App Frameworks #WWDC16

Crafting Modern Cocoa Apps

Session 239

Corbin Dunn AppKit Software Engineer
Jeff Nadeau AppKit Software Engineer

What to Expect

Creating a Modern Look with Modern Views

Modern Drag & Drop, and Event Tracking

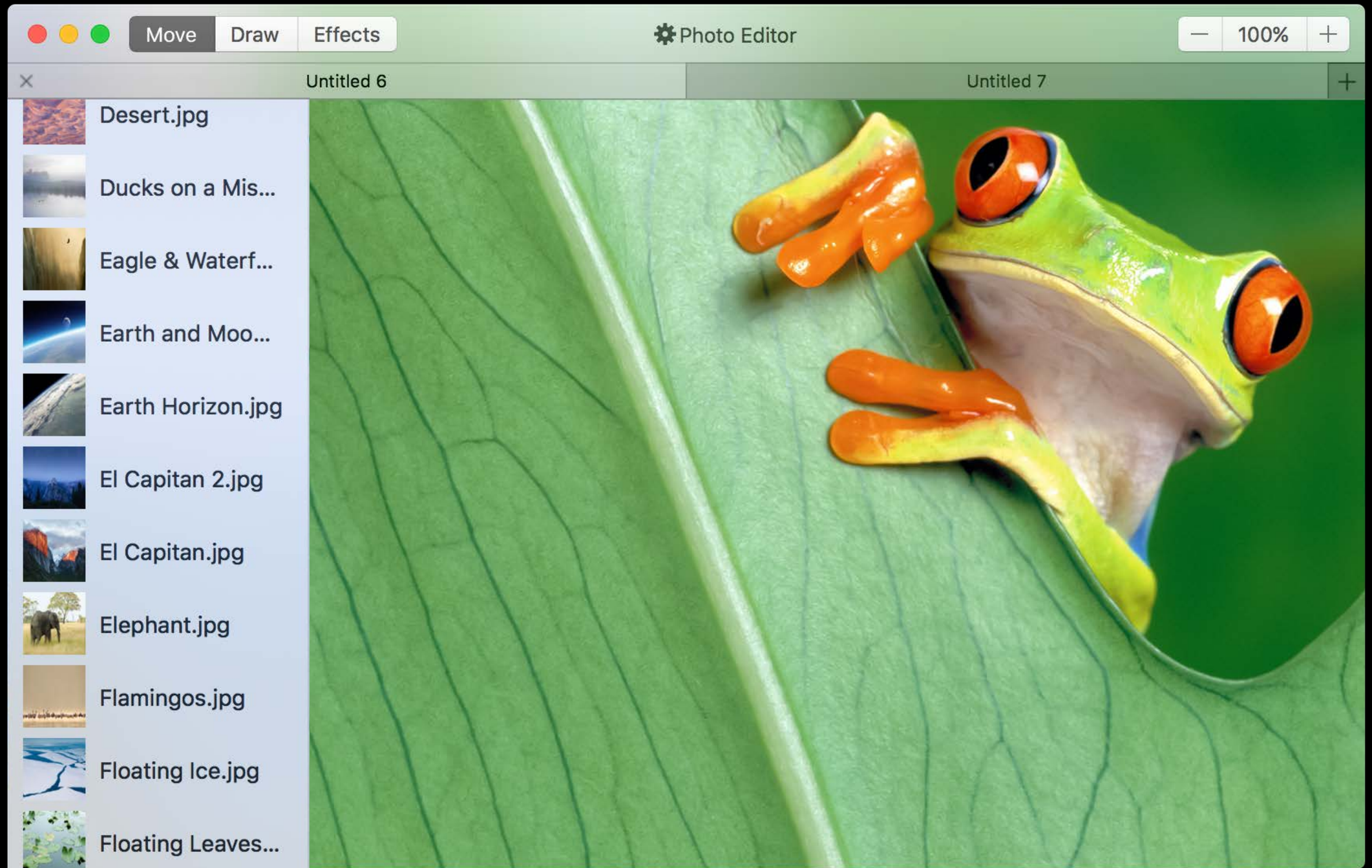
Container View Controls Handled Correctly

Using System Appearances

Designing with Storyboards

Modern Mac Features

Demo App


Move
Draw
Effects
Photo Editor
100%
Untitled 6
Untitled 7
Desert.jpg
Ducks on a Mis...
Eagle & Waterf...
Earth and Moo...
Earth Horizon.jpg
El Capitan 2.jpg
El Capitan.jpg
Elephant.jpg
Flamingos.jpg
Floating Ice.jpg
Floating Leaves...

What to Expect

Most all technologies and API available on 10.10 or 10.11

Things specific to 10.12 will be highlighted

Download the demo app—available soon

What to Expect

Creating a Modern Look with Modern Views

Modern Drag & Drop, and Event Tracking

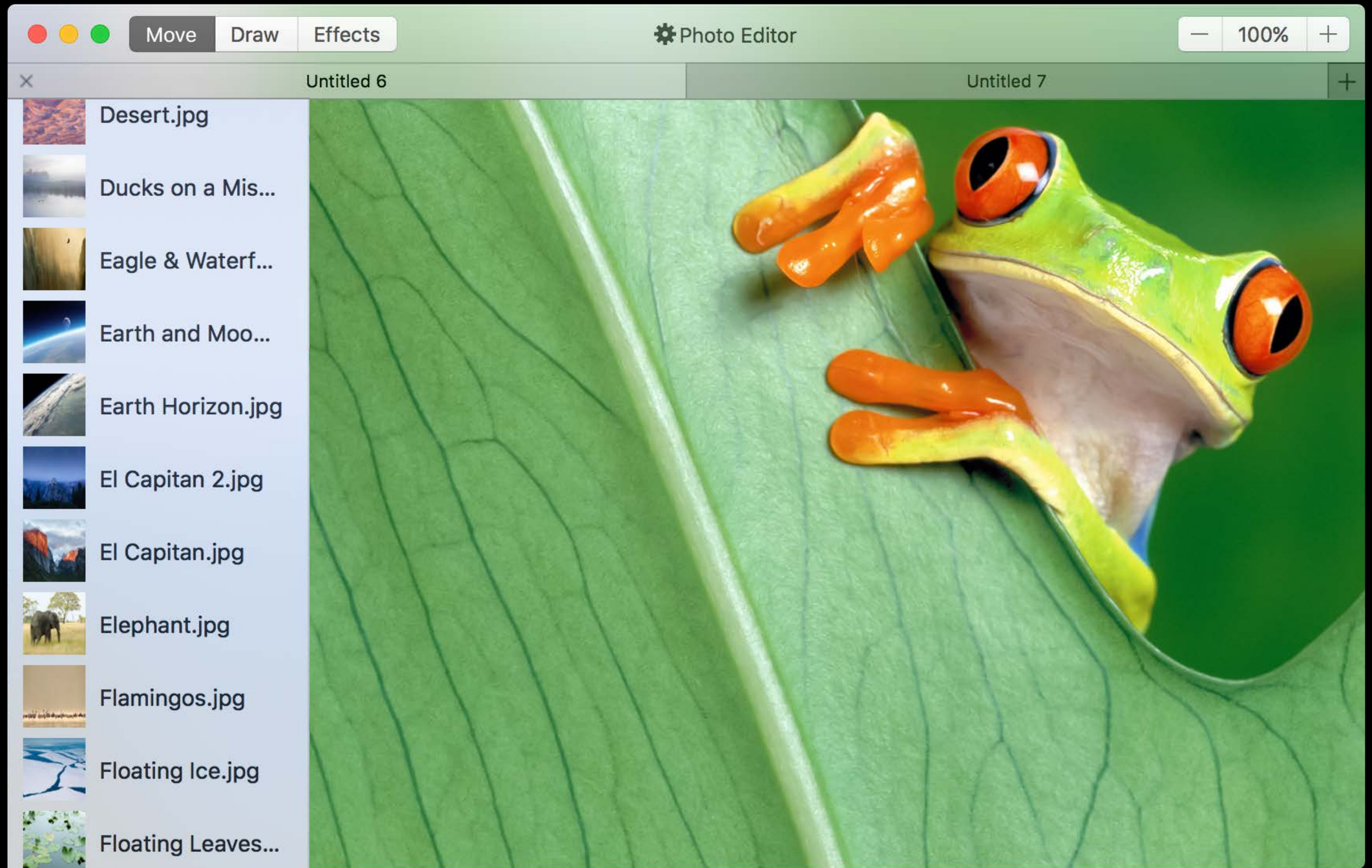
Container View Controls Handled Correctly

Using System Appearances

Designing with Storyboards

Modern Mac Features

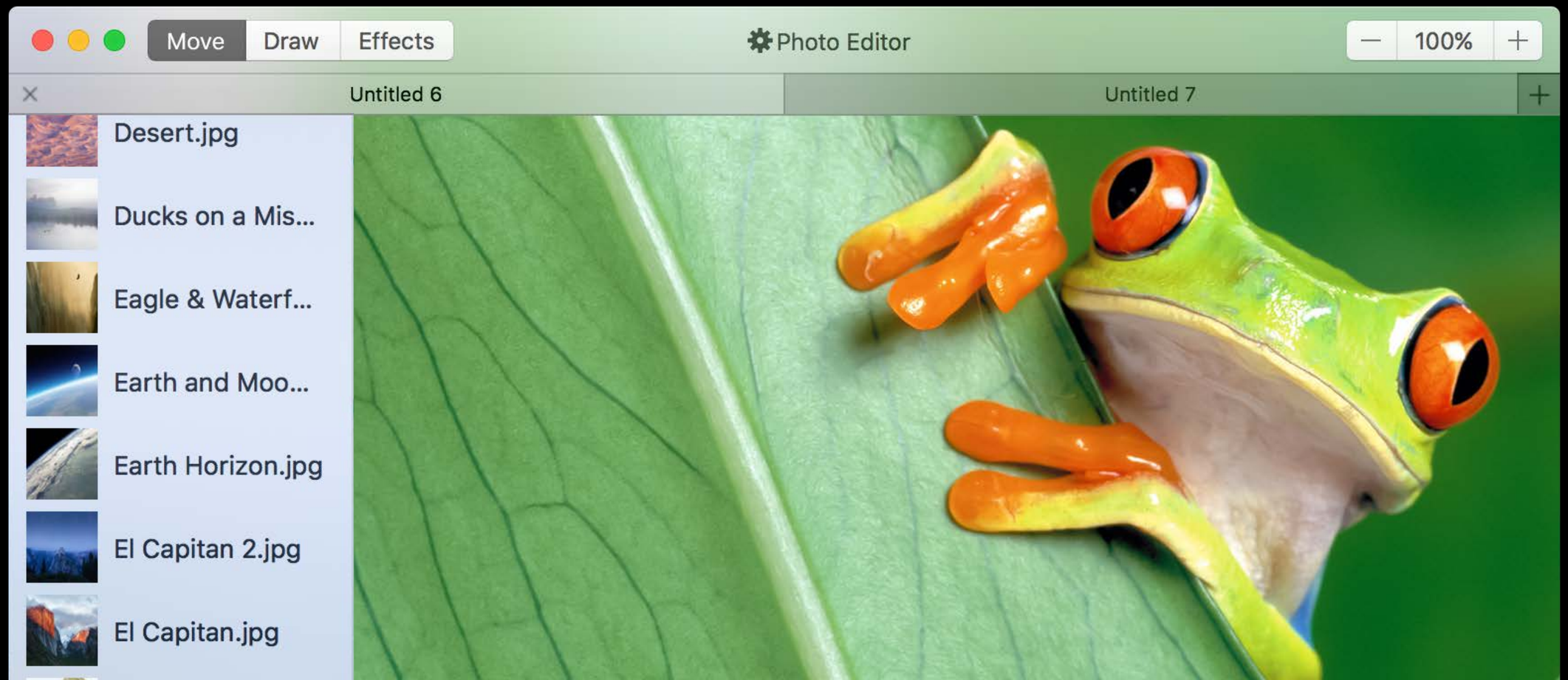
Creating a Modern Window and Toolbar



Full Size Content View

The content view should extend to underneath the toolbar area

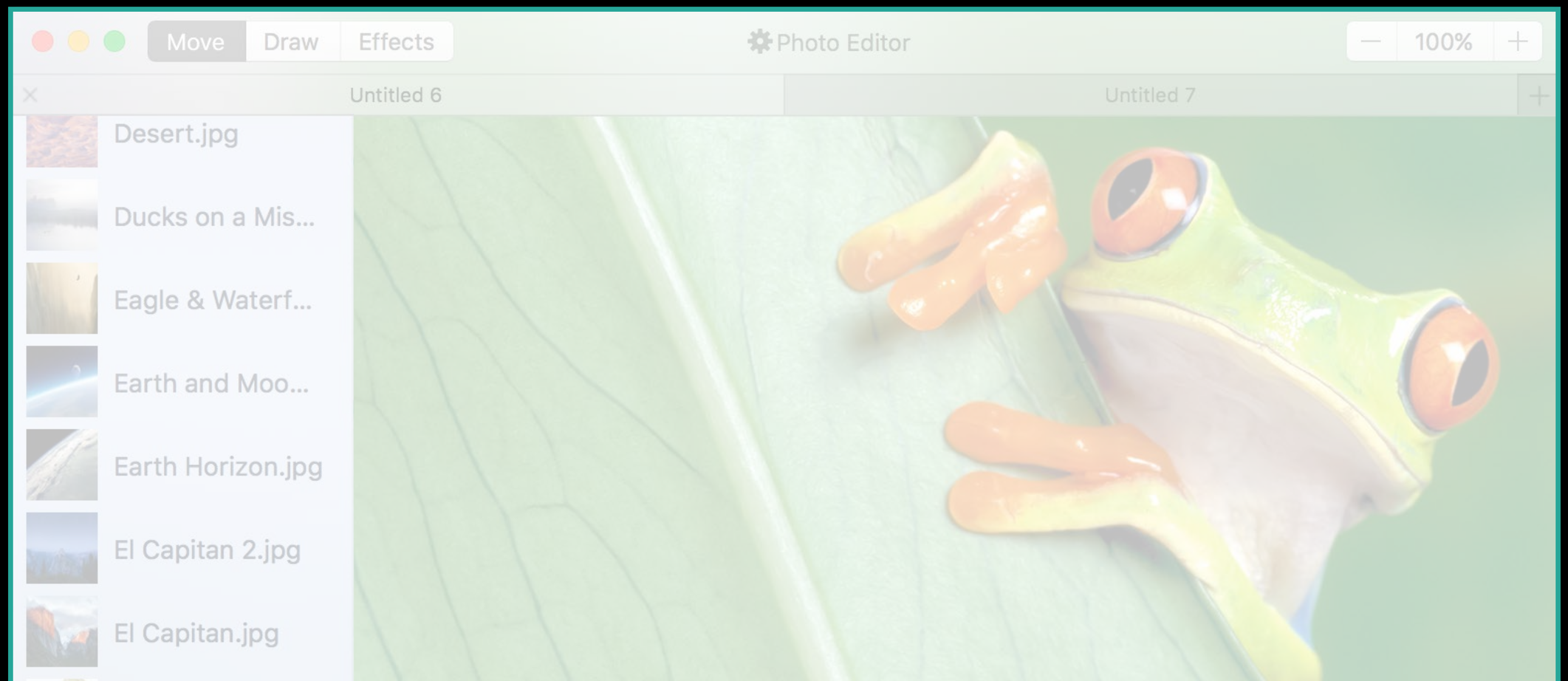
Automatically blurs content underneath it



Full Size Content View

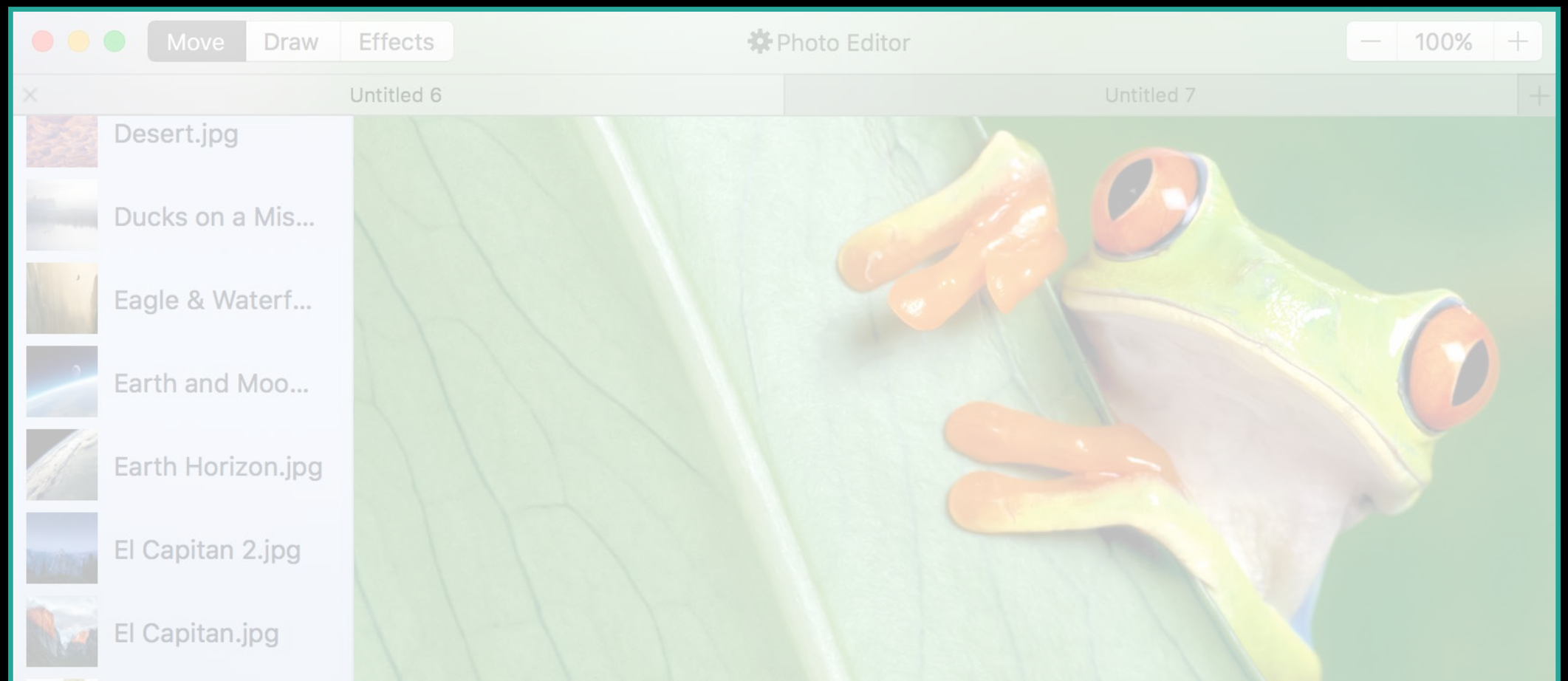
The content view should extend to underneath the toolbar area

Automatically blurs content underneath it

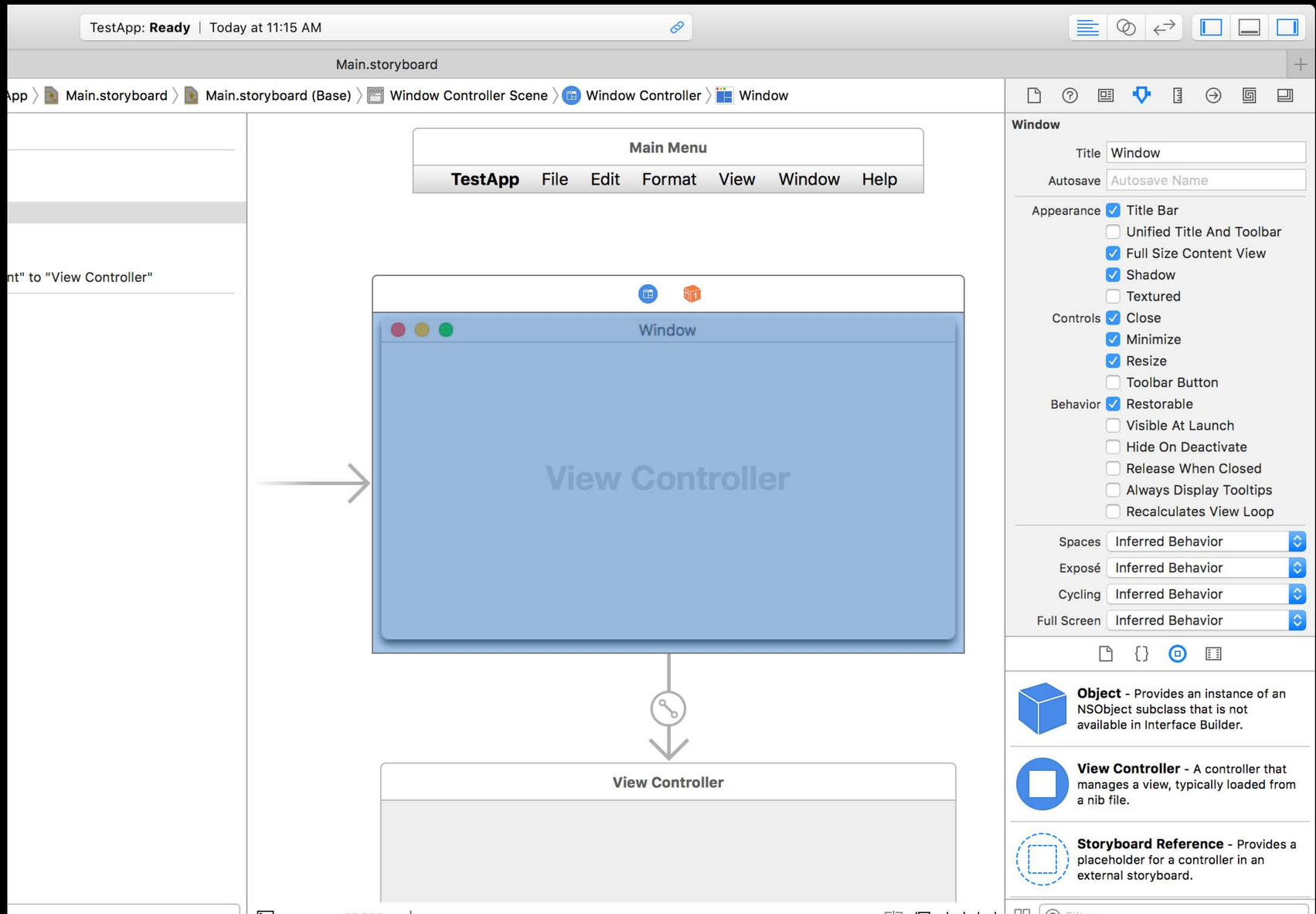


Full Size Content View

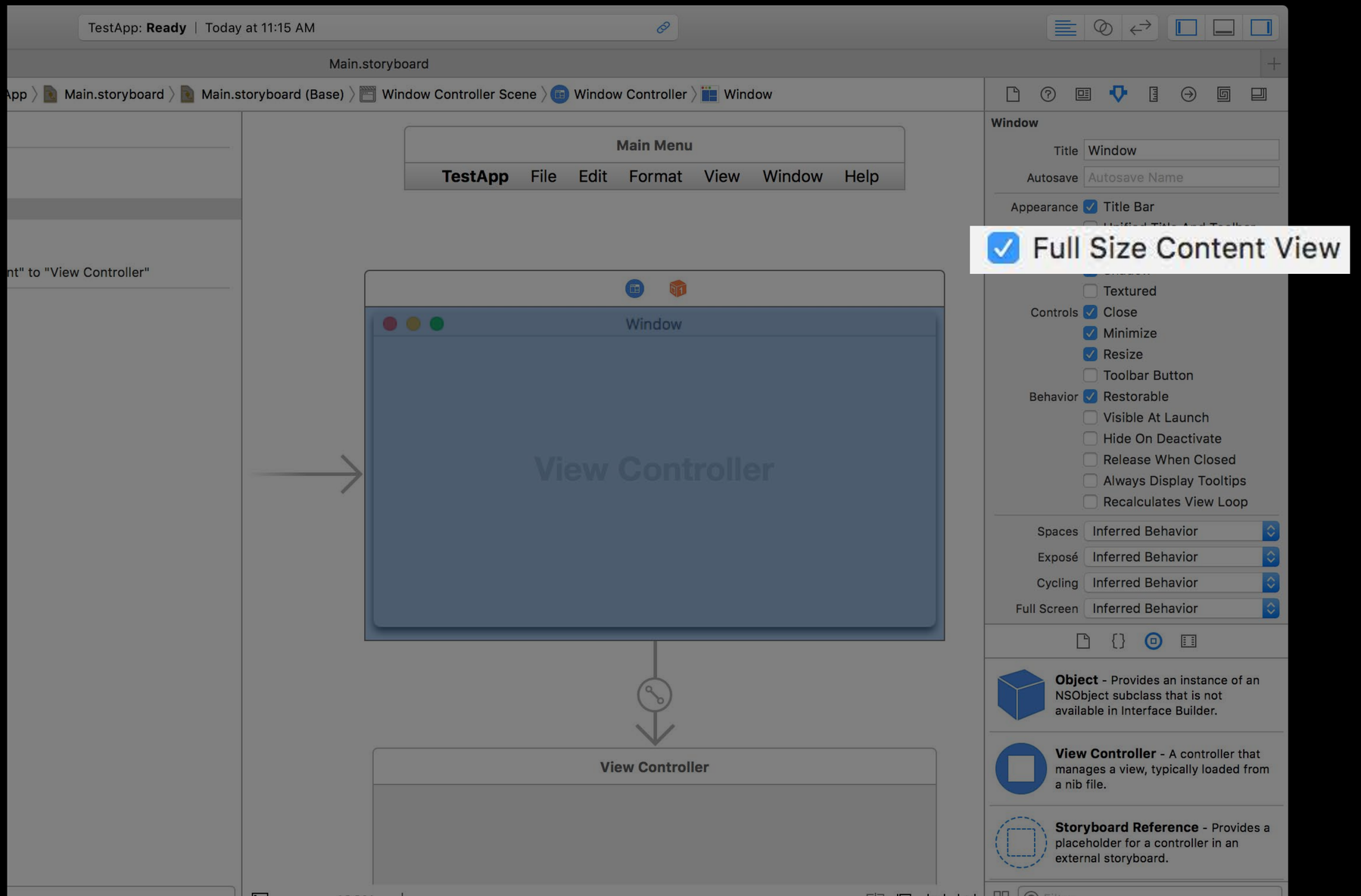
```
window.styleMask = [window.styleMask, .fullSizeContentView]
```



Full Size Content View

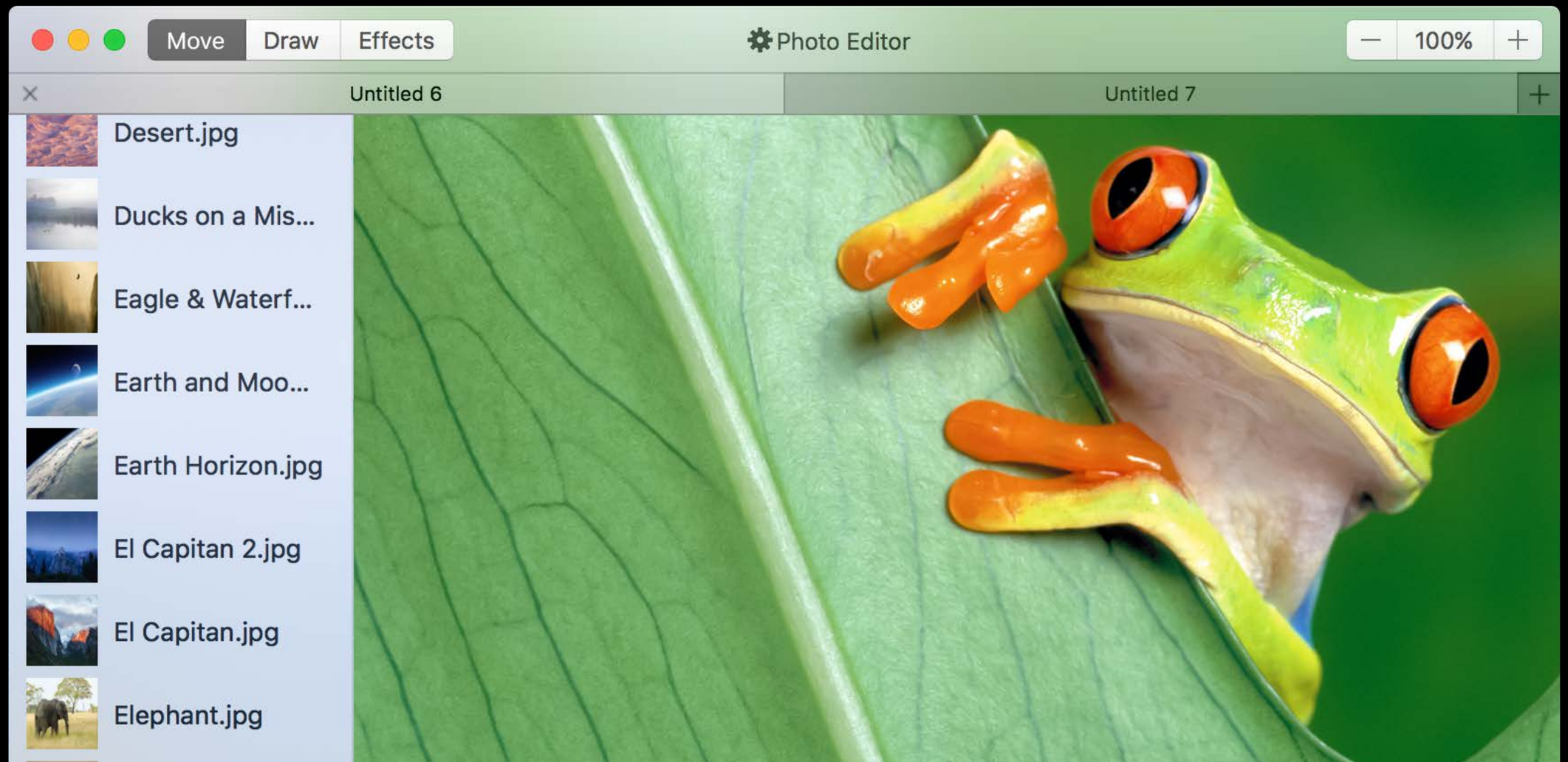


Full Size Content View



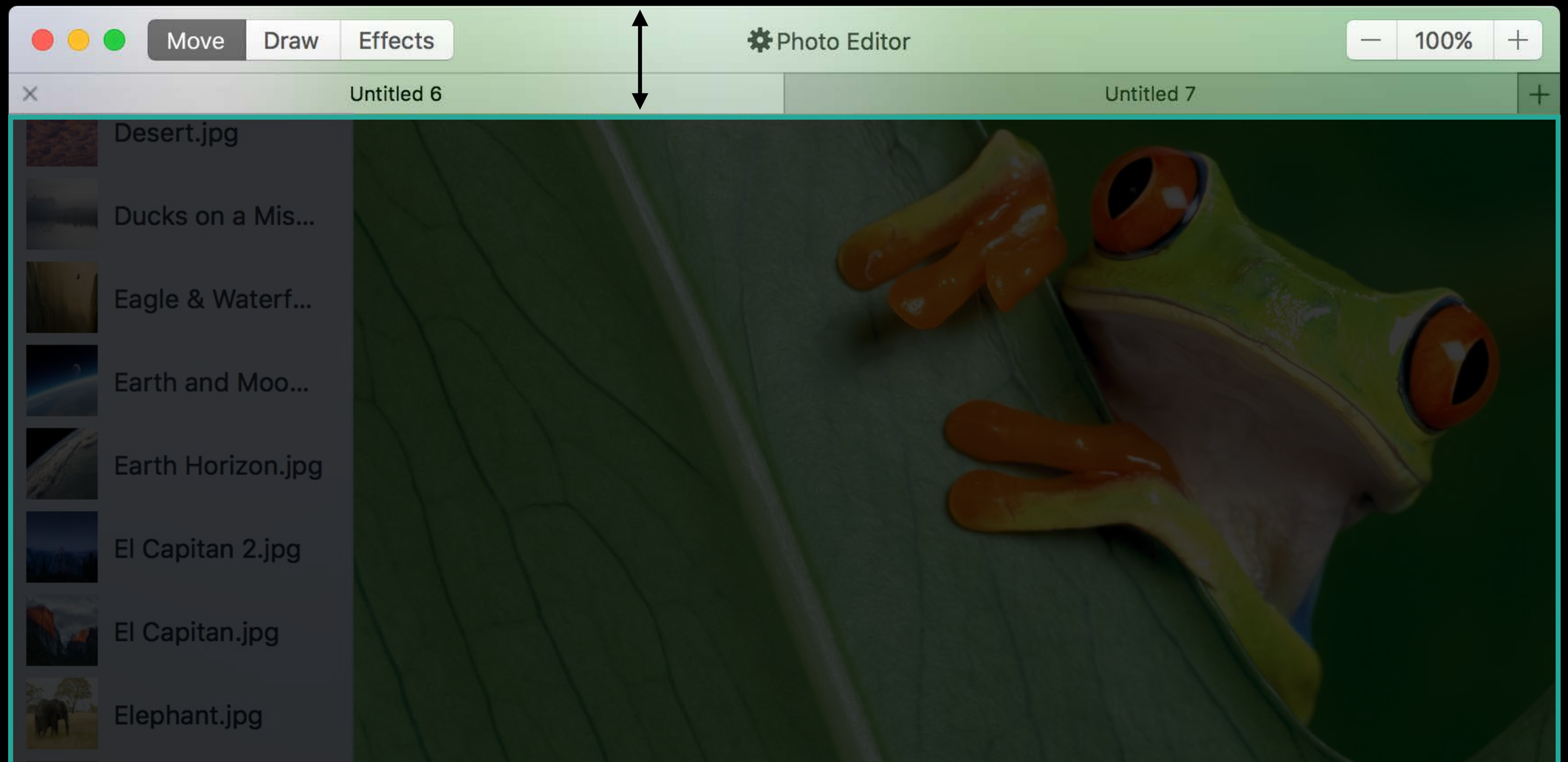
Full Size Content View

Offsetting content under the toolbar/titlebar



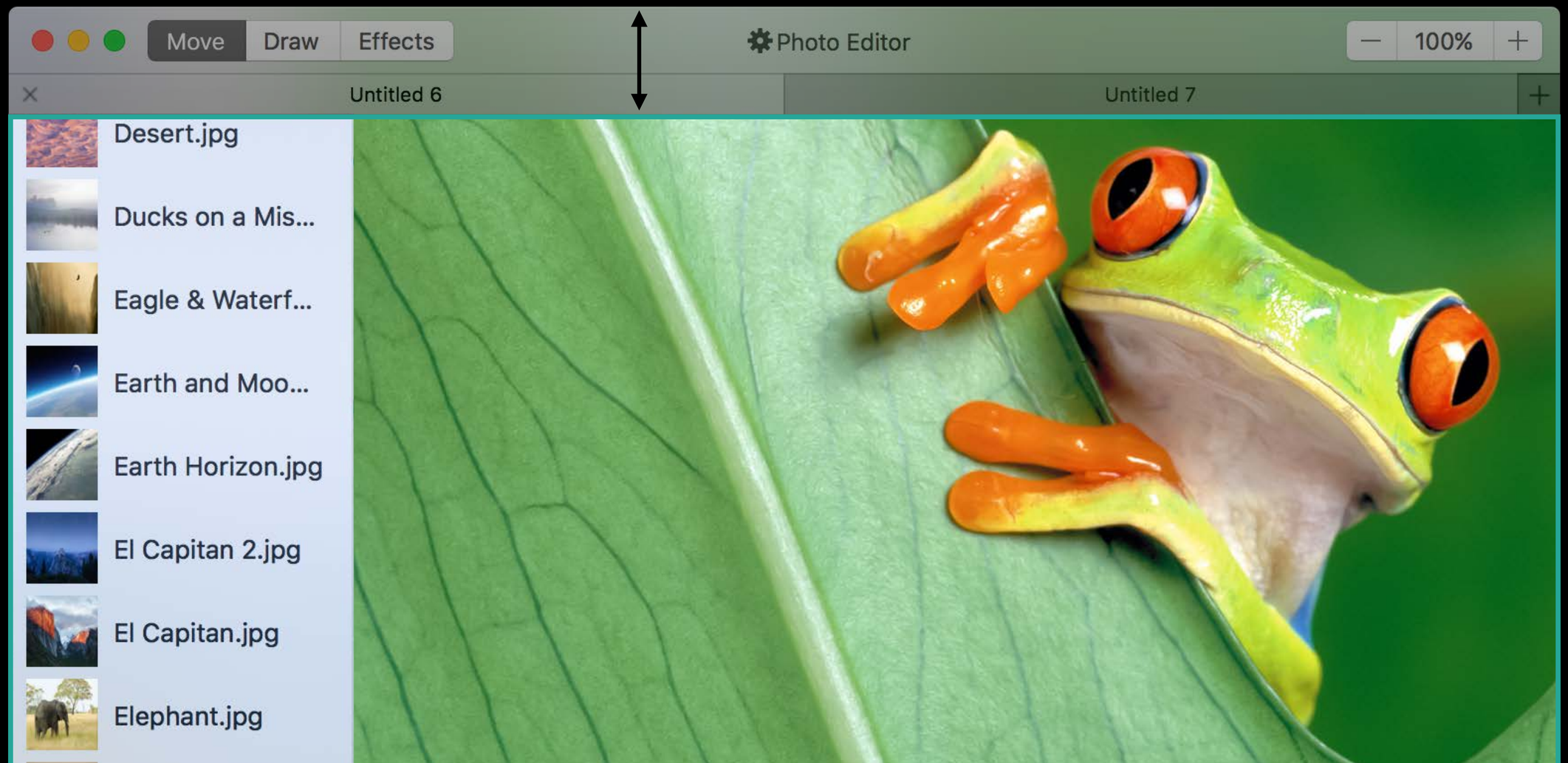
Full Size Content View

Offsetting content under the toolbar/titlebar



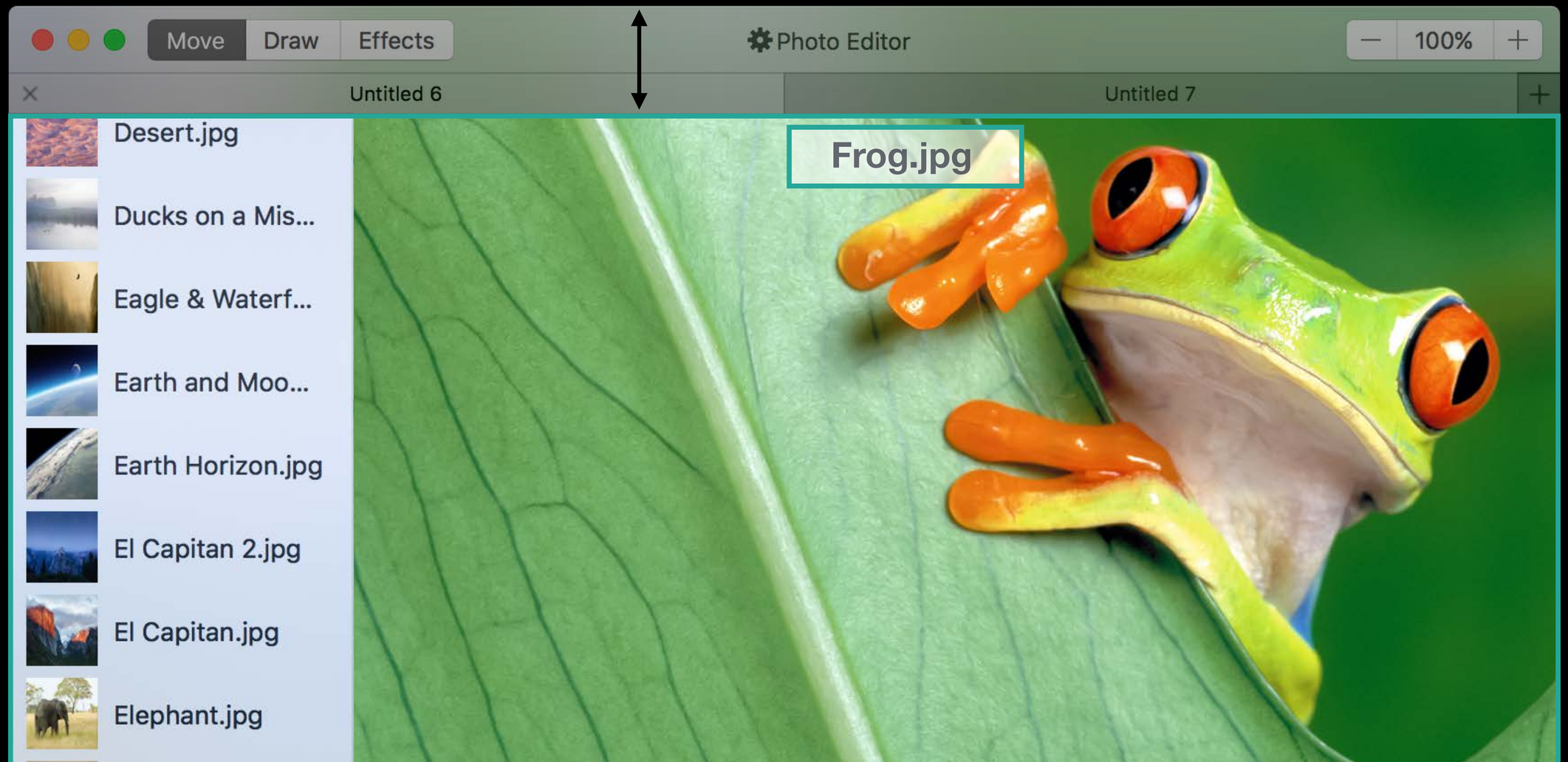
Full Size Content View

Offsetting content under the toolbar/titlebar



Full Size Content View

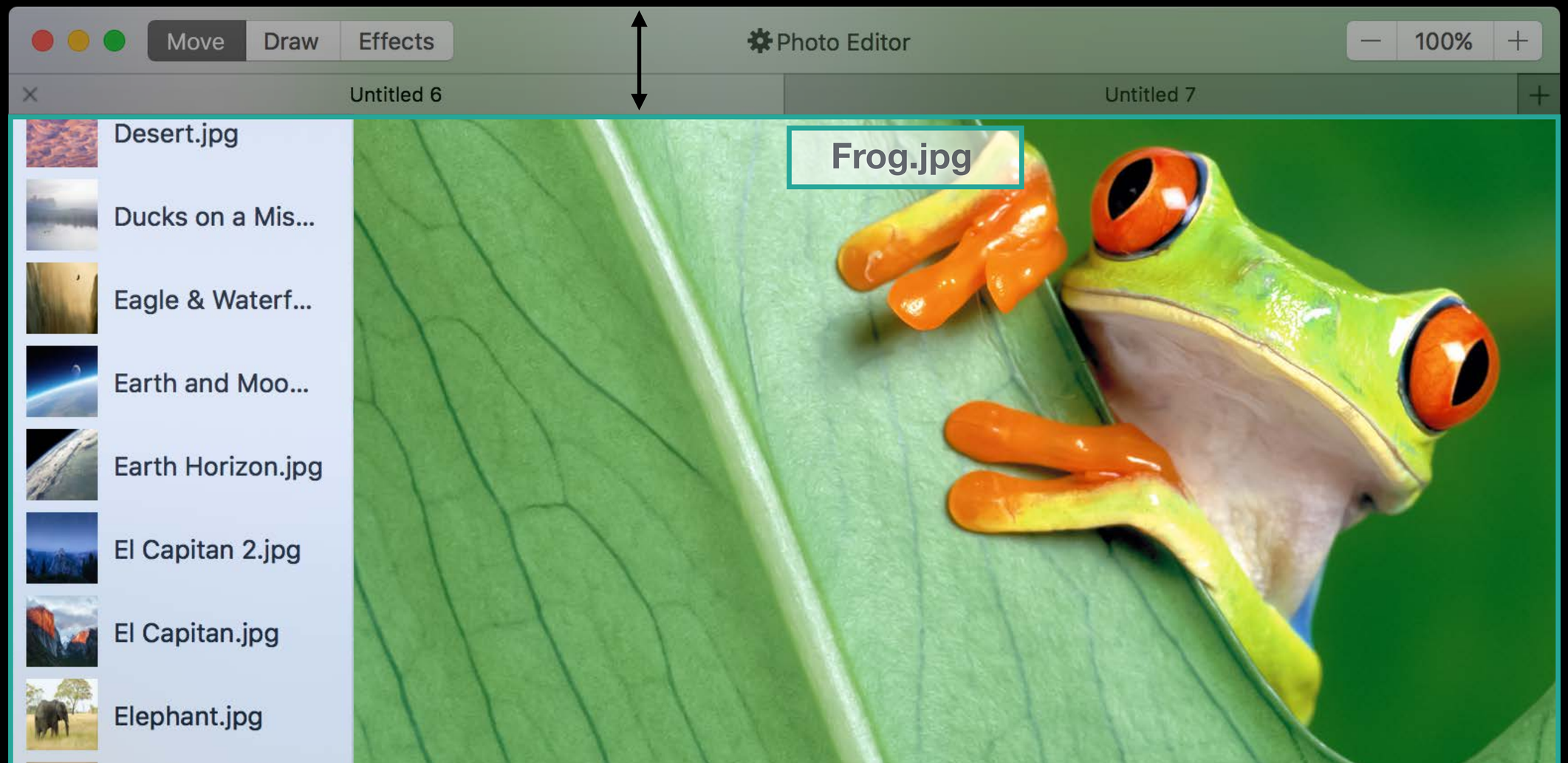
Offsetting content under the toolbar/titlebar



Full Size Content View

NSWindow property

```
public var contentLayoutRect: NSRect { get }
```



Full Size Content View

Prefer to use the contentLayoutGuide with Auto Layout

```
override func updateViewConstraints() {
   if topConstraint == nil {
     if let topAnchor = titleTextField.window?.contentLayoutGuide?.topAnchor {
       topConstraint = titleTextField.topAnchor.constraint(equalTo: topAnchor, constant: 2)
       topConstraint!.isActive = true
     }
   }
   super.updateViewConstraints()
}
```

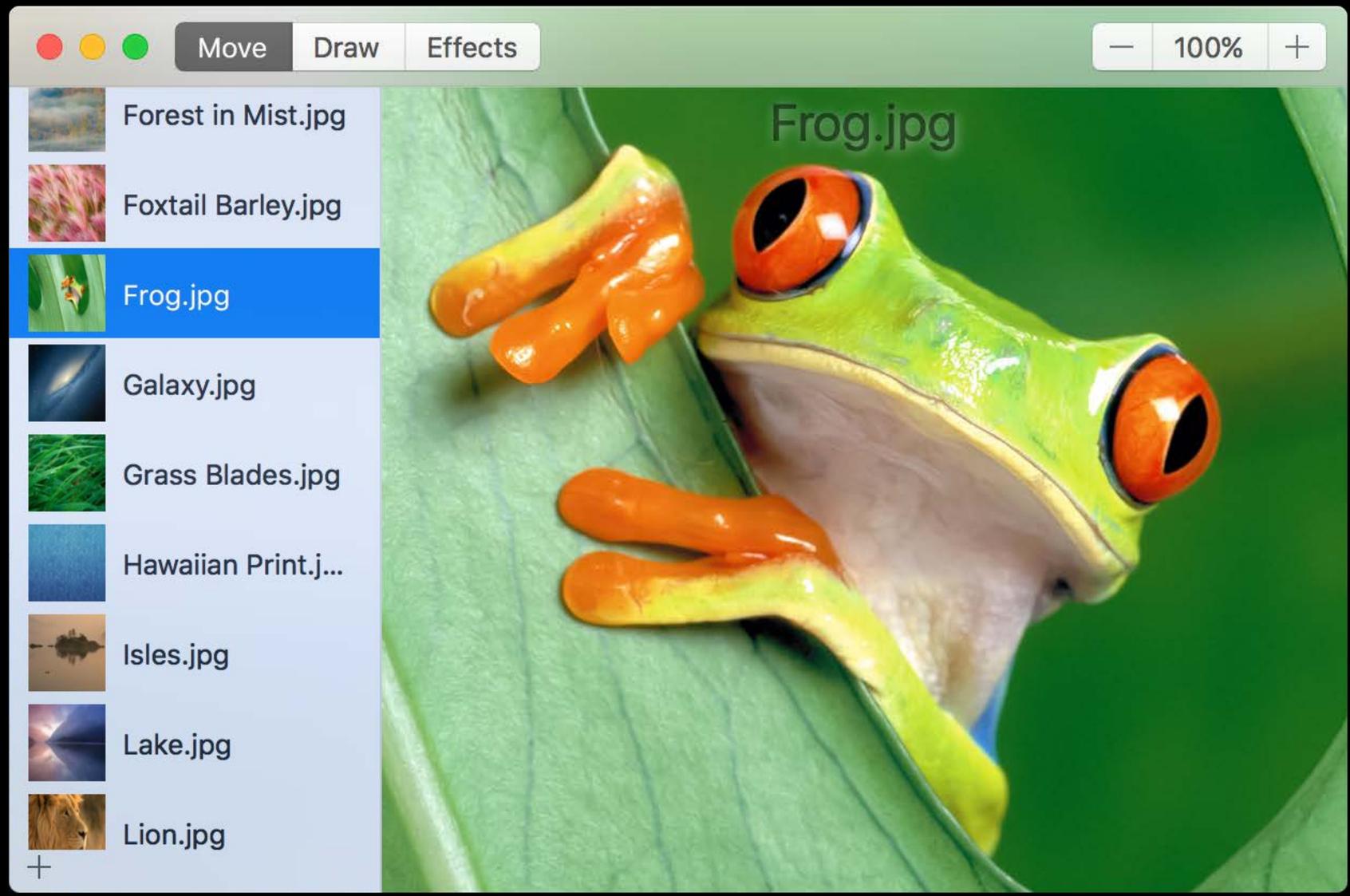
Full Size Content View

Prefer to use the contentLayoutGuide with Auto Layout

```
override func updateViewConstraints() {
   if topConstraint == nil {
     if let topAnchor = titleTextField.window?.contentLayoutGuide?.topAnchor {
       topConstraint = titleTextField.topAnchor.constraint(equalTo: topAnchor, constant: 2)
       topConstraint!.isActive = true
      }
   }
   super.updateViewConstraints()
}
```

Use a Streamlined Toolbar

Use a Streamlined Toolbar



Use a Streamlined Toolbar

When and how

Generally for non-document based or “shoebox” applications

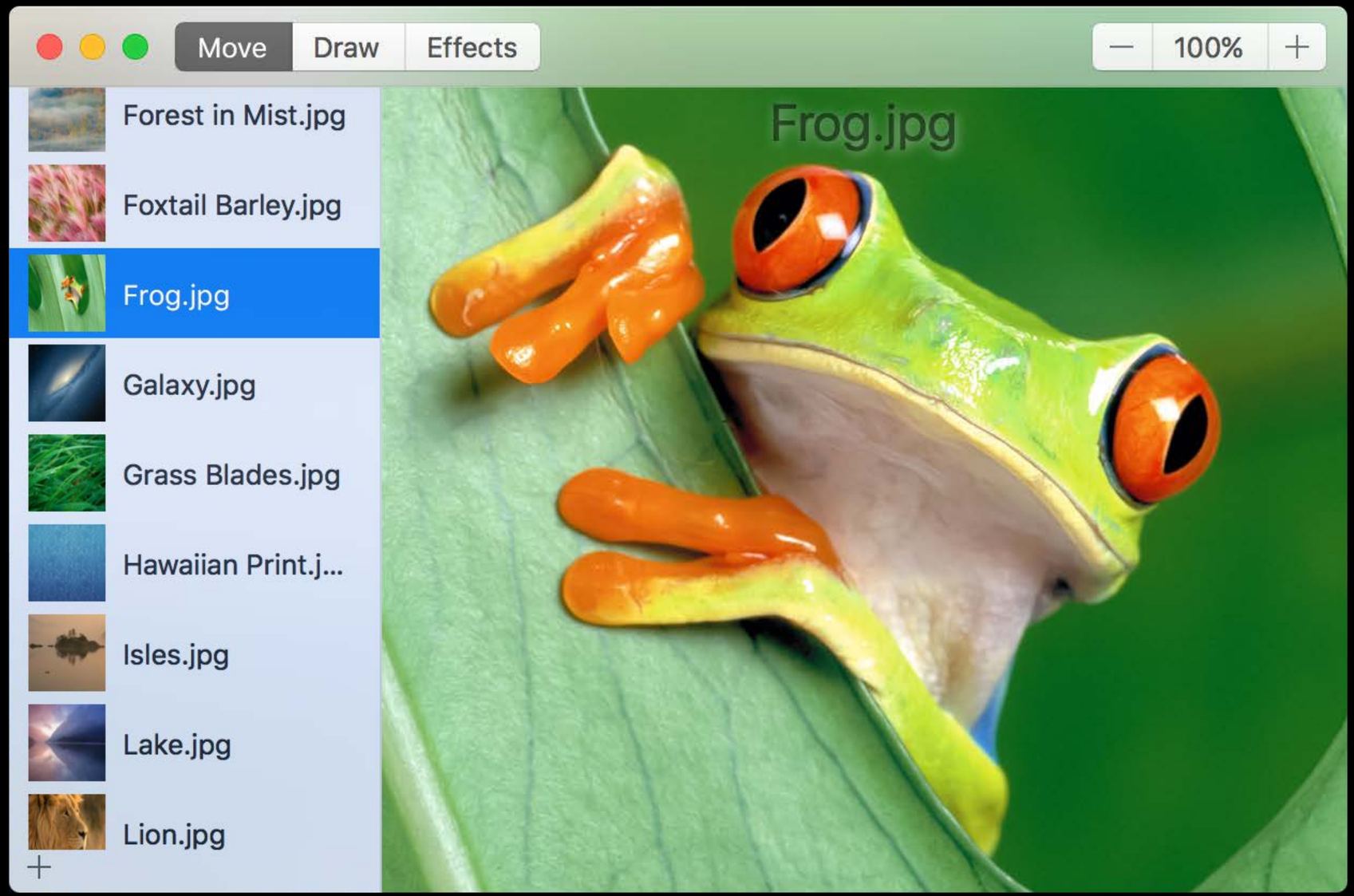
- Maps, Calendar, Safari, System Preferences, Photos

Easy to set in code

```
override func viewWillAppear() {
    super.viewWillAppear()
    self.view.window!.titleVisibility = .hidden
}
```

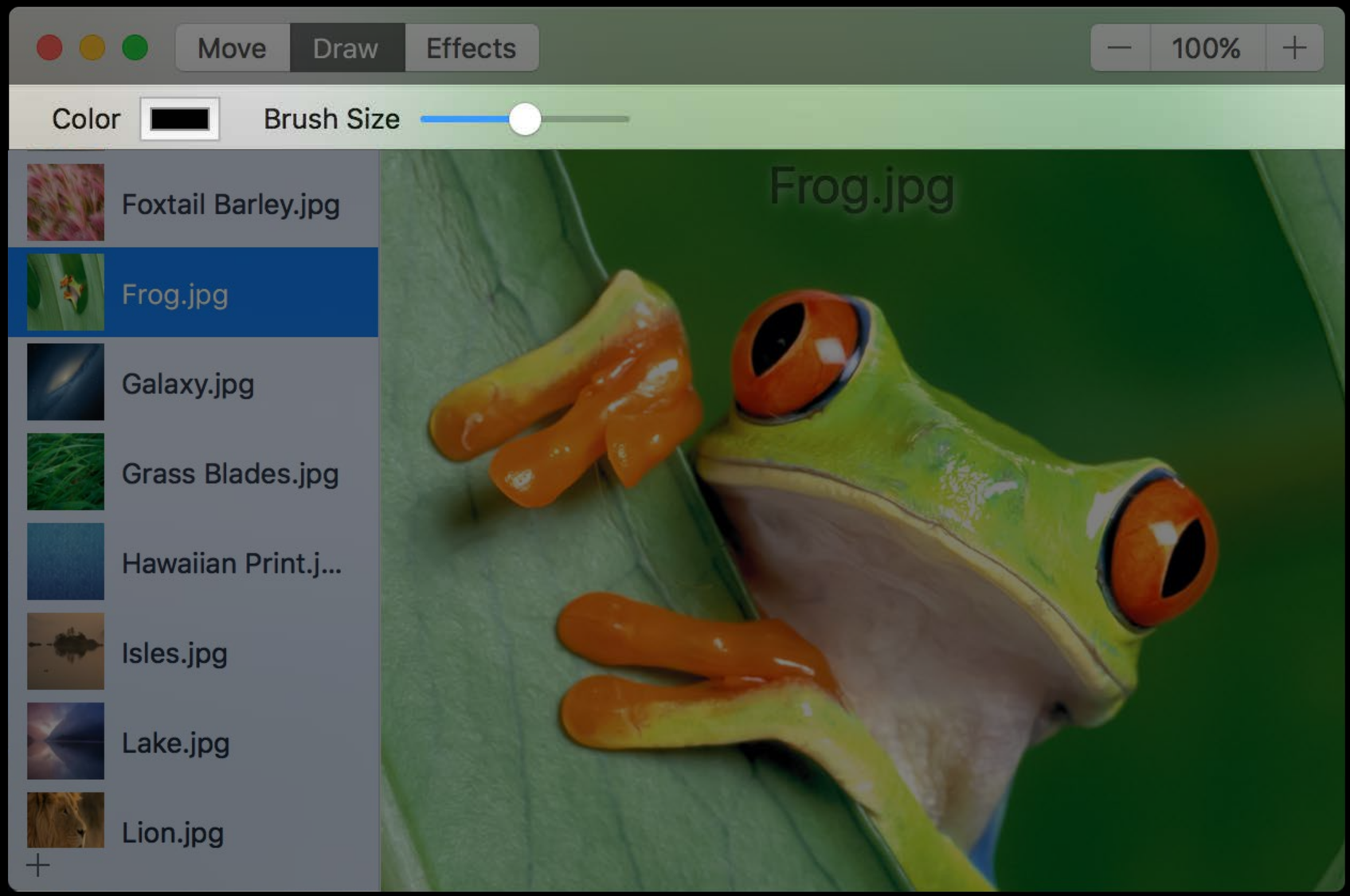
Creating a Modern Window and Toolbar

Complement with title bar accessory view controllers



Creating a Modern Window and Toolbar

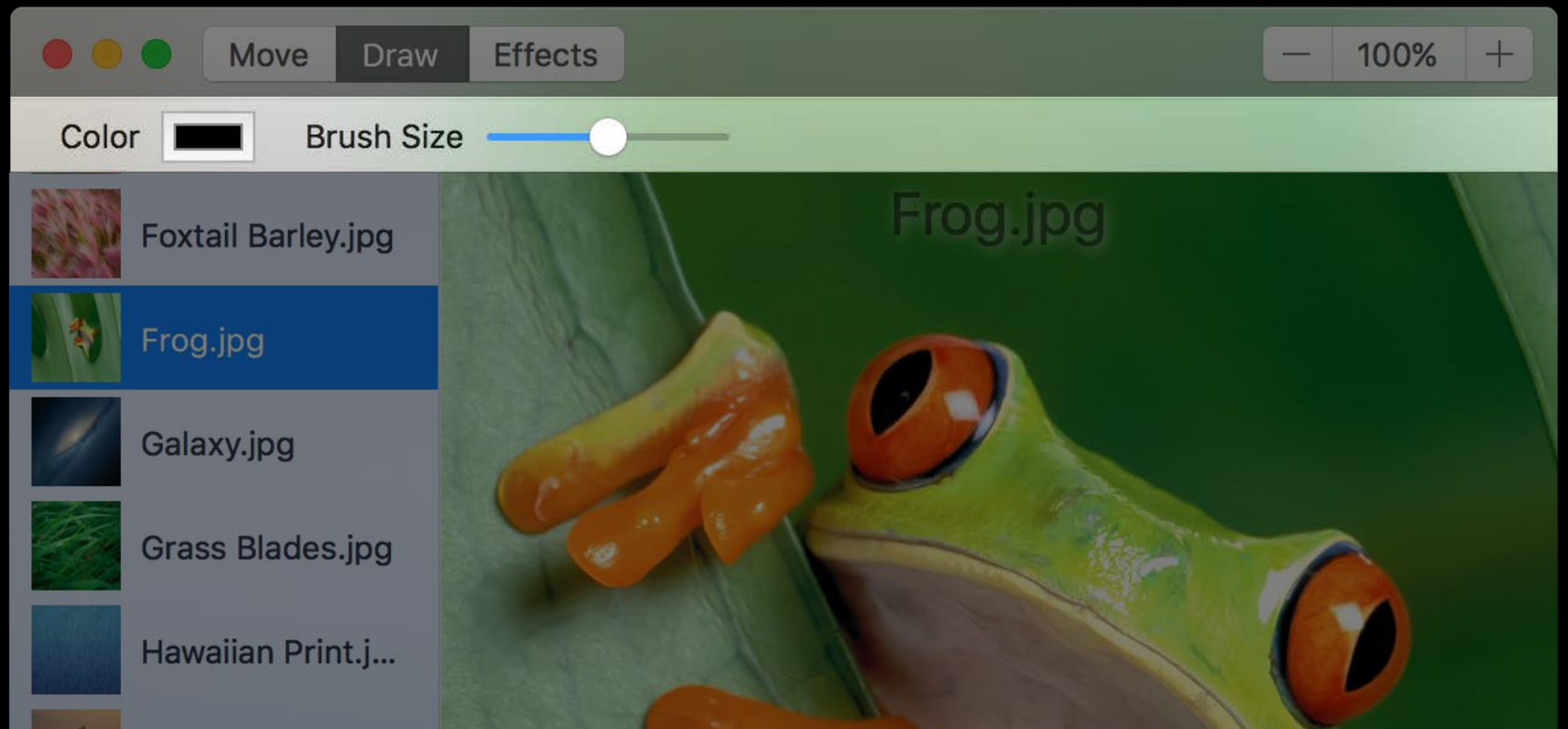
Complement with title bar accessory view controllers



Creating a Modern Window and Toolbar

Complement with title bar accessory view controllers

```
class NSTitlebarAccessoryViewController : NSViewController { … }


accessoryViewController.layoutAttribute = NSLayoutAttribute.bottom
```



Creating a Modern Window and Toolbar

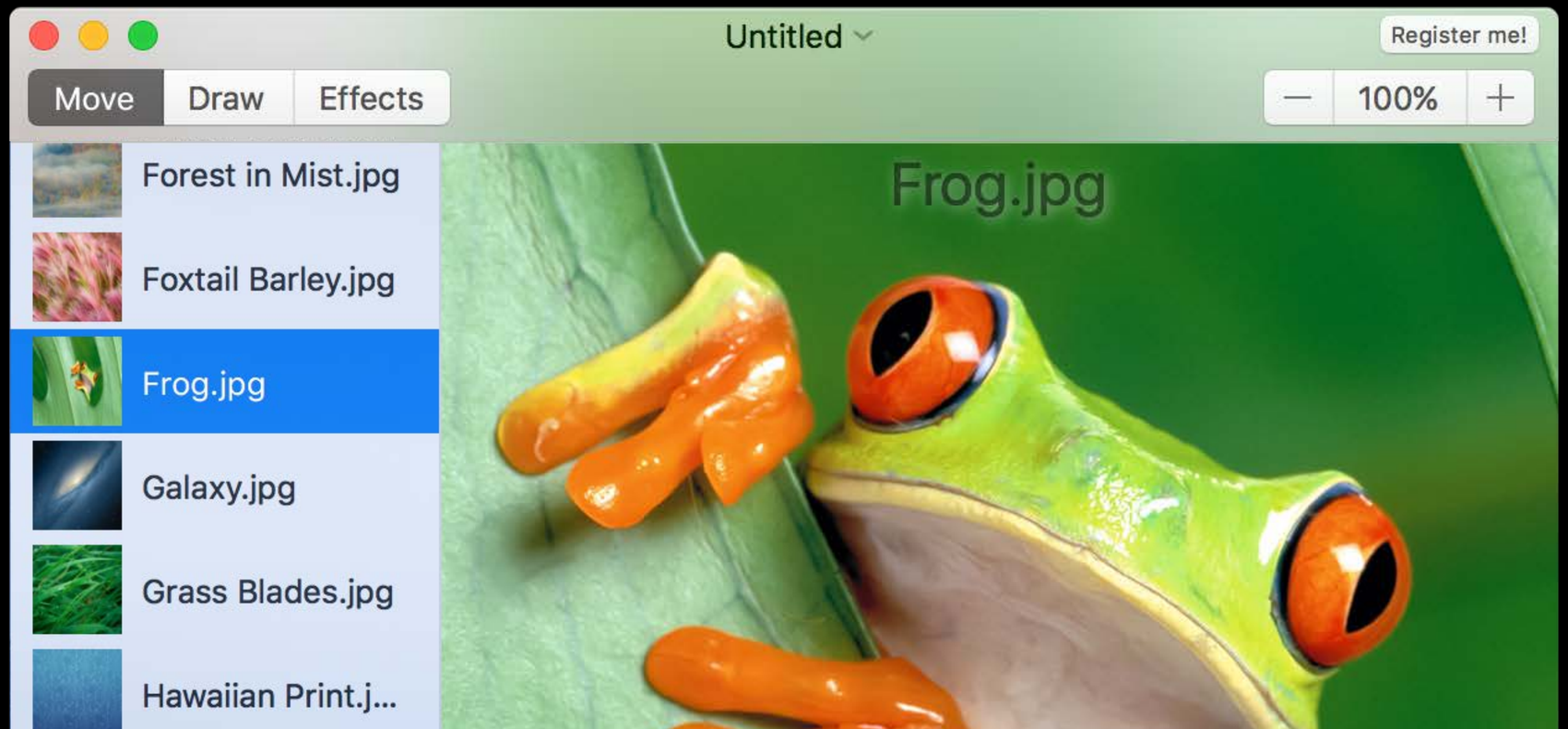
NEW

Complement with title bar accessory view controllers

```
accessoryViewController.layoutAttribute = NSLayoutAttribute.trailing
```

NSLayoutAttribute.trailing is new to 10.12 and replaces NSLayoutAttribute.right

Better abstraction for Right to Left (RTL) language localizations



Creating a Modern Window and Toolbar

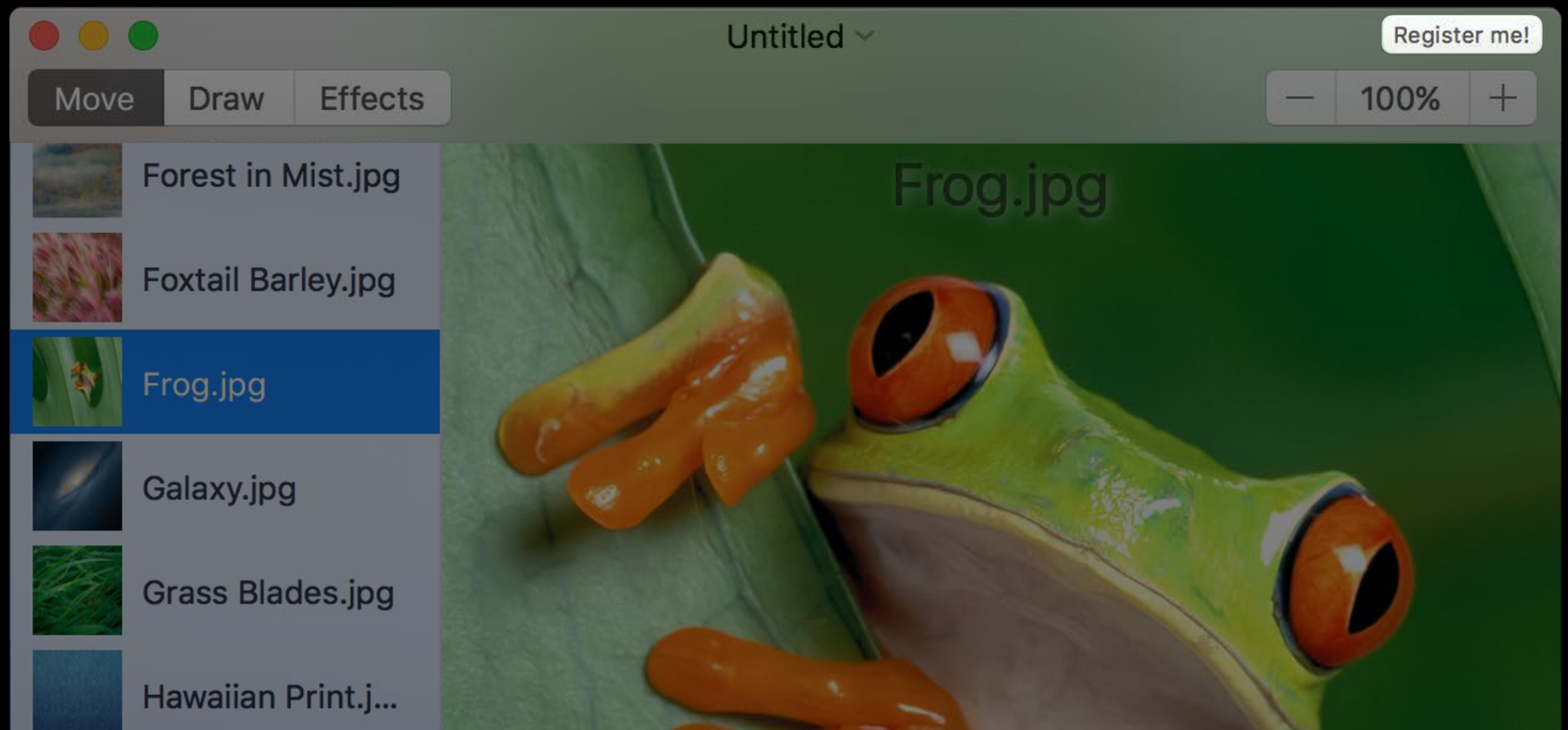
NEW

Complement with title bar accessory view controllers

```
accessoryViewController.layoutAttribute = NSLayoutAttribute.trailing
```

NSLayoutAttribute.trailing is new to 10.12 and replaces NSLayoutAttribute.right

Better abstraction for Right to Left (RTL) language localizations



Creating a Modern Window and Toolbar

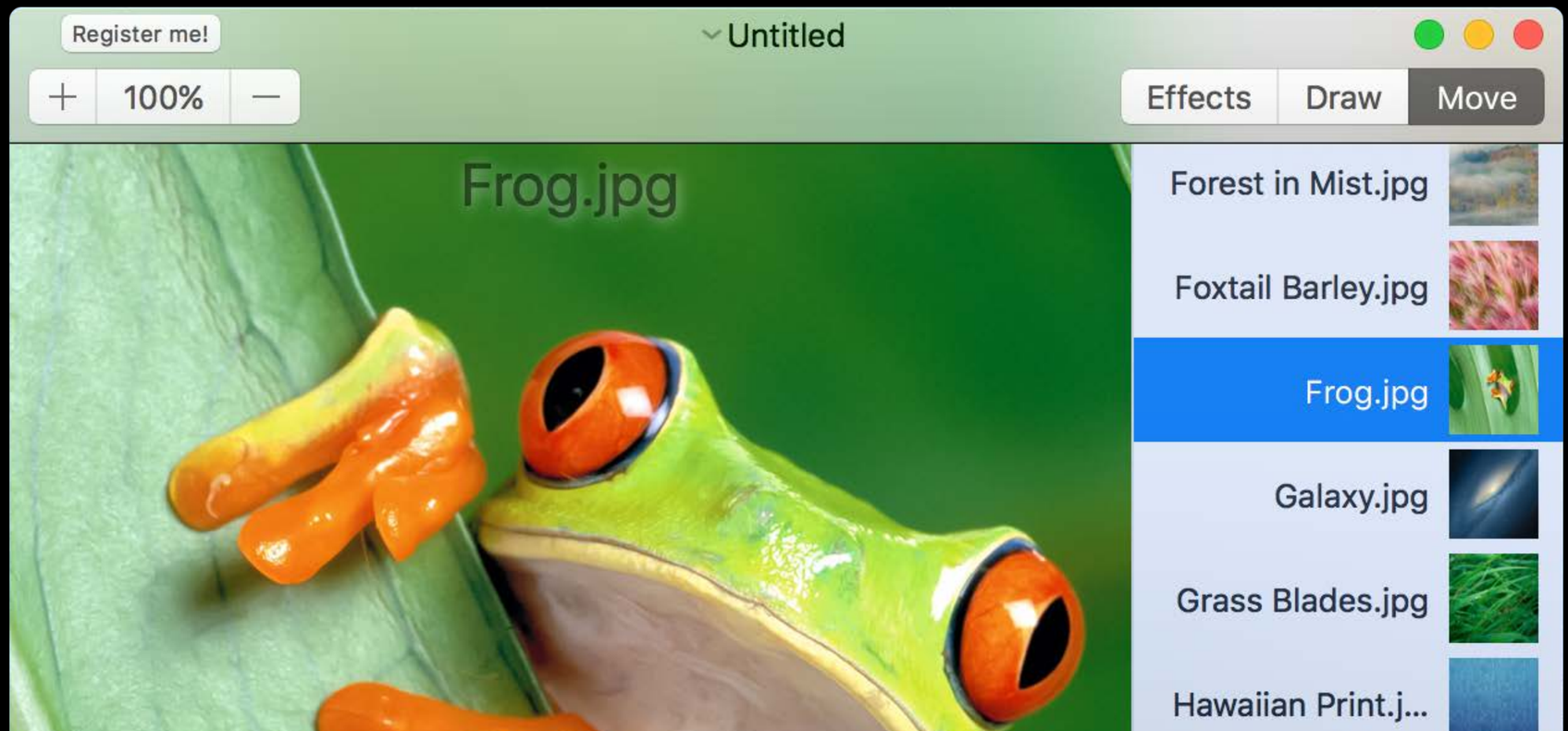
NEW

Complement with title bar accessory view controllers

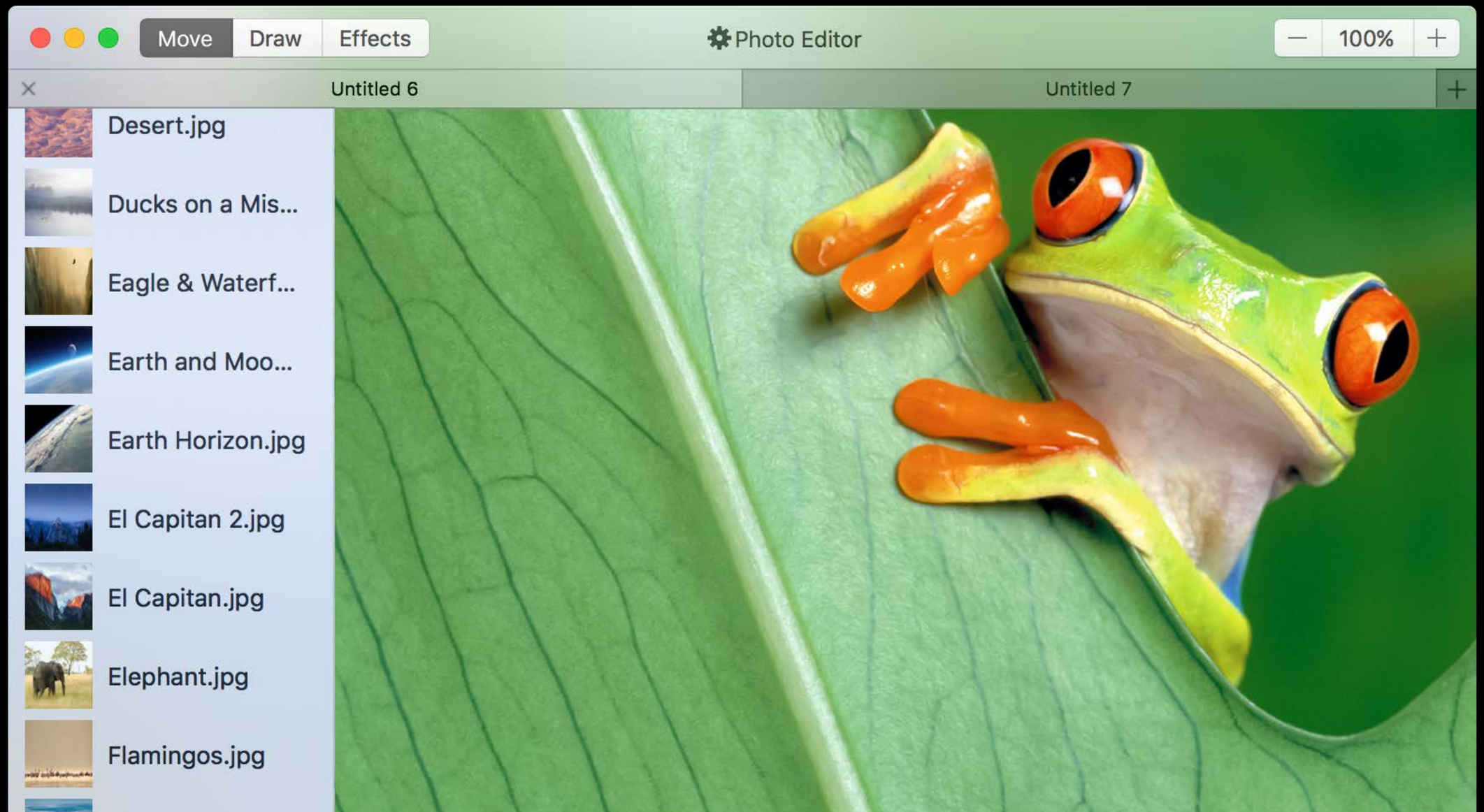
```
accessoryViewController.layoutAttribute = NSLayoutAttribute.trailing
```

NSLayoutAttribute.trailing is new to 10.12 and replaces NSLayoutAttribute.right

Better abstraction for Right to Left (RTL) language localizations



Tabbed Windows



Tabbed Windows

Windows are automatically tabbed on orderFront()

Windows are automatically un-tabbed on orderOut()

Conceptually, all windows are still considered .visible to code

- In reality, non-front most tabs are hidden via CoreGraphics

Use the -tabbingIdentifier to group together windows

- Non-NSDocument applications can enable the plus button by implementing

```
@IBAction public func newWindowForTab(sender: AnyObject?)
```

What's New in Cocoa Nob Hill Tuesday 11:00am

Creating a Modern Window and Toolbar

For more information

Improving the Full Screen Window Experience — WWDC 2015

Using Views and Core Animation

Core Animation

Graphics rendering and animation infrastructure

Drawing is done by the graphics hardware

- Allows smooth animations
- Allows fast responsive scrolling

Primary component is a CALayer

Using Views and Core Animation

UI should be composed of multiple views/layers

Using Views and Core Animation

UI should be composed of multiple views/layers

Using Views and Core Animation

Layer properties to provide content

```
public class CALayer {
    public var contents: AnyObject?

    public var backgroundColor: CGColor?
    public var cornerRadius: CGFloat
    public var borderWidth: CGFloat
    public var borderColor: CGColor?
…
}
```

Using Views and Core Animation

Layer properties to provide content

```
public class CALayer {
    public var contents: AnyObject?

    public var backgroundColor: CGColor?
    public var cornerRadius: CGFloat
    public var borderWidth: CGFloat
    public var borderColor: CGColor?
…
}
```

Using Views and Core Animation

Layer properties to provide content

```
public class CALayer {
    public var contents: AnyObject?

    public var backgroundColor: CGColor?
    public var cornerRadius: CGFloat
    public var borderWidth: CGFloat
    public var borderColor: CGColor?
…
}
```

Using Views and Core Animation

A View can be backed by a Layer

```
public class NSView {

   public var wantsLayer: Bool
   public var layer: CALayer?

   public func draw(_ dirtyRect: NSRect) // -drawRect:

   public var wantsUpdateLayer: Bool { get }
   public func updateLayer()

…
}
```

Using Views and Core Animation

A View can be backed by a Layer

```
public class NSView {

    public var wantsLayer: Bool
    public var layer: CALayer?

    public func draw(_ dirtyRect: NSRect) // -drawRect:

    public var wantsUpdateLayer: Bool { get }
    public func updateLayer()

…
}
```

Using Views and Core Animation

A View can be backed by a Layer

```
public class NSView {

   public var wantsLayer: Bool
   public var layer: CALayer?

   public func draw(_ dirtyRect: NSRect) // -drawRect:

   public var wantsUpdateLayer: Bool { get }
   public func updateLayer()

…
}
```

Using Views and Core Animation

A View can be backed by a Layer

```
public class NSView {

   public var wantsLayer: Bool
   public var layer: CALayer?

   public func draw(_ dirtyRect: NSRect) // -drawRect:

   public var wantsUpdateLayer: Bool { get }
   public func updateLayer()

…
}
```

Using Views and Core Animation

A View can be backed by a Layer

```
public class NSView {

   public var wantsLayer: Bool
   public var layer: CALayer?

   public func draw(_ dirtyRect: NSRect) // -drawRect:

   public var wantsUpdateLayer: Bool { get }
   public func updateLayer()

…
}
```

Using Views and Core Animation

A View can be backed by a Layer

```
public class NSView {

   public var wantsLayer: Bool
   public var layer: CALayer?

   public func draw(_ dirtyRect: NSRect) // -drawRect:

   public var wantsUpdateLayer: Bool { get }
   public func updateLayer()

…
}
```

Using Views and Core Animation

Recommend layer backing the window.contentView

- All child views will automatically inherit layer backing

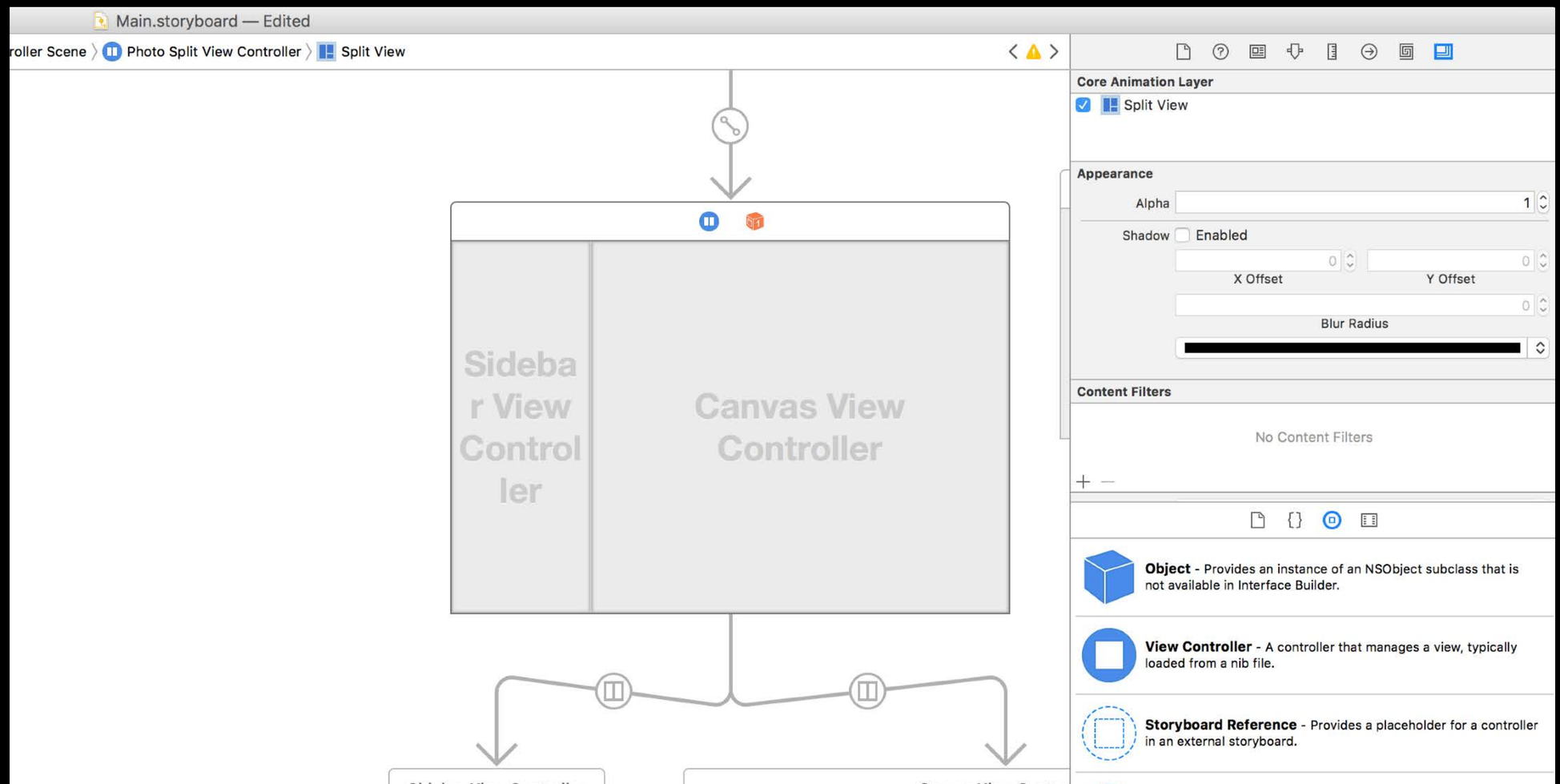
Use layer backed views

- Instead of directly adding a new CALayer as a sublayer
- Better for Retina display support

Using Views and Core Animation

How to turn on layer backing

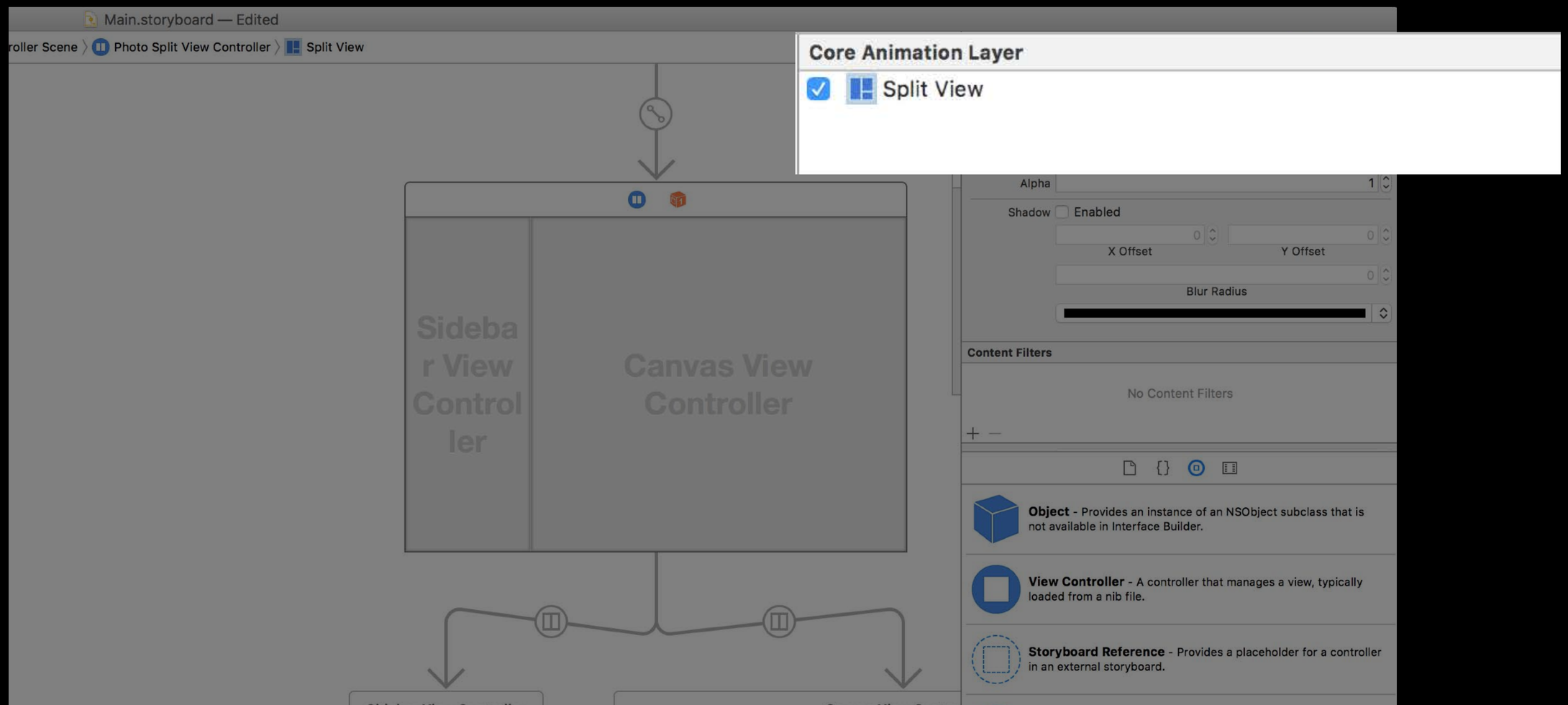
```
window.contentView!.wantsLayer = true
```



Using Views and Core Animation

How to turn on layer backing

```
window.contentView!.wantsLayer = true
```



Using Views and Layers

Redrawing views with layers

Set the layerContentsRedrawPolicy

```
view.layerContentsRedrawPolicy = .onSetNeedsDisplay
```

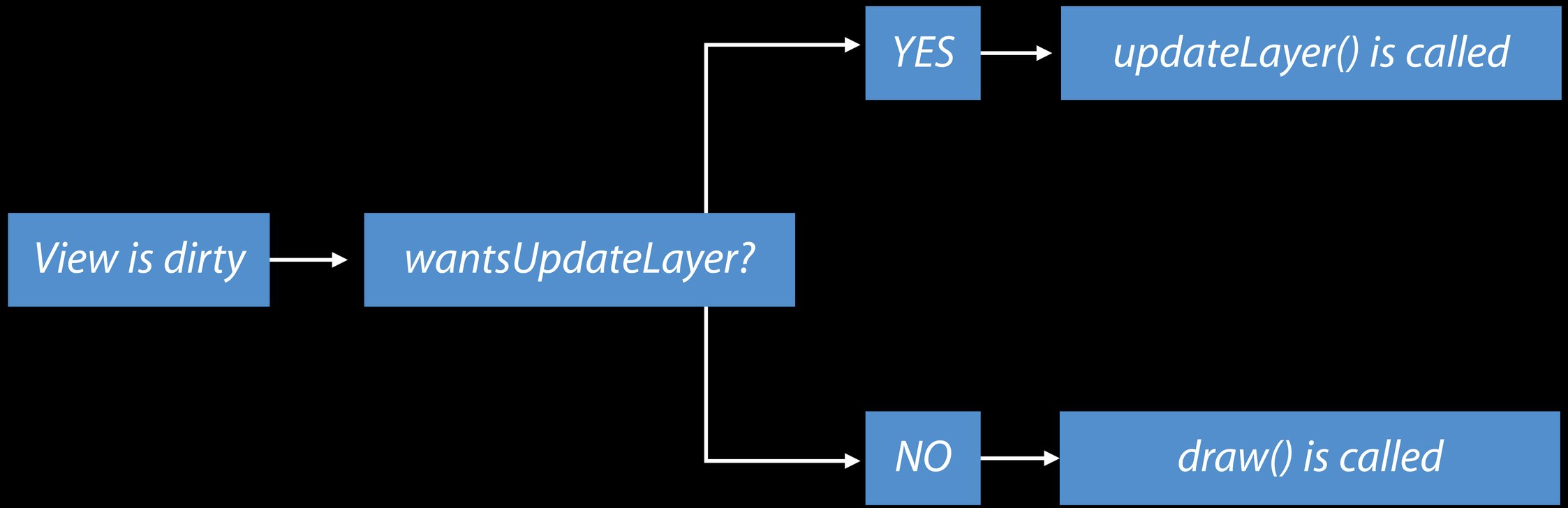
- Otherwise, the view is redrawn whenever its frame origin changes

Call needsDisplay when you need to redraw

- Such as when the content of the view changes

Using Views and Layers

How layer contents are provided for a view



Using Views and Layers

How layer contents are provided for a view

Use updateLayer() if you can represent your view's contents by setting layer properties

- Prefer updateLayer() over draw()
- Prefer to use the same layer.contents for things that look the same
- Prefer to compose UI using standard views (such as NSTextField)

Otherwise use draw(_ : NSRect)

- A unique backing store (image) is created for that view

Using Views and Layers

Using updateLayer()

```
override var wantsUpdateLayer: Bool {
    return true
}

override func updateLayer() {
    self.layer!.contents = NSImage(named: NSImageNameUserGroup)
    self.layer!.backgroundColor = NSColor.red().cgColor
}
```

Using Views and Layers

Using updateLayer()

```
override var wantsUpdateLayer: Bool {
    return true
}

override func updateLayer() {
    self.layer!.contents = NSImage(named: NSImageNameUserGroup)
    self.layer!.backgroundColor = NSColor.red().cgColor
}
```

Using Views and Layers

Using updateLayer()

```
override var wantsUpdateLayer: Bool {
    return true
}
```

```
override func updateLayer() {
    self.layer!.contents = NSImage(named: NSImageNameUserGroup)
    self.layer!.backgroundColor = NSColor.red().cgColor
}
```

What to Expect

Creating a Modern Look with Modern Views

Modern Drag & Drop, and Event Tracking

Container View Controls Handled Correctly

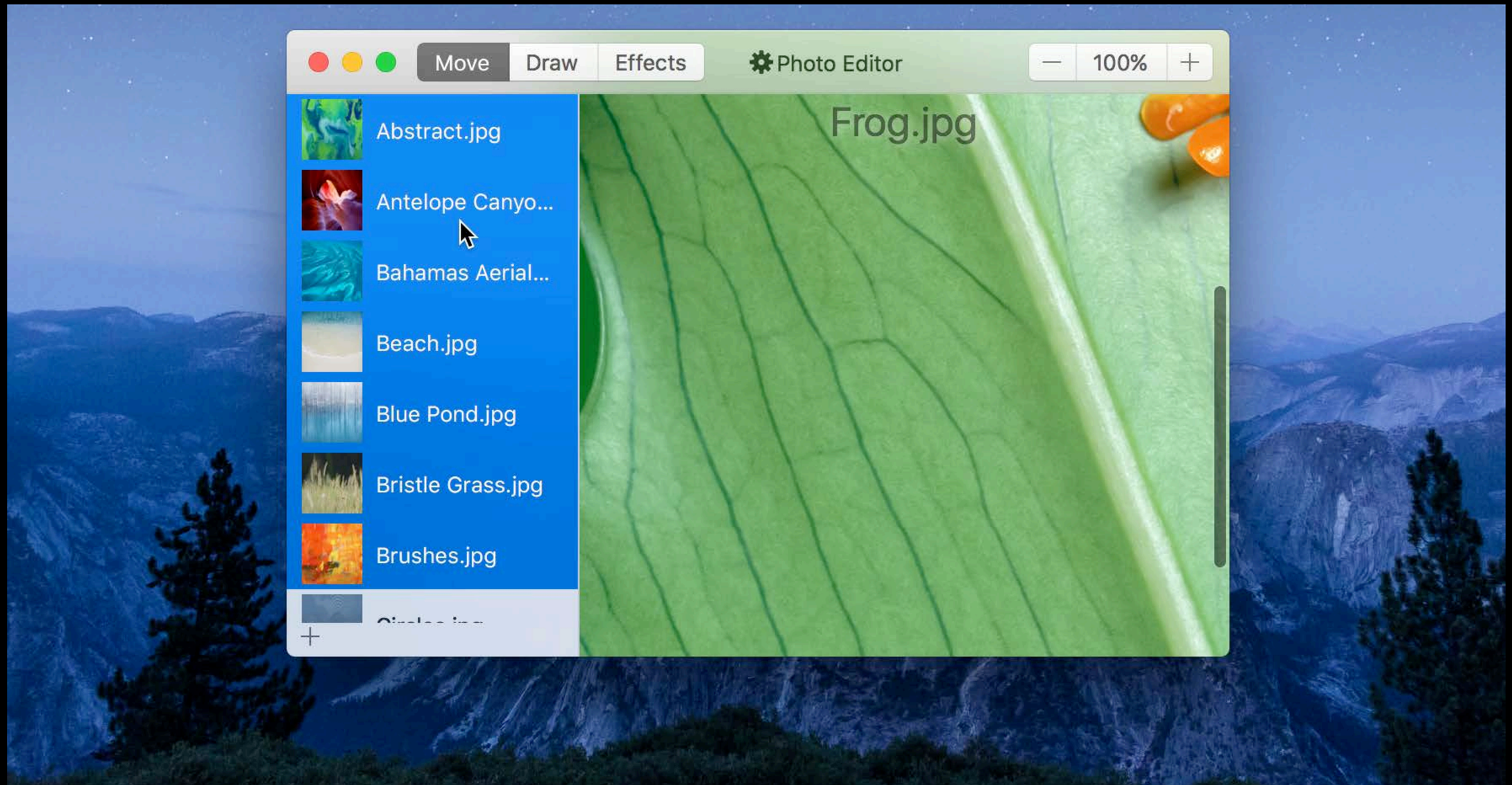
Using System Appearances

Designing with Storyboards

Modern Mac Features

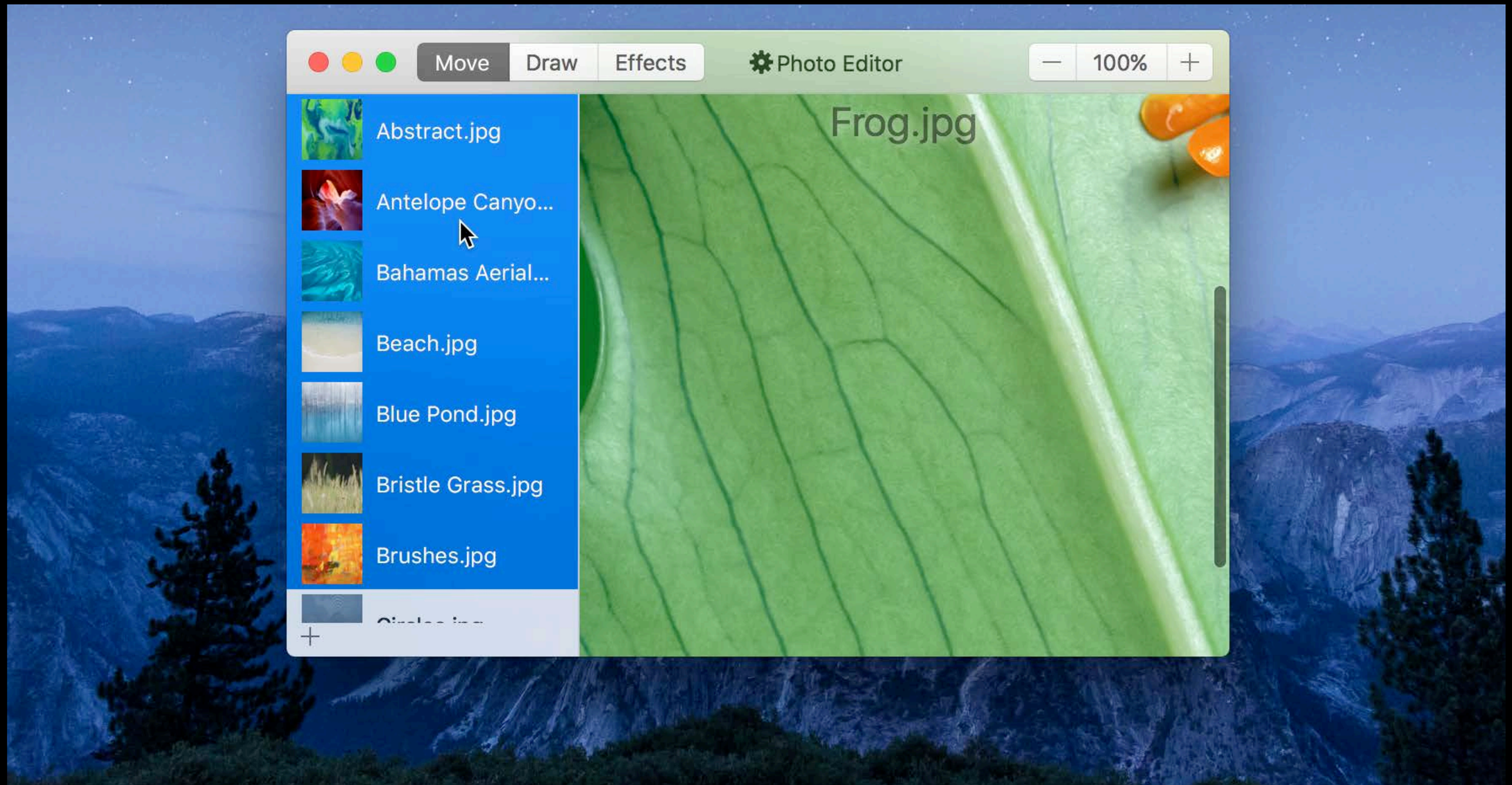
Modern Drag & Drop, and Event Tracking

Adopt drag flocking



Modern Drag & Drop, and Event Tracking

Adopt drag flocking



Adopt Drag Flocking

Start a drag with

```
extension NSView {
    public func beginDraggingSession(items: [NSDraggingItem],
                                     event: NSEvent,
                                     source: NSDraggingSource) -> NSDraggingSession

}
```

Adopt Drag Flocking

In a table view or outline view

Use the delegate method

```
optional public func tableView(_ tableView: NSTableView,
                      pasteboardWriterForRow: Int) -> NSPasteboardWriting?
```

Instead of the delegate method

```
optional public func tableView(_ tableView: NSTableView,
                               writeRowsWith: NSIndexSet,
                                         to: NSPasteboard) -> Bool
```

Adopt Drag Flocking

In a collection view

Use the delegate method

```
optional public func collectionView(_ collectionView: NSCollectionView,
                                    pasteboardWriterForItemAt: IndexPath) -> NSPasteboardWriting?
```

Instead of the delegate method

```
optional public func collectionView(_ collectionView: NSCollectionView,
                                    writeItemsAt: Set<IndexPath>,
                                    to: NSPasteboard) -> Bool
```

Drag Flocking and Drag File Promises

NEW

File promises avoid putting large files on the pasteboard

10.12 now supports file promises with drag flocking via

```
public class NSFilePromiseReceiver : NSObject, NSPasteboardReading { ... }

public class NSFilePromiseProvider : NSObject, NSPasteboardWriting { ... }
```

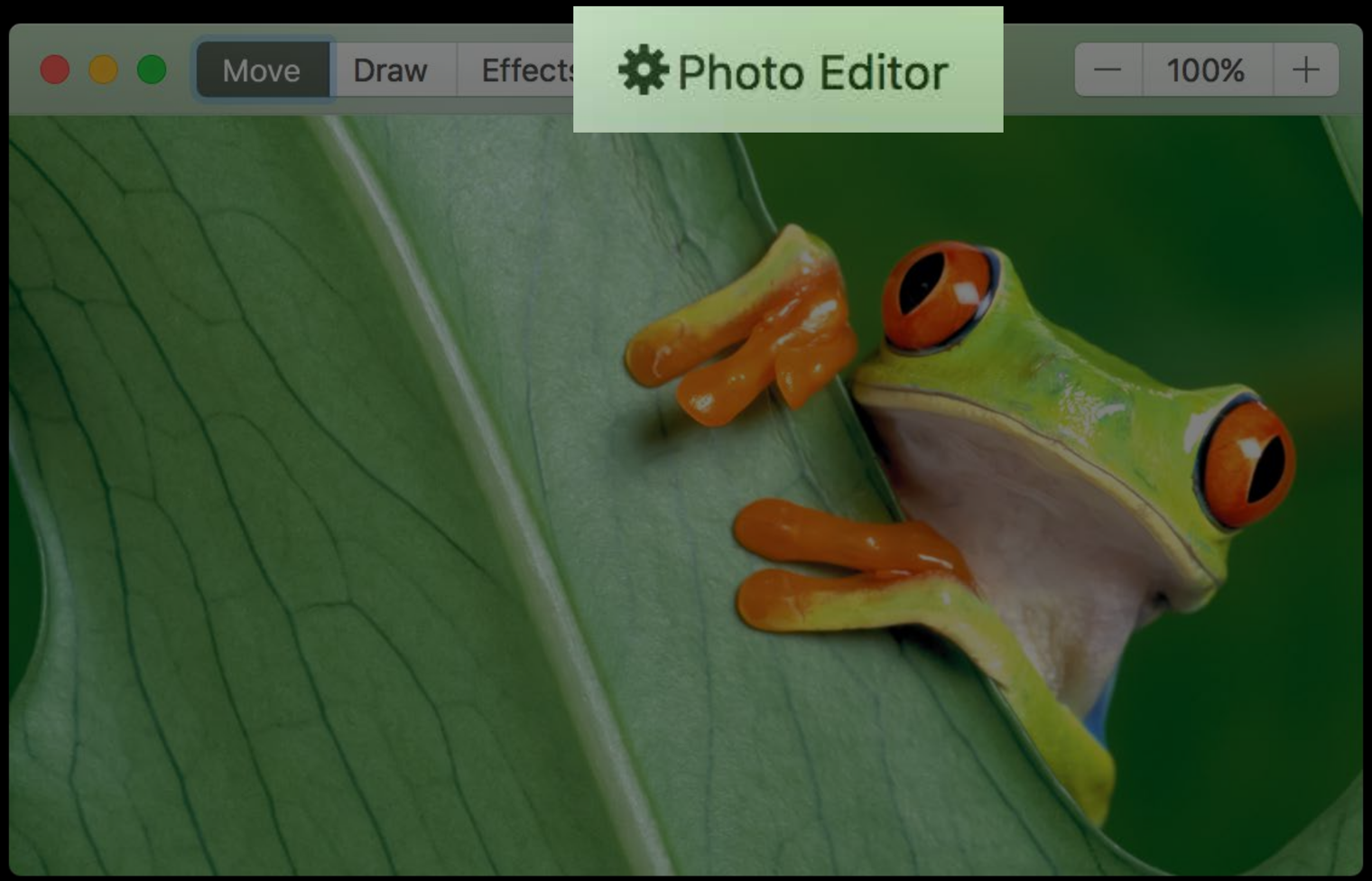
What's New in Cocoa Nob Hill Tuesday 11:00am

Modern Drag & Drop, and Event Tracking

Event tracking

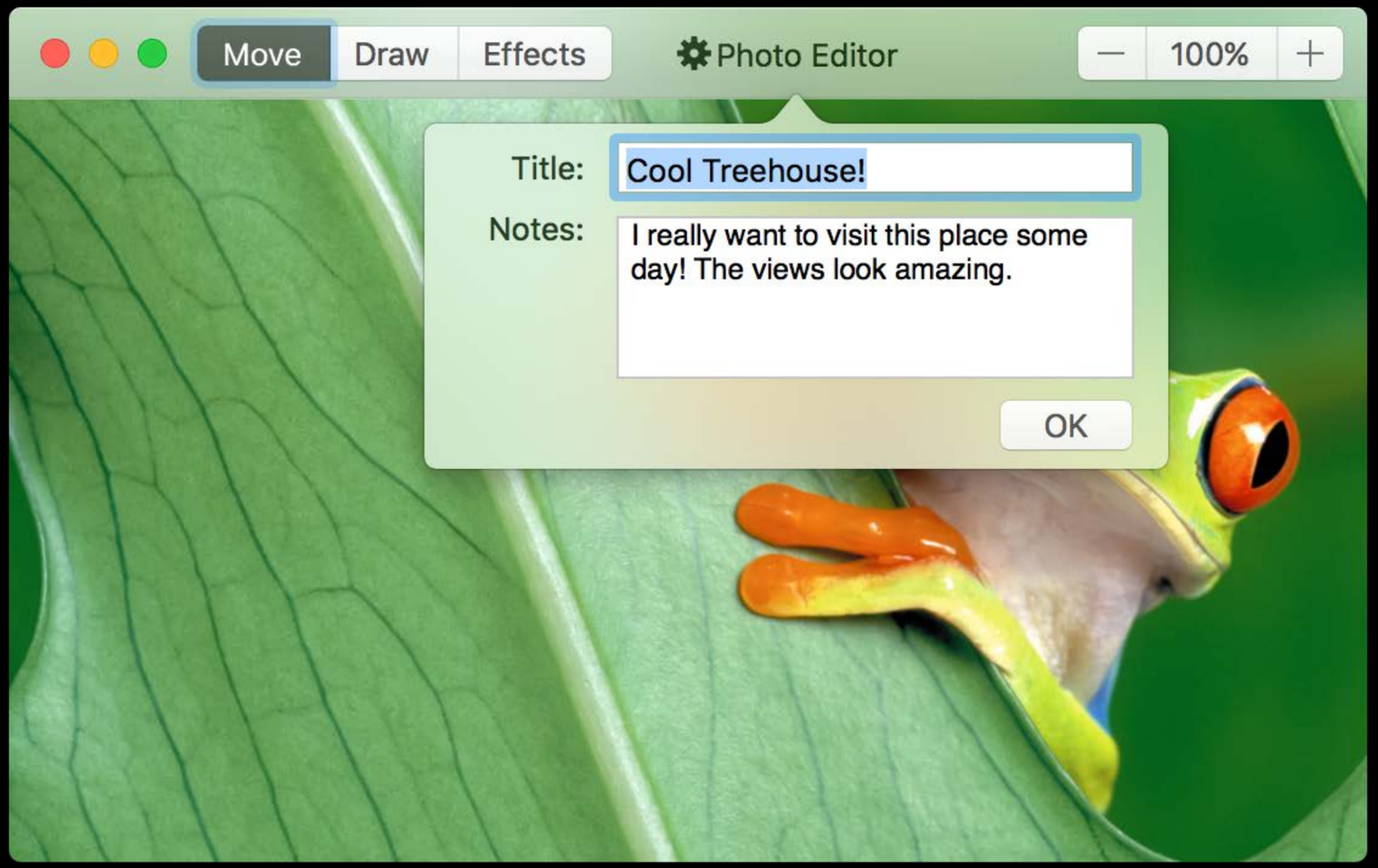
Modern Drag & Drop, and Event Tracking

Event tracking



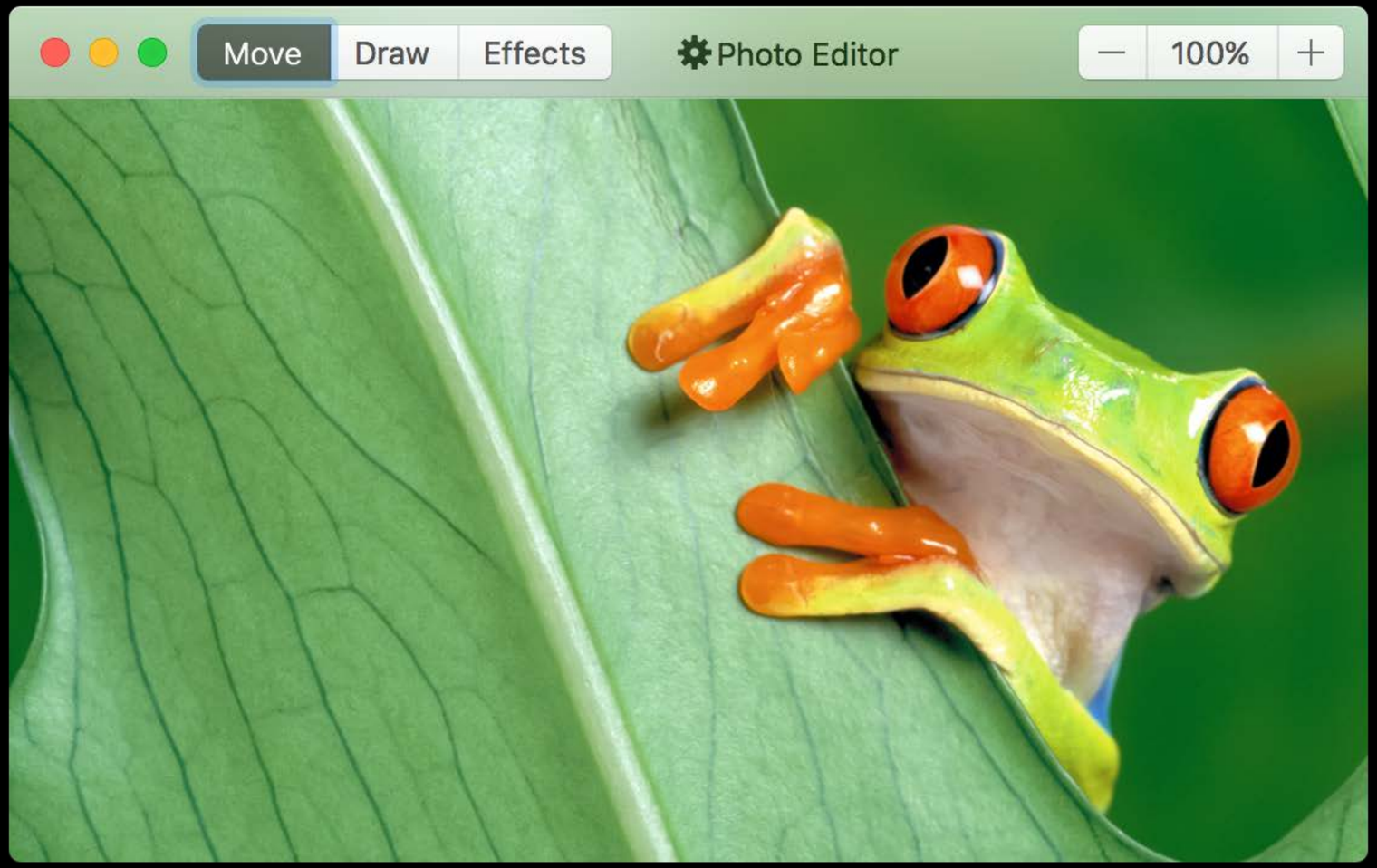
Modern Drag & Drop, and Event Tracking

Event tracking



Modern Drag & Drop, and Event Tracking

Event tracking



Modern Drag & Drop, and Event Tracking

Event tracking

Event Tracking

Two ways to pull events

Prefer to use

```
extension NSWindow {
  public func trackEvents(matching: NSEventMask, timeout: NSTimeInterval,
       mode: String, handler: (NSEvent, UnsafeMutablePointer<ObjCBool>) -> Void)
}
```

Instead of

```
extension NSApp {
   public func nextEvent(matching: NSEventMask, until: NSDate?,
                         inMode: String, dequeue: Bool) -> NSEvent?
}
```

Event Tracking

Prefer to use NSWindow.trackEvents()

```
override func mouseDown(_ mouseDownEvent: NSEvent) {
   var shouldCallSuper = false
   window.trackEvents(matching: [.leftMouseDragged, .leftMouseUp], timeout:...) {
                              (event: NSEvent?, stop: UnsafeMutablePointer<ObjCBool>) in
      if event.type == .leftMouseUp {
         stop.pointee = true
         shouldCallSuper = true
      } else if event.type == .leftMouseDragged {
          if (weMovedFarEnough) {
              stop.pointee = true
              window.performDrag(with: event)
          }
      }
```

Event Tracking

Prefer to use NSWindow.trackEvents()

```
override func mouseDown(_ mouseDownEvent: NSEvent) {
   var shouldCallSuper = false
   window.trackEvents(matching: [.leftMouseDragged, .leftMouseUp], timeout:...) {
                              (event: NSEvent?, stop: UnsafeMutablePointer<ObjCBool>) in
      if event.type == .leftMouseUp {
         stop.pointee = true
         shouldCallSuper = true
      } else if event.type == .leftMouseDragged {
          if (weMovedFarEnough) {
              stop.pointee = true
              window.performDrag(with: event)
          }
      }
```

Event Tracking

Prefer to use NSWindow.trackEvents()

```
override func mouseDown(_ mouseDownEvent: NSEvent) {
   var shouldCallSuper = false
   window.trackEvents(matching: [.leftMouseDragged, .leftMouseUp], timeout:...) {
                              (event: NSEvent?, stop: UnsafeMutablePointer<ObjCBool>) in
      if event.type == .leftMouseUp {
         stop.pointee = true
         shouldCallSuper = true
      } else if event.type == .leftMouseDragged {
          if (weMovedFarEnough) {
              stop.pointee = true
              window.performDrag(with: event)
          }
      }
```

Event Tracking

Prefer to use NSWindow.trackEvents()

```
override func mouseDown(_ mouseDownEvent: NSEvent) {
   var shouldCallSuper = false
   window.trackEvents(matching: [.leftMouseDragged, .leftMouseUp], timeout:...) {
                              (event: NSEvent?, stop: UnsafeMutablePointer<ObjCBool>) in
      if event.type == .leftMouseUp {
         stop.pointee = true
         shouldCallSuper = true
      } else if event.type == .leftMouseDragged {
          if (weMovedFarEnough) {
              stop.pointee = true
              window.performDrag(with: event)
          }
      }
```

Event Tracking

Prefer to use NSWindow.trackEvents()

```
override func mouseDown(_ mouseDownEvent: NSEvent) {
   var shouldCallSuper = false
   window.trackEvents(matching: [.leftMouseDragged, .leftMouseUp], timeout:...) {
                              (event: NSEvent?, stop: UnsafeMutablePointer<ObjCBool>) in
      if event.type == .leftMouseUp {
         stop.pointee = true
         shouldCallSuper = true
      } else if event.type == .leftMouseDragged {
          if (weMovedFarEnough) {
              stop.pointee = true
              window.performDrag(with: event)
          }
      }
```

Moving Windows / Window Dragging

Don’t manually drag a window by setting the frame

Instead pass window dragging off to the system with NSWindow’s

```
public func performDrag(event: NSEvent)
```

- Allows windows to move when the app is hung
- Space switching will work correctly
- Spaces Bar will properly drop down
- Window snapping/alignment works

What to Expect

Creating a Modern Look with Modern Views

Modern Drag & Drop, and Event Tracking

Container View Controls Handled Correctly

Using System Appearances

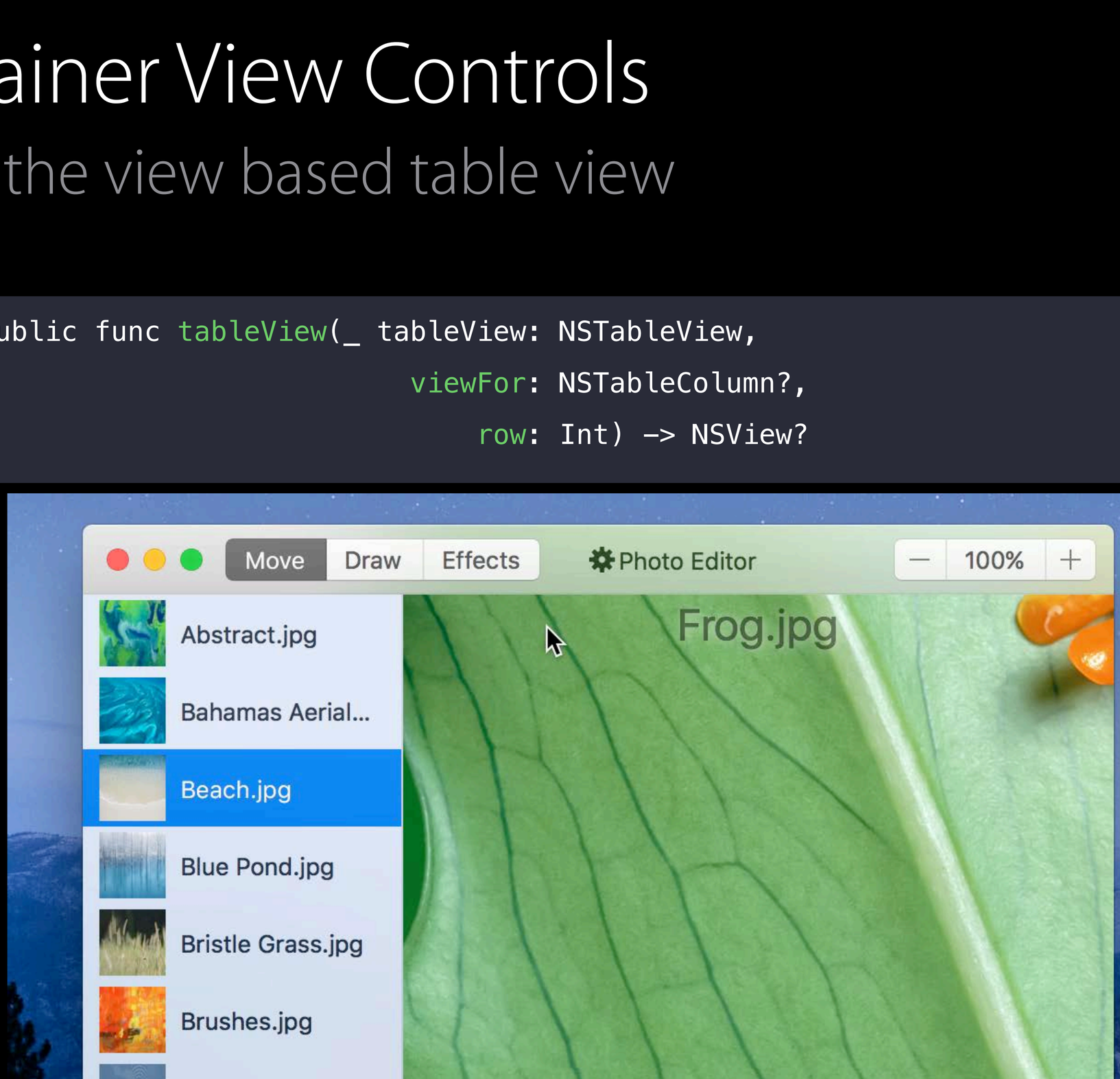
Designing with Storyboards

Modern Mac Features

Container View Controls

Adopt the view based table view

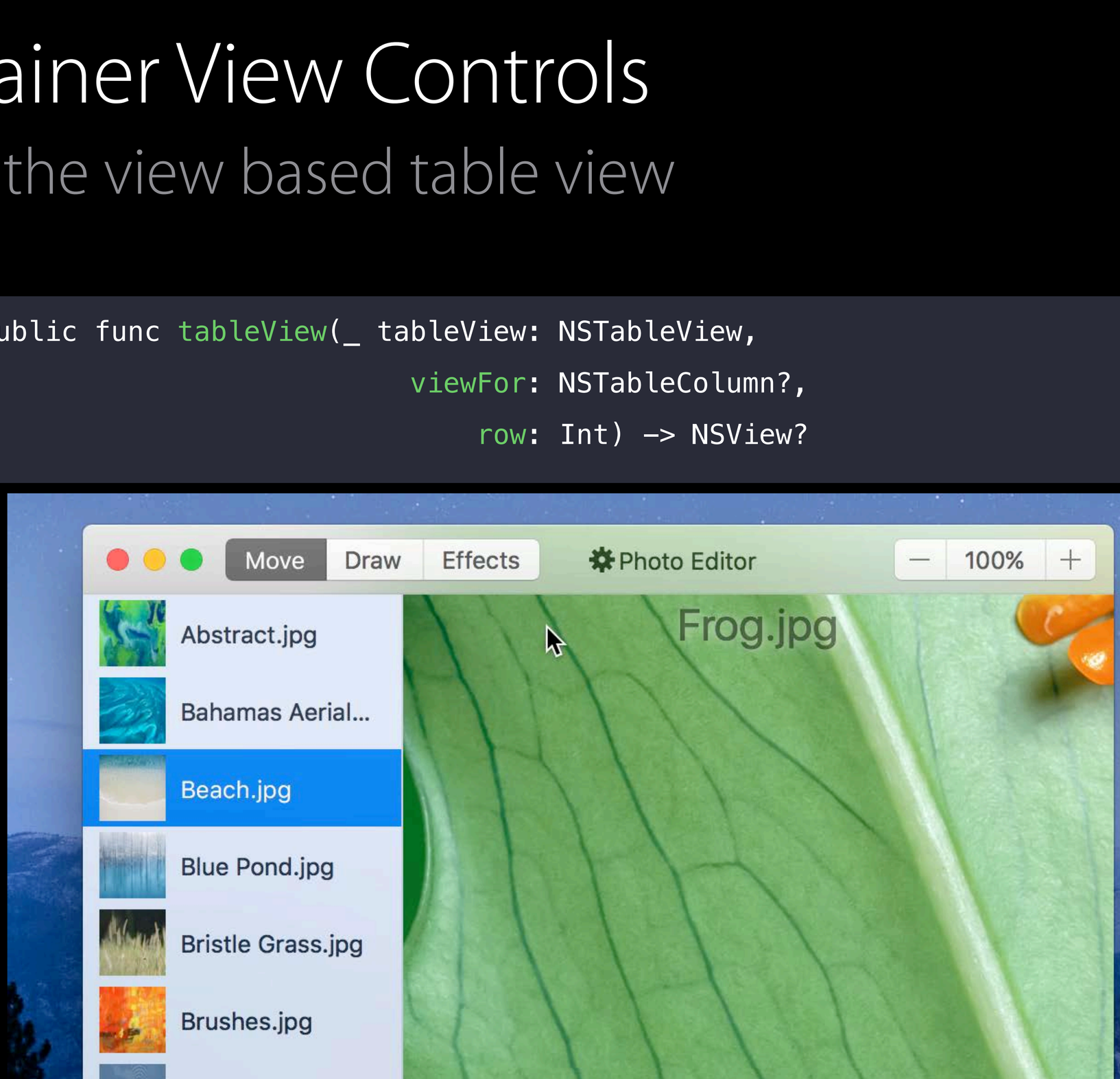
```
optional public func tableView(_ tableView: NSTableView,
                               viewFor: NSTableColumn?,
                                   row: Int) -> NSView?
```



Container View Controls

Adopt the view based table view

```
optional public func tableView(_ tableView: NSTableView,
                               viewFor: NSTableColumn?,
                                   row: Int) -> NSView?
```



View Based Table View

Swipe to delete

Implement the delegate method

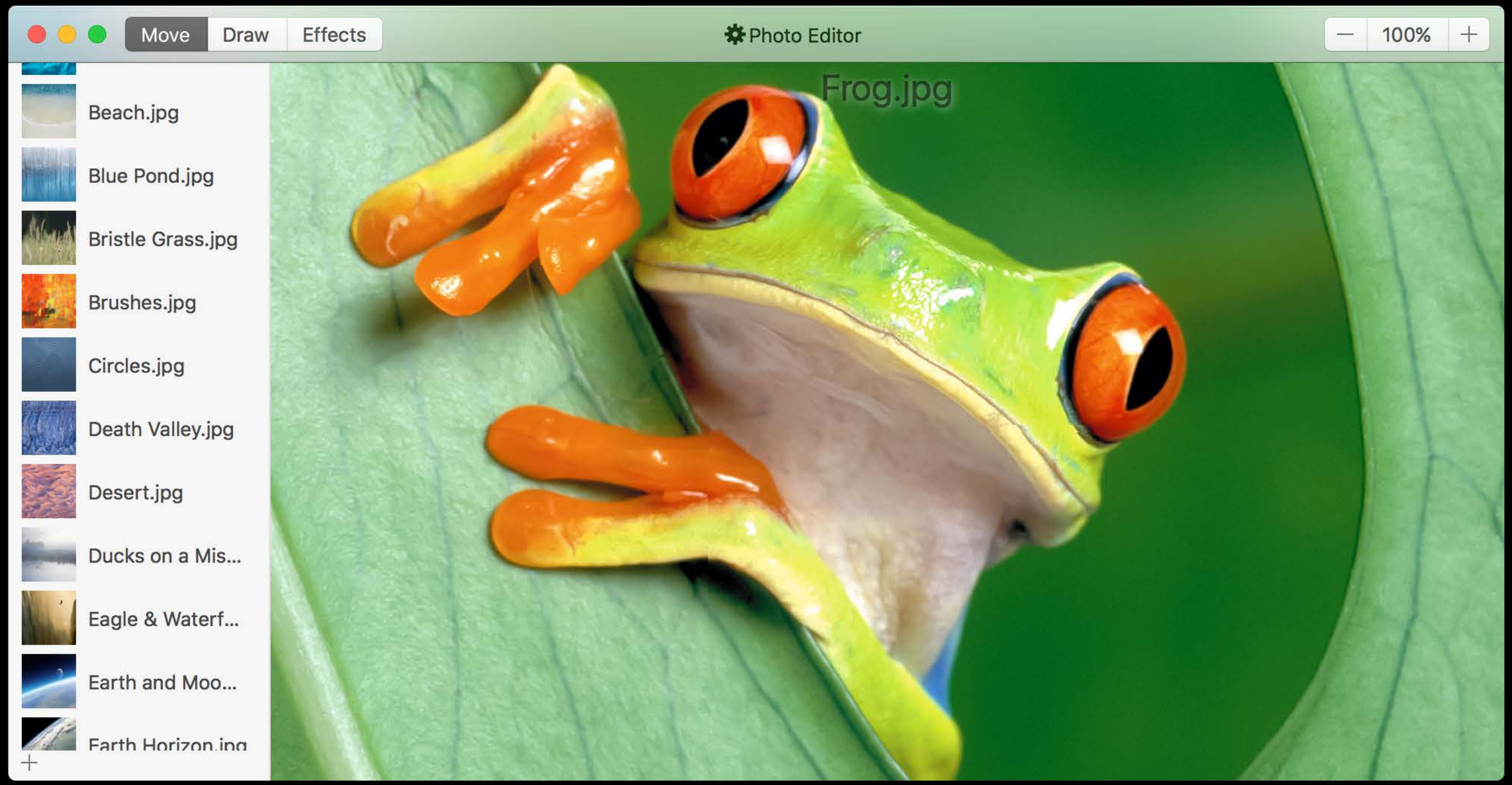
```
optional public func tableView(_ tableView: NSTableView,
                               rowActionsForRow: Int,
                                          edge: NSTableRowActionEdge) -> [NSTableViewRowAction]
```

using NSTableRowAction

```
public class NSTableViewRowAction : NSObject, NSCopying {
      public convenience init(style: NSTableViewRowActionStyle,
                                title: String,
                              handler: (NSTableViewRowAction, Int) -> Void)
... }
```

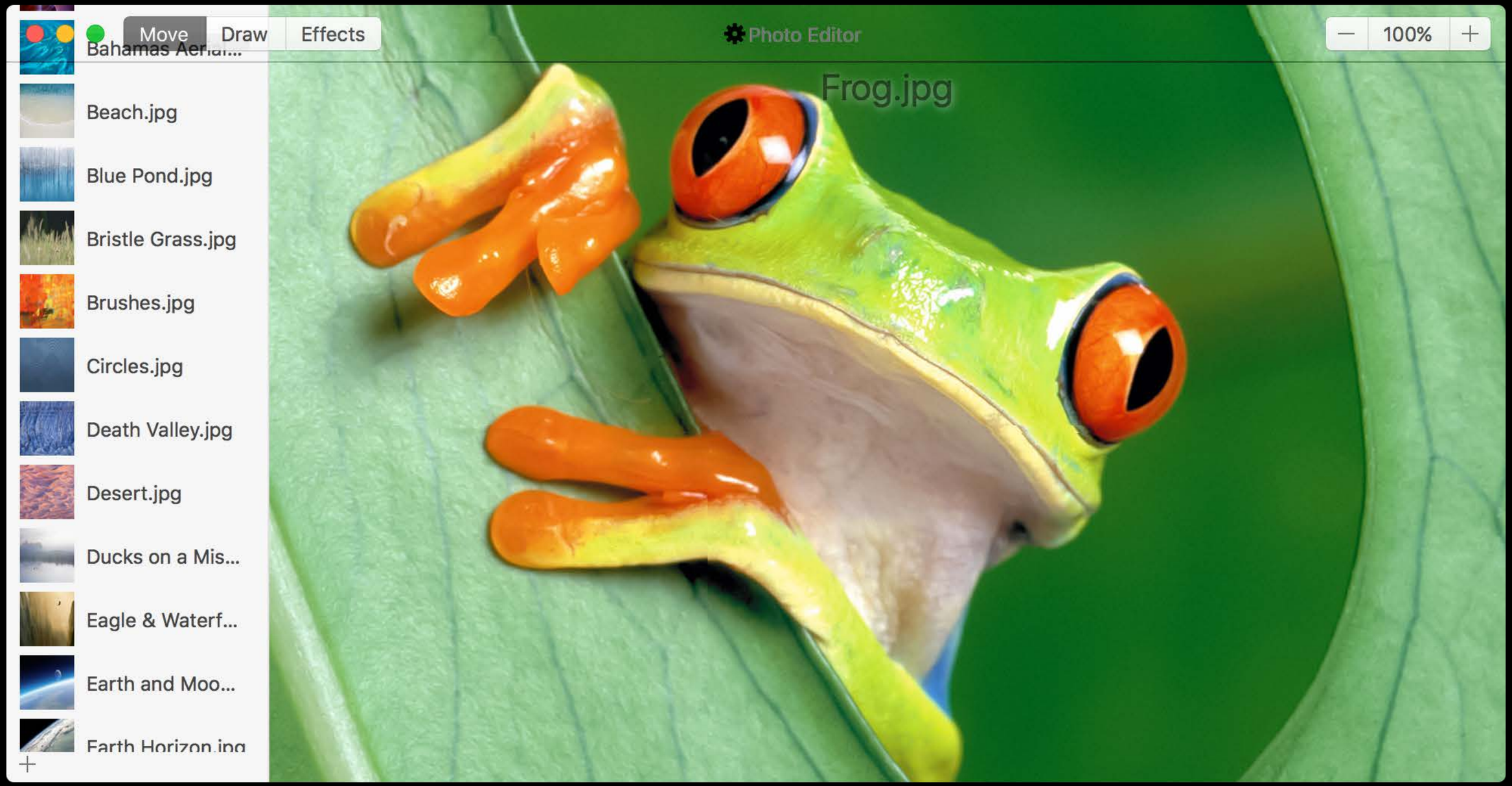
Container View Controls—NSScrollView

Insetting NSScrollView's starting offset



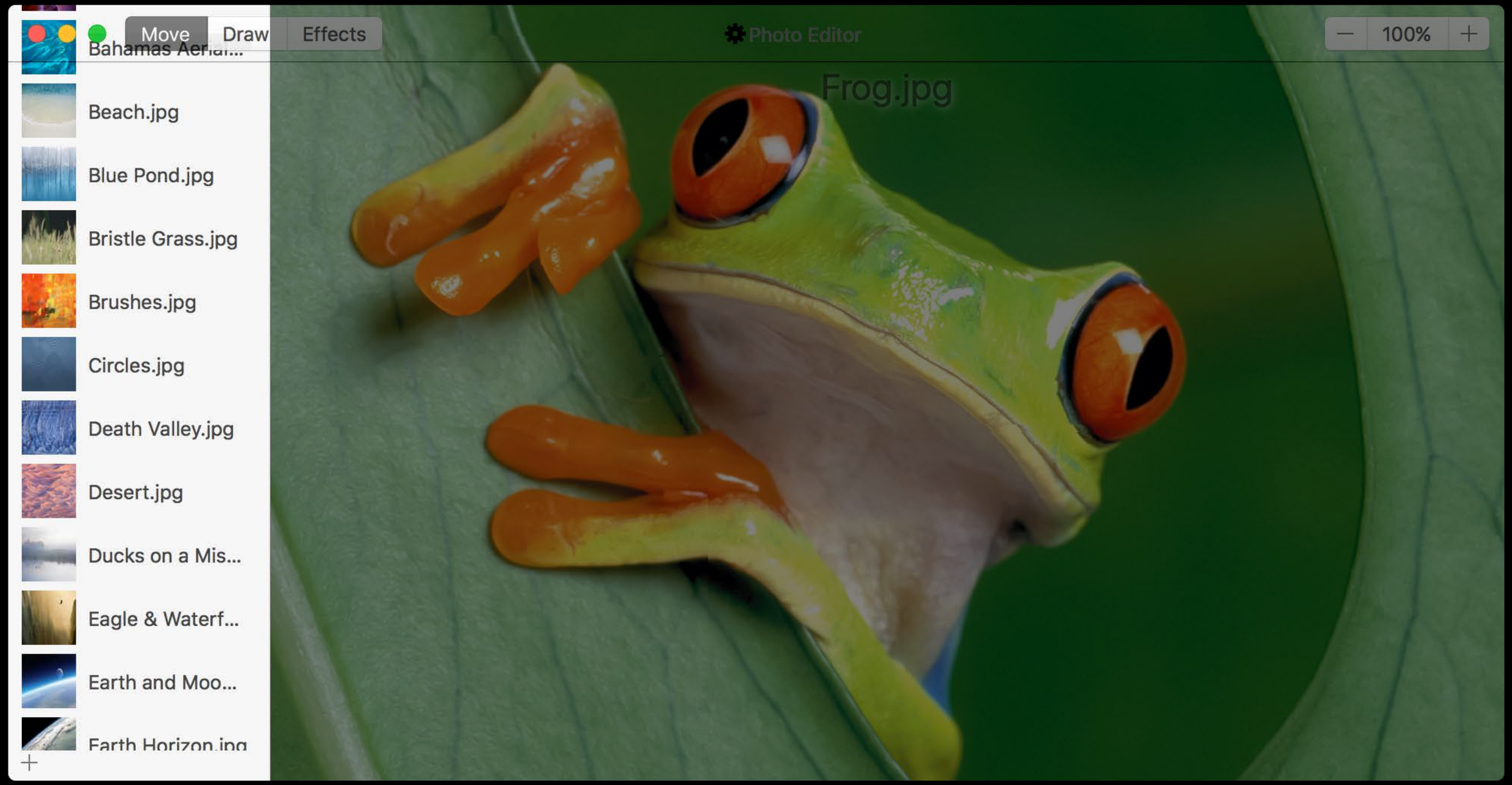
Container View Controls—NSScrollView

Insetting NSScrollView's starting offset



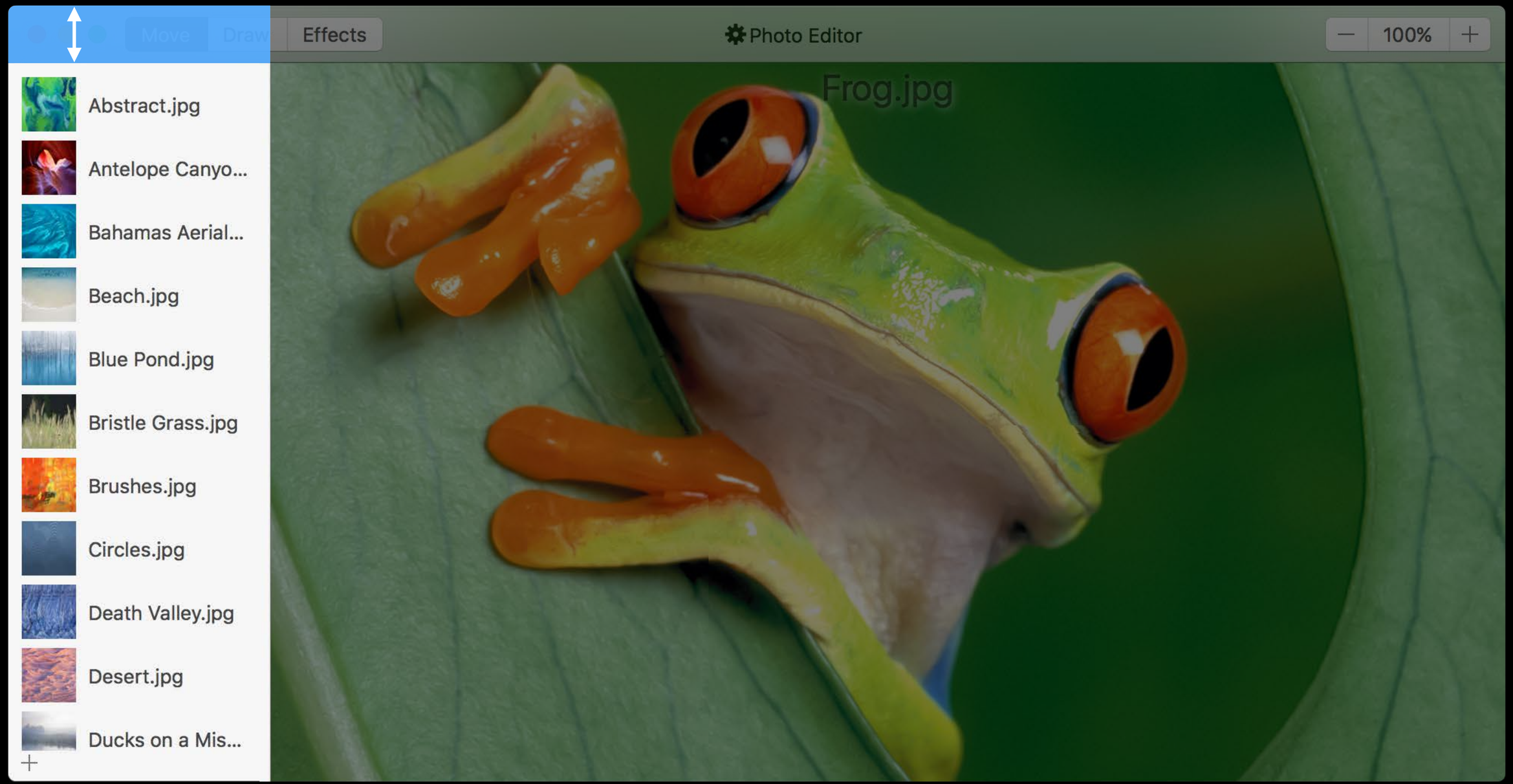
Container View Controls—NSScrollView

Insetting NSScrollView's starting offset



Container View Controls—NSScrollView

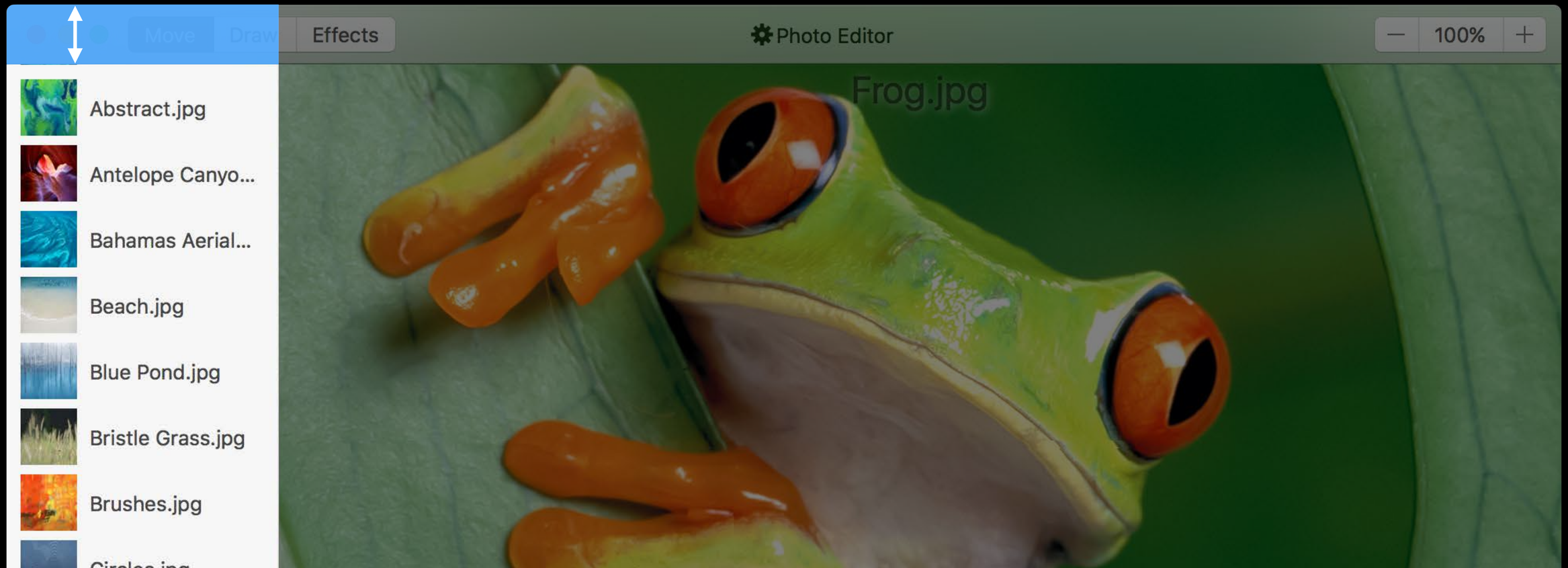
Insetting NSScrollView's starting offset



Container View Controls—NSScrollView

Insetting NSScrollView's starting offset

```
public class NSScrollView : NSView {
    public var contentInsets: NSEdgeInsets
    public var automaticallyAdjustsContentInsets: Bool
```

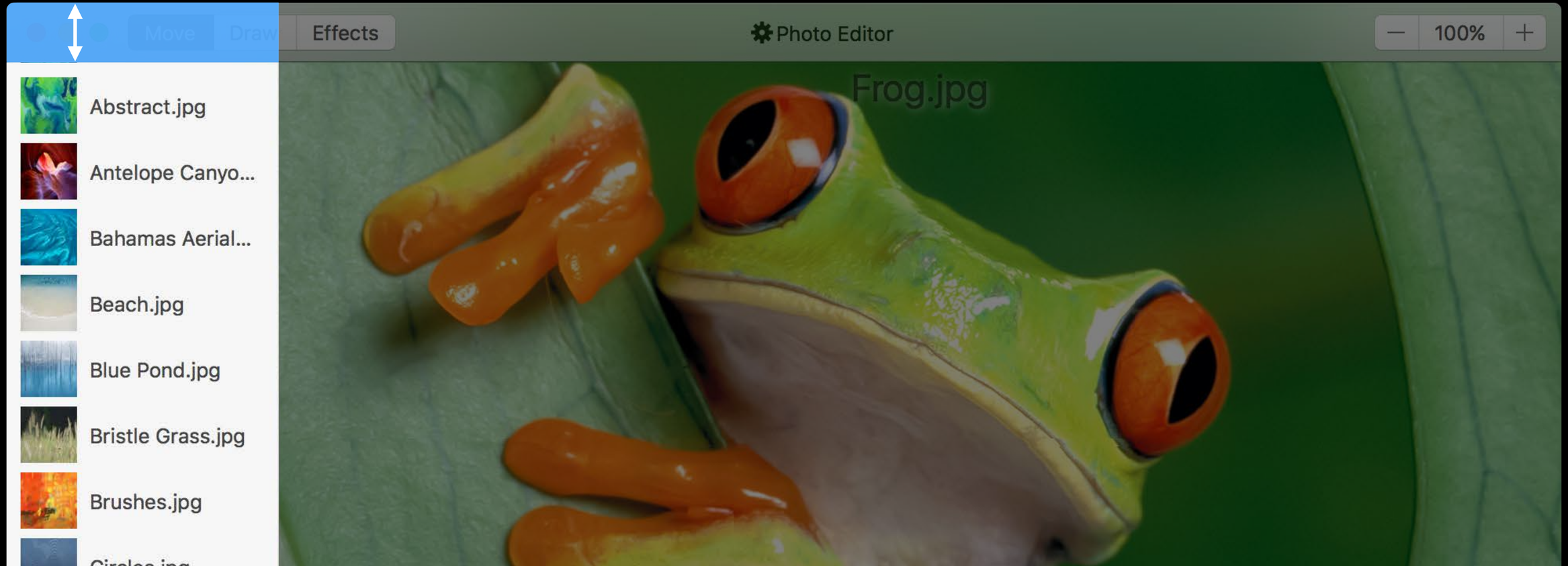


Container View Controls—NSScrollView

Insetting NSScrollView's starting offset

automaticallyAdjustsContentInsets tracks the contentLayoutRect

Scroll views will automatically start below the titlebar/toolbar area



Container View Controls—NSScrollView

Insetting NSScrollView's starting offset

Manual control can be used to add extra accessories before some content

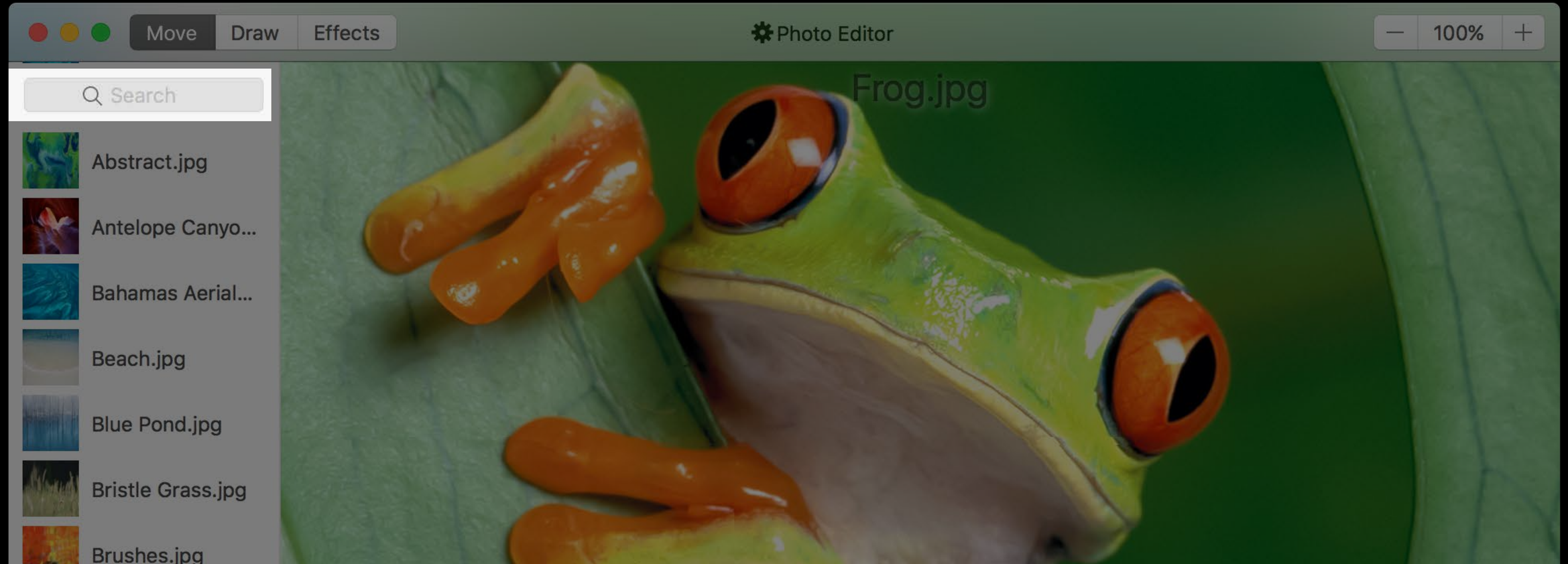
- Sort indicator view such as shown in Mail
- Search field, refresh button, etc

Container View Controls—NSScrollView

Insetting NSScrollView's starting offset

Manual control can be used to add extra accessories before some content

- Sort indicator view such as shown in Mail
- Search field, refresh button, etc



Controls and Localization

Use Auto Layout and Base.lproj

Enable Base Internationalization

- Use one (or more) Storyboards in Base.lproj instead of multiple localized copies
- Use language specific strings files

Don’t use fixed width constraints

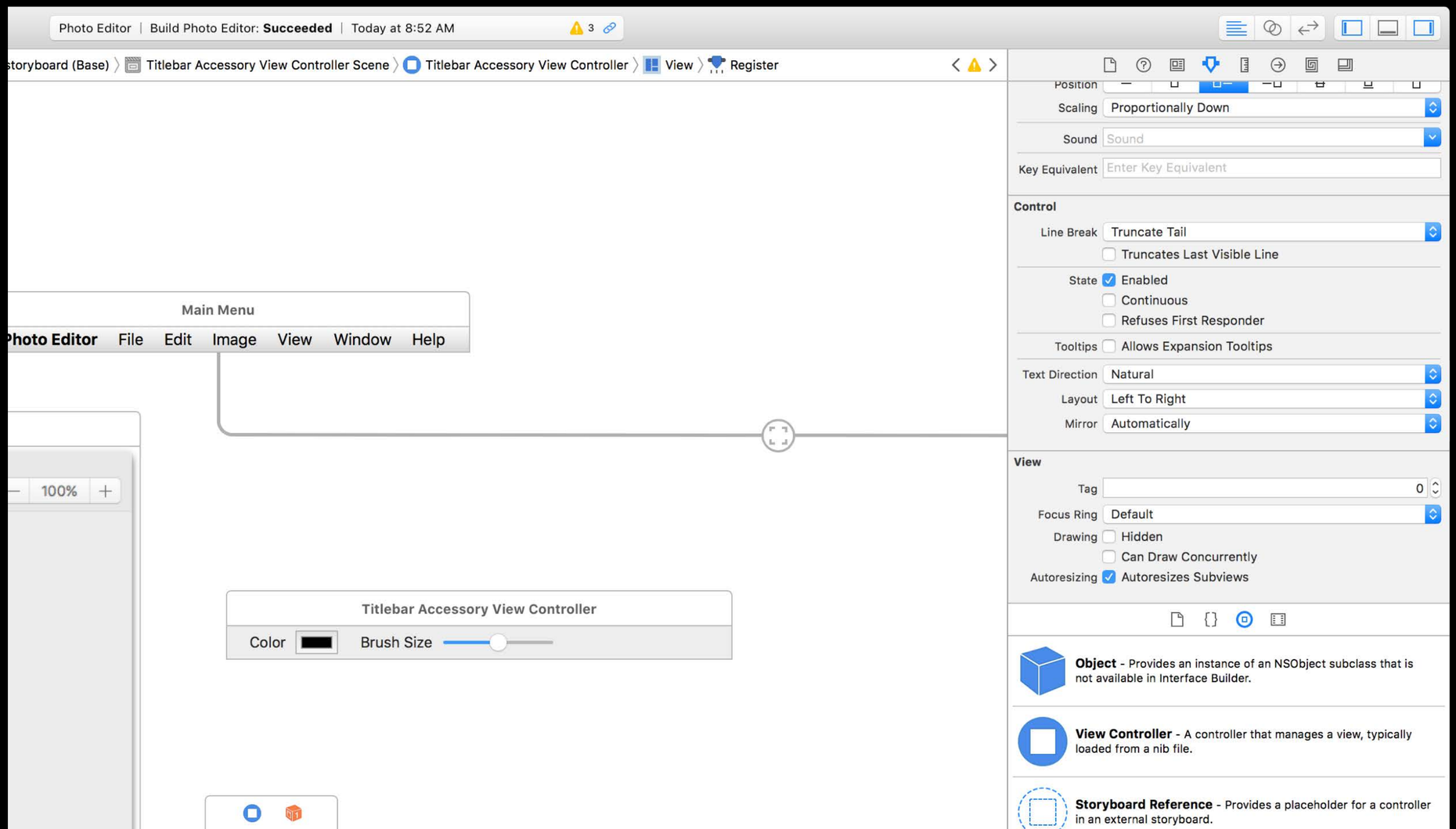
- Instead, use intrinsic content sizes

Use leading and trailing attributes

Utilize NSStackView to pin views together

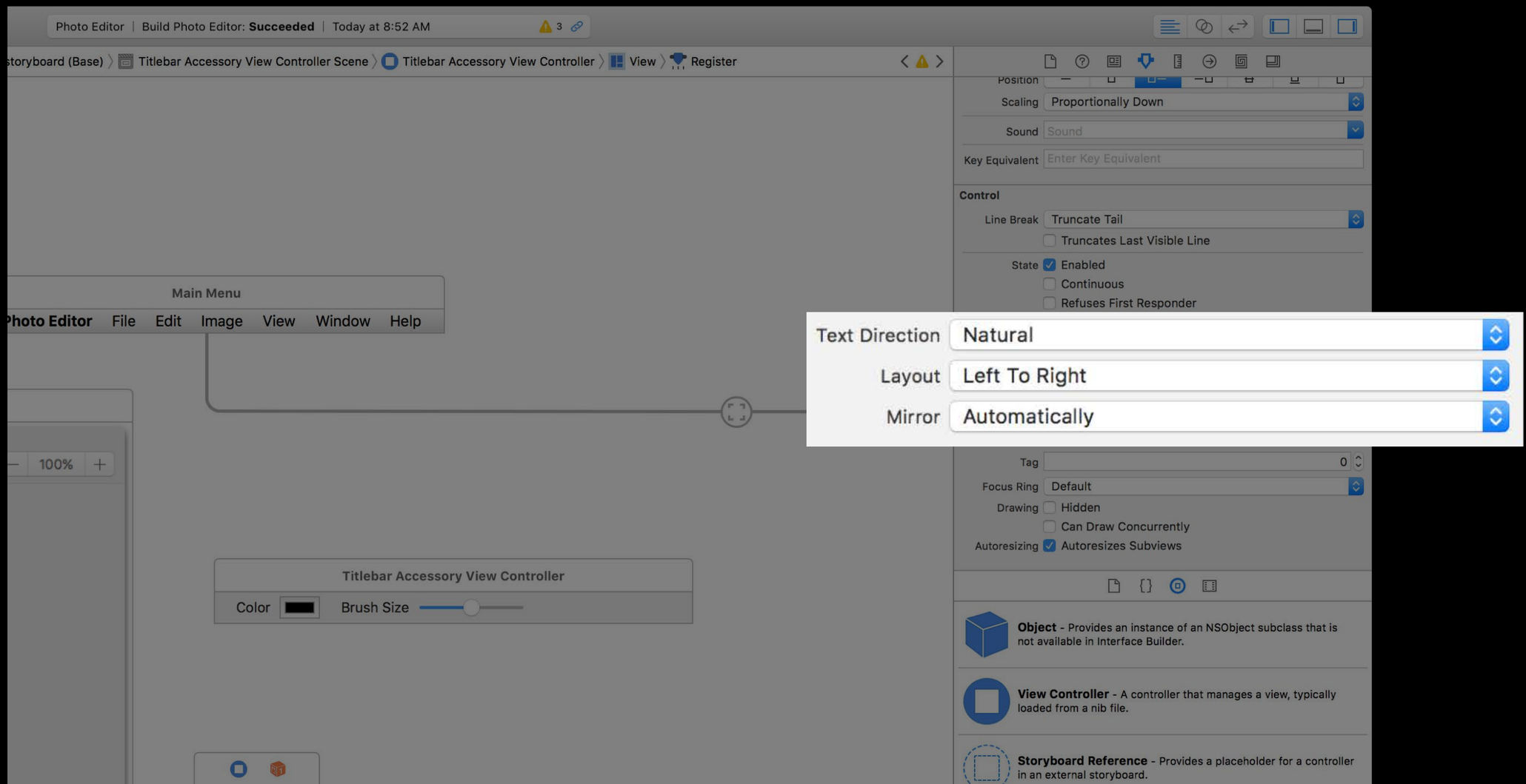
Localization in IB

Text, Layout, and Mirror



Localization in IB

Text, Layout, and Mirror

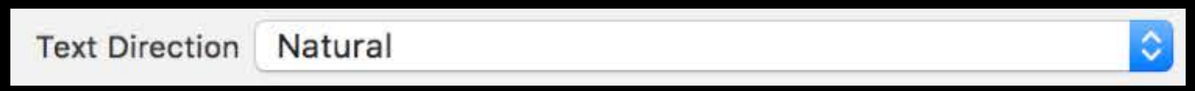


Localization in IB

Text direction

Text direction options

- Natural—the text layout in the control is based on the string assigned to the control
- Left to Right—the text layout is always left to right
- Right to Left—the text layout is always right to left



Localization in IB

View/Control layout

Layout refers to the **`NSView userInterfaceLayoutDirection`** property

```
NSUserInterfaceLayoutDirection.leftToRight
NSUserInterfaceLayoutDirection.rightToLeft
```

Normally you should not change this unless you always want a specific alignment

Most all controls support `.rightToLeft` by examining the property

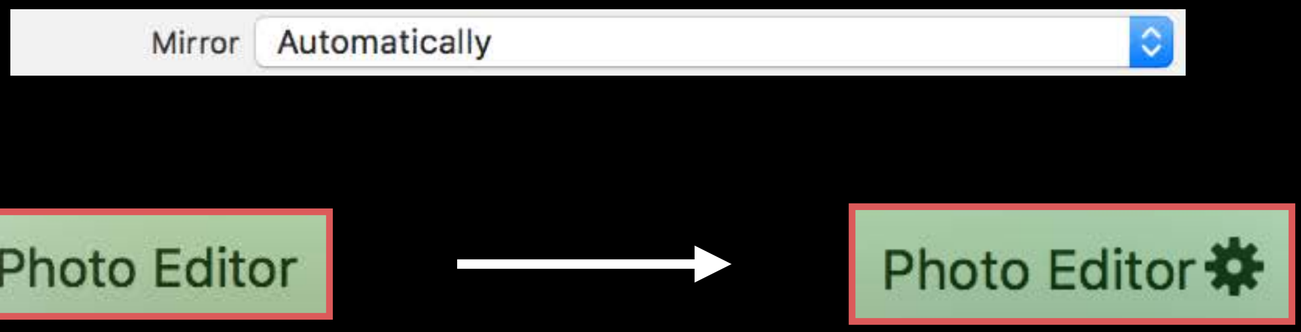
The default value is the NSApp value, which is based on the app localization being used

Localization in IB

Mirroring

Mirror tells the system to flip certain control properties

- NSUserInterfaceLayoutDirection, NSTextAlignment, NSCellImagePosition
  - Left goes to Right, and Right goes to Left
  - Example of the image position on a button mirroring

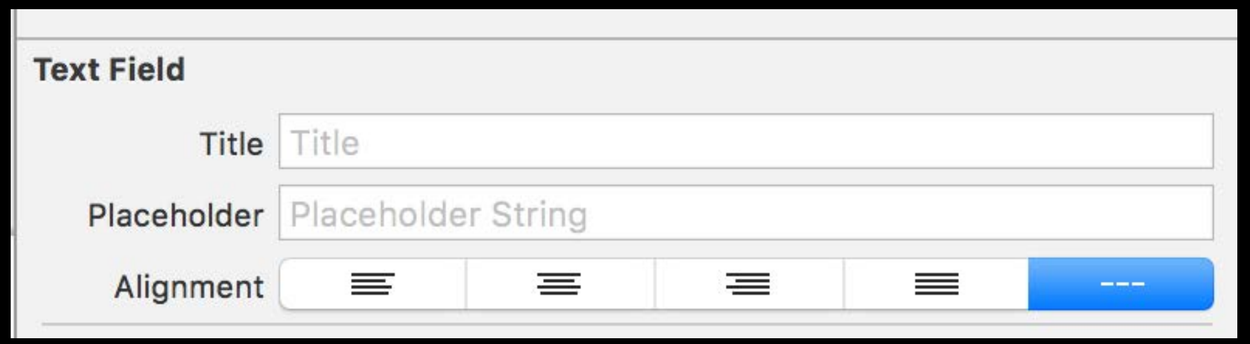


Localization in IB

Mirroring

Mirror tells the system to flip certain control properties, however

- NSTextAlignment will not mirror when it is set to—Center, Justified, or Natural

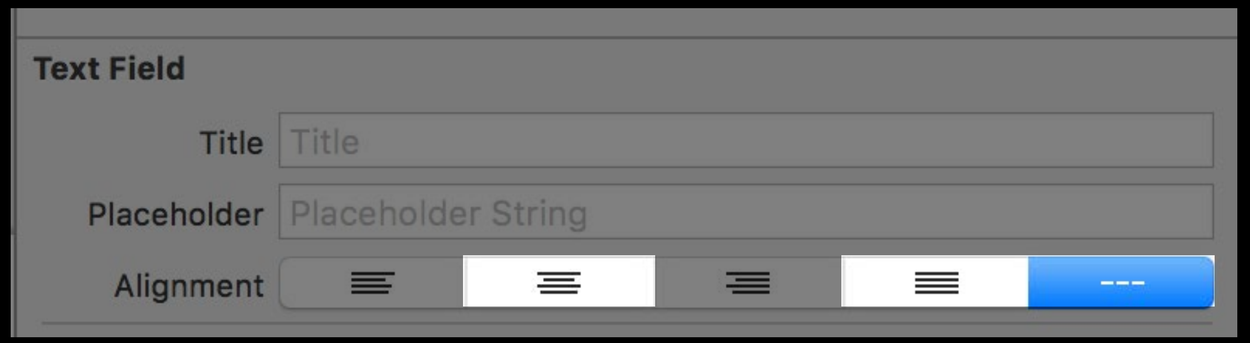


Localization in IB

Mirroring

Mirror tells the system to flip certain control properties, however

- NSTextAlignment will not mirror when it is set to—Center, Justified, or Natural



Localization in IB

Mirroring

Normally you won't ever set this unless you always require a specific layout

- Example—A play button that should always follow the physical representation

Localization in Code

Manually mirroring

Views created in code default to the NSApp userInterfaceLayoutDirection

Mirroring is not automatic, so you must do this

```
let button = NSButton(frame: buttonFrame)
if button.userInterfaceLayoutDirection == .leftToRight {
  button.alignment = .right
  button.imagePosition = .imageLeft
} else {
   button.alignment = .left
   button.imagePosition = .imageRight
}
```

Localization in Code

Manually mirroring

Views created in code default to the NSApp userInterfaceLayoutDirection

Mirroring is not automatic, so you must do this

```
let button = NSButton(frame: buttonFrame)
if button.userInterfaceLayoutDirection == .leftToRight {
  button.alignment = .right
  button.imagePosition = .imageLeft
} else {
   button.alignment = .left
   button.imagePosition = .imageRight
}
```

Localization in Code

Manually mirroring

Views created in code default to the NSApp userInterfaceLayoutDirection

Mirroring is not automatic, so you must do this

```
let button = NSButton(frame: buttonFrame)
if button.userInterfaceLayoutDirection == .leftToRight {
  button.alignment = .right
  button.imagePosition = .imageLeft
} else {
   button.alignment = .left
   button.imagePosition = .imageRight
}
```

Localization in Code

Automatic mirroring

NEW

Or use the new convenience initializer in 10.12

```
extension NSButton {
   public convenience init(title: String, image: NSImage,
                           target: AnyObject?, action: Selector?)
}
```

Left to Right—Image is on the left, text is on the right

Right to Left—Image is on the right, text is on the left

Appearances, Storyboards, and Mac Features

Jeff Nadeau AppKit

What to Expect

Creating a Modern Look with Modern Views

Proper Drag & Drop, and Event Tracking

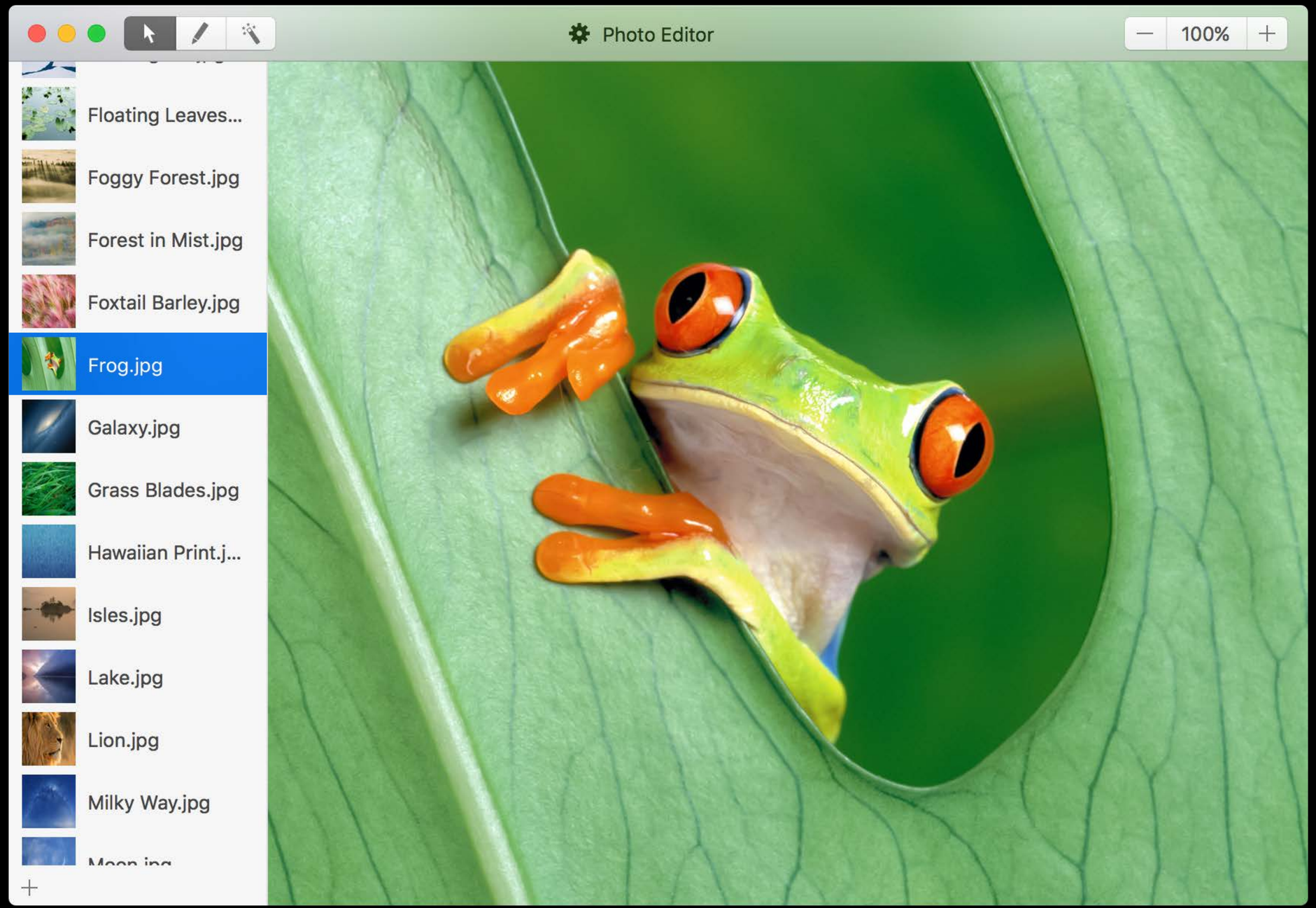
Complex Controls Handled Correctly

Using System Appearances

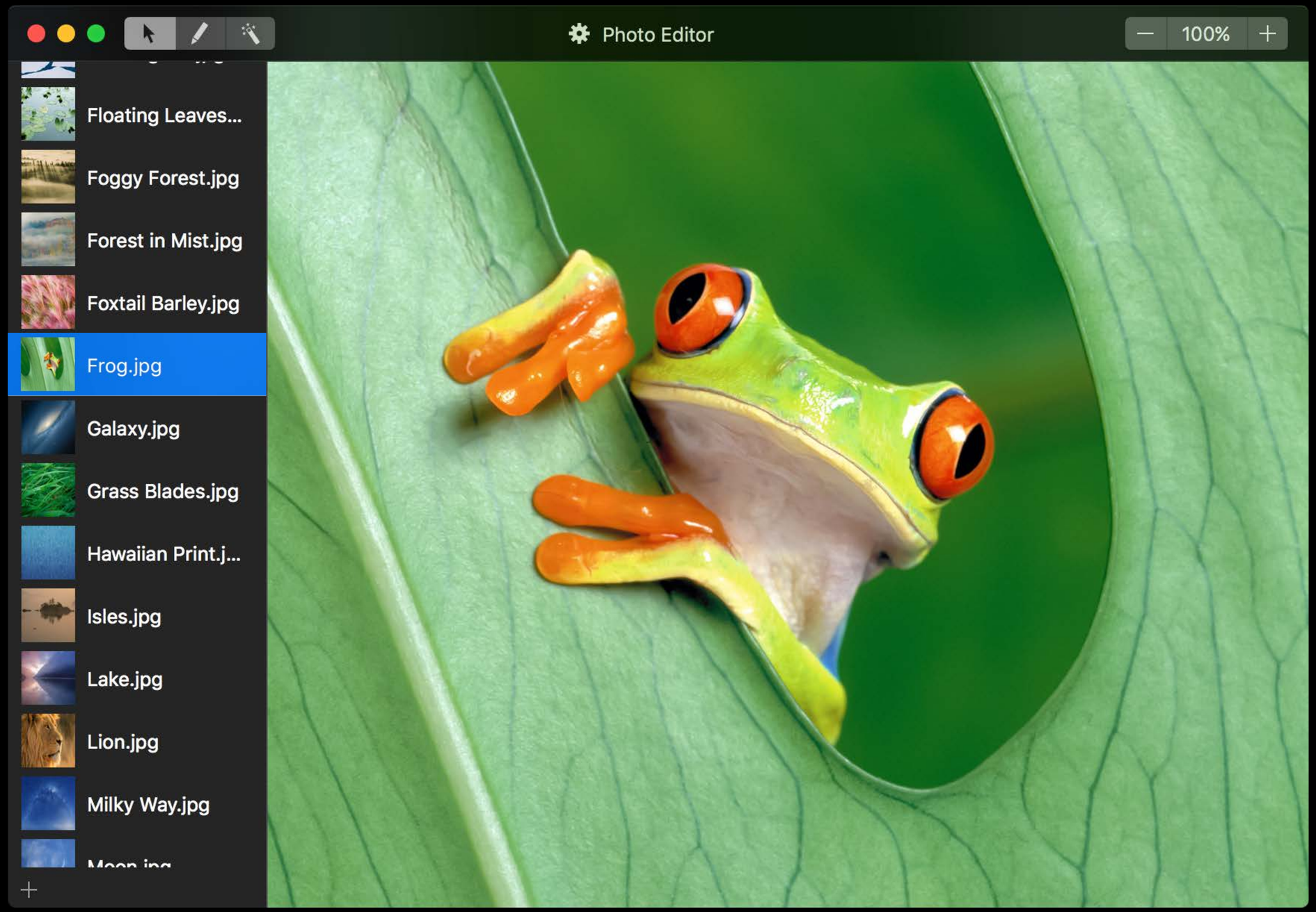
Designing with Storyboards

Modern Mac Features

System Appearances



System Appearances



System Appearances

Applying a built-in appearance

Simple to do with

```
window.appearance = NSAppearance(named: NSAppearanceNameVibrantDark)
```

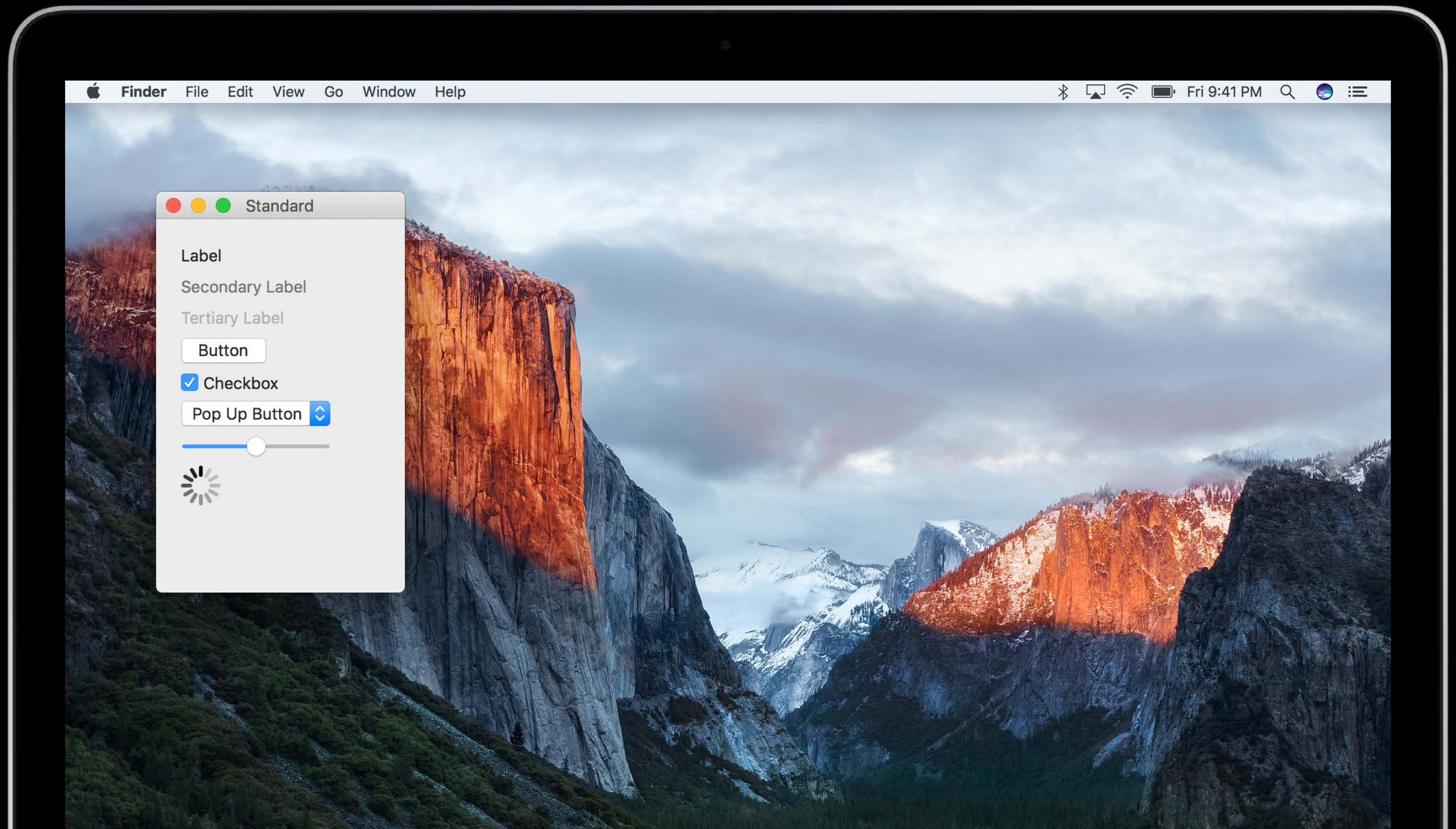
Prefer named colors over hardcoded values, e.g.

- labelColor
- selectedControlColor

Use system standard control styles when possible

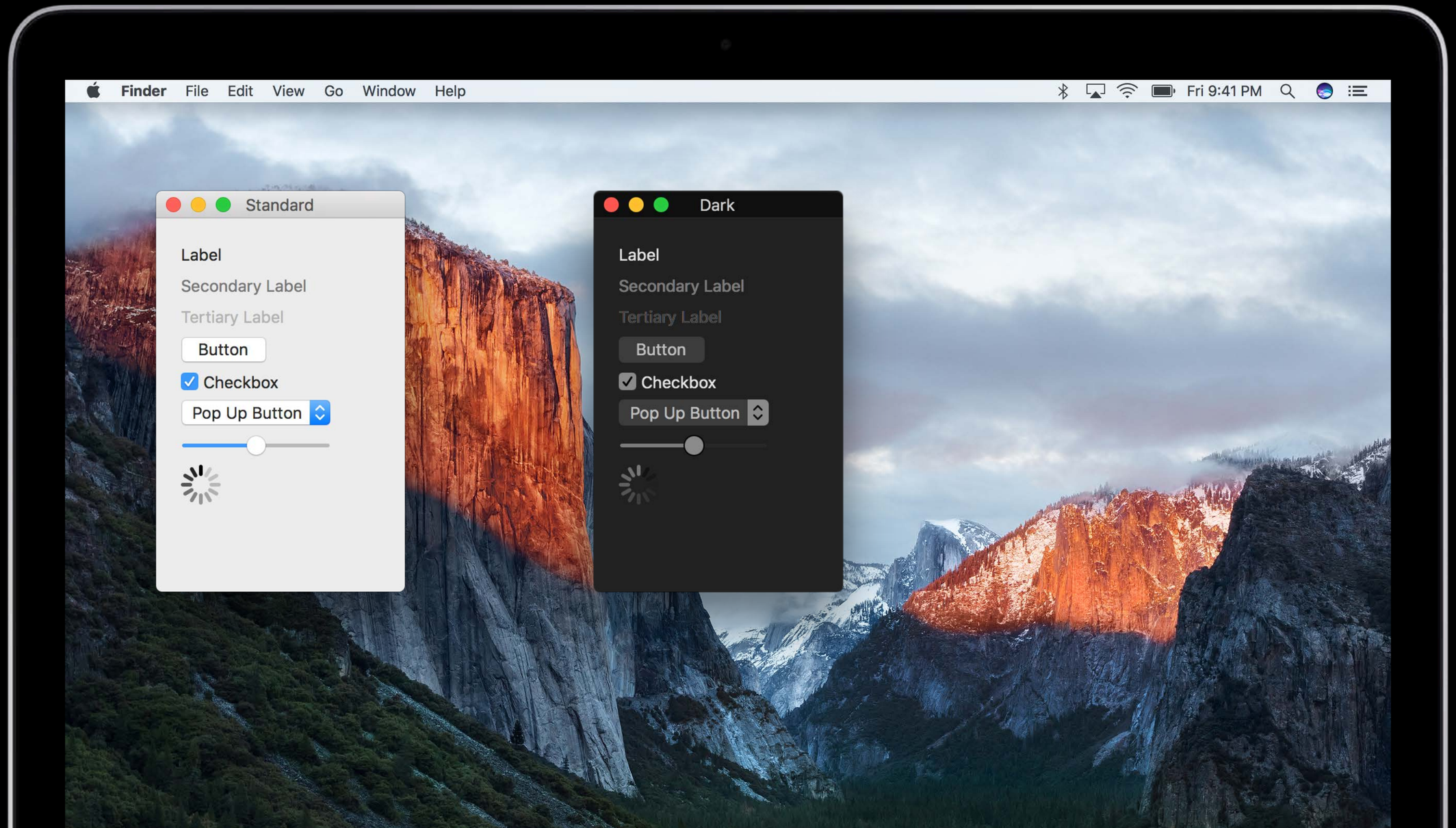
System Appearances

Standard



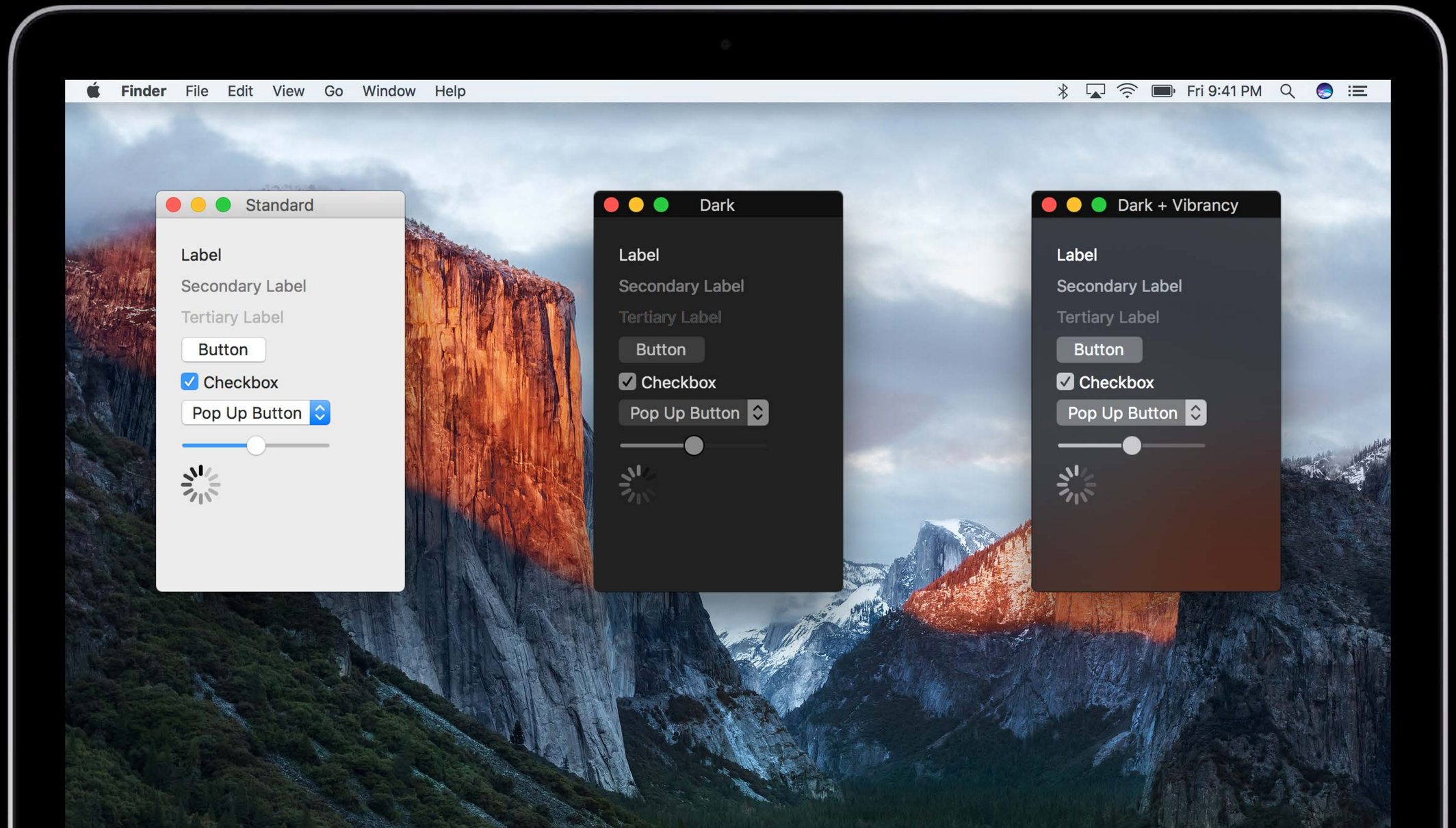
System Appearances

Standard



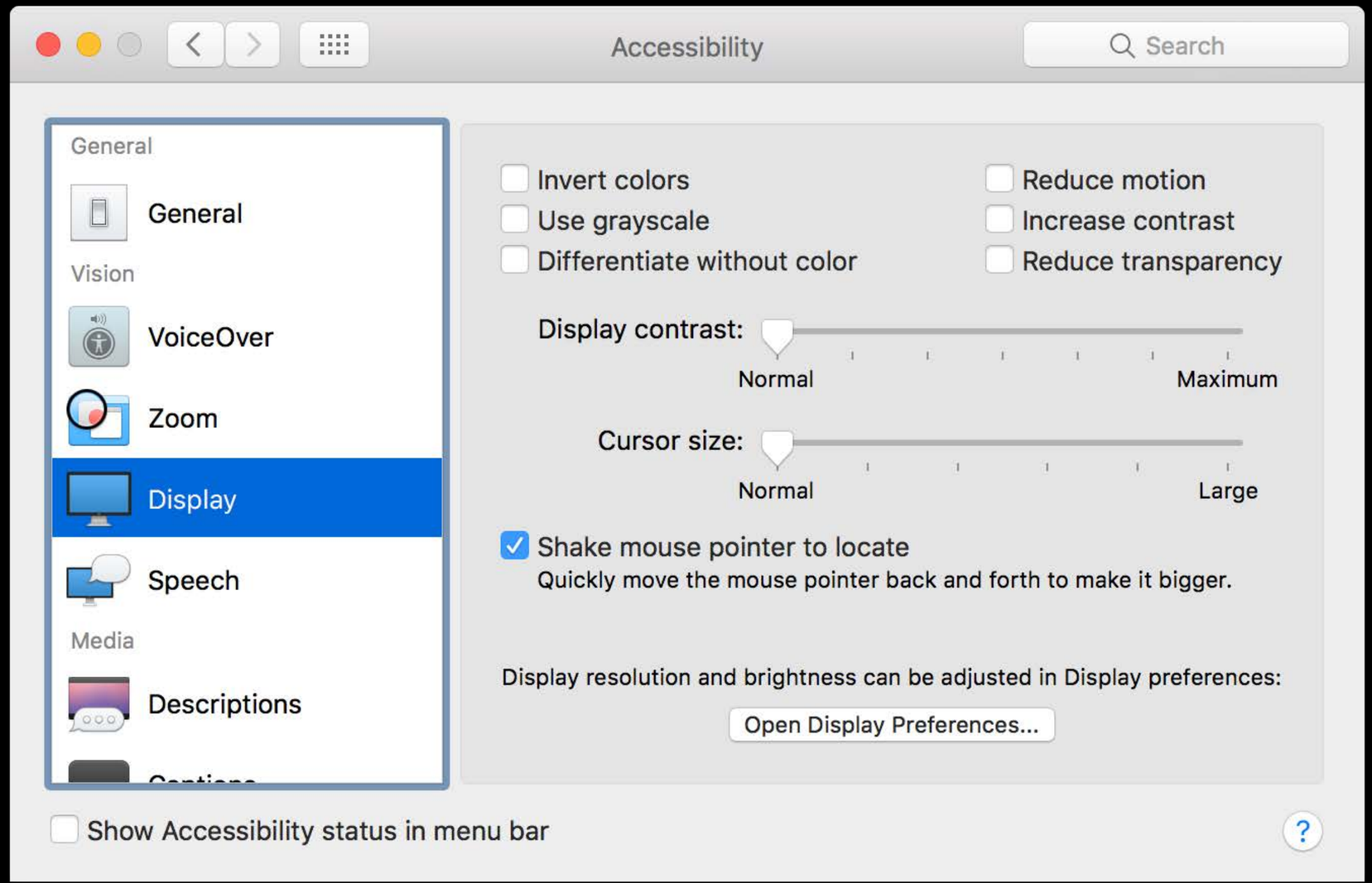
System Appearances

Standard



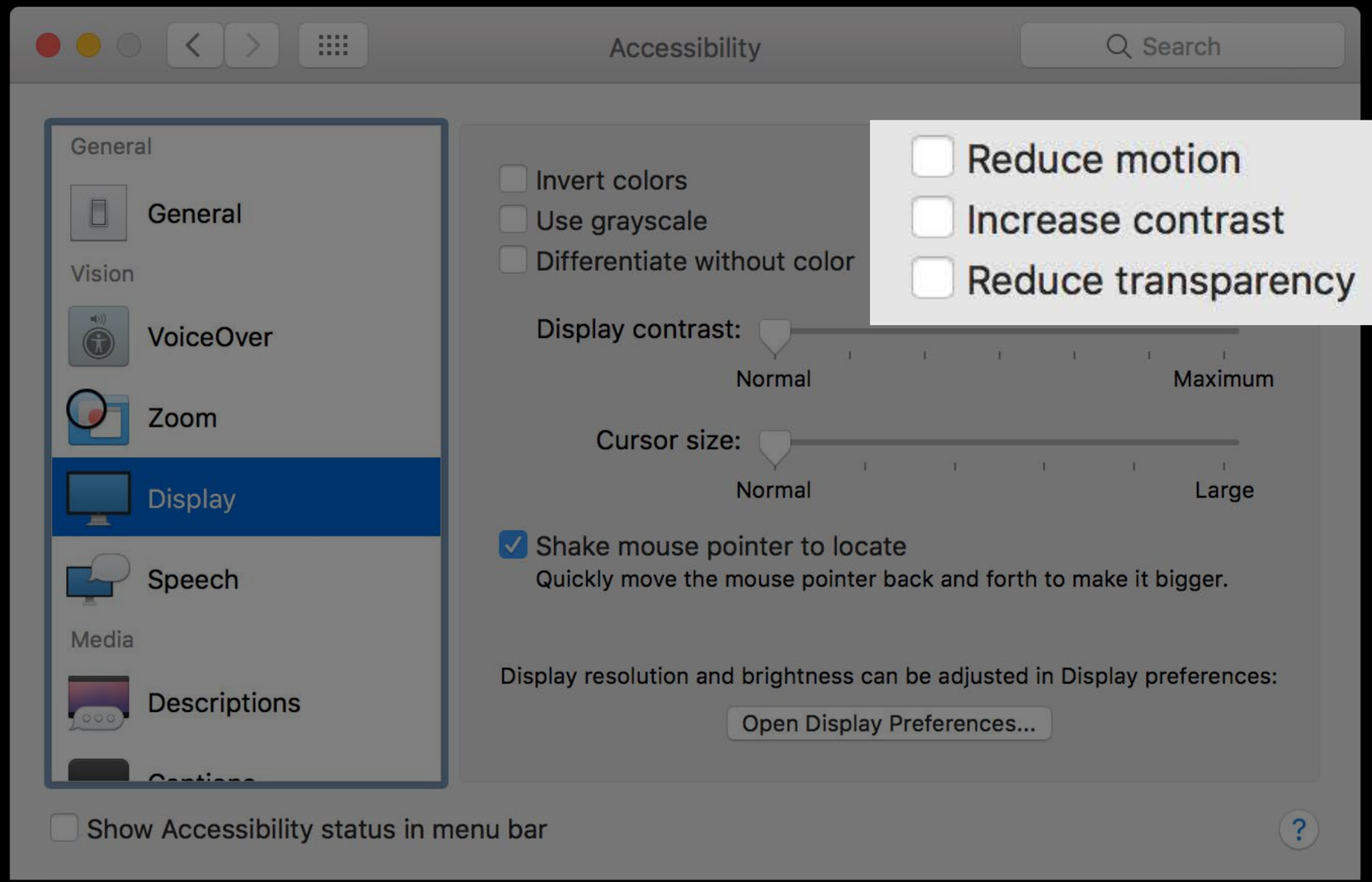
System Appearances

Increased contrast



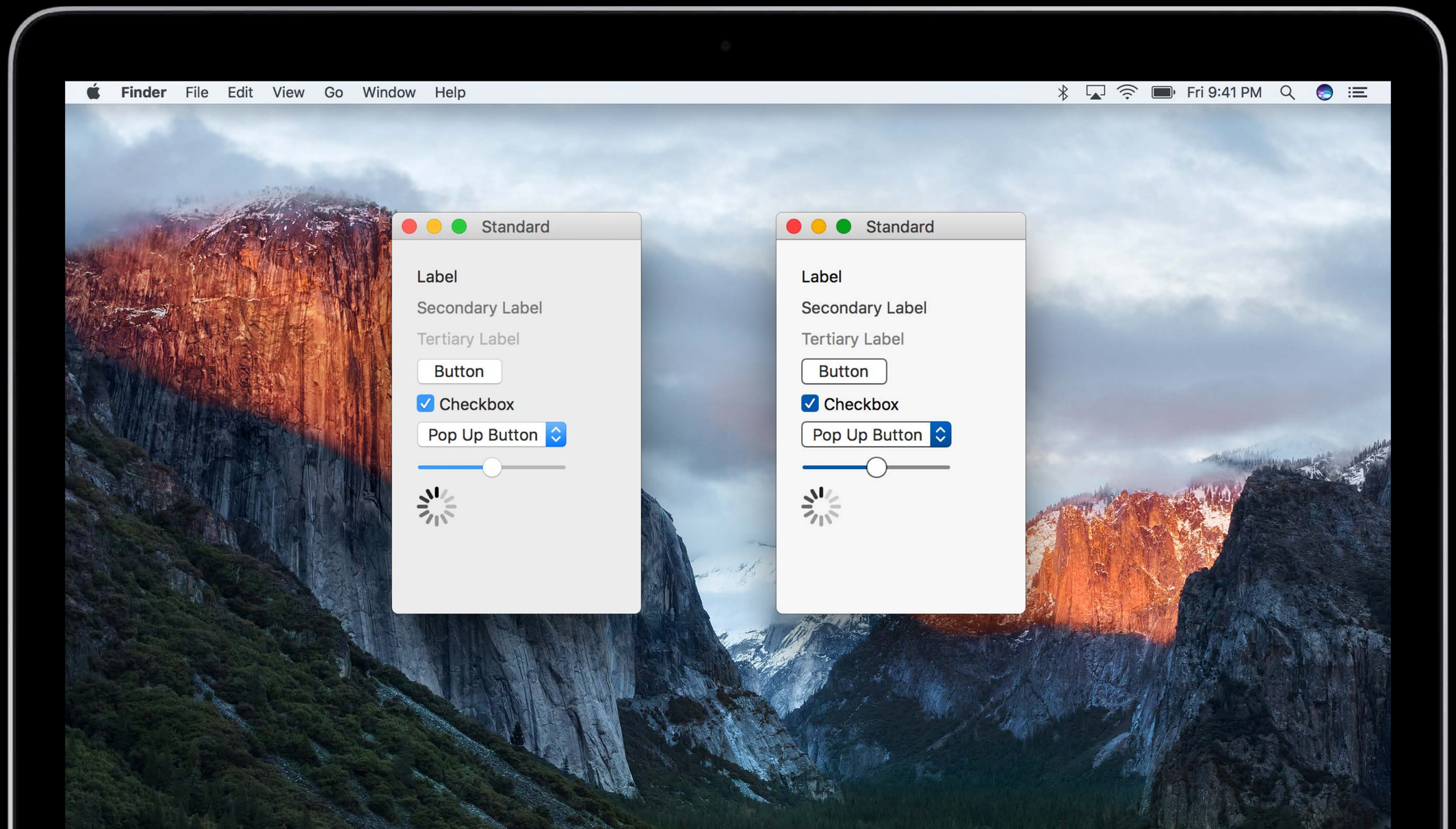
System Appearances

Increased contrast



System Appearances

Increased contrast



Visual Effect View, Materials, and Vibrancy

For more information

Adopting Advanced Features of the New UI of OS X Yosemite — WWDC 2014

What to Expect

Creating a Modern Look with Modern Views

Proper Drag & Drop, and Event Tracking

Complex Controls Handled Correctly

Using System Appearances

Designing with Storyboards

Modern Mac Features

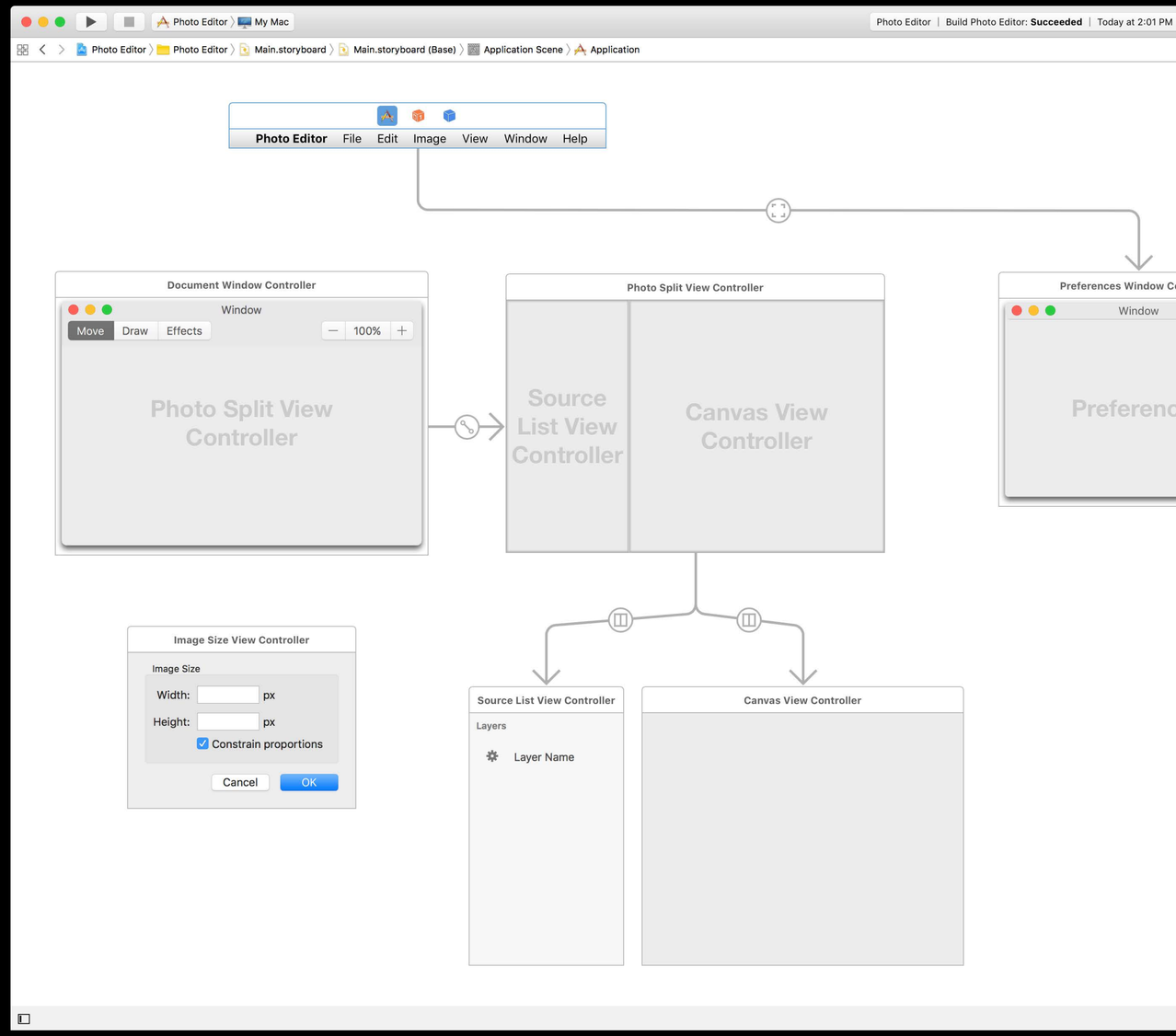
Using Storyboards

Designing with Storyboards

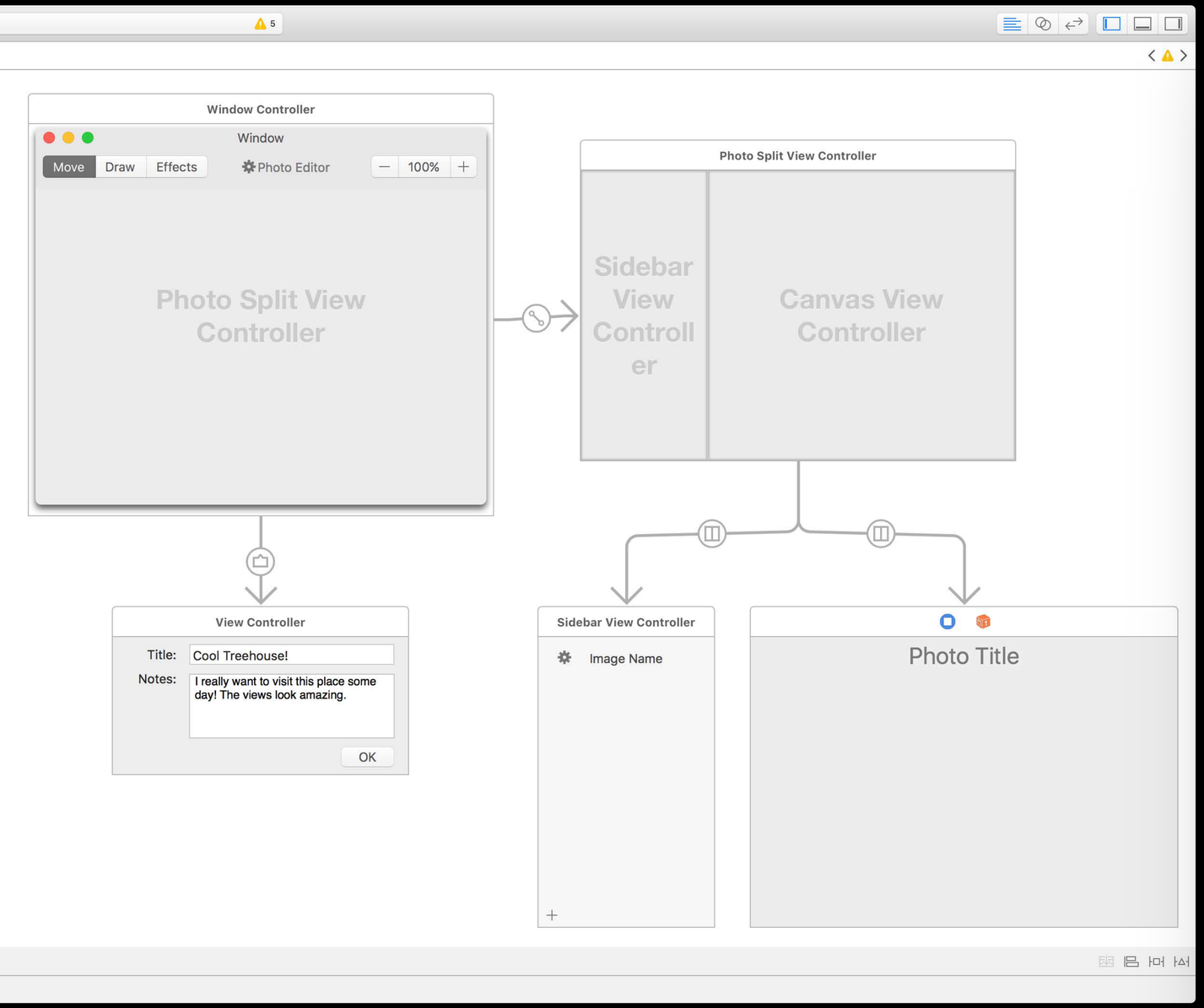
Visualize UI flow and hierarchy

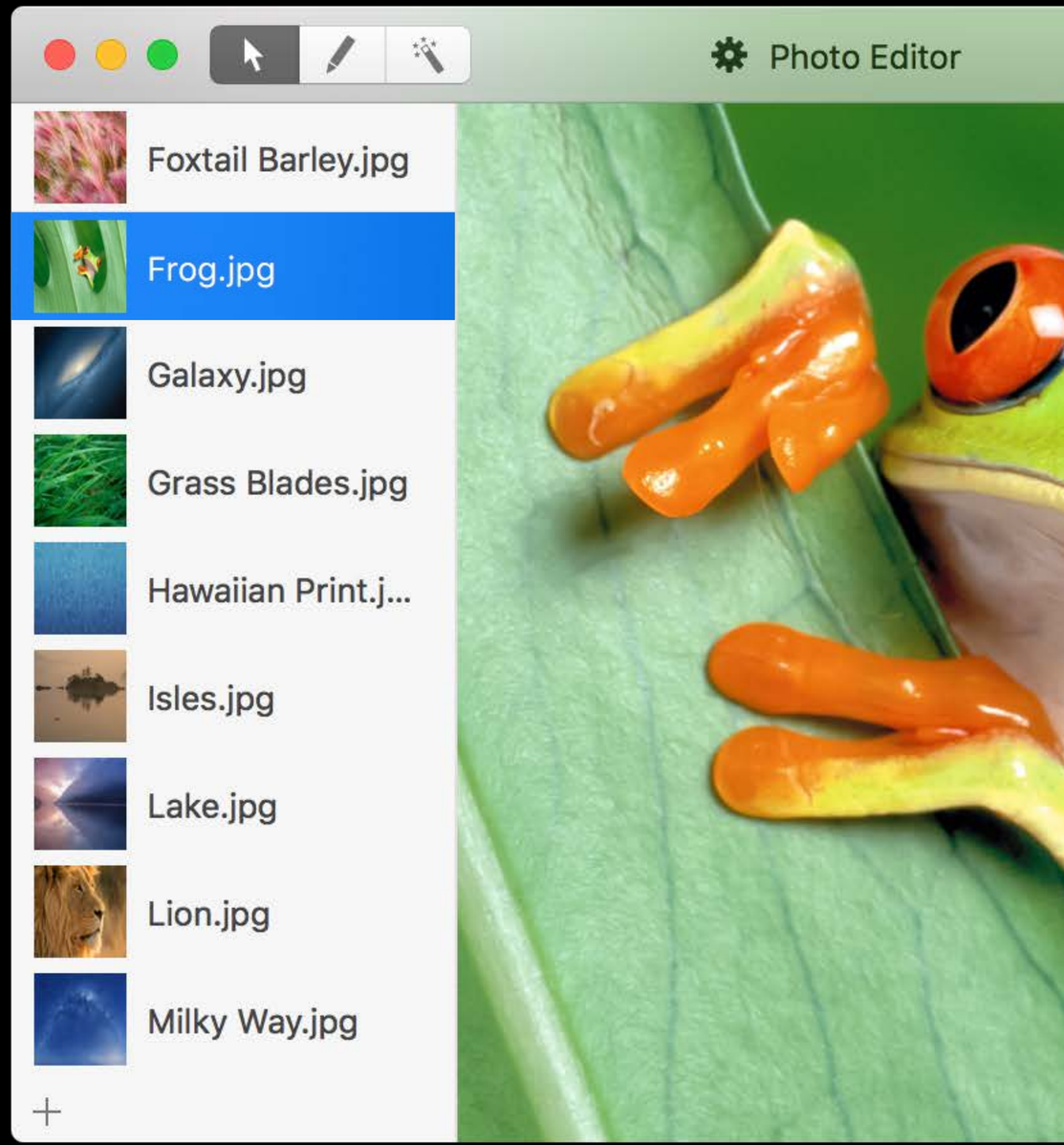
Use controllers as a unit of composition

Segues abstract away the "glue code" of managing the view hierarchy



Using Storyboards





Using Storyboards

Passing data between scenes

Dependencies should cascade downward

Hardcoding assumptions about UI structure reduces flexibility

Use protocol conformances to work generically

```swift
// Propagating dependencies generically from the NSWindowController subclass
var photoController: PhotoController? {
    didSet {
        propagate(photoController, toChildrenOf: self.contentViewController!)
    }
}

func propagate(_ photoController: PhotoController?, toChildrenOf parent: NSViewController) {
    if var consumer: PhotoControllerConsumer = parent as? PhotoControllerConsumer {
        consumer.photoController = photoController
    }
    for child in parent.childViewControllers {
        propagate(photoController, toChildrenOf: child)
    }
}
```

```swift
// Propagating dependencies generically from the NSWindowController subclass
var photoController: PhotoController? {
    didSet {
        propagate(photoController, toChildrenOf: self.contentViewController!)
    }
}

func propagate(_ photoController: PhotoController?, toChildrenOf parent: NSViewController) {
    if var consumer: PhotoControllerConsumer = parent as? PhotoControllerConsumer {
        consumer.photoController = photoController
    }
    for child in parent.childViewControllers {
        propagate(photoController, toChildrenOf: child)
    }
}
```

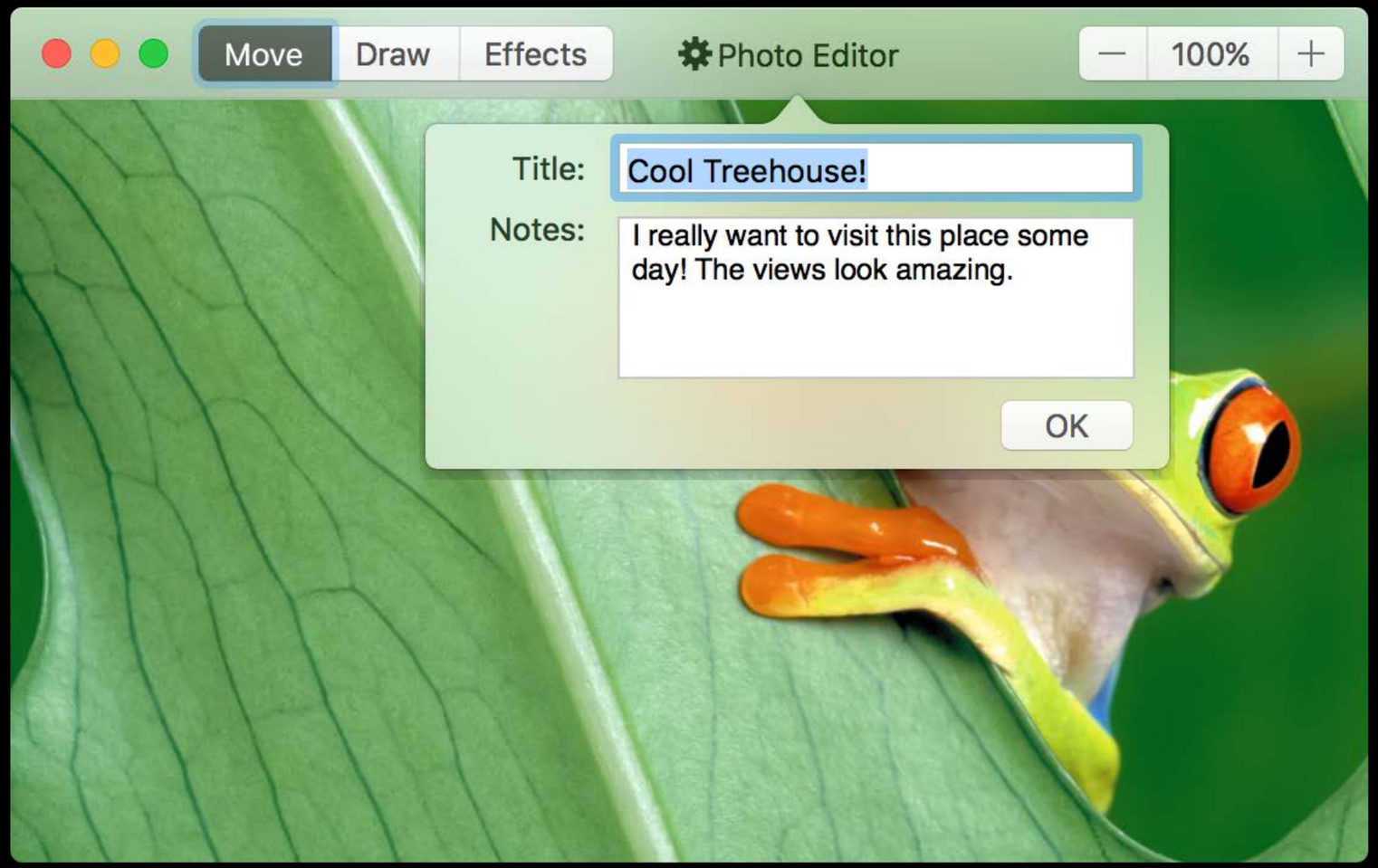
```swift
// Propagating dependencies generically from the NSWindowController subclass
var photoController: PhotoController? {
    didSet {
        propagate(photoController, toChildrenOf: self.contentViewController!)
    }
}

func propagate(_ photoController: PhotoController?, toChildrenOf parent: NSViewController) {
    if var consumer: PhotoControllerConsumer = parent as? PhotoControllerConsumer {
        consumer.photoController = photoController
    }
    for child in parent.childViewControllers {
        propagate(photoController, toChildrenOf: child)
    }
}
```

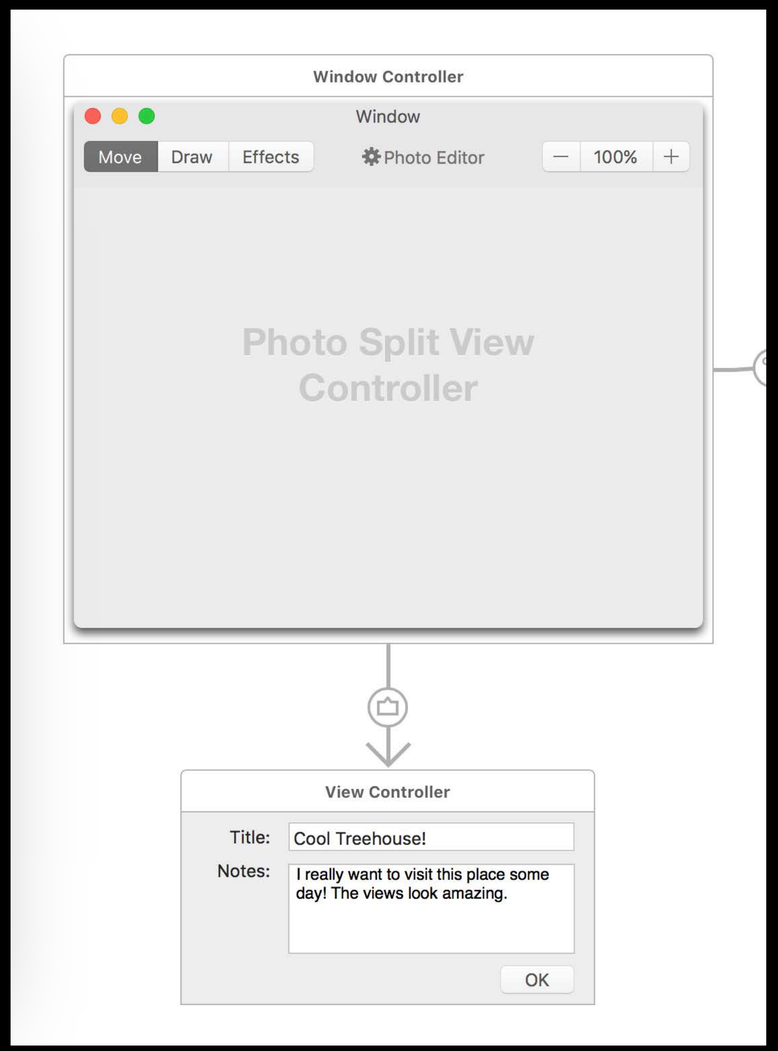
```swift
// Propagating dependencies generically from the NSWindowController subclass
var photoController: PhotoController? {
    didSet {
        propagate(photoController, toChildrenOf: self.contentViewController!)
    }
}

func propagate(_ photoController: PhotoController?, toChildrenOf parent: NSViewController) {
    if var consumer: PhotoControllerConsumer = parent as? PhotoControllerConsumer {
        consumer.photoController = photoController
    }
    for child in parent.childViewControllers {
        propagate(photoController, toChildrenOf: child)
    }
}
```

```swift
// Propagating dependencies generically from the NSWindowController subclass
var photoController: PhotoController? {
    didSet {
        propagate(photoController, toChildrenOf: self.contentViewController!)
    }
}

func propagate(_ photoController: PhotoController?, toChildrenOf parent: NSViewController) {
    if var consumer: PhotoControllerConsumer = parent as? PhotoControllerConsumer {
        consumer.photoController = photoController
    }
    for child in parent.childViewControllers {
        propagate(photoController, toChildrenOf: child)
    }
}
```

Using Storyboards

Passing data between scenes





Using Storyboards

Passing data between scenes

Use -prepareForSegue to provide data to presented controllers

```
override func prepare(for segue: NSStoryboardSegue, sender: AnyObject?) {
    if var photoConsumer = segue.destinationController as? PhotoControllerConsumer {
        photoConsumer.photoController = self.photoController
    }
}
```

Using Storyboards

Passing data between scenes

Use -prepareForSegue to provide data to presented controllers

```
override func prepare(for segue: NSStoryboardSegue, sender: AnyObject?) {
    if var photoConsumer = segue.destinationController as? PhotoControllerConsumer {
        photoConsumer.photoController = self.photoController
    }
}
```

Using Storyboards

Passing data between scenes

Use -prepareForSegue to provide data to presented controllers

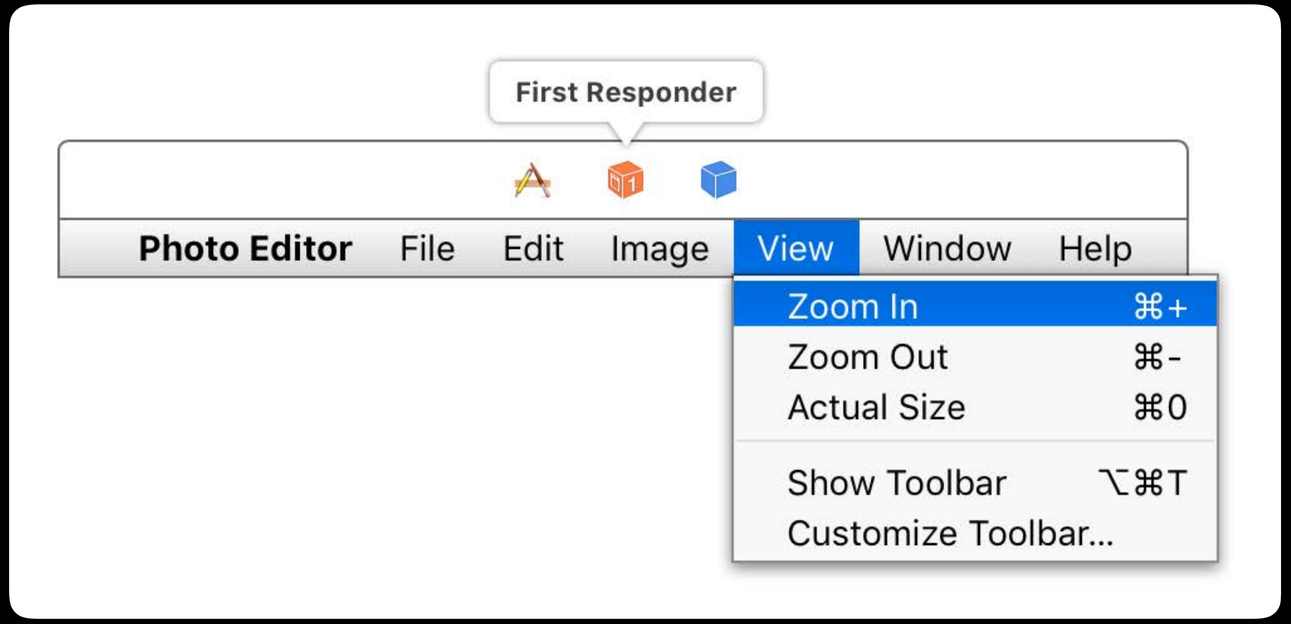
```
override func prepare(for segue: NSStoryboardSegue, sender: AnyObject?) {
    if var photoConsumer = segue.destinationController as? PhotoControllerConsumer {
        photoConsumer.photoController = self.photoController
    }
}
```

Using Storyboards

Propagating actions

The best handler for an action might be elsewhere in the UI hierarchy

Send actions up the responder chain



Using Storyboards

Propagating actions

Use UI validation to determine if a control is actionable

Using Storyboards

Propagating actions

Use UI validation to determine if a control is actionable

```
func controlIsValid(_ control: NSValidatedUserInterfaceItem) -> Bool {
    switch NSApp.target(forAction: control.action!, to: nil, from: control) {
    case let validator as NSUserInterfaceValidations:
        return validator.validateUserInterfaceItem(control)
    case .some(_):
        return true  // The target unconditionally handles this action
    case .none:
        return false // There is no target that handles this action
    }
}
```

Using Storyboards

Propagating actions

Use UI validation to determine if a control is actionable

```
func controlIsValid(_ control: NSValidatedUserInterfaceItem) -> Bool {
    switch NSApp.target(forAction: control.action!, to: nil, from: control) {
    case let validator as NSUserInterfaceValidations:
        return validator.validateUserInterfaceItem(control)
    case .some(_):
        return true  // The target unconditionally handles this action
    case .none:
        return false // There is no target that handles this action
    }
}
```

Using Storyboards

Propagating actions

Use UI validation to determine if a control is actionable

```
func controlIsValid(_ control: NSValidatedUserInterfaceItem) -> Bool {
    switch NSApp.target(forAction: control.action!, to: nil, from: control) {
    case let validator as NSUserInterfaceValidations:
        return validator.validateUserInterfaceItem(control)
    case .some(_):
        return true  // The target unconditionally handles this action
    case .none:
        return false // There is no target that handles this action
    }
}
```

Using Storyboards

Propagating actions

Use UI validation to determine if a control is actionable

```
func controlIsValid(_ control: NSValidatedUserInterfaceItem) -> Bool {
    switch NSApp.target(forAction: control.action!, to: nil, from: control) {
    case let validator as NSUserInterfaceValidations:
        return validator.validateUserInterfaceItem(control)
    case .some(_):
        return true  // The target unconditionally handles this action
    case .none:
        return false // There is no target that handles this action
    }
}
```

What to Expect

Creating a Modern Look with Modern Views

Proper Drag & Drop, and Event Tracking

Complex Controls Handled Correctly

Using System Appearances

Designing with Storyboards

Modern Mac Features

User Activities

Defining an activity

NSUserActivity describes what your app is viewing or editing

Used by Handoff to move activities between devices

User Activities

Defining an activity

NSUserActivity describes what your app is viewing or editing

Used by Handoff to move activities between devices

```
let activity = NSUserActivity(activityType: "com.example.calendar.view")
activity.title = NSLocalizedString("View Calendar", comment: "…")
activity.userInfo = [ "scope" : "day" ]
activity.delegate = self

/* NSUserActivityDelegate */
func userActivityWillSave(_ userActivity: NSUserActivity) {
    userActivity.addUserInfoEntries(from: [ "date" : self.currentDate ])
}
```

User Activities

Defining an activity

NSUserActivity describes what your app is viewing or editing

Used by Handoff to move activities between devices

```
let activity = NSUserActivity(activityType: "com.example.calendar.view")
activity.title = NSLocalizedString("View Calendar", comment: "…")
activity.userInfo = [ "scope" : "day" ]
activity.delegate = self

/* NSUserActivityDelegate */
func userActivityWillSave(_ userActivity: NSUserActivity) {
    userActivity.addUserInfoEntries(from: [ "date" : self.currentDate ])
}
```

User Activities

Defining an activity

NSUserActivity describes what your app is viewing or editing

Used by Handoff to move activities between devices

```
let activity = NSUserActivity(activityType: "com.example.calendar.view")
activity.title = NSLocalizedString("View Calendar", comment: "…")
activity.userInfo = [ "scope" : "day" ]
activity.delegate = self

/* NSUserActivityDelegate */
func userActivityWillSave(_ userActivity: NSUserActivity) {
    userActivity.addUserInfoEntries(from: [ "date" : self.currentDate ])
}
```

User Activities

Defining an activity

NSUserActivity describes what your app is viewing or editing

Used by Handoff to move activities between devices

```
let activity = NSUserActivity(activityType: "com.example.calendar.view")
activity.title = NSLocalizedString("View Calendar", comment: "…")
activity.userInfo = [ "scope" : "day" ]
activity.delegate = self

/* NSUserActivityDelegate */
func userActivityWillSave(_ userActivity: NSUserActivity) {
    userActivity.addUserInfoEntries(from: [ "date" : self.currentDate ])
}
```

User Activities

Defining an activity

NSUserActivity describes what your app is viewing or editing

Used by Handoff to move activities between devices

```
let activity = NSUserActivity(activityType: "com.example.calendar.view")
activity.title = NSLocalizedString("View Calendar", comment: "…")
activity.userInfo = [ "scope" : "day" ]
activity.delegate = self

/* NSUserActivityDelegate */
func userActivityWillSave(_ userActivity: NSUserActivity) {
    userActivity.addUserInfoEntries(from: [ "date" : self.currentDate ])
}
```

User Activities

Determining the current activity

Manually manage with -becomeCurrent and -resignCurrent

Attach activities to the responder chain for automatic management

User Activities

Determining the current activity

Manually manage with -becomeCurrent and -resignCurrent

Attach activities to the responder chain for automatic management

User Activities

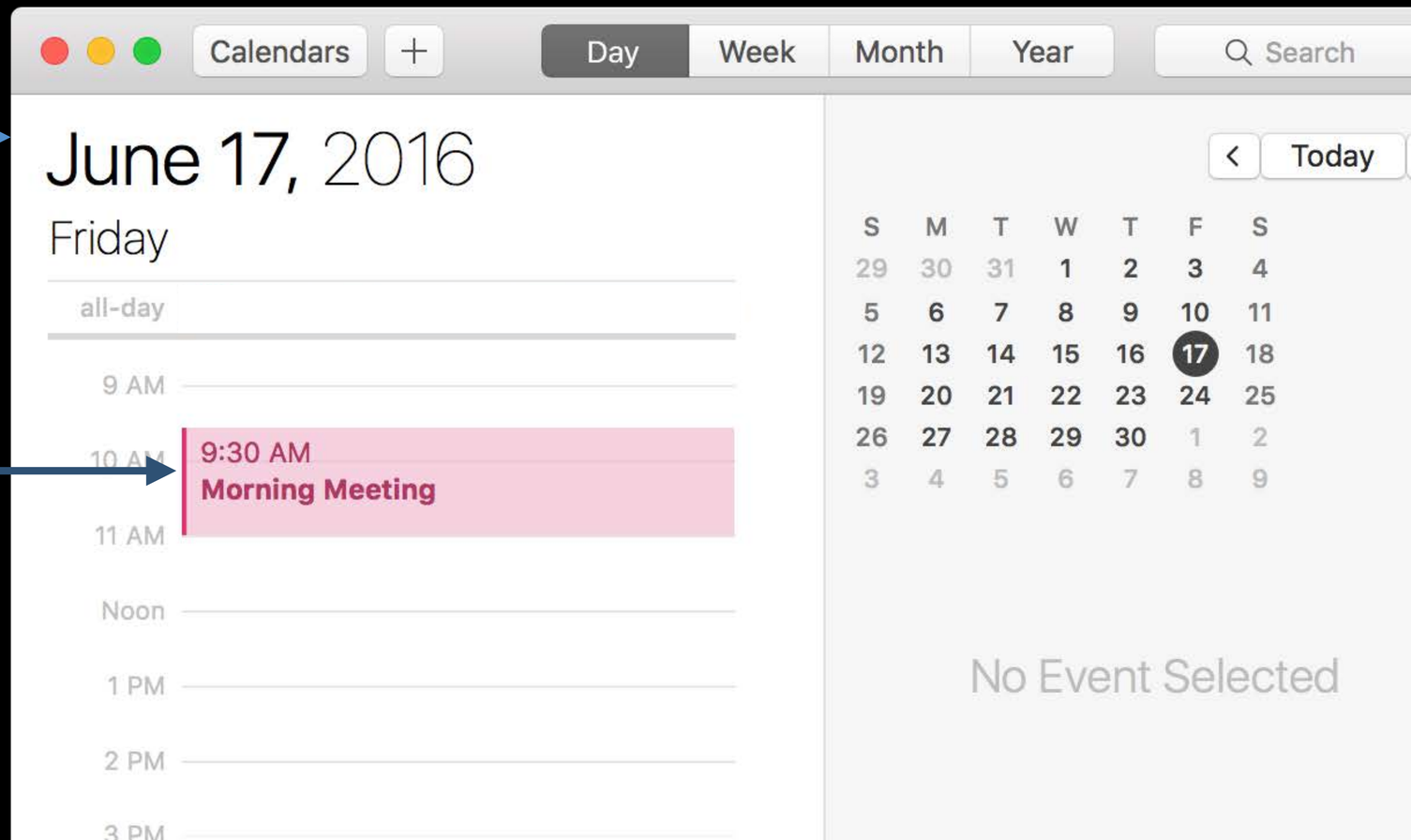
Determining the current activity

Manually manage with -becomeCurrent and -resignCurrent

Attach activities to the responder chain for automatic management

com.example.calendar.view

com.example.calendar.viewEvent



User Activities

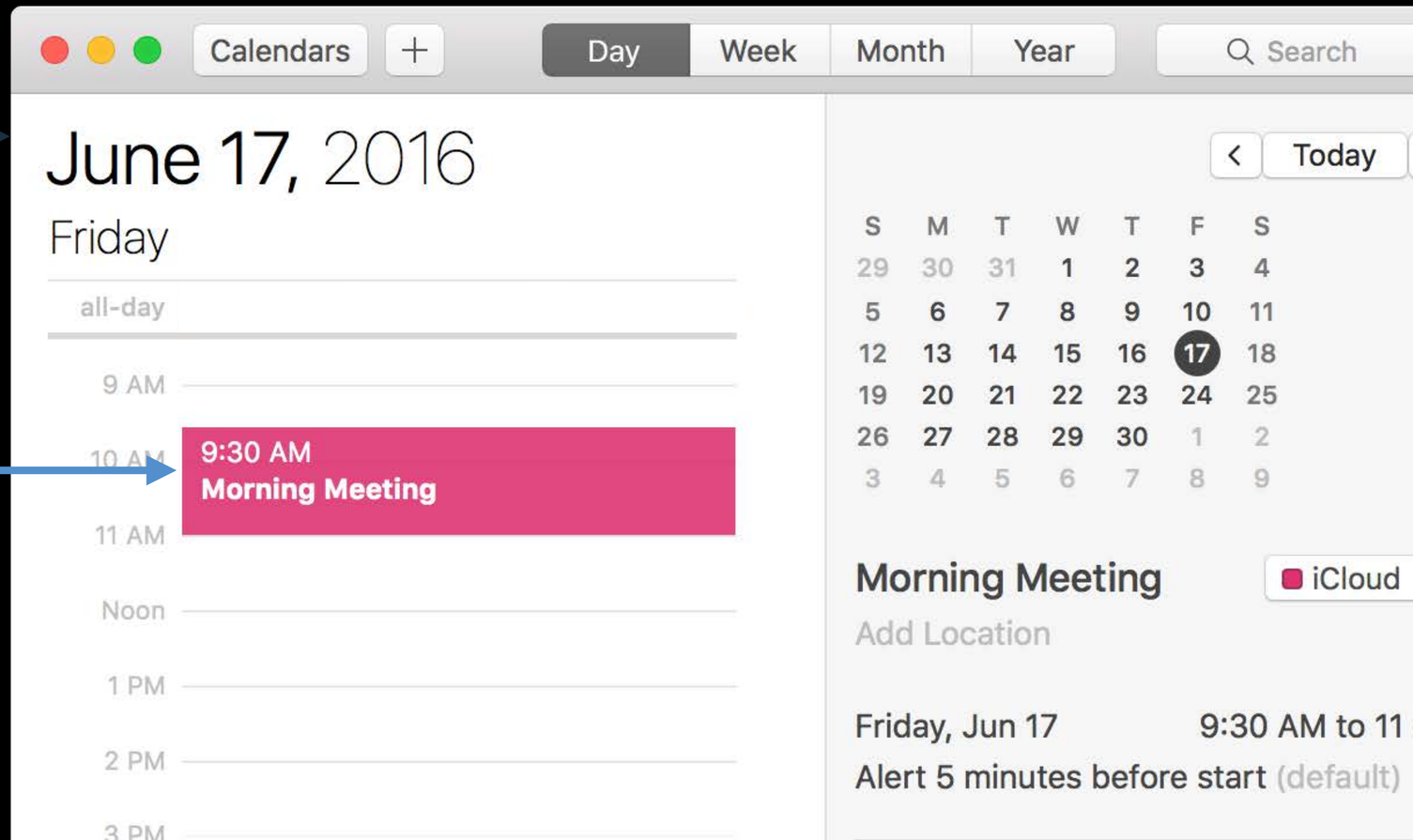
Determining the current activity

Manually manage with -becomeCurrent and -resignCurrent

Attach activities to the responder chain for automatic management

com.example.calendar.view

com.example.calendar.viewEvent

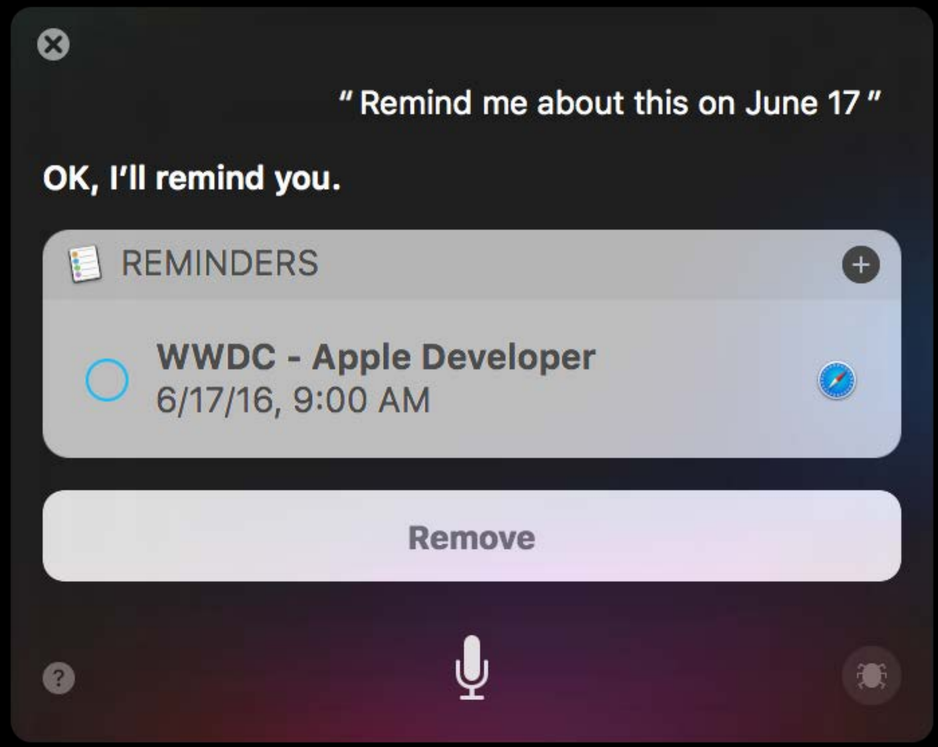


User Activities

Providing context for Siri

NEW

Siri uses the current activity to provide contextual commands



User Activities

For more information

Adopting Handoff on iOS and OS X — WWDC 2014

Resume

Picking up where you left off

State restoration allows apps to seamlessly quit and relaunch

The user interface contains state that doesn't belong in the model

State restoration encodes and restores transient UI state separately

Enabled on a per-window basis

Resume

Picking up where you left off

State restoration allows apps to seamlessly quit and relaunch

The user interface contains state that doesn't belong in the model

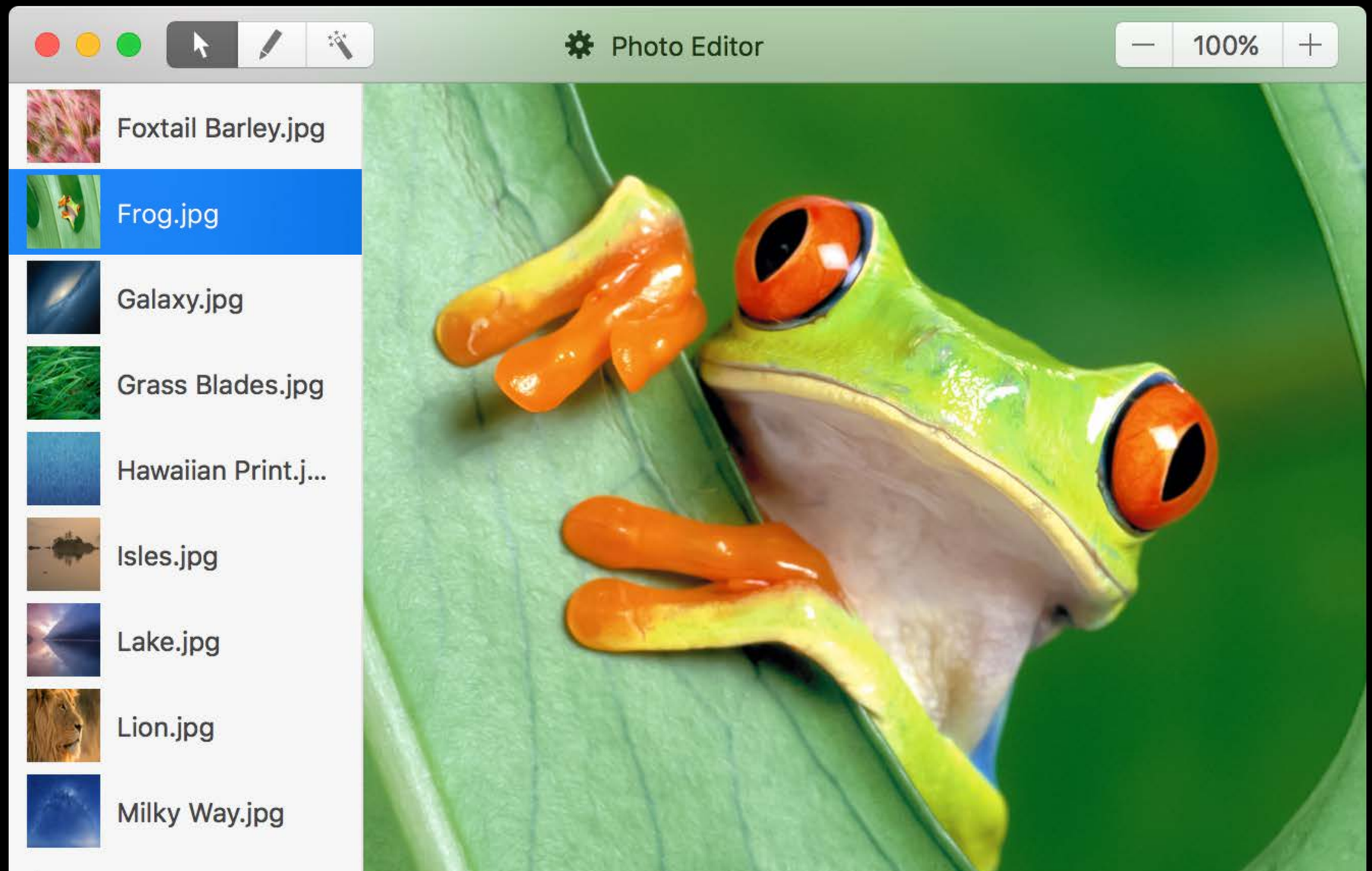
State restoration encodes and restores transient UI state separately

Enabled on a per-window basis

```
window.isRestorable = true
window.restorationClass = SomeClass.self // Conforms to NSWindowRestoration
```

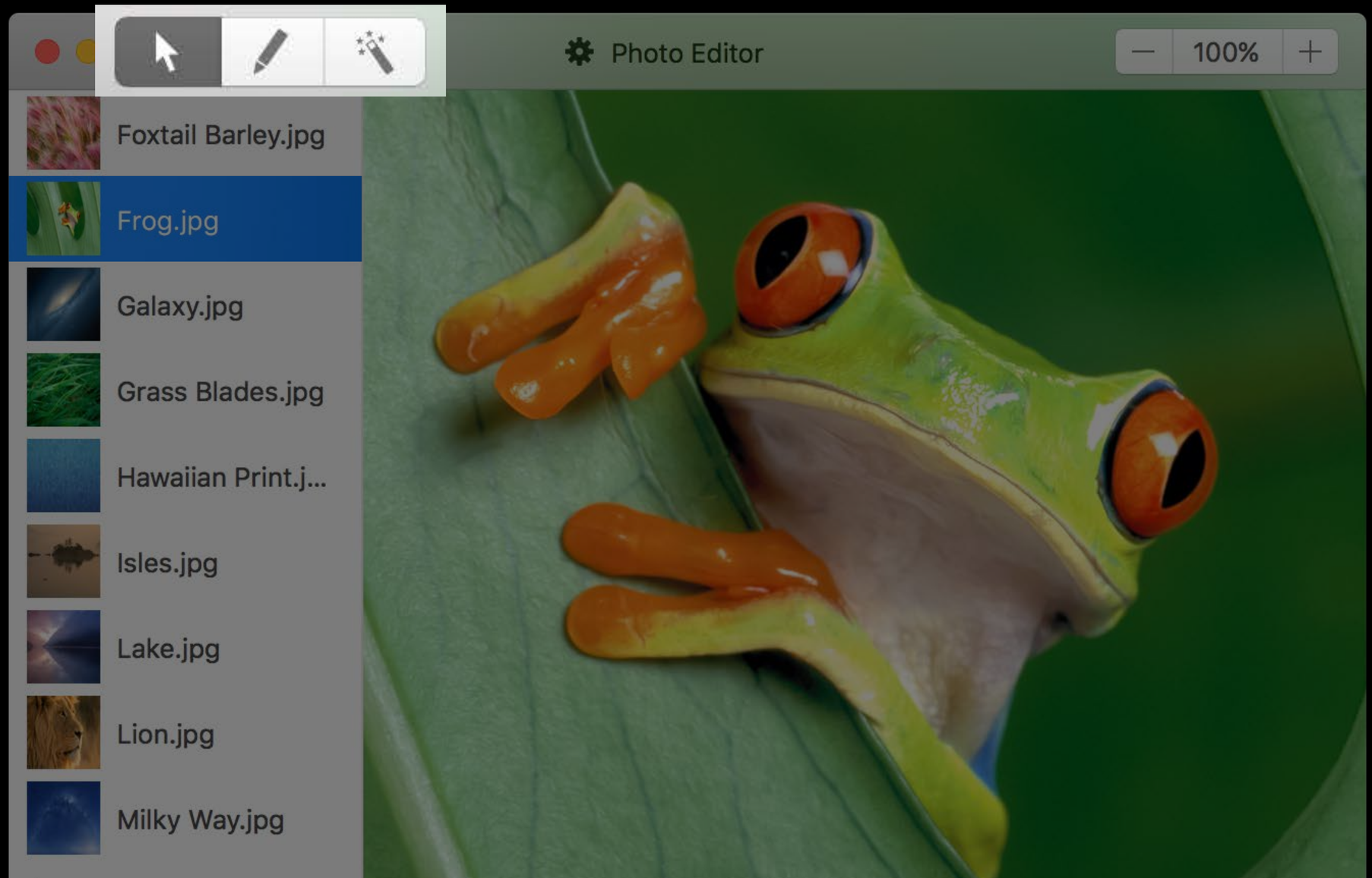
Resume

Picking up where you left off



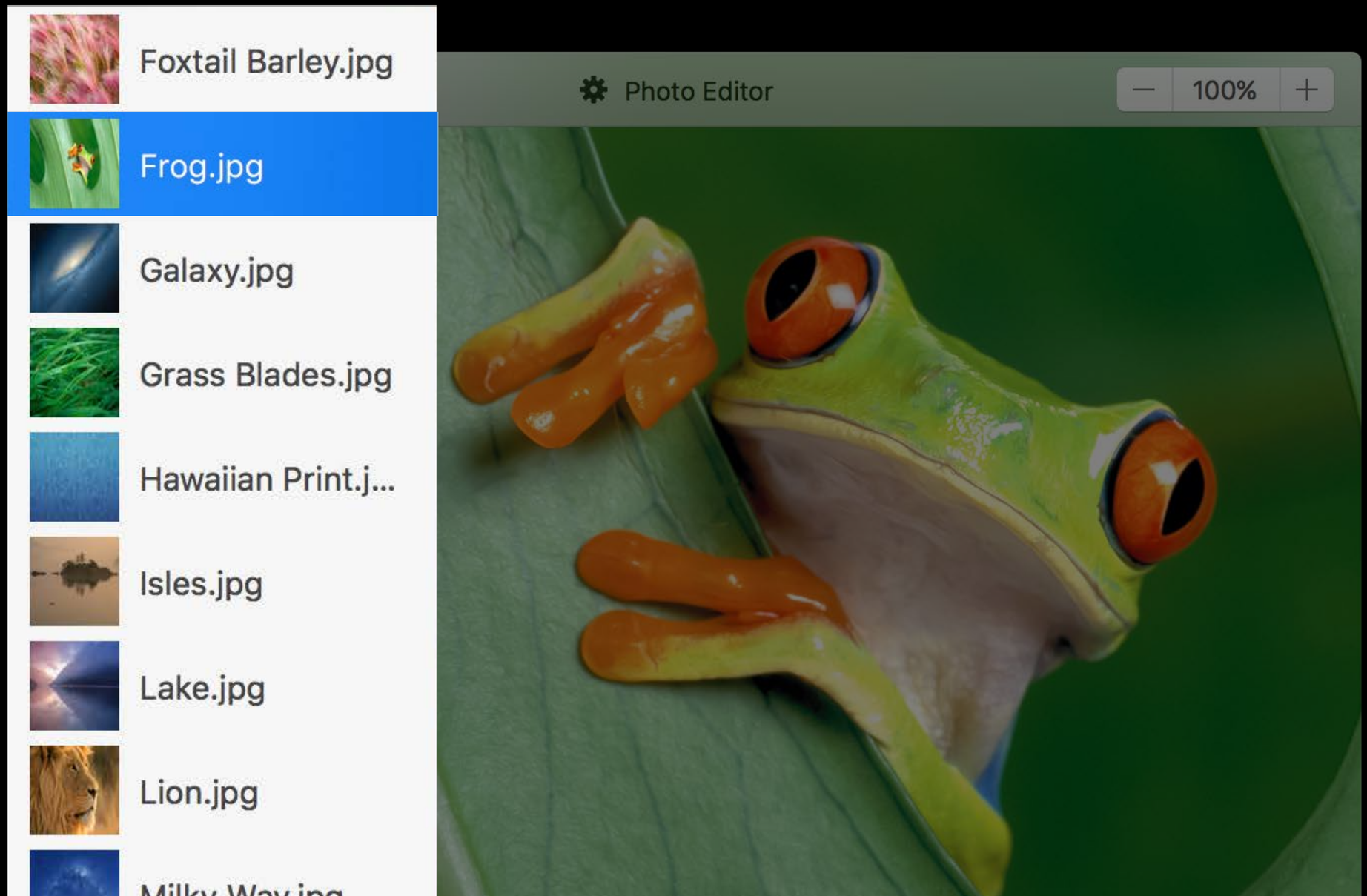
Resume

Picking up where you left off



Resume

Picking up where you left off



```
// Encoding UI state

override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)
    if let bookmarkData = self.bookmarkData {
        coder.encode(bookmarkData, forKey: "BookmarkData")
    }
}

// State invalidation
private func saveURLAsBookmarkData(_ url: URL) {
    do {
        bookmarkData = try url.bookmarkData([.withSecurityScope],
                                            includingResourceValuesForKeys: nil, relativeTo: nil)
        self.invalidateRestorableState()
    } catch let error as NSError  {
        presentError(error)
    }
}
```

```swift
// Encoding UI state

override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)
    if let bookmarkData = self.bookmarkData {
        coder.encode(bookmarkData, forKey: "BookmarkData")
    }
}

// State invalidation
private func saveURLAsBookmarkData(_ url: URL) {
    do {
        bookmarkData = try url.bookmarkData([.withSecurityScope],
                                            includingResourceValuesForKeys: nil, relativeTo: nil)
        self.invalidateRestorableState()
    } catch let error as NSError  {
        presentError(error)
    }
}
```

```
// Encoding UI state

override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)
    if let bookmarkData = self.bookmarkData {
        coder.encode(bookmarkData, forKey: "BookmarkData")
    }
}

// State invalidation
private func saveURLAsBookmarkData(_ url: URL) {
    do {
        bookmarkData = try url.bookmarkData([.withSecurityScope],
                                            includingResourceValuesForKeys: nil, relativeTo: nil)
        self.invalidateRestorableState()
    } catch let error as NSError  {
        presentError(error)
    }
}
```

```swift
// Encoding UI state

override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)
    if let bookmarkData = self.bookmarkData {
        coder.encode(bookmarkData, forKey: "BookmarkData")
    }
}

// State invalidation
private func saveURLAsBookmarkData(_ url: URL) {
    do {
        bookmarkData = try url.bookmarkData([.withSecurityScope],
                                            includingResourceValuesForKeys: nil, relativeTo: nil)
        self.invalidateRestorableState()
    } catch let error as NSError  {
        presentError(error)
    }
}
```

```
// Encoding UI state

override func encodeRestorableState(with coder: NSCoder) {
    super.encodeRestorableState(with: coder)
    if let bookmarkData = self.bookmarkData {
        coder.encode(bookmarkData, forKey: "BookmarkData")
    }
}

// State invalidation
private func saveURLAsBookmarkData(_ url: URL) {
    do {
        bookmarkData = try url.bookmarkData([.withSecurityScope],
                                            includingResourceValuesForKeys: nil, relativeTo: nil)
        self.invalidateRestorableState()
    } catch let error as NSError  {
        presentError(error)
    }
}
```

```swift
// Restoring UI state

override func restoreState(with coder: NSCoder) {
    super.restoreState(with: coder)
    bookmarkData = coder.decodeObject(forKey: "BookmarkData") as? Data
    guard let bookmarkData = self.bookmarkData else { return }
    /* Resolve the bookmark data and initialize the UI */
}
```

```swift
// Restoring UI state

override func restoreState(with coder: NSCoder) {
    super.restoreState(with: coder)
    bookmarkData = coder.decodeObject(forKey: "BookmarkData") as? Data
    guard let bookmarkData = self.bookmarkData else { return }
    /* Resolve the bookmark data and initialize the UI */
}
```

```swift
// Restoring UI state

override func restoreState(with coder: NSCoder) {
    super.restoreState(with: coder)
    bookmarkData = coder.decodeObject(forKey: "BookmarkData") as? Data
    guard let bookmarkData = self.bookmarkData else { return }
    /* Resolve the bookmark data and initialize the UI */
}
```

State Restoration

Restorable state key paths

For simple properties, declare restorableStateKeyPaths

```
override class func restorableStateKeyPaths() -> [String] {
    return super.restorableStateKeyPaths() + ["selectedIndex", "brushColor"]
}
```

Not mutually exclusive with manual encoding and decoding

Automatically invalidates state when properties change

Documents in the Cloud

With iCloud Drive, any document can end up in the cloud

macOS Sierra brings important changes to iCloud Drive

- The ~/Desktop and ~/Documents folders may be synced via iCloud
- Local copies of documents may be evicted to free up space

Documents in the Cloud

Using file coordination for cloud documents

Register a NSFilePresenter to ensure that the document isn't evicted out from under you

Use NSFileCoordinator to coordinate reading and writing

- File coordination will make sure the file is downloaded and up to date

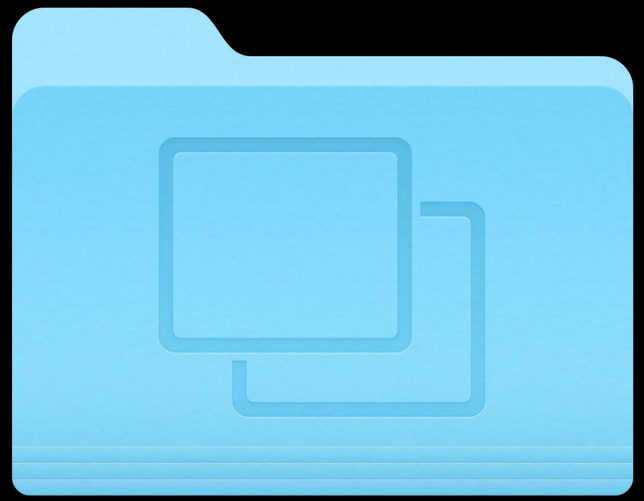
Further Reading

Key technologies for modern apps

Further Reading

Key technologies for modern apps

Asset Catalogs



Further Reading

Key technologies for modern apps

Asset Catalogs

Accessibility

Further Reading

Key technologies for modern apps

Asset Catalogs

Accessibility

Sandboxing

Further Reading

Key technologies for modern apps

Asset Catalogs

Accessibility

Sandboxing

XPC Services

Summary

Creating a Modern Look with Modern Views

Modern Drag & Drop, and Event Tracking

Container View Controls Handled Correctly

Using System Appearances

Designing with Storyboards

Modern Mac Features

More Information

https://developer.apple.com/wwdc16/239

Related Sessions

What's New In Cocoa	Nob Hill	Tuesday 11:00AM
Working with Wide Color	Mission	Thursday 1:40PM
What's New in Auto Layout	Presidio	Friday 3:00PM
Creating Modern Cocoa Apps		WWDC 2014

WWDC16