FACTORY
SERVICES WORK ABOUT CAREERS BLOG PIMCORE KNOWLEDGE BASE CONTACT US

BLOG

Making custom UISwitch Part 2

Mobile Development

Last updated on December 23, 2016

6 min read

Share

In the second part of *Making custom UISwitch* we will make this control more customizable with IBDesignable and IBInspectable, add an image to thumb view of the switch, on/off labels and on/off thumb images.

IBDesignable and IBInspectable are very handy features that are introduced with Xcode 6. They allow us to create custom controls that are rendered in real-time in Interface builder. IBDesignable adds that kind of option to your UIView class.

By adding keyword IBInspectable to a variable, we make that variable customizable in Interface builder.
If you miss the first part of Making custom UISwitch, please read it for better understanding this one.

But enough with definitions, let's start coding.

Add *@IBDesignable* before defining class and add *@IBInspectable* in front of every public property that you want to customize.

```
@IBDesignable
final class CustomSwitch: UIControl
@IBInspectable public var padding: CGFloat = 1 {
   didSet {
      self.layoutSubviews()
   }
}
@IBInspectable public var onTintColor = UIColor(red: 144/255, green: 202/255, blue: 119/255, alpha
   didSet {
      self.setupUI()
   }
}
```

Go to Main.storyboard and drag and drop to UIViewController UIView. In Identity Inspector set CustomSwitch to be class of this view. After a few seconds of rendering, you will see an image similar to the one shown below.

In Attributes Inspector you will see that multiple properties appeared under the *Custom Switch* section on top. The picture below can be different, depending on which property has the attached keyword *@IBInspectable.*

If you add prefix *@IBInspectable* to every property some of them won't be shown in Attributes Inspector (*thumbShadowColor*). To make them visible, define their type (*: UIColor*) and they will be rendered too.

```
@IBInspectable public var thumbShadowColor: UIColor = UIColor.black {
   didSet {
      self.thumbView.layer.shadowColor = self.thumbShadowColor.cgColor
   }
}
```

thumbShadowColor is specific because it is CGColor, you must change it to UIColor in order to show it in Interface builder.

You can play with these properties, increase and decrease values. Some of them will give you results that are acceptable, some of them won't (*cornerRadius, thumbCornerRadius*).

That's because every time you change a value, *didSet{}* of that property is called and it is live rendered in Interface builder. To fix this behavior, we will override *set{}* and *get{}* for *cornerRadius* and *thumbCornerRadius* property and add private properties.

```
@IBInspectable public var cornerRadius: CGFloat {
   get {
      return self.privateCornerRadius
   }
   set {
      if newValue > 0.5 || newValue < 0.0 { privateCornerRadius = 0.5 } else { privateCornerRadius =
         privateThumbCornerRadius = 0.5
      } else {
         privateThumbCornerRadius = newValue
      }
   }
}
private var privateThumbCornerRadius: CGFloat = 0.5 {
   didSet {
      self.layoutSubviews()
   }
}
```

You can now test all properties in Interface builder and create a switch with any color and shape you want.
If you are bored of playing with these properties, we can go on with other customizations.

If you are for a while in iOS development, undoubtedly, you wanted to replace a boring thumb circle in the native switch with an image? In this part of the tutorial, we will add simple UIImageView to thumb view.
Create a simple class that will add ImageView to itself. Open File/New/File pick Cocoa Touch Class, make it a subclass of UIView and set its name (CustomThumbView).

Create UIImageView property and add it as subView in the initialization of the class.

```
fileprivate(set) var thumbImageView = UIImageView(frame: CGRect.zero)
override init(frame: CGRect) {
   super.init(frame: frame)
   self.addSubview(self.thumbImageView)
}
required init?(coder aDecoder: NSCoder) {
   super.init(coder: aDecoder)
   self.addSubview(self.thumbImageView)
}
```

If you noticed there is no set of thumbImageView frame, that is because it will be done in layoutSubViews() method (frames are most accurate in that method and every update will be translated to frame of thumbImageView).
Override layoutSubviews() and add it to the extension of CustomThumbView or to class.

```
extension CustomThumbView {
   override func layoutSubviews() {
      super.layoutSubviews()
      self.thumbImageView.frame = CGRect(x: 0, y: 0, width: self.frame.width, height: self.frame.hei
      self.thumbImageView.layer.cornerRadius = self.layer.cornerRadius
      self.thumbImageView.clipsToBounds = self.clipsToBounds
   }
}
```

In CustomSwitch class replace initialization of thumbView with a new one.
fileprivate var thumbView = CustomThumbView(frame: CGRect.zero)

Everything is ready for the image and we can add it by setting the thumbImage property.

switch.thumbImage =UIImage(named: "CustomSwitchImg.png")
If the image has a transparent background, the background of thumbView will be seen. If you don't want that kind of result just set switch.thumbTintColor = UIColor.clear.

Looks nice, don't you agree? There are many custom switches that have state descriptions (on/off). Now we will add some labels to create that kind of layout. First, add two labels and one Bool property to CustomSwitch class.

```
// labels
public var labelOff:UILabel = UILabel()
public var labelOn:UILabel = UILabel()
public var areLabelsShown: Bool = false {
   didSet {
      self.setupUI()
   }
}
```

After that, create setupLabel() method that will set label values and add it to the view under thumbView. Example is in code snippet below but you can customize it yourself.

```
 fileprivate func setupLabels() {
      guard self.areLabelsShown else {
         self.labelOff.alpha = 0
         self.labelOn.alpha = 0
         return
      }
      self.labelOff.alpha = 1
      self.labelOn.alpha = 1
      let labelWidth = self.bounds.width / 2 - self.padding * 2
      self.labelOn.frame = CGRect(x: 0, y: 0, width: labelWidth, height: self.frame.height)
      self.labelOff.frame = CGRect(x: self.frame.width - labelWidth, y: 0, width: labelWidth, height: s
      self.labelOn.font = UIFont.boldSystemFont(ofSize: 12)
      self.labelOff.font = UIFont.boldSystemFont(ofSize: 12)
      self.labelOn.textColor = UIColor.white
      self.labelOff.textColor = UIColor.white
      self.labelOff.sizeToFit()
      self.labelOff.text = "Off"
      self.labelOn.text = "On"
      self.labelOff.textAlignment = .center
      self.labelOn.textAlignment = .center
      self.insertSubview(self.labelOff, belowSubview: self.thumbView)
      self.insertSubview(self.labelOn, belowSubview: self.thumbView)
   }
On the bottom of layoutSubviews() method add condition for updating the labels frame.
//label frame
if self.areLabelsShown {
   let labelWidth = self.bounds.width / 2 - self.padding * 2
   self.labelOn.frame = CGRect(x: 0, y: 0, width: labelWidth, height: self.frame.height)
   self.labelOff.frame = CGRect(x: self.frame.width - labelWidth, y: 0, width: labelWidth, height: self
}
```

That's it, just set in UIViewController switch.areLabelsShown = true
to true and labels will be shown in switch layout. Labels are intentionally public so you can access their properties (color, font…) outside CustomSwitch class.

Last but not least, we will create a simple smooth image animation for changing the state of control.
First we add two private properties to CustomSwitch class and two public UIImage properties with overridden didSet().

```
fileprivate var onImageView = UIImageView(frame: CGRect.zero)
fileprivate var offImageView = UIImageView(frame: CGRect.zero)
public var onImage:UIImage? {
   didSet {
      self.onImageView.image = onImage
      self.layoutSubviews()
   }
}
public var offImage:UIImage? {
   didSet {
      self.offImageView.image = offImage
      self.layoutSubviews()
   }
}
```

Add onImageView and offImageView as subViews at the bottom of setupUI() method.

self.addSubview(self.onImageView)
self.addSubview(self.offImageView)
At the end of *layoutSubviews()* add this code snippet which sets the start frame and alpha value of on/off images. This guard method prevents the user to set just one image since both of them must be set to set the right frame for them.

```
guard onImage != nil && offImage != nil else {
   return
}
let frameSize = thumbSize.width > thumbSize.height ? thumbSize.height * 0.7 : thumbSize.width * 0
let onOffImageSize = CGSize(width: frameSize, height: frameSize)
self.onImageView.center = CGPoint(x: self.onPoint.x + self.thumbSize.width / 2, y: self.onPoint.y + s
self.offImageView.center = CGPoint(x: self.offPoint.x + self.thumbSize.width / 2, y: self.offPoint.y +
self.onImageView.frame.size = onOffImageSize
self.offImageView.frame.size = onOffImageSize
self.onImageView.alpha = self.isOn ? 1.0 : 0.0
self.offImageView.alpha = self.isOn ? 0.0 : 1.0
```

To create animations, add setOnOffImageFrame() method which moves the center of images depending on its isOn property. You can add this method to the bottom of the animation block (they will have the same animation speed as the rest of the objects).

```
fileprivate func setOnOffImageFrame() {
   self.onImageView.center.x = self.isOn ? self.onPoint.x + self.thumbSize.width / 2 : self.frame.widt
   self.offImageView.center.x = !self.isOn ? self.offPoint.x + self.thumbSize.width / 2 : 0
   self.onImageView.alpha = self.isOn ? 1.0 : 0.0
   self.offImageView.alpha = self.isOn ? 0.0 : 1.0
}
```

Finally, add UIImage to switches property onImage and offImage in UIViewController and animation will be set automatically.

That's it for now. Hope this tutorial will help you create pretty switches in future projects.
You can find all source code of this tutorial on GitHub Factory repository.

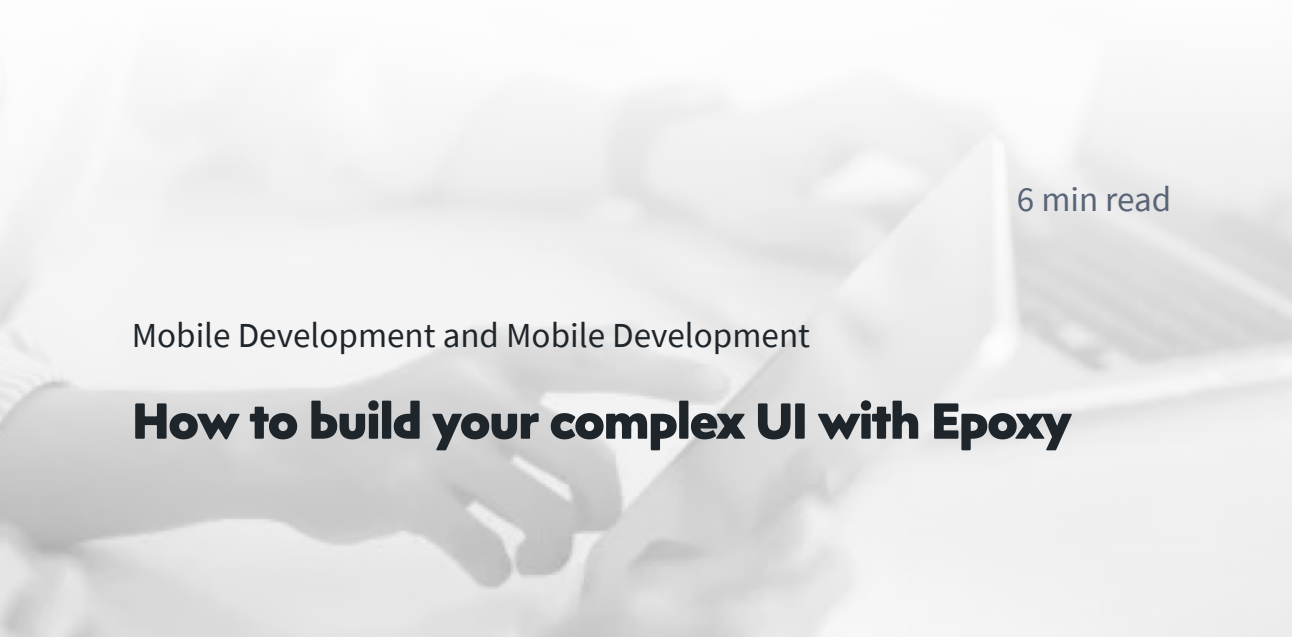
To stay updated, follow us on our social media channels – Facebook – Twitter – LinkedIn – Instagram

If you liked this, you might like…

How to use dynamic quick actions in iOS
How we created custom scale view element in iOS
What to avoid when submitting a mobile app to iTunes
A Quick Guide to App Store Optimization

Like Be the first of your friends to like this.

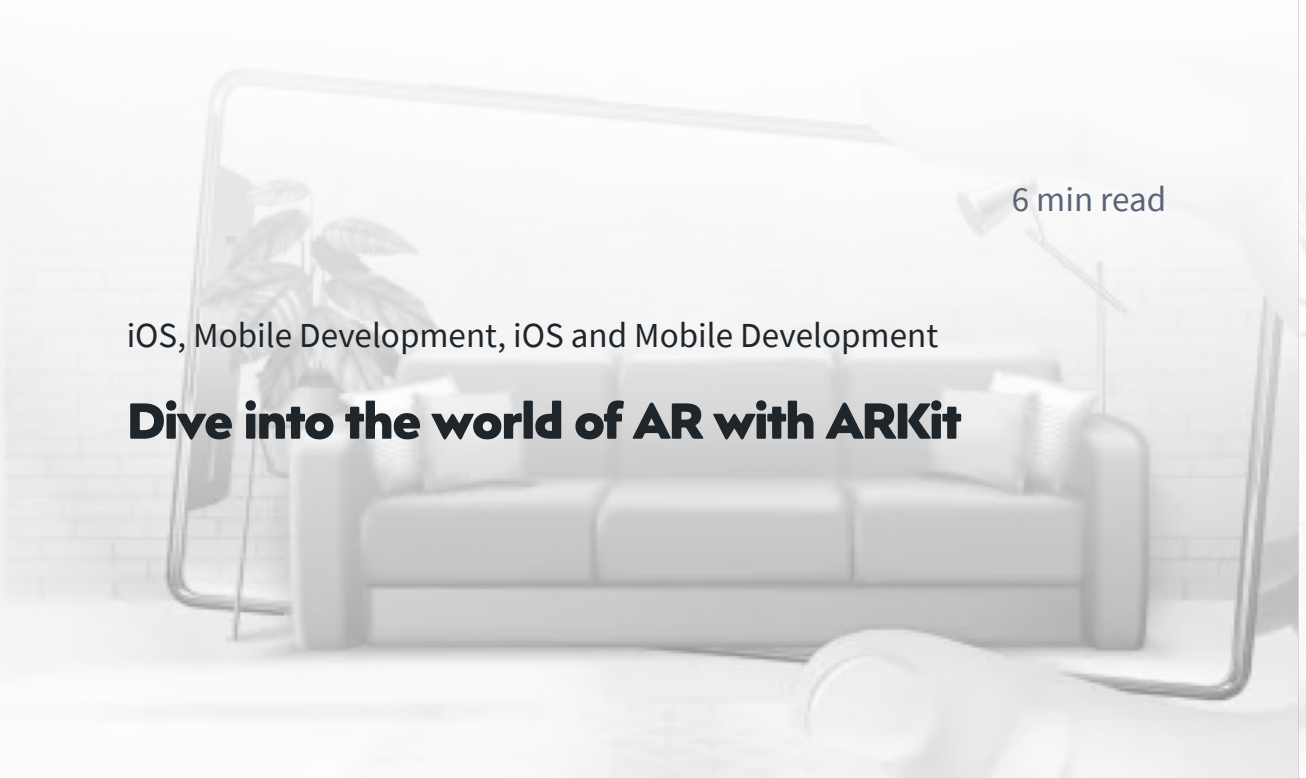
We prepared more good reads for you!



6 min read

Mobile Development and Mobile Development

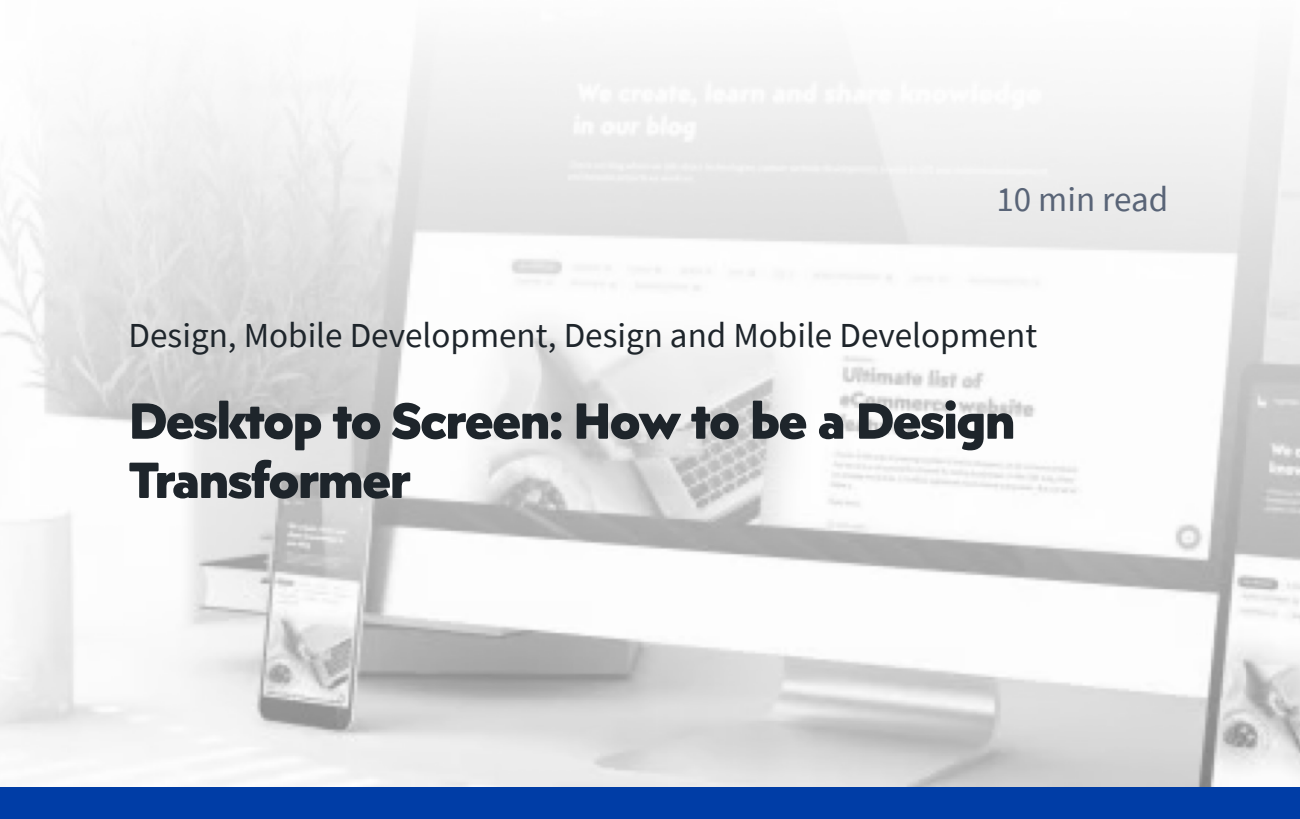
How to build your complex UI with Epoxy



6 min read

iOS, Mobile Development, iOS and Mobile Development

Dive into the world of AR with ARKit



10 min read

Design, Mobile Development, Design and Mobile Development

Desktop to Screen: How to be a Design Transformer

EXPLORE NEXT

Contact

Are you looking to improve your online presence by building a new website or making sure your service/product is available to your customers on their mobile phones or just looking to say hi? We love discussing new ideas and bringing them to life so please feel free to get in touch!

Discover more

CONTACT US

Say Hello to Boris.

BORIS IS OUR SALES MANAGER

NAME *
Name

EMAIL *
Email

HOW CAN WE HELP YOU? *
How can we help you?

Fields marked with * are required.

☐ I understand the information above will be stored only for business purposes. Check our Privacy Policy for more info.

SUBMIT