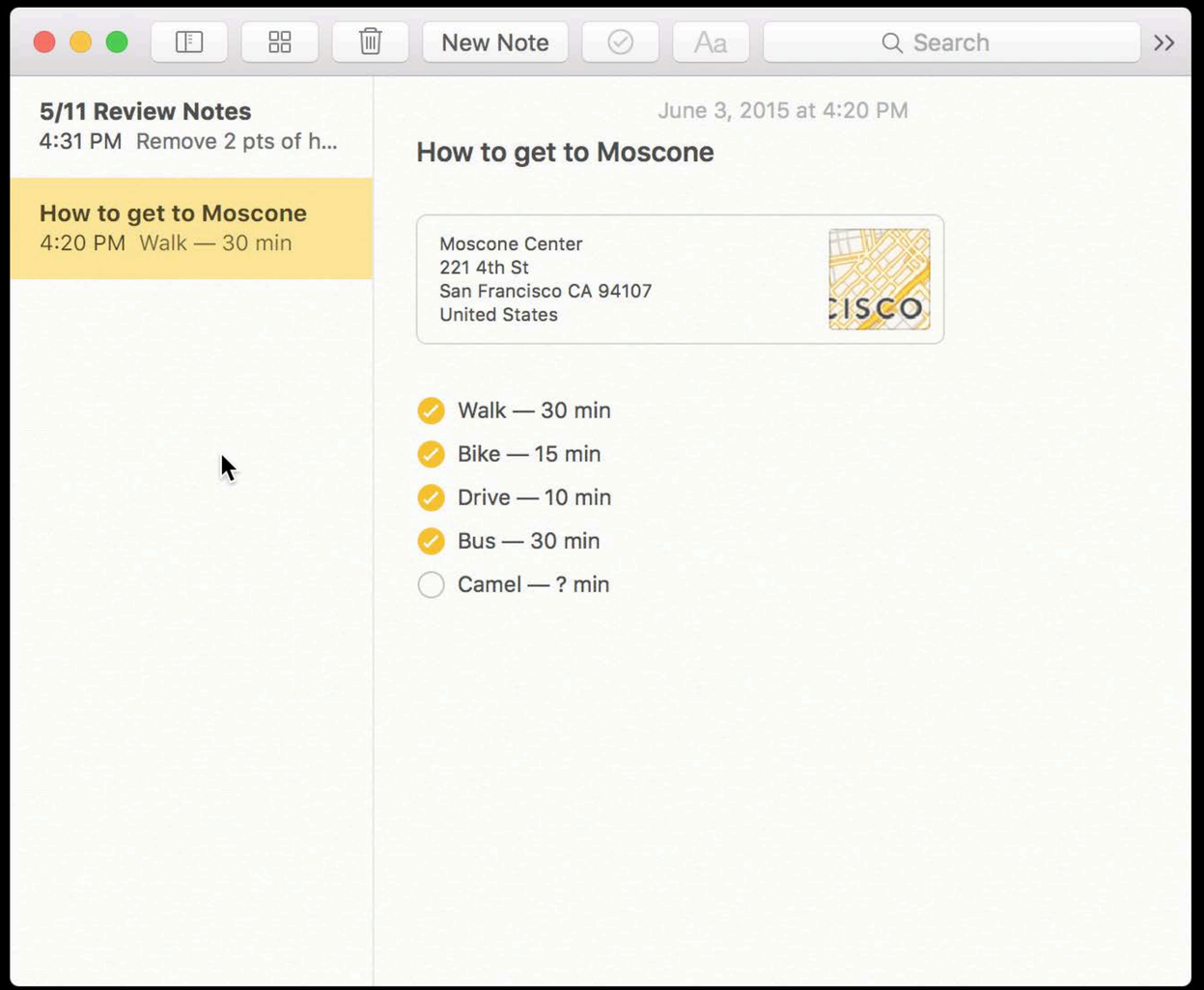
```
class NSSplitViewController {
  @IBAction func toggleSidebar(AnyObject?)
```



Spring Loading



Spring Loading



Spring Loading

```
class NSSplitViewItem {
  var springLoaded: Bool
```

Spring Loading

```
class NSSplitViewItem {
  var springLoaded: Bool
```

Transiently uncollapse during drags

Spring Loading

```
class NSSplitViewItem {
  var springLoaded: Bool
```

Transiently uncollapse during drags

Both sidebars and non-sidebars

Metrics

```
class NSSplitViewItem {
  var minimumThickness: CGFloat
  var maximumThickness: CGFloat
  var holdingPriority: NSLayoutPriority
  var preferredThicknessFraction: CGFloat
  var automaticMaximumThickness: CGFloat

class NSSplitViewController {
  var minimumThicknessForInlineSidebars: CGFloat
```

Metrics

```
class NSSplitViewItem {
  var minimumThickness: CGFloat                    (points)
  var maximumThickness: CGFloat                    (points)
```

Metrics

```
class NSSplitViewItem {
  var minimumThickness: CGFloat                    (points)
  var maximumThickness: CGFloat                    (points)
```

Metrics

```
class NSSplitViewItem {
  var minimumThickness: CGFloat                    (points)
  var maximumThickness: CGFloat                    (points)
```

Metrics

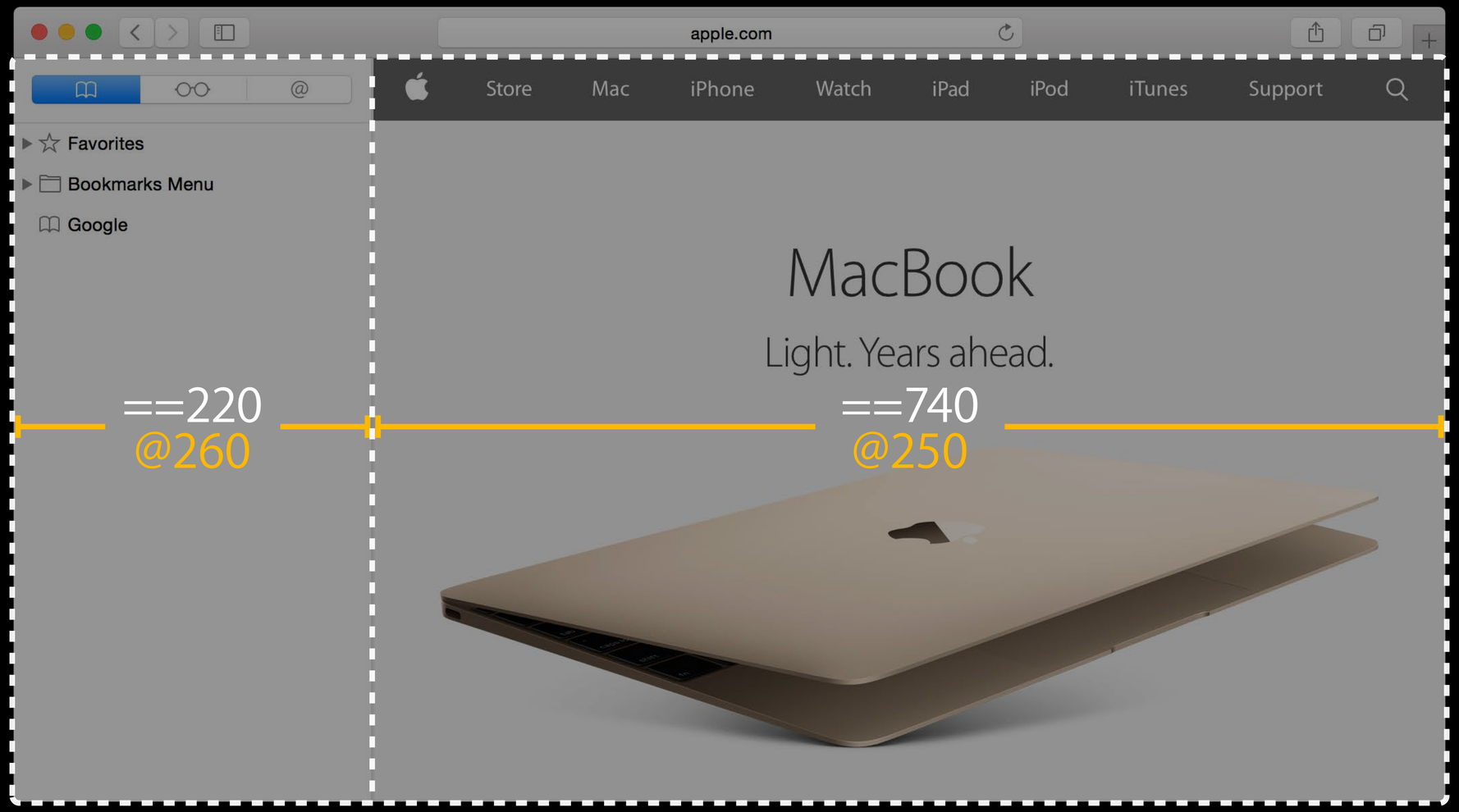
```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

(constraint priority)

Metrics

```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

(constraint priority)



Metrics

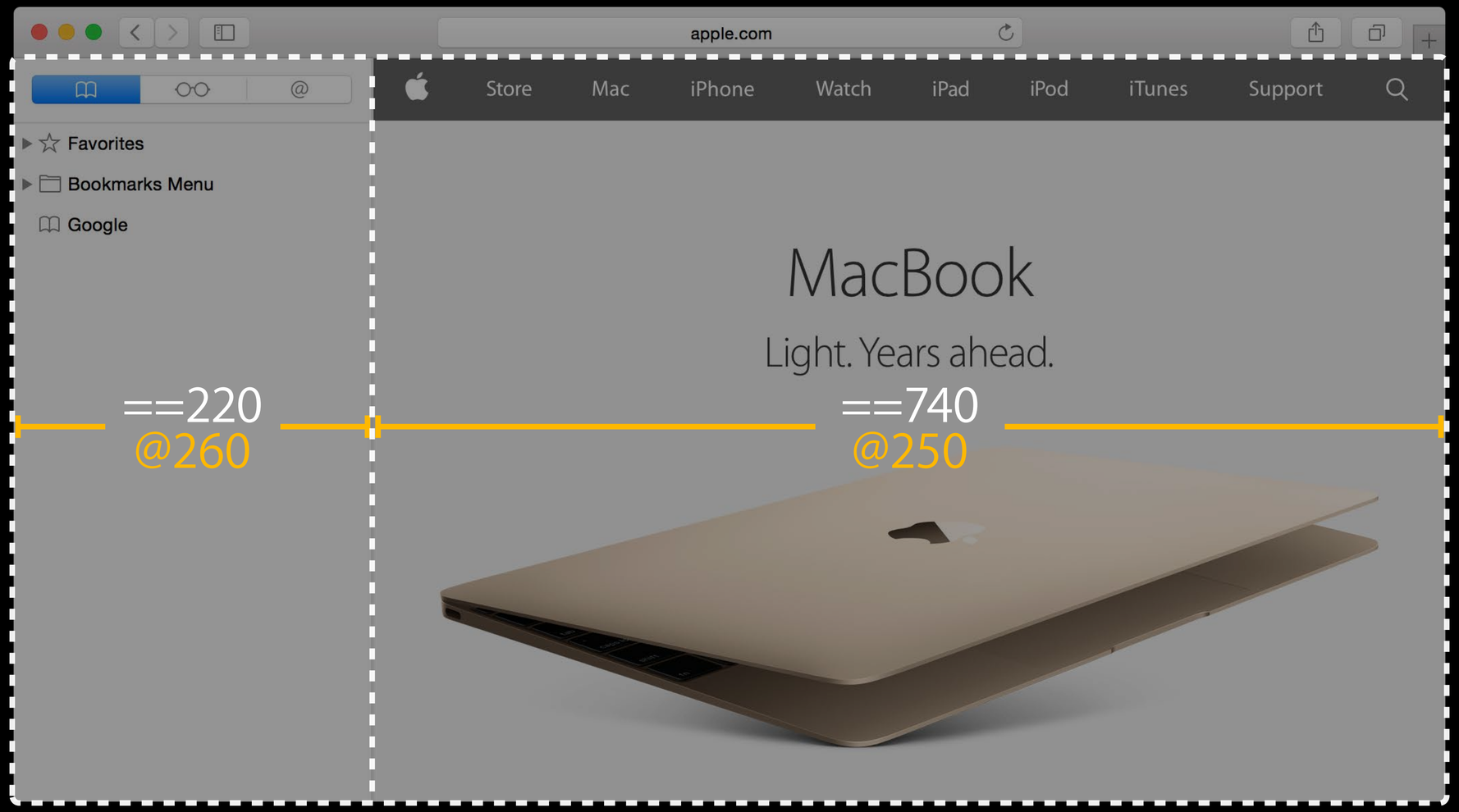
```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

(constraint priority)

Metrics

```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

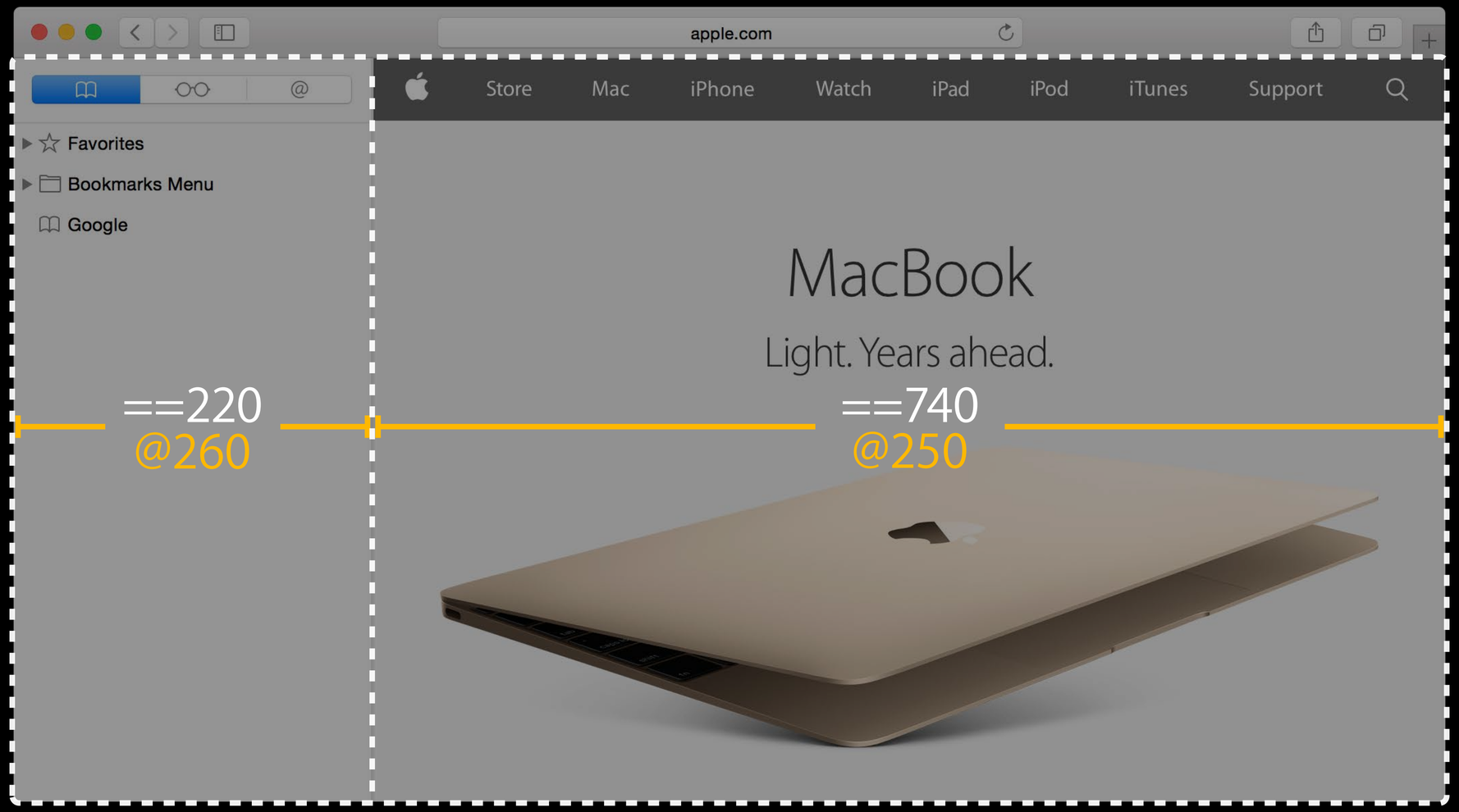
(constraint priority)



Metrics

```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

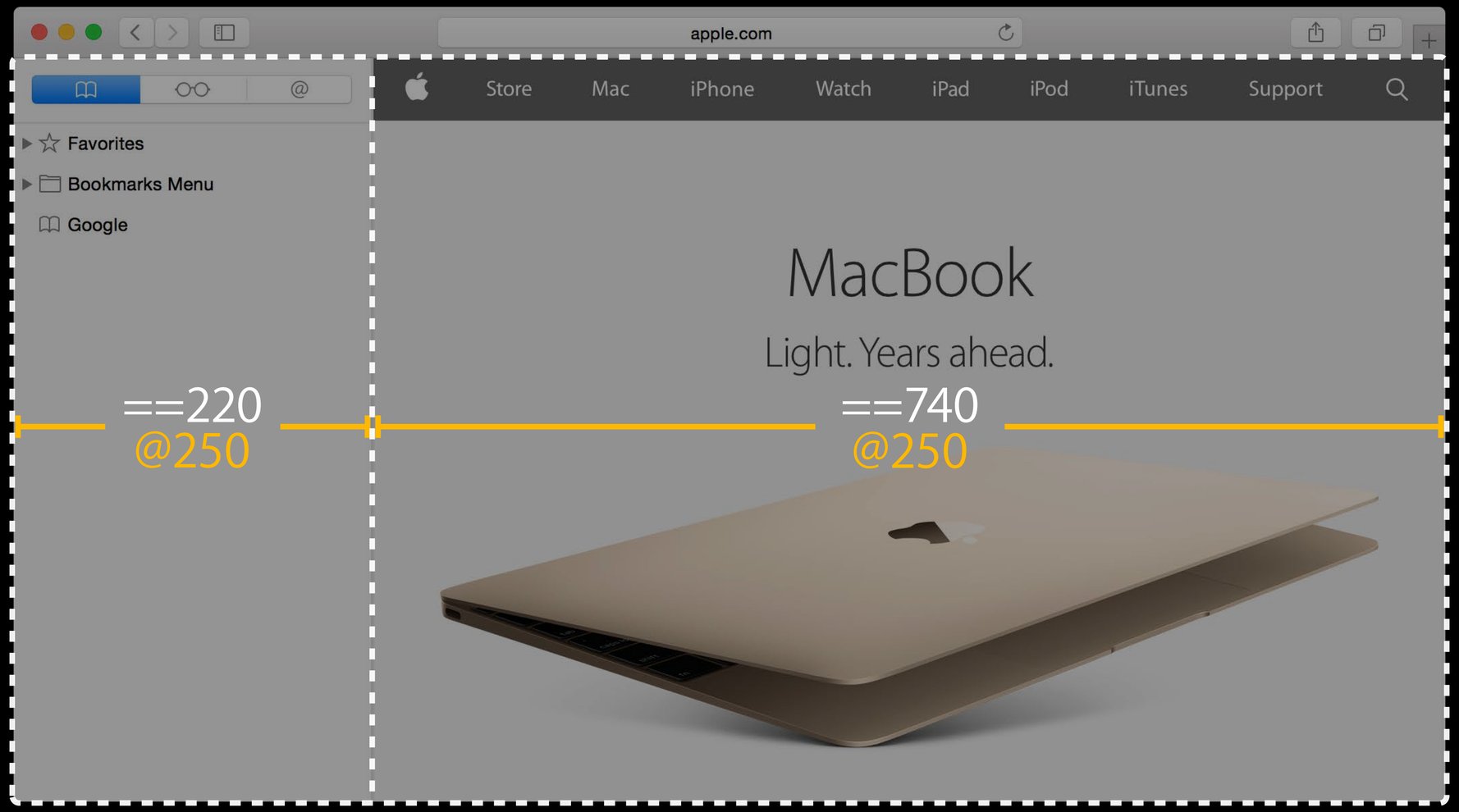
(constraint priority)



Metrics

```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

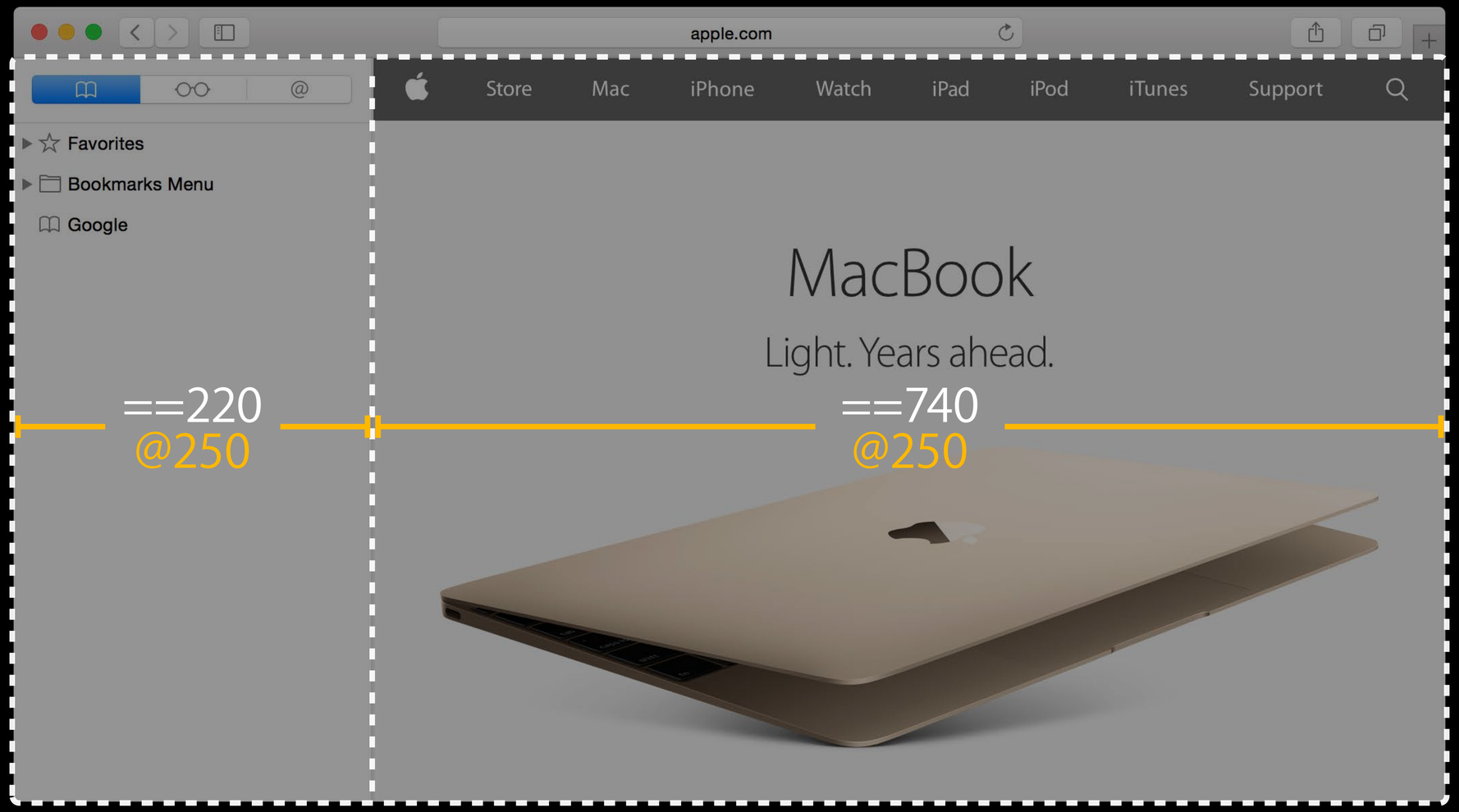
(constraint priority)



Metrics

```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

(constraint priority)



Metrics

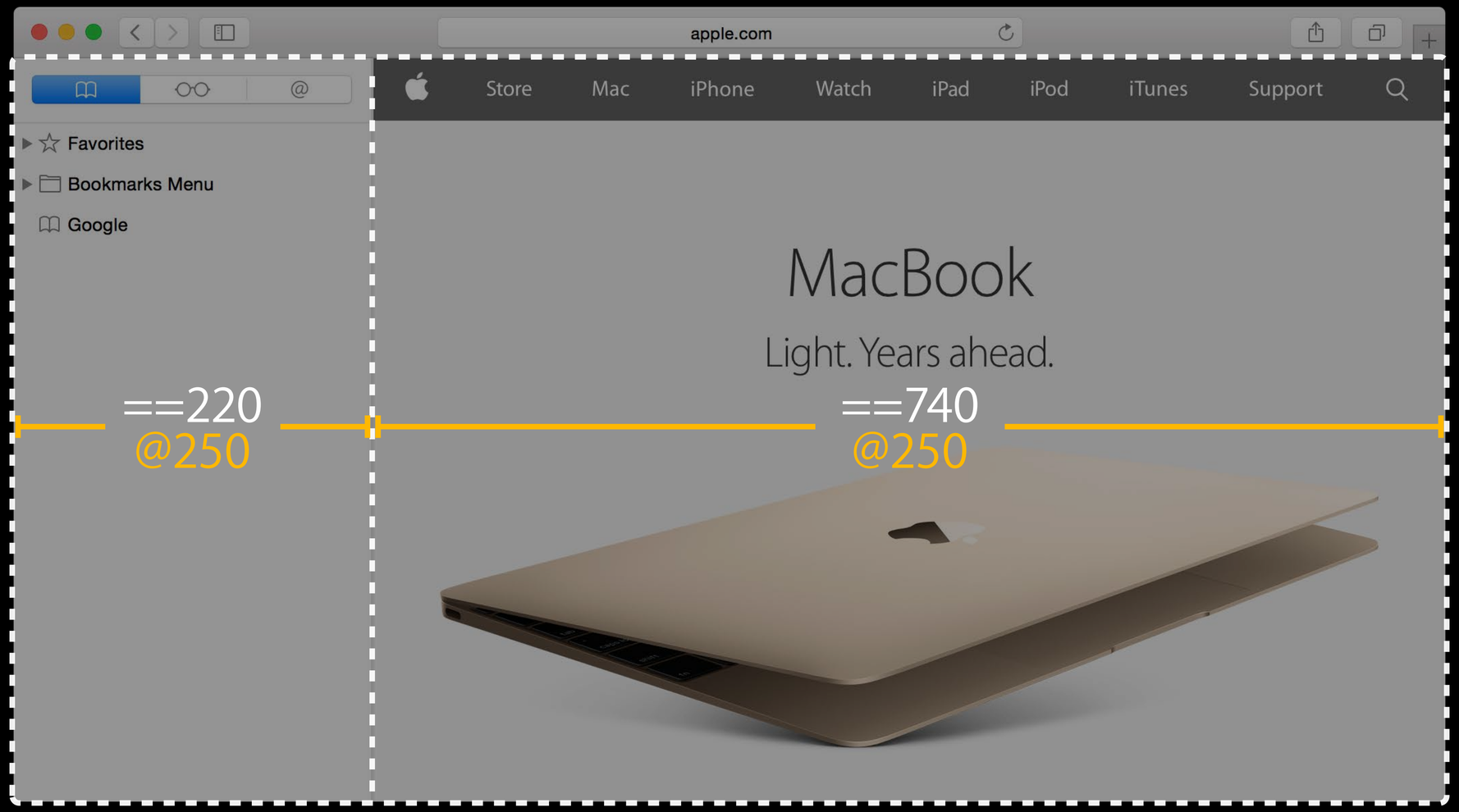
```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

(constraint priority)

Metrics

```
class NSSplitViewItem {
  var holdingPriority: NSLayoutPriority
```

(constraint priority)



Metrics

```
class NSSplitViewItem {
  var preferredThicknessFraction: CGFloat
  var automaticMaximumThickness: CGFloat
```

(percentage)

(points)

Metrics

```
class NSSplitViewItem {
  var preferredThicknessFraction: CGFloat
  var automaticMaximumThickness: CGFloat
```

(percentage)

(points)

Metrics

```
class NSSplitViewItem {
  var preferredThicknessFraction: CGFloat
  var automaticMaximumThickness: CGFloat
```

(percentage)

(points)

Metrics

```
class NSSplitViewItem {
  var preferredThicknessFraction: CGFloat
  var automaticMaximumThickness: CGFloat
```

(percentage)

(points)

Metrics

```
class NSSplitViewController {
  var minimumThicknessForInlineSidebars: CGFloat
```

(points)

Metrics

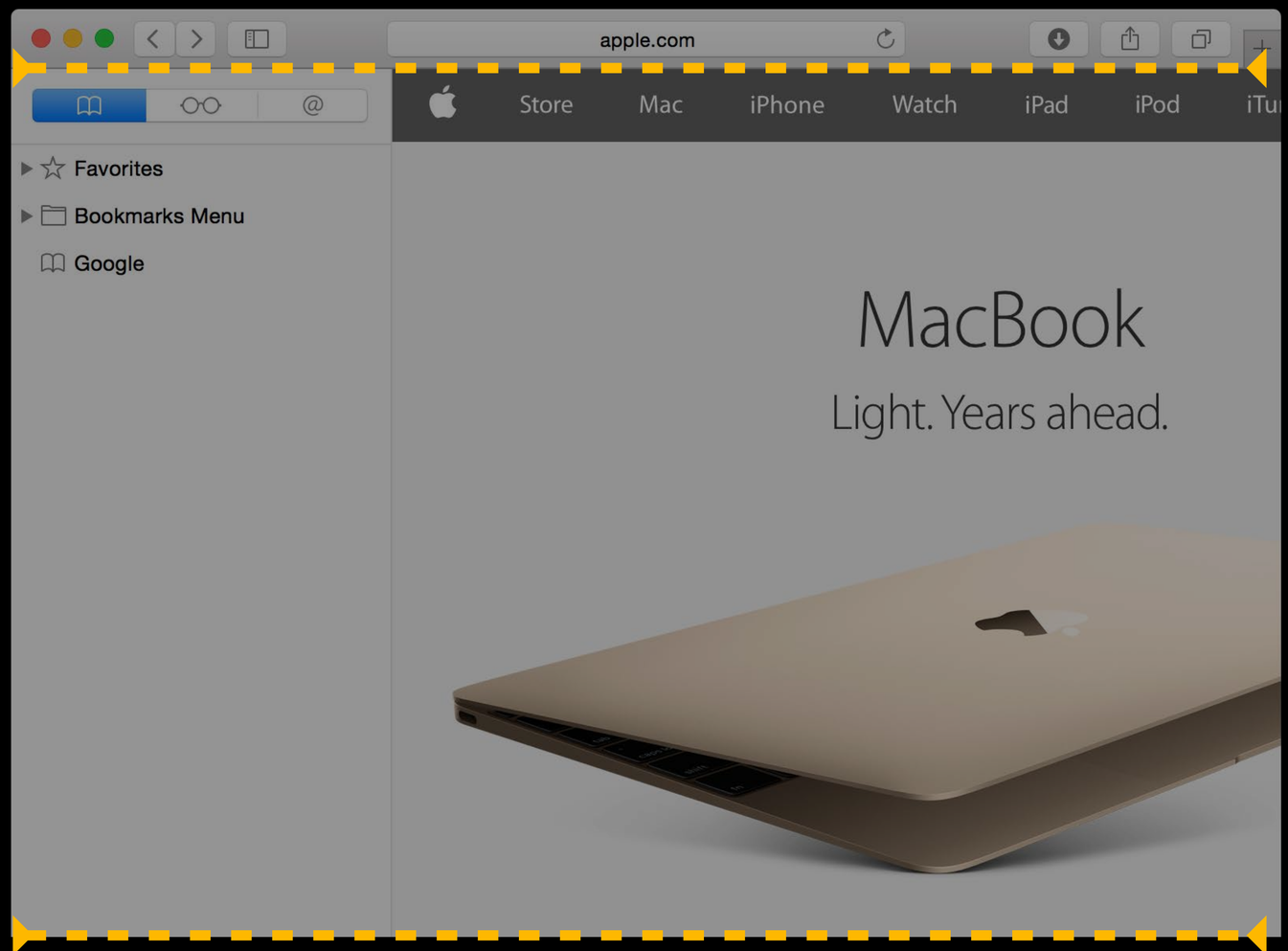
```
class NSSplitViewController {
  var minimumThicknessForInlineSidebars: CGFloat
```

(points)

Metrics

```
class NSSplitViewController {
  var minimumThicknessForInlineSidebars: CGFloat
```

(points)



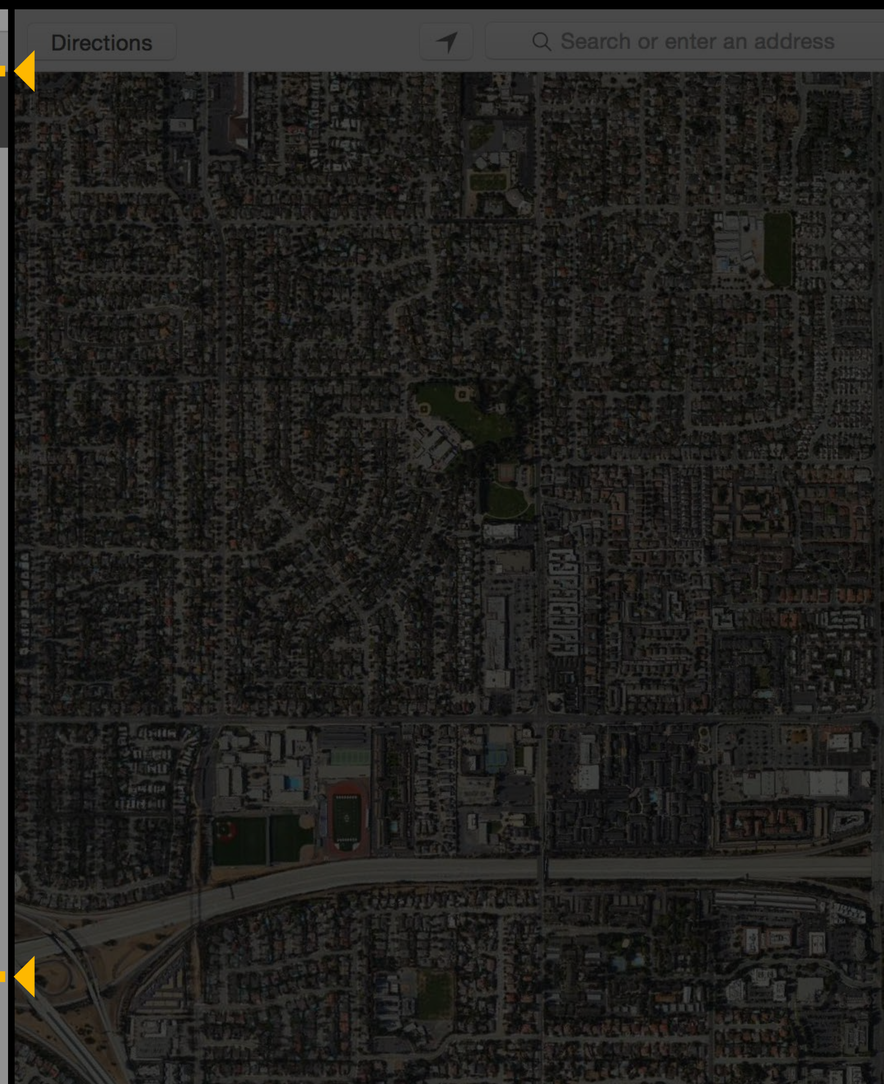
Metrics

```
class NSSplitViewController {
  var minimumThicknessForInlineSidebars: CGFloat
```

(points)

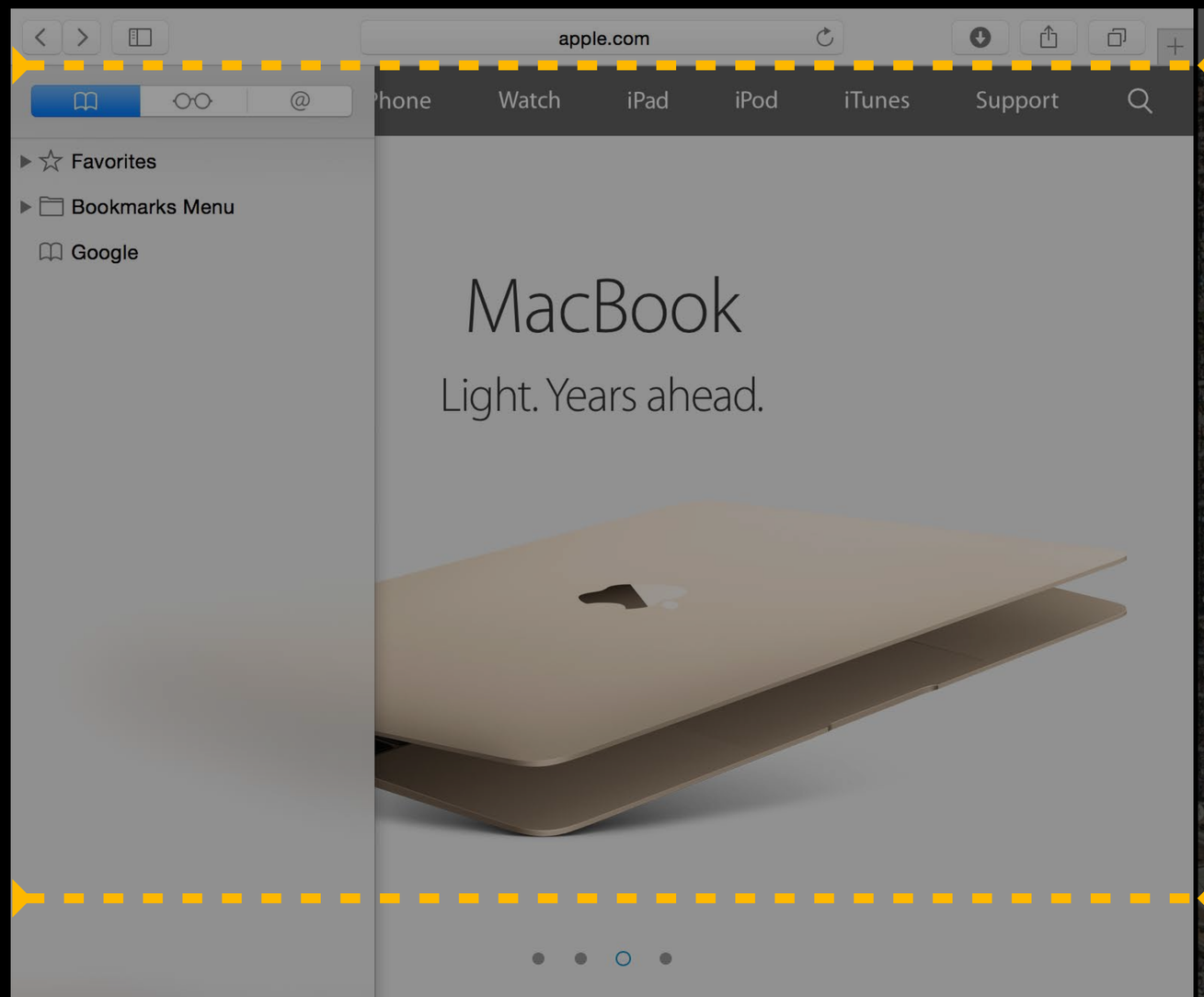
Metrics

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



Metrics

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



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()
```

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

```
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

NEW

NSSplitView

Arranged subviews

NEW

With 10.11, subviews are designated as arranged

NSSplitView

Arranged subviews

NEW

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

NEW

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
```

NSSplitView

Arranged subviews

NEW

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`

NSSplitView

Arranged subviews

NEW

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`

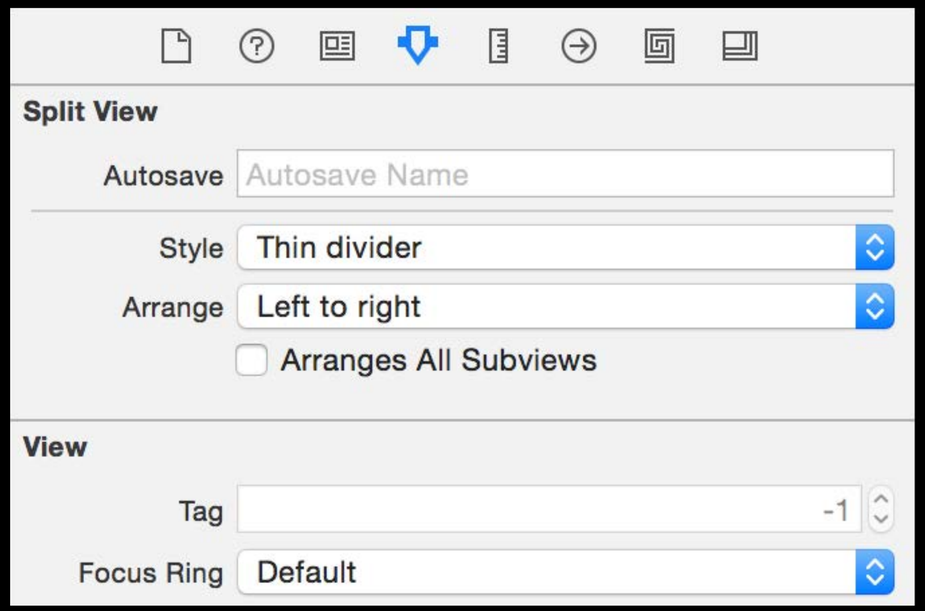
`arrangedSubviews` is always a subset of `subviews`

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

NSSplitView

Arranged subviews

NEW



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

`subviews == arrangedSubviews`

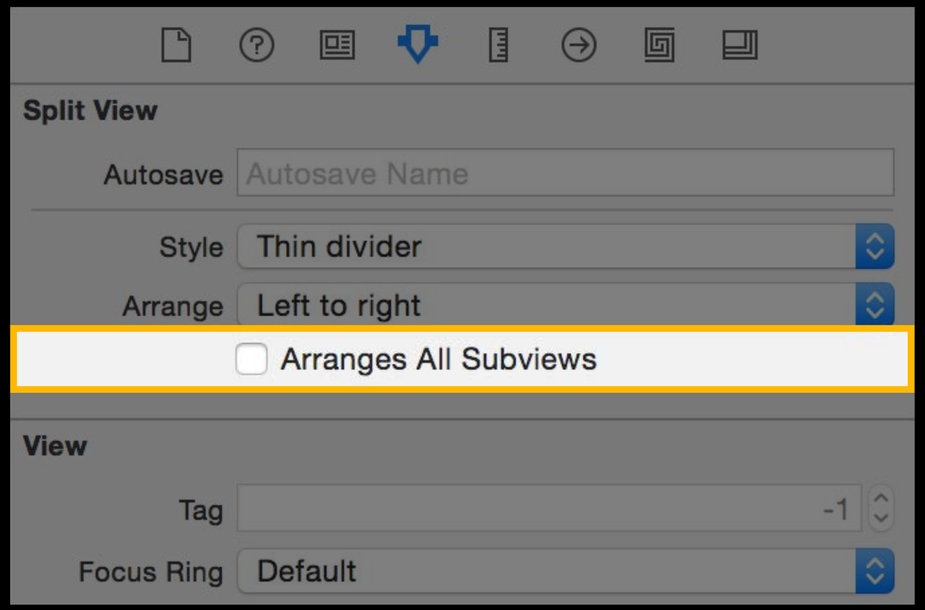
`arrangedSubviews` is always a subset of `subviews`

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

NSSplitView

Arranged subviews

NEW



`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)0xeeb00"
 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

<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>
```

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

Mysteries of Auto Layout, Part 1	Presidio	Thursday 11:00AM

NSStackView

Distributions

NEW

NSStackView

Distributions

NEW

```
enum NSStackViewDistribution : Int {
  case GravityAreas
  case Fill
  case FillEqually
  case FillProportionally
  case EqualSpacing
  case EqualCentering
}
```

NSStackView

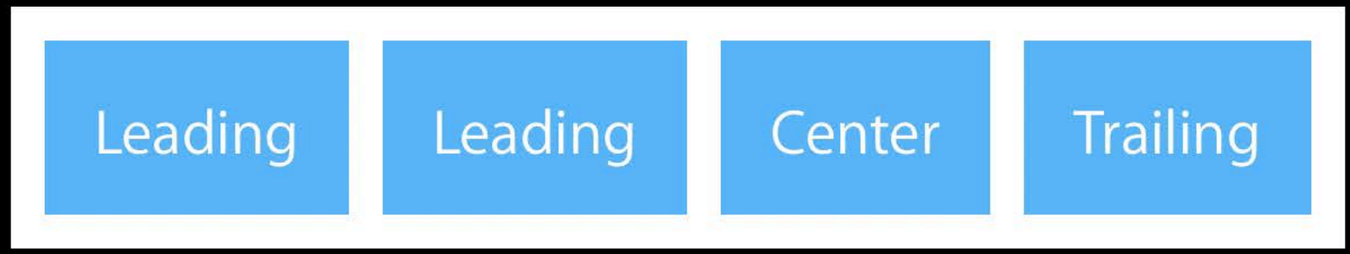
Distributions

NEW

```
enum NSStackViewDistribution : Int {
  case GravityAreas
  case Fill
  case FillEqually
  case FillProportionally
  case EqualSpacing
  case EqualCentering
}


class NSStackView {
  var distribution: NSStackViewDistribution
```

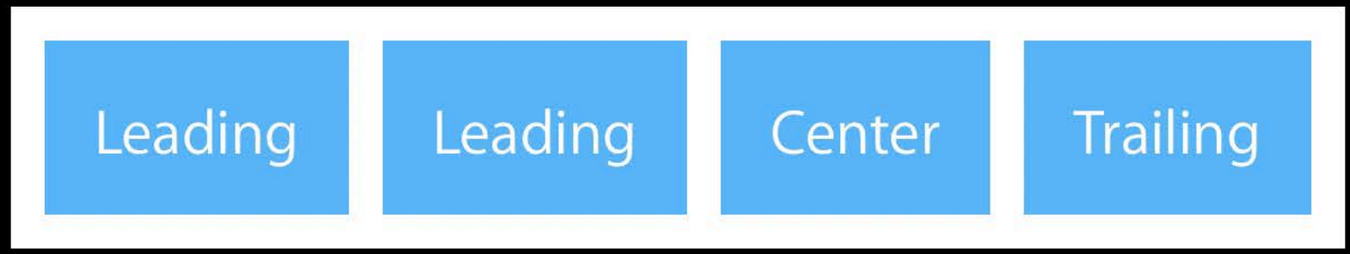
NSStackView

.GravityAreas



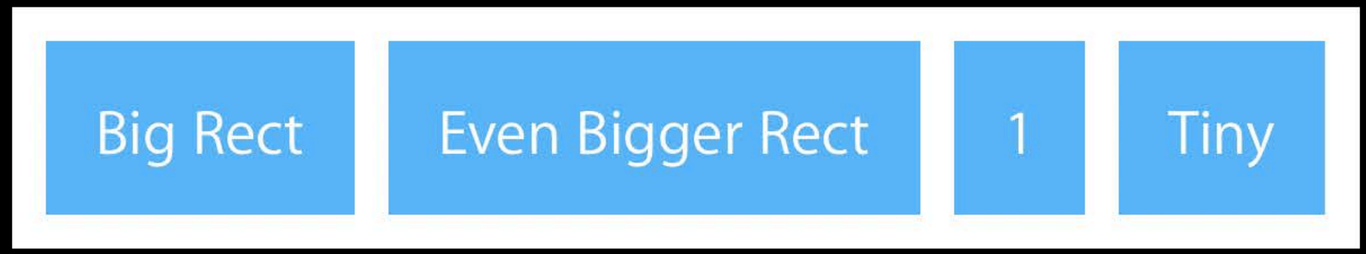
NSStackView

.GravityAreas



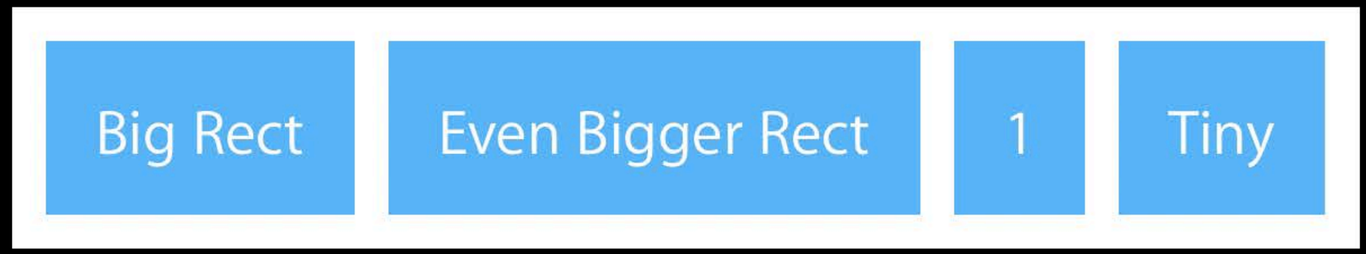
NSStackView

.Fill



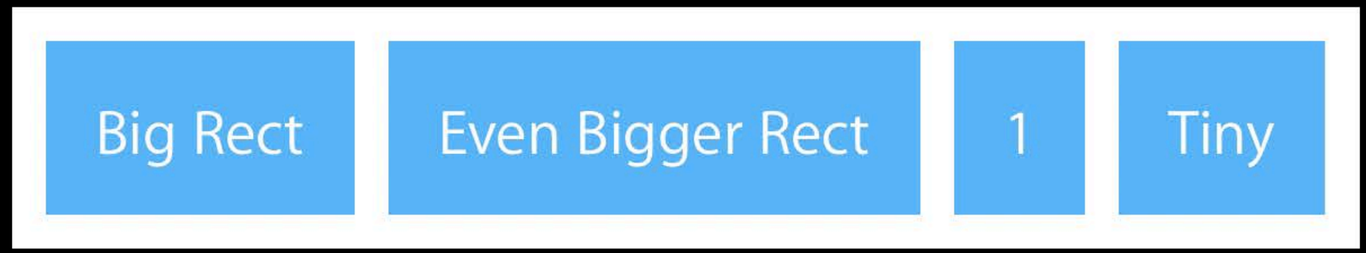
NSStackView

.Fill



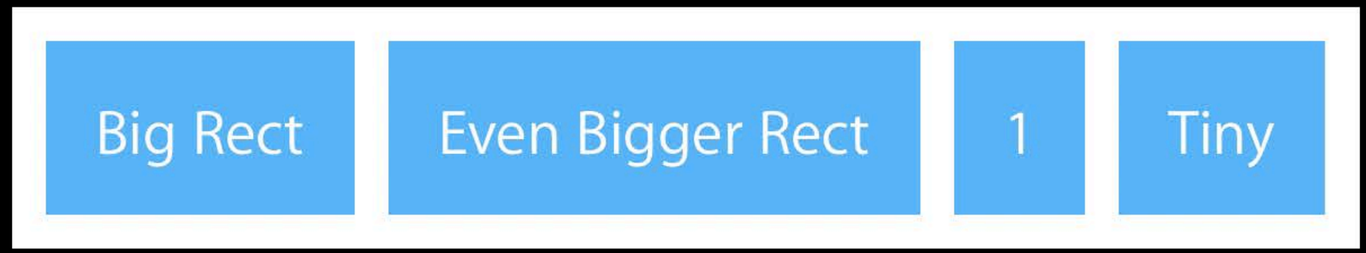
NSStackView

.FillEqually



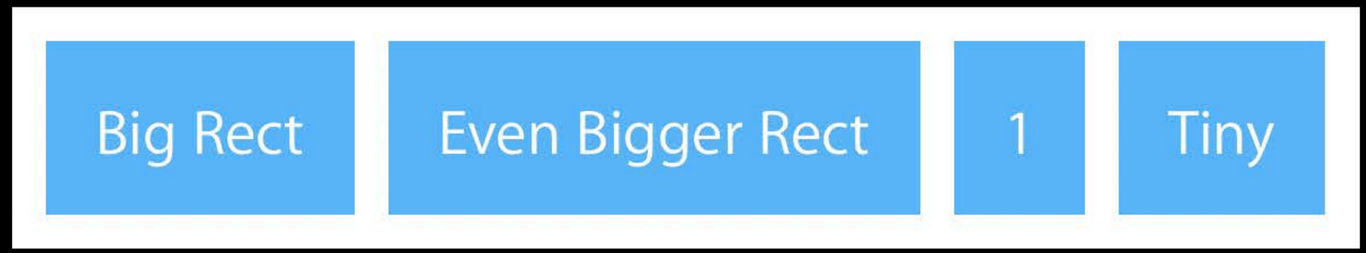
NSStackView

.FillEqually



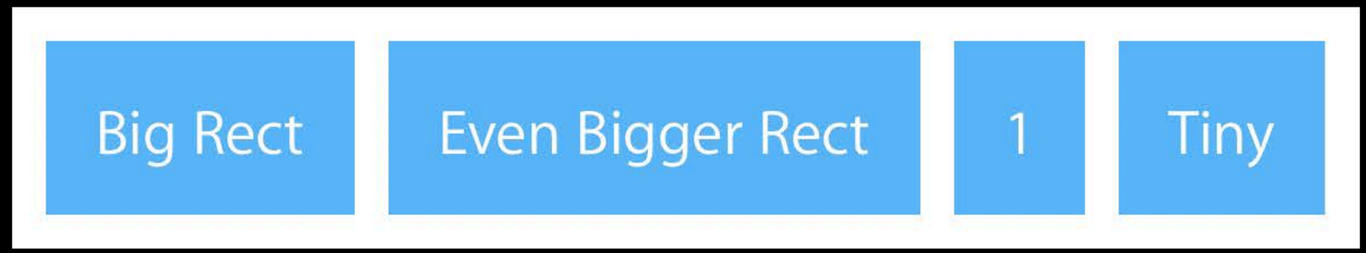
NSStackView

.FillProportionally



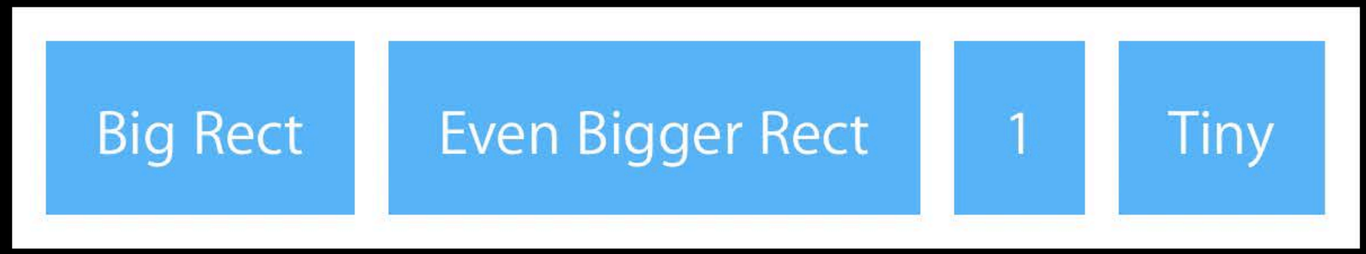
NSStackView

.FillProportionally



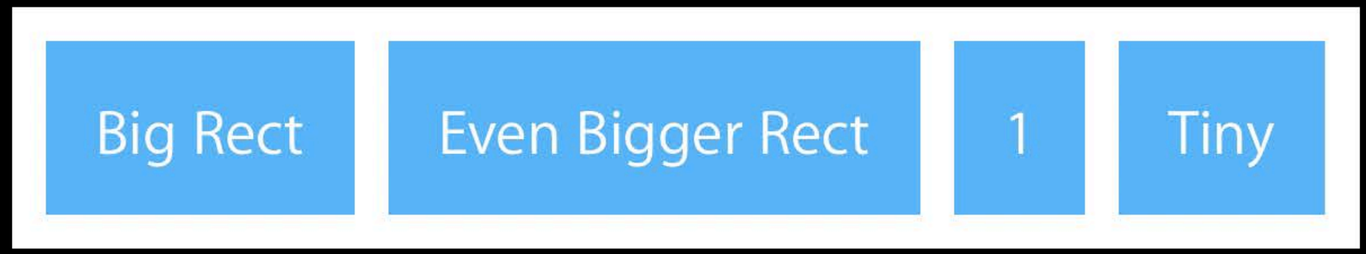
NSStackView

.EqualSpacing



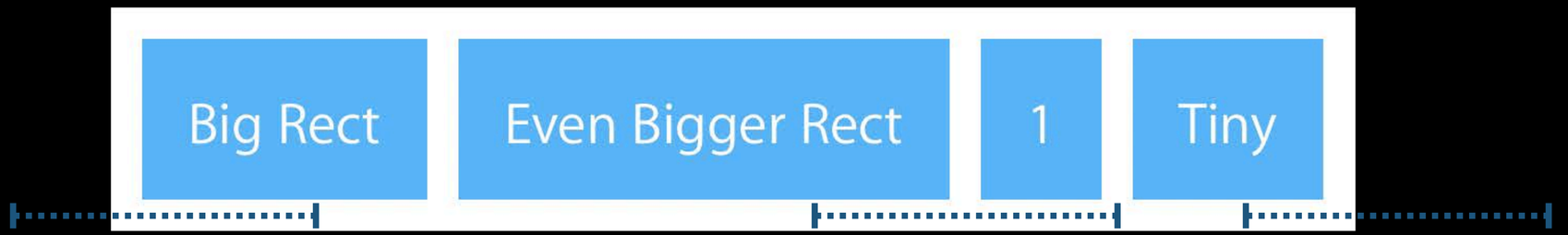
NSStackView

.EqualSpacing



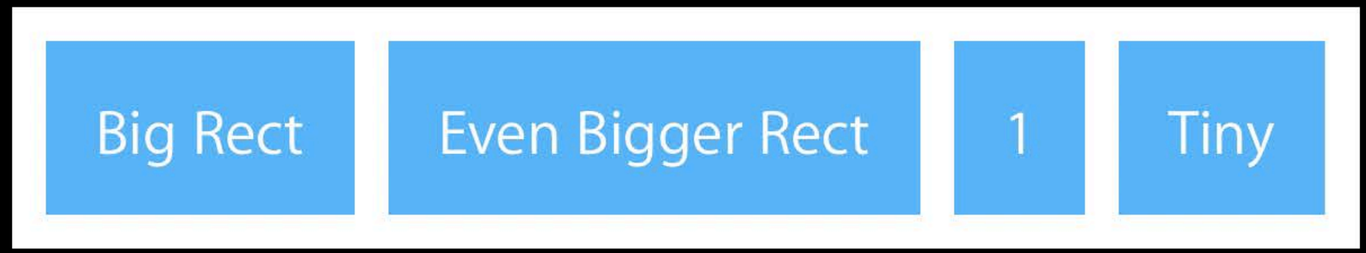
NSStackView

.EqualSpacing



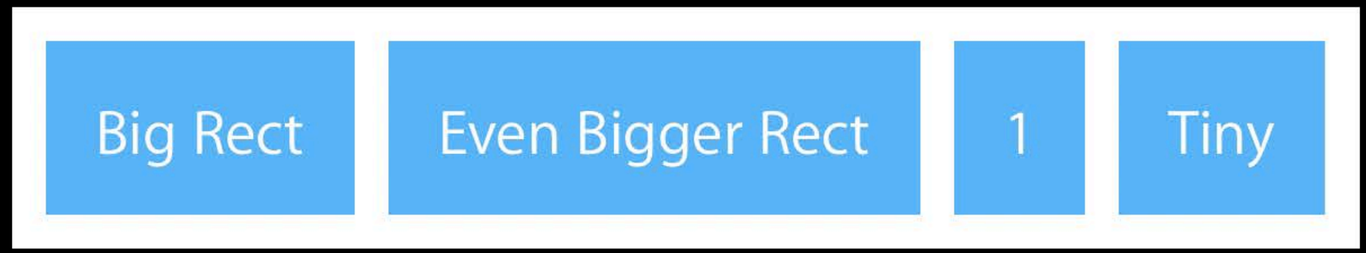
NSStackView

.EqualCentering



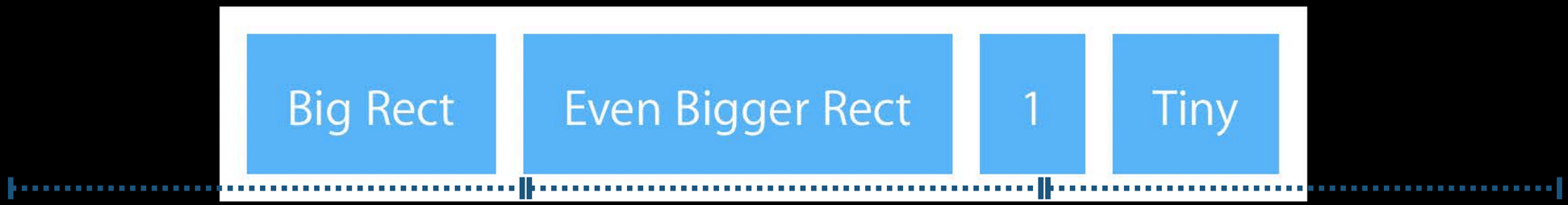
NSStackView

.EqualCentering



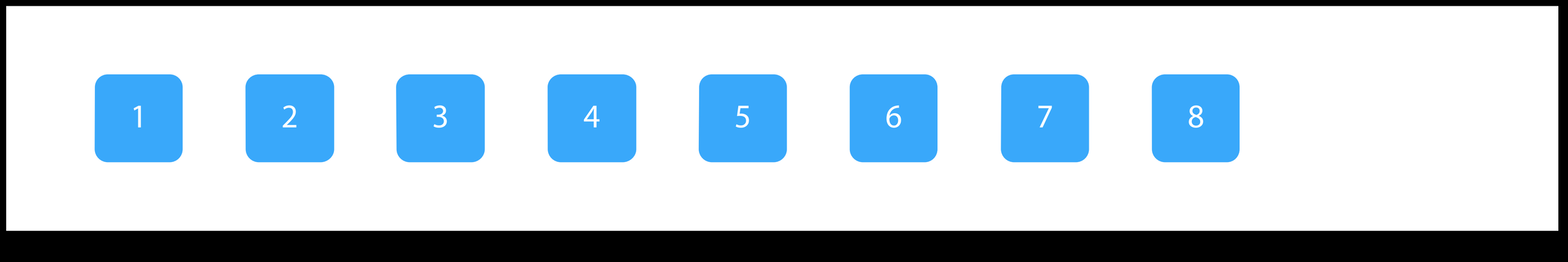
NSStackView

.EqualCentering



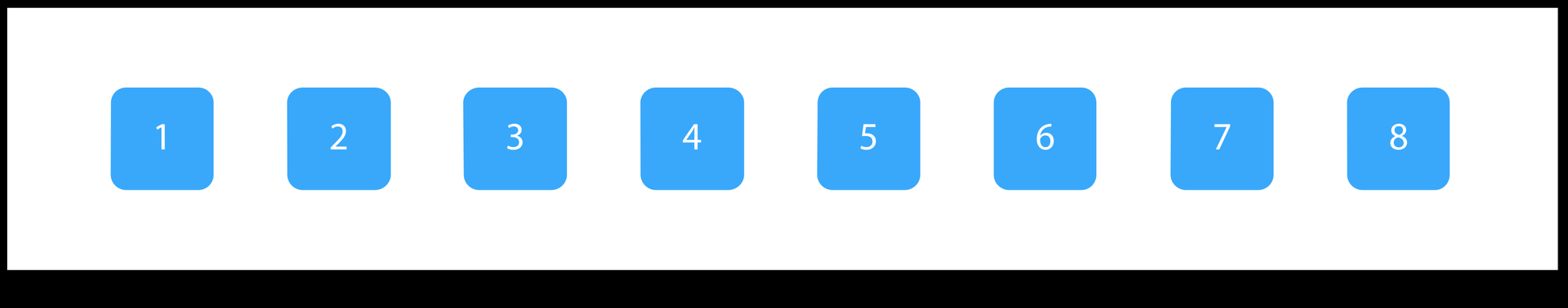
NSStackView

Visibility Priorities



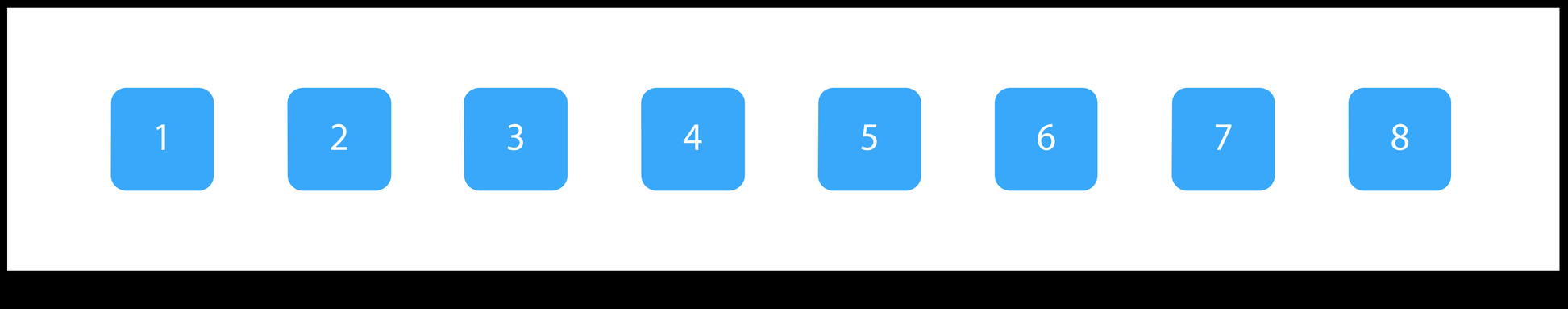
NSStackView

Visibility Priorities



NSStackView

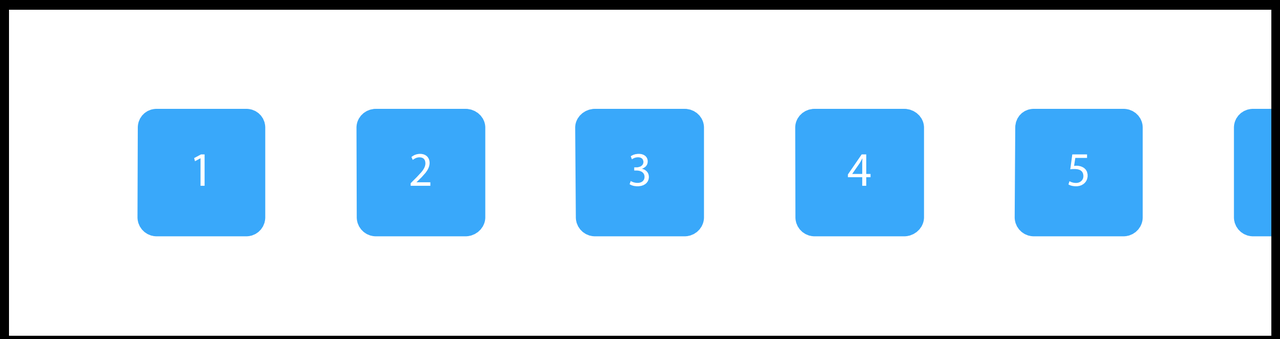
Visibility Priorities




```
stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
```

NSStackView

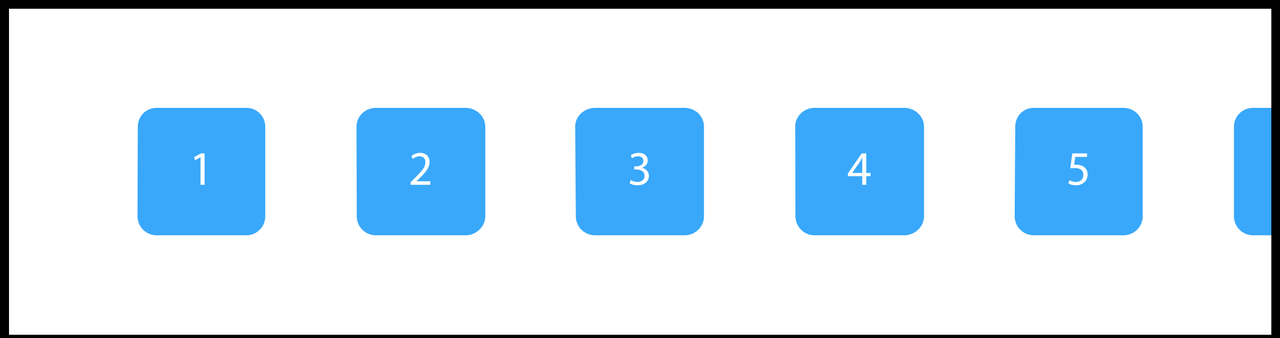
Visibility Priorities




```
stackView.setClippingResistancePriority(250, forOrientation: .Horizontal)
```

NSStackView

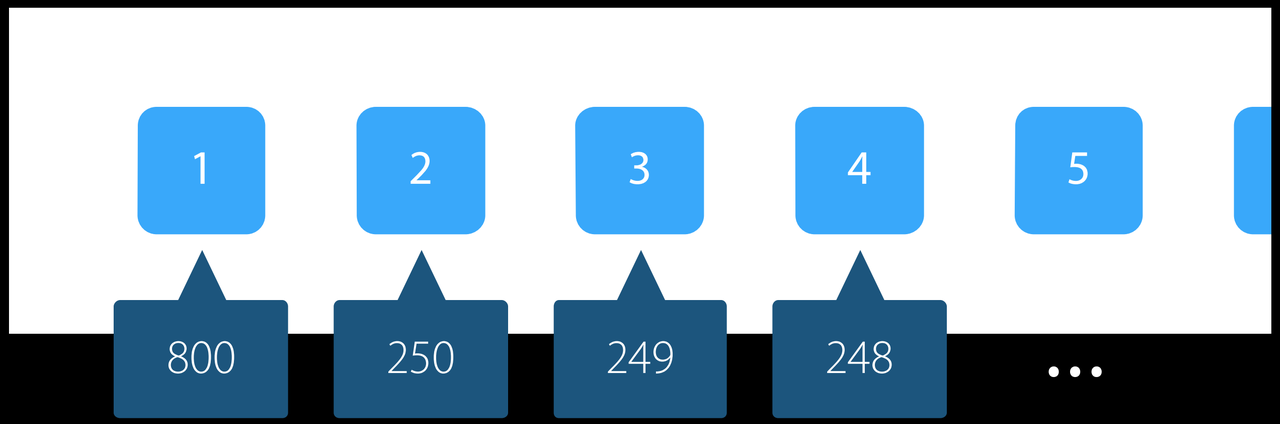
Visibility Priorities




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

NSStackView

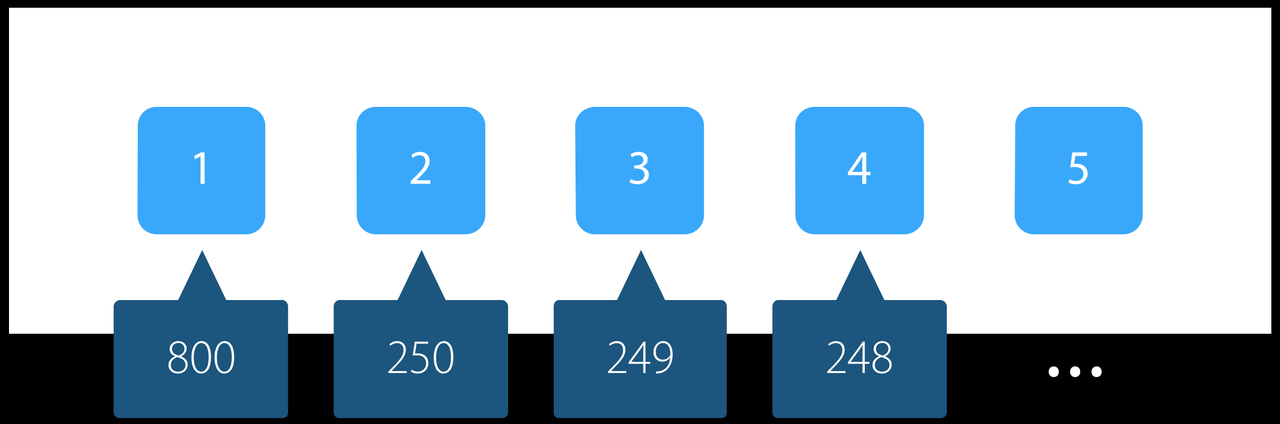
Visibility Priorities




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

NSStackView

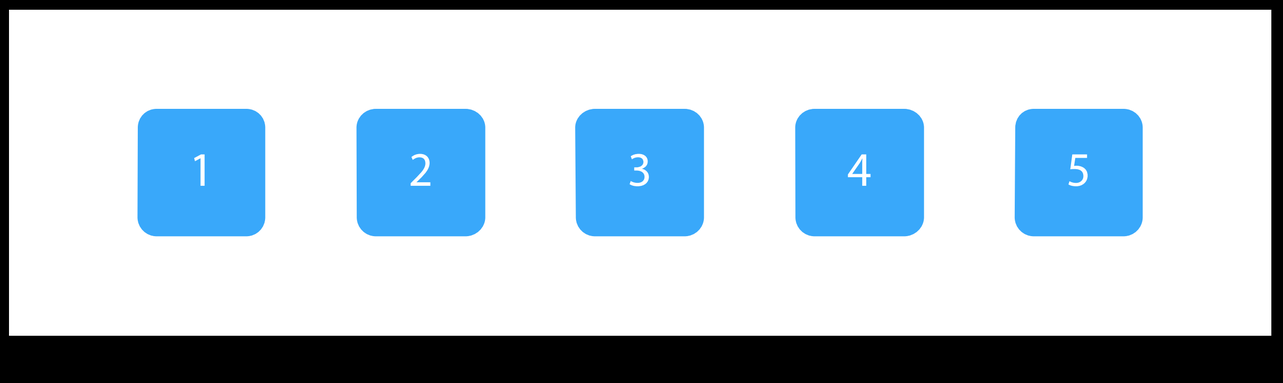
Visibility Priorities




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

NSStackView

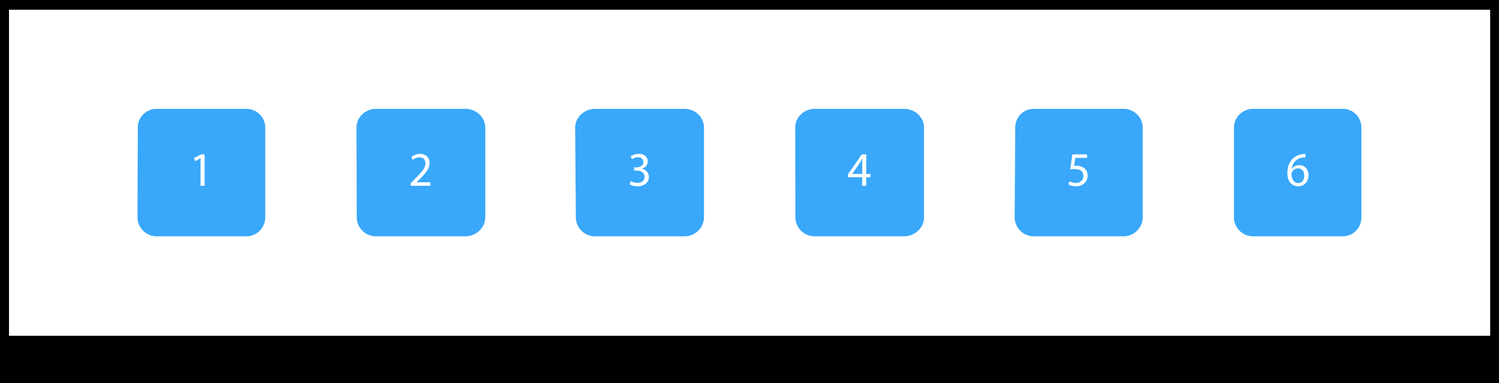
Visibility Priorities




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

NSStackView

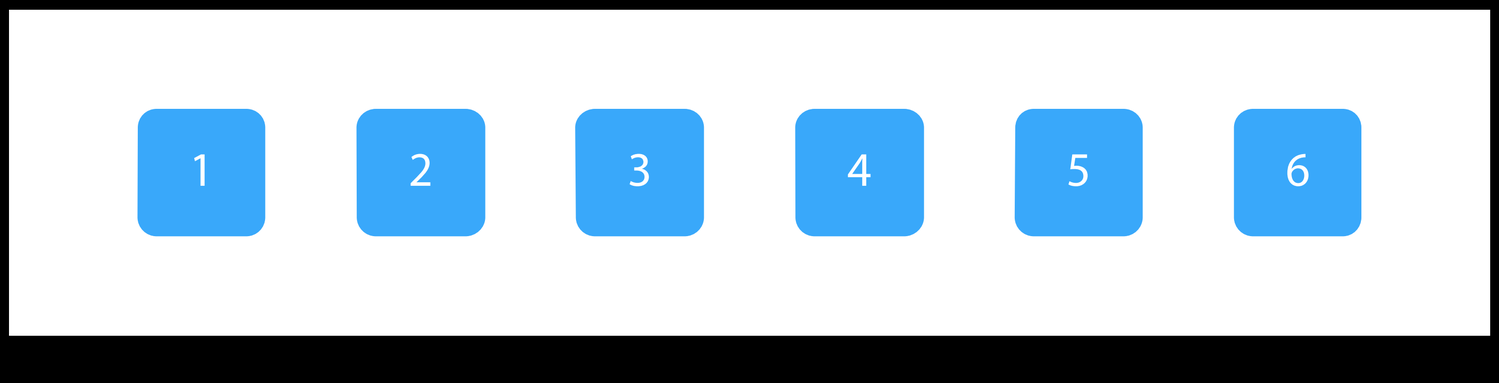
Visibility Priorities




```
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

What's New in NSCollectionView	Mission	Thursday 4:30PM

Sample Code

Exhibition — An Adaptive OS X App

https://developer.apple.com/library/prerelease/mac/samplecode/Exhibition

Exhibition

Exhibition

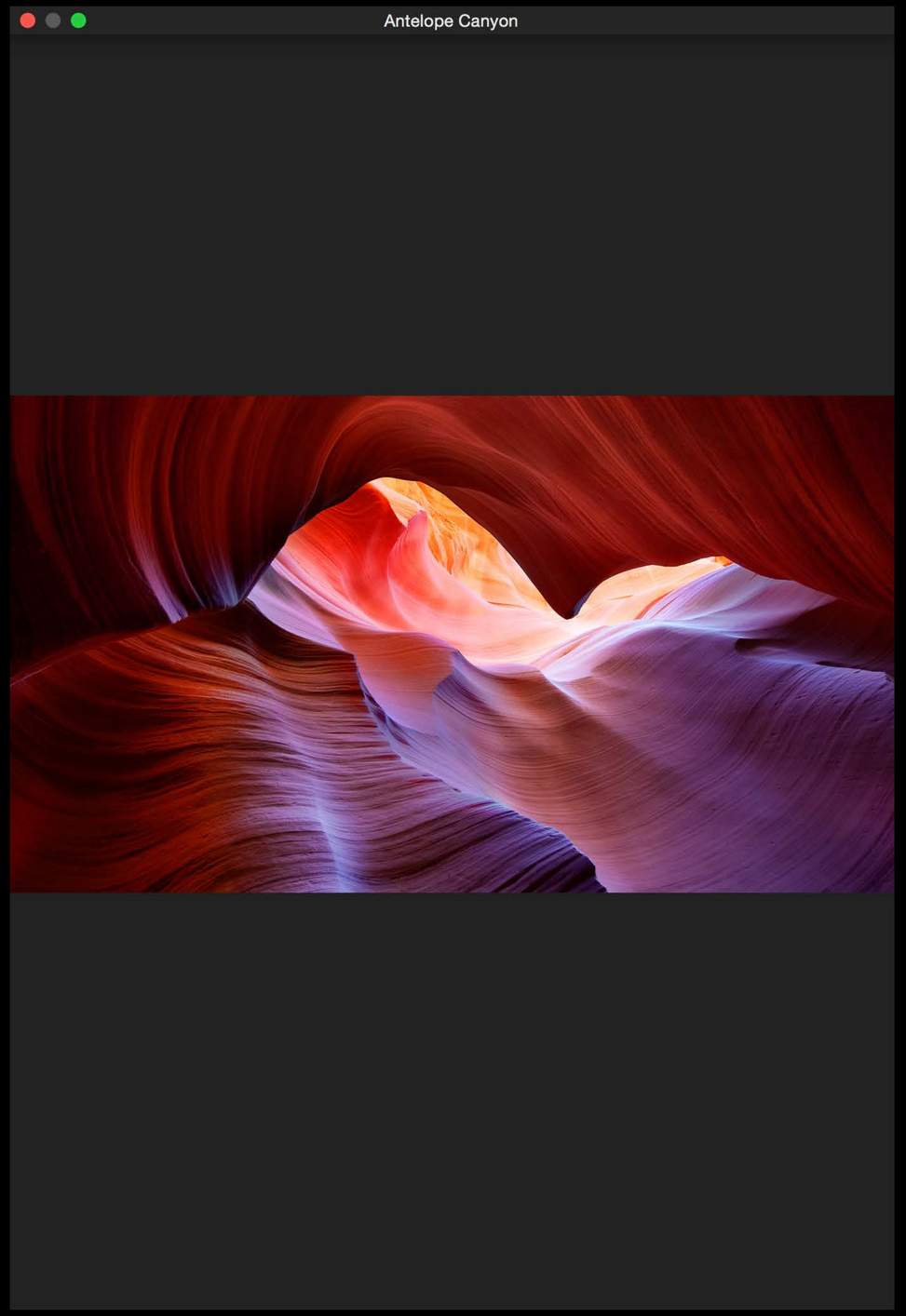
Full Screen API

Exhibition

Full Screen API

Exhibition

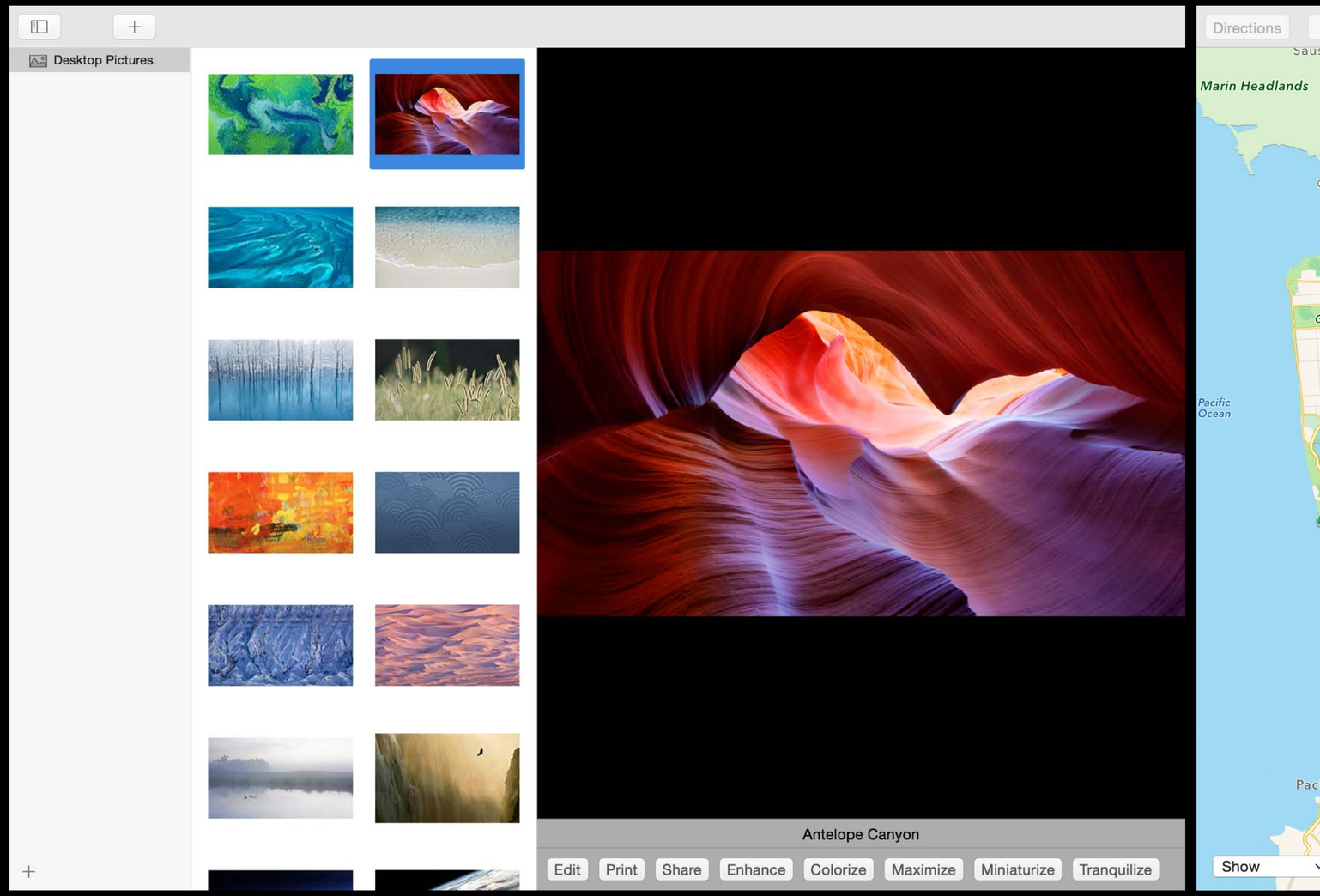
Full Screen API



Exhibition

Full Screen API

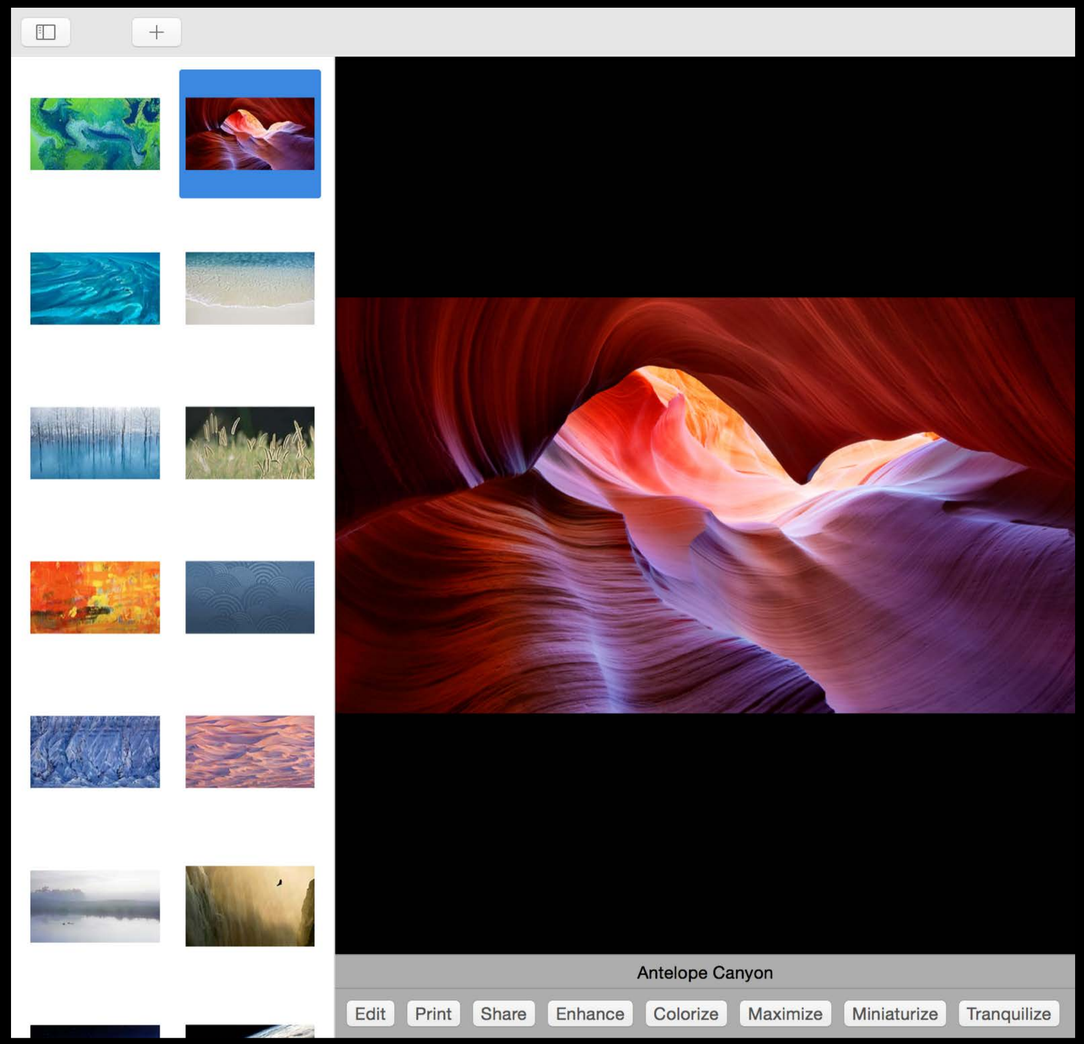
NSSplitViewController

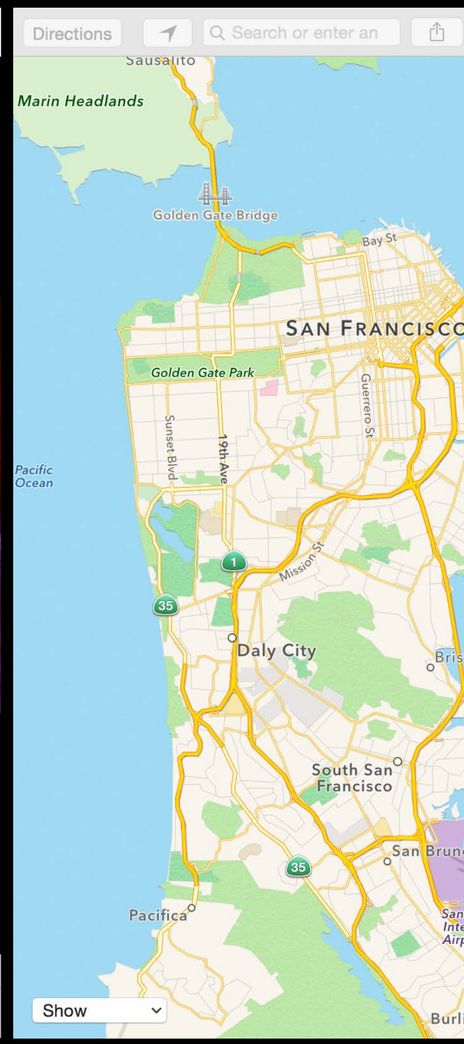


Exhibition

Full Screen API

NSSplitViewController

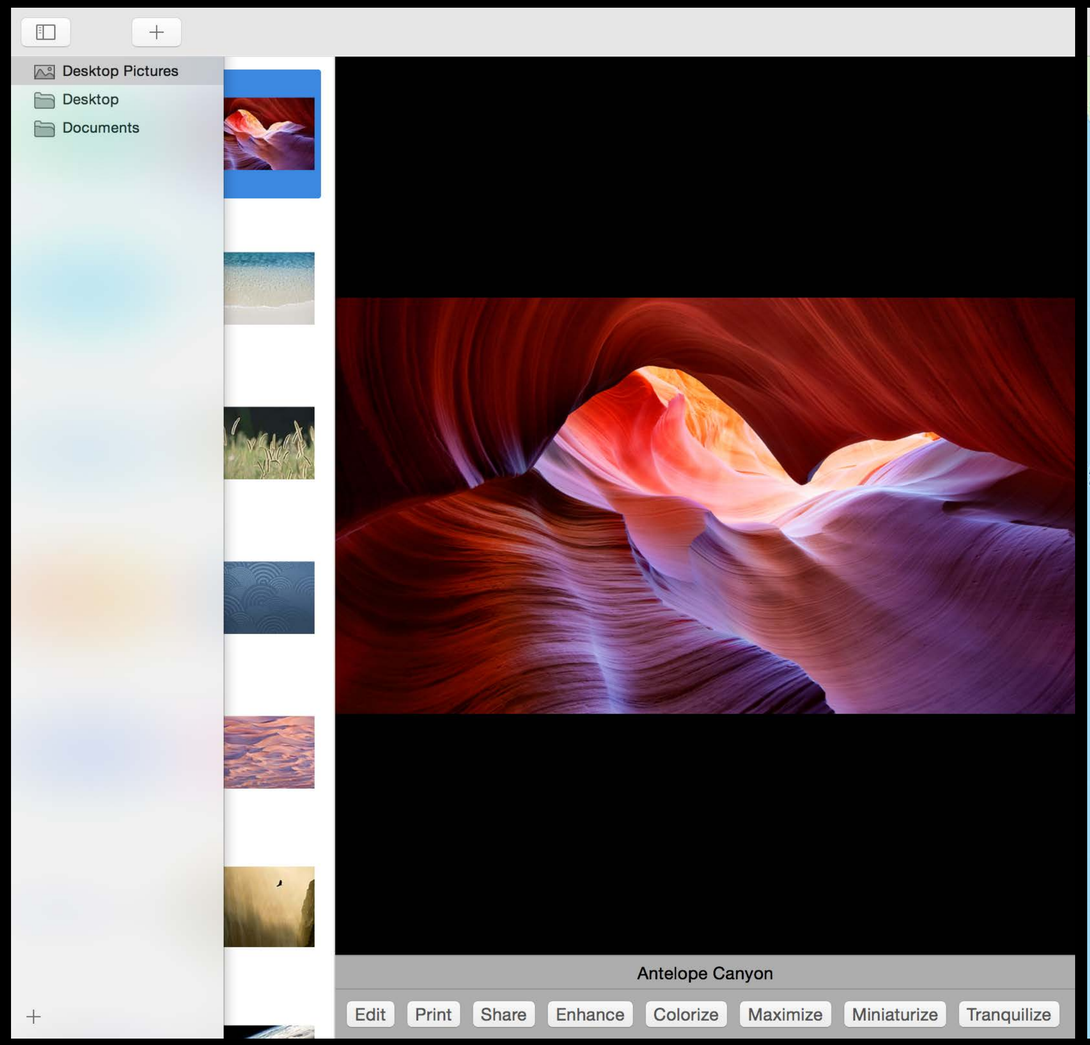




Exhibition

Full Screen API

NSSplitViewController

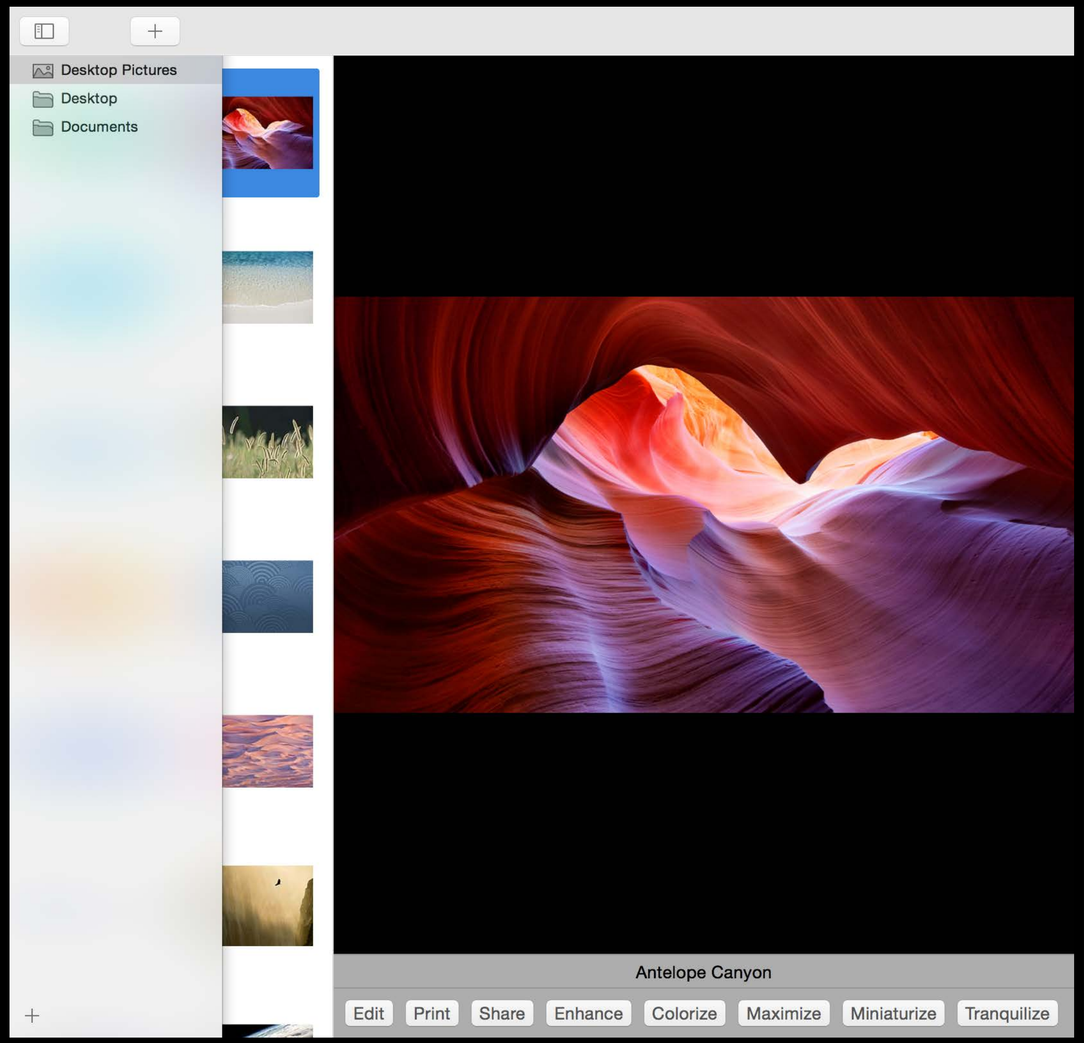


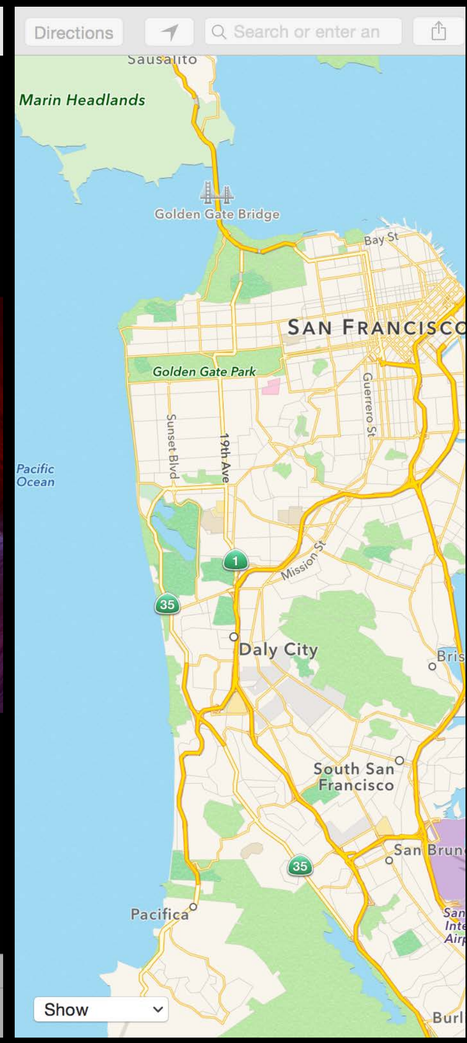
Exhibition

Full Screen API

NSSplitViewController

NSStackView



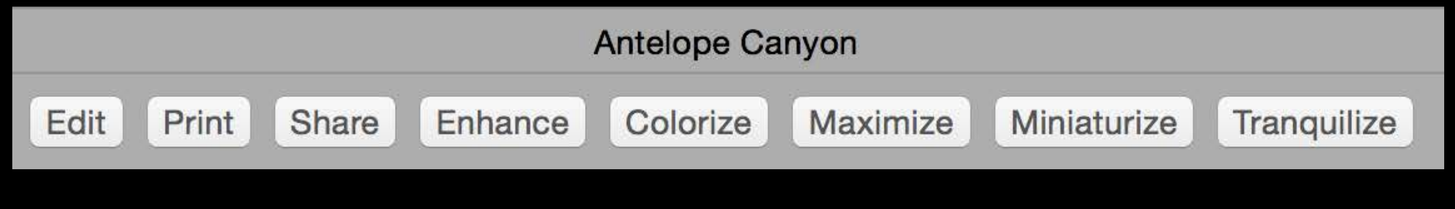


Exhibition

Full Screen API

NSSplitViewController

NSStackView

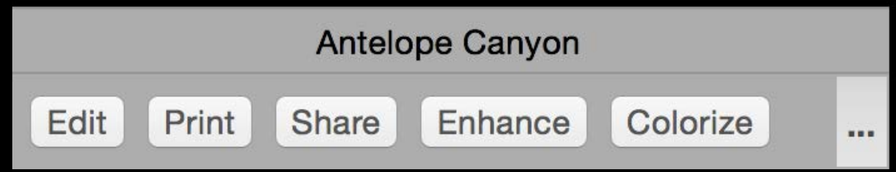


Exhibition

Full Screen API

NSSplitViewController

NSStackView

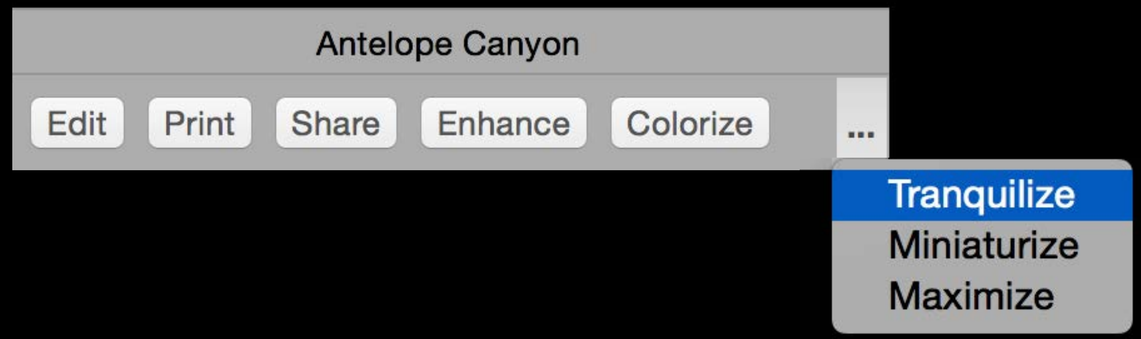


Exhibition

Full Screen API

NSSplitViewController

NSStackView

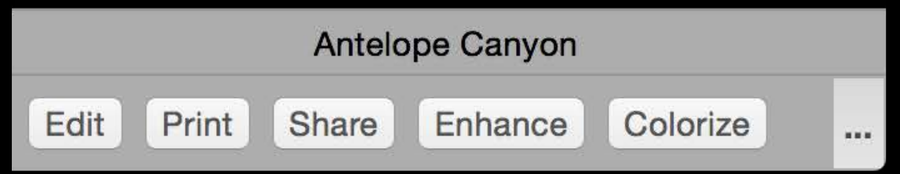


Exhibition

Full Screen API

NSSplitViewController

NSStackView



NSCollectionView

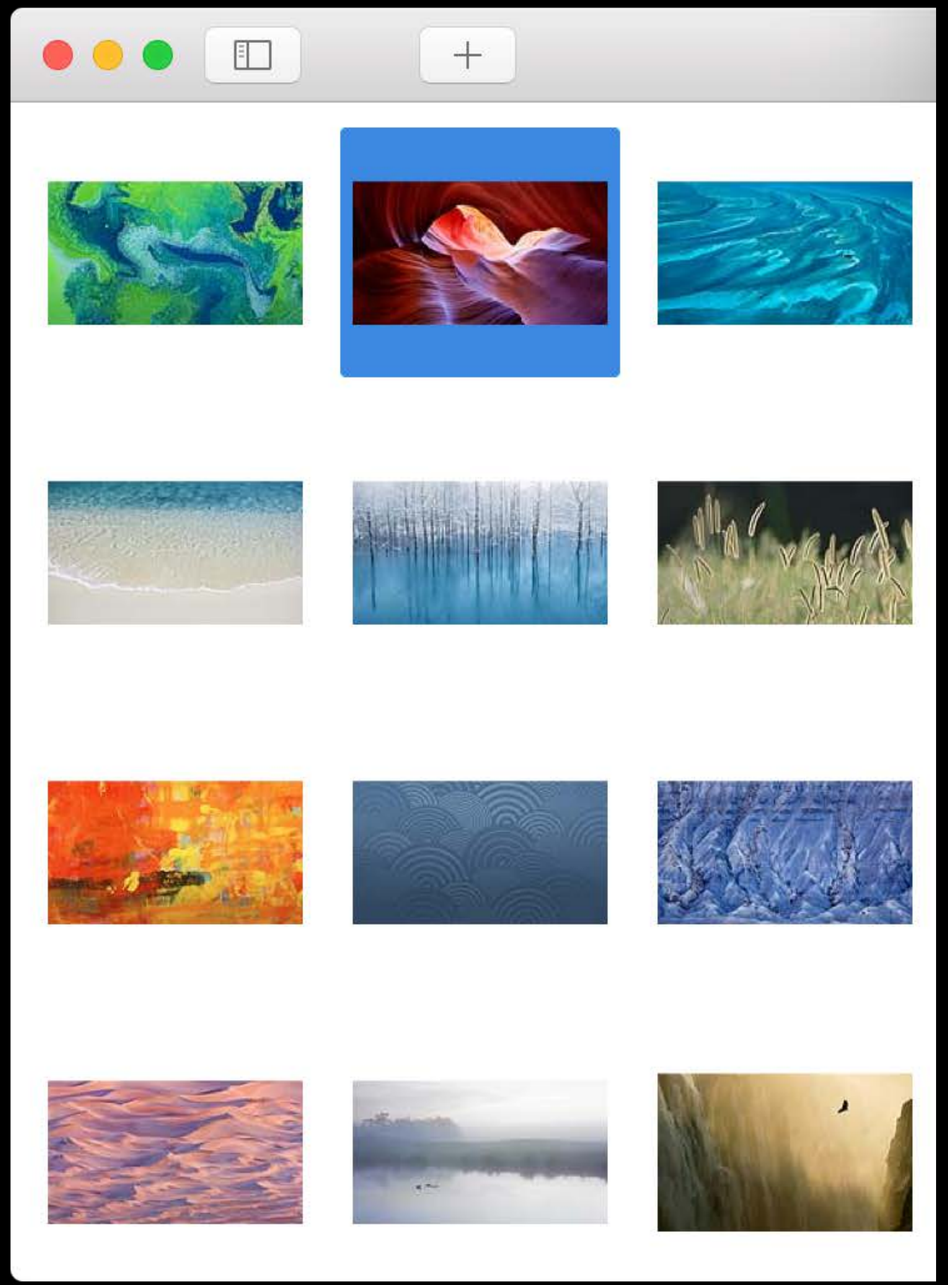
Exhibition

Full Screen API

NSSplitViewController

NSStackView

NSCollectionView



Summary

Summary

Full Screen

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
http://developer.apple.com/library/mac

Technical Support

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

General Inquiries

Paul Marcos, App Frameworks Evangelist
pmarcos@apple.com

Related Sessions

Mysteries of Auto Layout, Part 1	Presidio	Thursday 11:00AM
Mysteries of Auto Layout, Part 2	Presidio	Thursday 1:30PM
What's New in NSCollectionView	Mission	Thursday 4:30PM

Labs

Interface Builder and Auto Layout Lab	Developer Tools Lab C	Thursday 2:30PM
Cocoa and Full Screen Support Lab	Frameworks Lab D	Thursday 3:30PM
Cocoa and NSCollectionView Lab	Frameworks Lab B	Friday 9:00AM

WWDC15