# 親切すぎる iPhone アプリ開発の本

## Xcode 7（Swift 2）への対応

2015.11.08

目的

## Xcode 7 で「親切すぎる iPhone アプリ開発の本」のサンプルを動かすためにおこなわれた変更点を説明する。

Xcode 7 では、Swift 言語が 1.2 から 2 にバージョンアップされました。

そのため Xcode 7 で「親切すぎる iPhone アプリ開発の本」のサンプルを動かすためには、Swift 2 の文法への変更が必要になります。

ダウンロードできるサンプル自体は、すでに Xcode 7 に対応済みです。

このドキュメントは、サンプルを Xcode 7 に対応させるために、どのような変更をおこなったかを説明するために用意しました。

「親切すぎる iPhone アプリ開発の本」の関連ページを示しながら説明しますので、Swift 2 への移行の参考に使ってください。

24 ページ

## println が廃止された

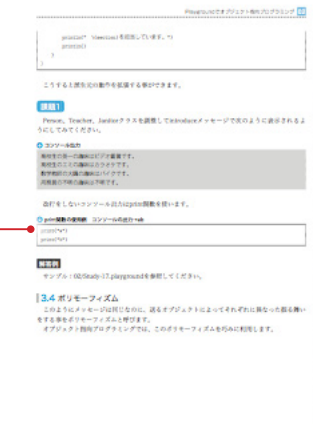
println は無くなりました。代わりに print を使います。

28 ページ

引数は必須となったので、以下の記述はエラーとなります。

```
print()
```

79 ページ



このように Swift 2 では、print は何も指定しないと最後に改行をするようになっています。Swift 1.2 までの print のように改行をさせたくない場合、引数 terminator: に空文字「""」を与えます。例えば、課題 1 の

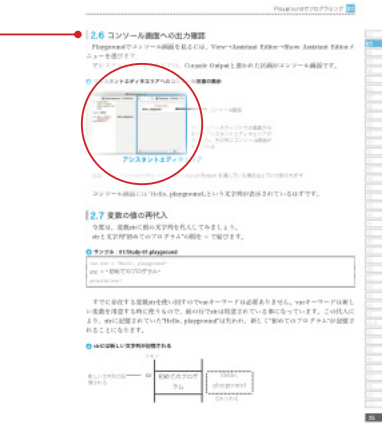
```
print("a")
print("b")
```

は

```
print("a", terminator: "")
print("b", terminator: "")
```

というようにします。

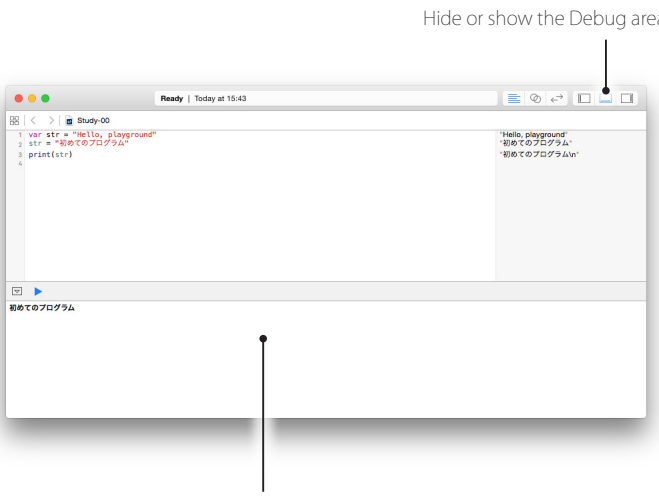
25 ページ



## Playground のコンソール画面出力がデバッグエリアに変更された

Playground のコンソール画面出力はデバッグエリアに表示されるようになりました。ツールバー自体が表示されない時は、View → Show Toolbar メニューを選びます。

**アシスタントエディタエリアへのコンソール画面の表示**



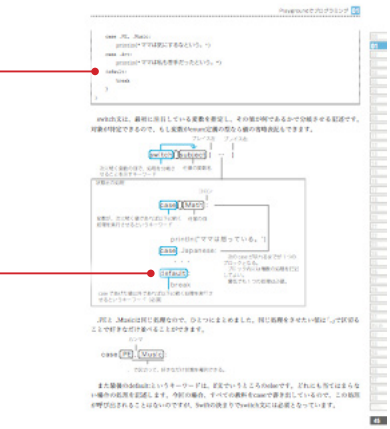

注意）コンソールに以下のエラーが表示される場合がありますが、無視してかまいません。

<Error>: CGContextSaveGState: invalid context 0x0.・・・

<Error>: CGContextTranslateCTM: invalid context 0x0.・・・

<Error>: CGContextRestoreGState: invalid context 0x0.・・・

45 ページ



## すべての case 文を用意した場合、default は不要となった

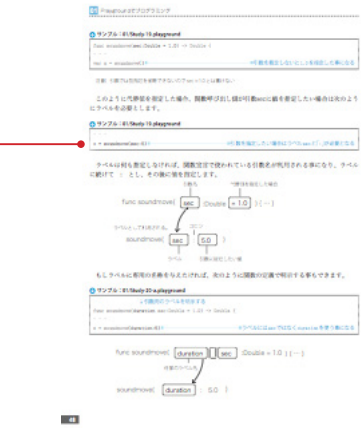
例えば 01/Study-17.playground のように、すべての列挙の case が書かれていた場合、default はあってもなくてもよくなりました。

46 ページ

## 定数の利用を推奨される

値が変更されない変数は、定数として宣言するよう注意されるようになりました。そのため、例えば 01/Study-18.playground では、var distance を let に変更した方がいいと注意されます。

48 ページ



## 関数のラベルの扱いがメソッドと同じになった

関数のラベルはメソッドと同じ扱いに（124 ページ参照）なりました。このため第 1 引数は、特に指定しない限りラベルが使われなくなり、第 2 引数以降は特に指定がない限り引数名がラベルに使われます。

代替値を指定された場合も自動でラベルはつきません。そのため、例えば 01/Study-19.playground でおこなっているようなラベル名指定は不要になりました。

```
x = soundmove(sec:5)
```

は

```
x = soundmove(5)          (sec:5) と書かなくてよい
```

となります。

50 ページ

同じ理由で、例えば 01/Study-20-d.playground でおこなっている _ によるラベル利用禁止指定は不要になりました。

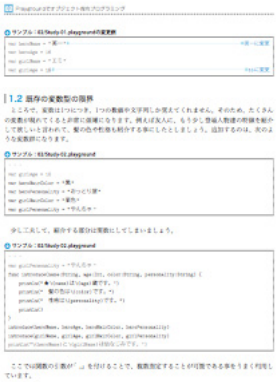
```
func soundmove(_ sec:Double = 1.0)->Double {
```

は

```
func soundmove(sec:Double)->Double {  _ は不要
```

となります。

58 ページ



そして 02/Study-02.playground での introduce 関数呼び出しは

```
introduce(heroName, heroAge, heroHairColor, heroPersonality)
```

が

```
introduce(heroName, age: heroAge, color: heroHairColor,
  personality: heroPersonality)
```

となり、第２引数以降に age: や color:、personality: といったラベルが必要にます。

84 ページ

## do while ループ文が repeat while ループ文となった

do while ループ文は廃止され、かわりに repeat while ループ文が用意されました。そのため、例えば、4.5 do while ループ文のサンプルとして示した

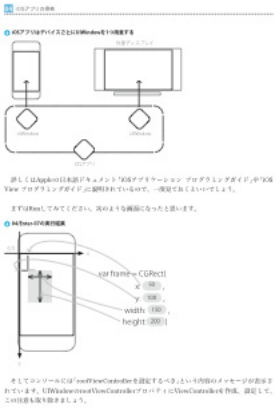
```
do {
    appearances[i++].introduce()
} while i <= 3
```

は

```
repeat {
    appearances[i++].introduce()
} while i <= 3
```

となります。

154 ページ



## UIWindow の rootViewController プロパティ設定が必須となった

UIWindow の rootViewController プロパティの設定は必須となりました。そのため 04/Enter-07 は実行すると実行時エラーで止まります。本文中に

**まずは Run してみてください。次のような画面になったと思います。**

とありますが、実行時エラーで止まり画面は確認できません。ここでは Stop ボタンで停止するようにしてください。

次の 155 ページに rootViewController の設定が説明されています。

163 ページ

## touchesBegan 〜 touchesEnded の引数の型が変更された

touchesBegan 〜 touchesEnded の引数の型が変更されました。
そのため、例えば 05/Interactive-01 の

```
override func touchesBegan(touches: Set<NSObject>, withEvent event: UIEvent) {
```

は

```
override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
```

となります。

164 ページ

これに合わせ 05/Interactive-02 のような、as? UITouch は不要となり

```
if let touch = touches.first as? UITouch
```

は

```
if let touch = touches.first      ← as? UITouch は不要
```

となります。

180 ページ

## CGRect 構造体のメソッドが変更された

CGRect 構造体の rectByOffsetting は offsetBy に変更されました。
そのため、例えば 06/ViewInView-04 の

```
overView.frame = overView.frame.rectByOffsetting(dx: 200, dy: 0)
```

は

```
overView.frame = overView.frame.offsetBy(dx: 200, dy: 0)
```

となります。

208 ページ

また、offset は offsetInPlace に変更されました。そのため、例えば 08/DelegateButton-06 の

```
r.offset(dx: r.size.width + 10, dy: 0)
```

は

```
r.offsetInPlace(dx: r.size.width + 10, dy: 0)
```

となります。

257 ページ

加えて、rectByInsetting は insetBy に変更されました。そのため、例えば 10/Pallet-06-indicator の

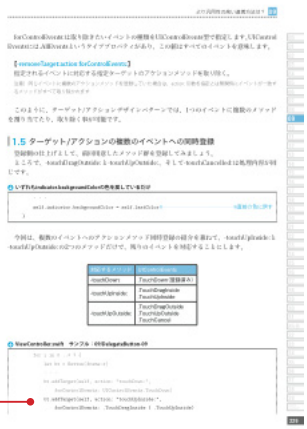
```
self.indicator.frame = r.rectByInsetting(dx: -3, dy: -3)
```

は

```
self.indicator.frame = r.insetBy(dx: -3, dy: -3)
```

となります。

229 ページ



## UIControlEvents の複数指定方法が変更された

UIControlEvents の複数指定方法が変更され、OR ビット演算子を使うのではなく、Set というコレクション型で羅列するようになりました。そのため、例えば 09/DelegateButton-09 の

```
bt.addTarget(self, action: "touchUpInside:",
        forControlEvents: .TouchDragInside | .TouchUpInside)
bt.addTarget(self, action: "touchUpOutside:",  forControlEvents:
        .TouchDragOutside | .TouchUpOutside | .TouchCancel)
```

は

```
bt.addTarget(self, action: "touchUpInside:",
        forControlEvents: [.TouchDragInside, .TouchUpInside])
bt.addTarget(self, action: "touchUpOutside:",  forControlEvents:
        [.TouchDragOutside, .TouchUpOutside, .TouchCancel])
```

となります。

243 ページ

## UIView の init(coder:) が init?(coder:) となった

UIView の init(coder aDecoder: NSCoder) が変更され、戻り値がオプショナルとなりました。そのため、例えば 10/Pallet-00 の

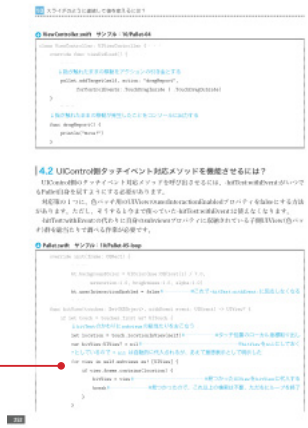
```
required init(coder aDecoder: NSCoder)
```

は

```
required init?(coder aDecoder: NSCoder)      ← nil が戻される場合がある
```

となります。

253 ページ



## UIView の subviews プロパティの型は [UIView] となった

UIView の subviews プロパティの型は [UIView] となったので、例えば 10/Pallet-05-loop の

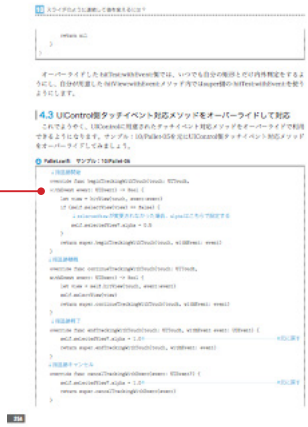
```
            for view in self.subviews as! [UIView] {
```

は

```
            for view in self.subviews {          ← as! [UIView] は不要
```

となります。

254 ページ



## beginTrackingWithTouch 〜 endTrackingWithTouch の引数の型が変更された

beginTrackingWithTouch 〜 endTrackingWithTouch の event: 引数の型が UIEvent から UIEvent? に変更されました。そのため 10/Pallet-06 の

```
    override func beginTrackingWithTouch(touch: UITouch,
        withEvent event: UIEvent) -> Bool {
```

が

```
    override func beginTrackingWithTouch(touch: UITouch,
        withEvent event: UIEvent?) -> Bool {← event はオプショナルとなった
```

となります。これに合わせ

255 ページ

```
    func hitView(touch: UITouch, event: UIEvent) -> UIView? {
```

を

```
    func hitView(touch: UITouch, event: UIEvent?) -> UIView? {
                                          ↑ 受け取る側も型を合わせる
```

とします。

266 ページ

## CALayer の sublayers プロパティの型は [CALayer]? となった

CALayer の sublayers プロパティは [CALayer]? となったので、11/Pallet-07-alt の

```
        for layer in self.layer.sublayers as! [CALayer] {
```

は

```
        if let sublayers = self.layer.sublayers {      ←アンラップの確認が必要
            for layer in sublayers {      ← as! [CALayer] は不要
```

もしくは、self.layer.sublayers が存在することがはっきりしているなら、次のように、if let を使わず、直接　！　でアンラップしたものを使ってもいいでしょう。

```
        ~~if let sublayers = self.layer.sublayers {~~
        for layer in self.layer.sublayers! {              ←確認なしのアンラップ
```

286 ページ

## UIButton.buttonWithType は UIButton(type:) に変更された

UIButton.buttonWithType は UIButton(type:) に変更されました。そのため 12/Switch-06 の

```
let bt = UIButton.buttonWithType(.System) as! UIButton
```

は

```
let bt = UIButton(type: .System)
```

となります。

292 ページ

## CALayerDelegate の drawLayer は引数の型が変更された

CALayerDelegate の drawLayer は引数の型が変更されました。
そのため 13/CustomLayer-01 の

```
override func drawLayer(layer: CALayer!, inContext ctx: CGContext!) {
```

は

```
override func drawLayer(layer: CALayer, inContext ctx: CGContext) {
        layer、ctx ともアンラップ済みオプショナルではなくなった
```

となります。

296 ページ

## CALayer の drawInContext は引数の型が変更された

CALayer の drawInContext は引数の型が変更されました。
そのため 13/CustomLayer-03 の

```
override func drawInContext(ctx: CGContext!) {
```

は

```
override func drawInContext(ctx: CGContext) {
        アンラップ済みオプショナルではなくなった
```

となります。

301 ページ

## CGGradientDrawingOptions の複数指定方法が変更された

CGGradientDrawingOptions の複数指定方法が変更され、OR ビット演算子を使うのではなく、Set というコレクション型で羅列するようになりました。そのため 13/CustomLayer-05 の

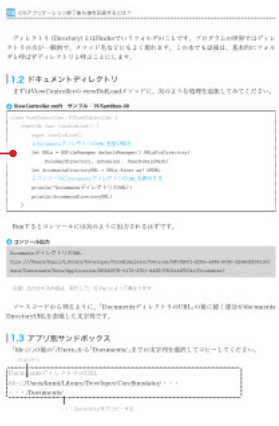
```
CGContextDrawLinearGradient(context, gradient, startPoint, endPoint,
    CGGradientDrawingOptions(
            CGGradientDrawingOptions.DrawsBeforeStartLocation
            | CGGradientDrawingOptions.DrawsAfterEndLocation
    )
)
```

は

```
CGContextDrawLinearGradient(context, gradient, startPoint, endPoint,
    [.DrawsBeforeStartLocation, .DrawsAfterEndLocation]
)
```

となります。

388 ページ



## NSFileManager の URLsForDirectory の戻り値の型が変更された

NSFileManager の URLsForDirectory の戻り値の型が [NSURL] となりました。そのため、例えば 19/Sandbox-00 の

```
let URLs = NSFileManager.defaultManager().URLsForDirectory・・・
let documentsDirectoryURL = URLs.first as! NSURL
```

は

```
let URLs = NSFileManager.defaultManager().URLsForDirectory・・・
let documentsDirectoryURL = URLs.first　← as! NSURL は不要
```

となります。

ただし as! NSURL としないので、そのままだと documentsDirectoryURL はラップされた状態となり、利用する時は次のようにアンラップの必要があります。

```
let URLs = NSFileManager.defaultManager().URLsForDirectory・・・
if let documentsDirectoryURL = URLs.first {　←アンラップとチェック
    アンラップが成功した時の処理
}
```

また、guard キーワードによりアンラップが失敗した時に、処理を続けずに終わらせる書き方ができるようになりました。そのため

```
let URLs = NSFileManager.defaultManager().URLsForDirectory・・・
guard let documentsDirectoryURL = URLs.first else {
    失敗した時の処理
}
アンラップが成功した時の処理
```

とも書け、if let 文の中に長々と処理を書かなくて済むようになっています。

406 ページ

## UIImagePNGRepresentation の戻り値の型が変更された

UIImagePNGRepresentation は NSData? を返すようになったので、使う時はアンラップする必要があります。そのため 20/Imaging-01 の

```
let imageData = UIImagePNGRepresentation(image)
    ・・・
imageData.writeToURL(fileURL, atomically:true)
```

は

```
let imageData = UIImagePNGRepresentation(image)
    ・・・
imageData?.writeToURL(fileURL, atomically:true)
                ↑ imageData が有効な時だけ実行
```

となります。

補足）本文やサンプルでは、createImage が成功する前提で記述していますが、これも厳密には成功したかどうかチェックするべきです。Swift 2 から追加された guard 文を使い

```
let image = self.createImage(CGSize(width: 100, height: 100))
let imageData = UIImagePNGRepresentation(image)
```

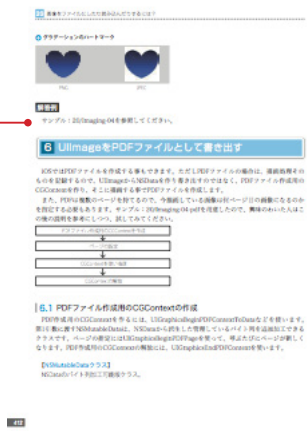
ではなく

```
    guard let image = self.createImage(CGSize(width: 100, height: 100)) else {
        必要なら、失敗時の後始末をここに記述する。
        return
    }
    let imageData = UIImagePNGRepresentation(image)
```

とした方が、より安全なプログラムとなります。

## UIGraphicsGetCurrentContext の戻り値の型が変更された

412 ページ



UIGraphicsGetCurrentContext は CGContext? を返すようになったので、これに合わせ、ダウンロードサンプル 20/Imaging-04 の

```
func drawHeart(ctx:CGContextRef) {
        ・・・
func fillGradation(context:CGContextRef, startPoint:CGPoint, endPoint:CGPoint) {
```

は

```
func drawHeart(ctx:CGContext?) {
        ・・・
func fillGradation(context:CGContext?, startPoint:CGPoint, endPoint:CGPoint) {
```

としました。
注意）CGContextRef と CGContext は同じ意味です。本書では CGContextRef を使っている部分もありますが、修正に合わせ CGContext を使うようにします。

430 ページ

## 例外処理が追加された

Swift 2 では例外処理が記述できるようになりました。例外処理は

```
do {
        try 処理
} catch let error as NSError {      投げられた例外の受け取り
        例外発生時の処理（error に例外の理由が入る）
}
```

といった書き方になります。この変更に合わせ、エラーを引数で受け取っていたメソッドは、引数からエラー項目が排除され、do catch でエラーを受け取るようになりました。このため、22/Filename-01 の

```
var error:NSError?
let result = NSFileManager.defaultManager().createDirectoryAtURL(
        imageDirURL, withIntermediateDirectories: true, attributes: nil, error: &error)
if (result == false) {
    NSLog(" ディレクトリ作成 (%d):%@", error!.code, error!.localizedDescription)
}
```

は

```
do {
    try NSFileManager.defaultManager().createDirectoryAtURL(
        imageDirURL, withIntermediateDirectories: true, attributes: nil)
                                                                    ↑
                ↓   引数から error: 部が無くなり catch で受け取るようになった
} catch let error as NSError {
    NSLog(" ディレクトリ作成 (%d):%@", error.code, error.localizedDescription)
}
```

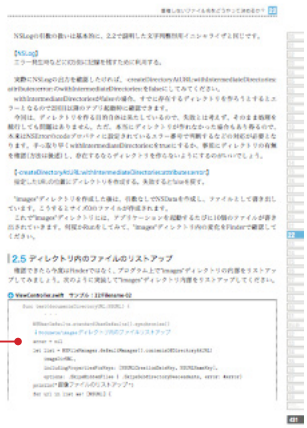
となります。

NSFileManager の contentsOfDirectoryAtURL も例外を投げるようになったので、同じように対応しています。そして成功時は [NSURL] が戻るので as! [NSURL は不要となります。

そのため 22/Filename-02 の

```
error = nil
let list = NSFileManager.defaultManager().contentsOfDirectoryAtURL(
    imageDirURL,
    includingPropertiesForKeys: [NSURLCreationDateKey, NSURLNameKey],
    options: .SkipsHiddenFiles | .SkipsSubdirectoryDescendants, error: &error)
println(" 画像ファイルのリストアップ ")
for url in list as! [NSURL] {
```

431 ページ



```
        ・・・
    }
```

は

```
    do {
        let list = try NSFileManager.defaultManager().contentsOfDirectoryAtURL(
            imageDirURL,
            includingPropertiesForKeys: [NSURLCreationDateKey, NSURLNameKey],
            options: [.SkipsHiddenFiles, .SkipsSubdirectoryDescendants])
        print(" 画像ファイルのリストアップ ")
        for url in list {
                    ・・・
        }
    } catch let error as NSError {
        NSLog(" ディレクトリリストアップ (%d):%@",
                    error.code, error.localizedDescription)
    }
```

となります。

　また、例外処理とは関係ありませんが options: の指定も OR ビット演算子ではなく Set というコレクション型の形になります。

```
    let list = NSFileManager.defaultManager().contentsOfDirectoryAtURL(
        imageDirURL,
        includingPropertiesForKeys: [NSURLCreationDateKey, NSURLNameKey],
        options: .SkipsHiddenFiles | .SkipsSubdirectoryDescendants, error: &error)
```

↓

```
        let list = try NSFileManager.defaultManager().contentsOfDirectoryAtURL(
            imageDirURL,
            includingPropertiesForKeys: [NSURLCreationDateKey, NSURLNameKey],
            options: [.SkipsHiddenFiles, .SkipsSubdirectoryDescendants])
```

contentsOfDirectoryAtURL から戻された [NSURL] を調べる for url in list ループ内でも、NSURL の resourceValuesForKeys が例外を投げます。
そのため

```
    for url in list {
        if let dic = url.resourceValuesForKeys(
                [NSURLCreationDateKey, NSURLNameKey], error:nil) {
```

は

```
    for url in list {
        do {
            if let dic = url.resourceValuesForKeys(
                    [NSURLCreationDateKey, NSURLNameKey], error:nil) {
        } catch _ {      ←投げられた例外をすべて受け取る指定
            例外への対応
        }
```

といった記述が必要なのですが、こちらは try? キーワードを使い、例外が発生した時は nil を戻すようにし、if let 文と組み合わせて、有効な dic が得られたらという表現にしました。

```
    for url in list {
        if let dic = try? url.resourceValuesForKeys(
                [NSURLCreationDateKey, NSURLNameKey]) {
            let date = dic[NSURLCreationDateKey] as! NSDate
                ・・・
        }
}
```

例外処理の詳細は iBooks の The Swift Programming Language を参照してください。

433 ページ

## Array の sort は sortInPlace に変更された

Array の sort は sortInPlace になりました。そのため 22/Filename-02-alt

```
array.sort {
    $0.lastPathComponent < $1.lastPathComponent
}
```

は

```
array.sortInPlace {
    $0.lastPathComponent < $1.lastPathComponent
}
```

となります。

522 ページ

## UIViewController の supportedInterfaceOrientations の戻り値の型が変更された

UIViewController の supportedInterfaceOrientations の戻り値の型は Int から変更されて、UIInterfaceOrientationMask になりました。そのため 28/Thumbnail-09 の

```
override func supportedInterfaceOrientations() -> Int {
    return Int(UIInterfaceOrientationMask.All.rawValue)
}
```

はキャストが不要になり

```
override func supportedInterfaceOrientations()->UIInterfaceOrientationMask {
    return .All      ← Int へのキャストが不要
}
```

となります。

522 ページ

また、複数の向きをサポートする場合、OR ビット演算子の合成ではなく Set というコレクション型で羅列するようになり

```
override func supportedInterfaceOrientations() -> Int {
    return Int((UIInterfaceOrientationMask.Portrait | .LandscapeLeft).rawValue)
}
```

は

```
override func supportedInterfaceOrientations()->UIInterfaceOrientationMask {
//        return [.Portrait, .LandscapeLeft]      ← Set で羅列する
    return .All      ← Int へのキャストが不要
}
```

となります。

553 ページ

## UIView の setTranslatesAutoresizingMaskIntoConstraints がプロパティに変更された

UIView の setTranslatesAutoresizingMaskIntoConstraints はプロパティになりました。そのため 29/Autolayout-02 の

```
label.setTranslatesAutoresizingMaskIntoConstraints(false)
```

は

```
label.translatesAutoresizingMaskIntoConstraints = false
```

となります。

559 ページ

## 「3.9 サイズクラスの利用」で紹介している UIViewController の updateViewConstraints は自動的に呼ばれなくなった

iOS 9.0 から traitCollectionDidChange の後に updateViewConstraints が自動的に呼ばれなくなります。そのため 29/Autolayout-08 の

```
override func traitCollectionDidChange(previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)
    self.needUpdateConstraintsForTraitCollection = true
}
```

で、self.view に setNeedsUpdateConstraints が必要になり

```
override func traitCollectionDidChange(previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)
    self.needUpdateConstraintsForTraitCollection = true
    self.view.setNeedsUpdateConstraints()    ←追加
}
```

となります。

598 ページ

## UIAlertAction の handler: 引数のブロック側引数の型が変更された

UIAlertAction の handler: 引数のブロック側引数の型が変更になりました。UIAlertAction がアンラップ済みのオプショナルではなくなります。そのため 31/Alert-02 の

```
let action1 = UIAlertAction(title: " アクション 1", style: .Default ) {
    (_:UIAlertAction!) -> Void in
```

は

```
let action1 = UIAlertAction(title: " アクション 1", style: .Default ) {
    (_:UIAlertAction) -> Void in
                    ↑ UIAlertAction! から UIAlertAction となった
```

になります。

662 ページ

## UITableView の dequeueReusableCellWithIdentifier の戻り値の型が変更された

UITableView の dequeueReusableCellWithIdentifier の戻り値の型は UITableViewCell? となったので、34/Table-01 の

```
var cell:UITableViewCell! = tableView.dequeueReusableCellWithIdentifier(
    CellIdentifier) as? UITableViewCell
```

は

```
var cell:UITableViewCell! = tableView.dequeueReusableCellWithIdentifier(
    CellIdentifier)          ← as? UITableViewCell は不要
```

となります。

689 ページ

## UICollectionViewCell の selectedBackgroundView の型が変更された

UICollectionViewCell の selectedBackgroundView の型は UIView? となりました。そのため 34/Collection-03 の

```
self.selectedBackgroundView.backgroundColor = UIColor.blueColor()
```

は、アンラップが必要となり

```
self.selectedBackgroundView!.backgroundColor = UIColor.blueColor()
```

となります。もし selectedBackgroundView の存在が保証できない場合は、有効かどうかの確認もしてください。

705 ページ

## UITableView の indexPathForSelectedRow はプロパティに変更された

UITableView の indexPathForSelectedRow はプロパティとなりました。
そのため 35/Table-12

```
    if let indexPath = self.tableView.indexPathForSelectedRow() {
```

は

```
    if let indexPath = self.tableView.indexPathForSelectedRow {
                                        ↑プロパティになった
```

となります。

731 ページ

## UITextView の caretRectForPosition で渡す引数の型が変更された

UITextView の caretRectForPosition で渡す引数の型が、アンラップ済みのオプショナルではなくなったので、36/Table-15 の

```
    let r = memoView.caretRectForPosition(memoView.selectedTextRange?.start)
```

は

```
    var textPosition = UITextPosition()
    if self.memoView.selectedTextRange != nil {
        textPosition = self.memoView.selectedTextRange!.start
    }
    let r = self.memoView.caretRectForPosition(textPosition)
```

としました。

743 ページ

## AVAudioPlayer の init(contentsOfURL:) が例外を投げるようになった

AVAudioPlayer の init(contentsOfURL:) は例外を投げるようになりました。そのため 37/Camera-02 の

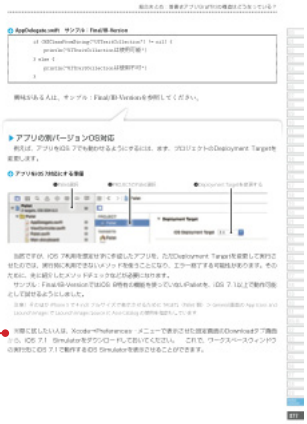
```
self.audioPlayer = AVAudioPlayer(contentsOfURL:soundURL, error:nil)
```

は try? キーワードを使い、例外発生時は nil が設定されるよう

```
self.audioPlayer = try? AVAudioPlayer(contentsOfURL:soundURL)
```

としています。

877 ページ



## 設定画面の Download リストから iOS 7 で動作する Simulator がなくなった

Xcode 7 では iOS 7 で動作する Simulator が設定画面の Download リストからなくなりました。

そのため Xcode 7 で簡単に違いを試せる iOS のバージョンは、iOS 8 と iOS 9 となります。

これに合わせ、-respondsToSelector: の使用例には、iOS 9 で使えて、iOS 8 で使えないメソッドやクラスを使うことにします。

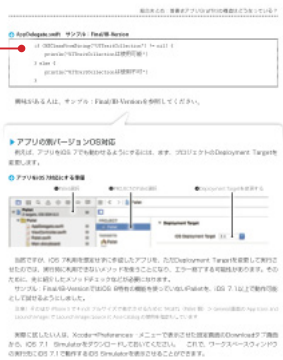
876 ページ

サンプル：Final/IB-Version では -restoreUserActivityState: メソッドのかわりに

```
if self.respondsToSelector("restoreUserActivityState:") {
    ・・・
```

-pressesBegan:withEvent:（iOS 9 から使用可能）を調べることにし

```
if self.respondsToSelector("pressesBegan:withEvent:") {
    ・・・
```



UITraintCollection クラスのかわりに

```
if (NSClassFromString("UITraintCollection") != nil) {
    ・・・
```

UIPressesEvent クラス（iOS 9 から使用可能）を調べることにします。

```
if (NSClassFromString("UIPressesEvent") != nil) {
    ・・・
```

また、Swift 1.2 までは、設定した Deployment Target のバージョンでは利用できないメッセージも、記述することが許されていました。

そのため、-respondsToSelector: でメソッドの存在を確認して、存在すれば使うといった記述ができたのですが、この記述は Swift 2 からはエラーになります。

例えば、iOS 9 以降の UIViewController では、viewIfLoaded プロパティを使い、自身の view が、すでに作成済みかどうか確認できるようになったのですが、これを利用するために次のように記述すると、Swift 2 では Deployment Target を iOS 9 以上にしない限りエラーになります。

サンプル：Final/IB-Version

```
class AppDelegate: UIResponder, UIApplicationDelegate {
        ・・・
    if (NSClassFromString("UIPressesEvent") != nil) {
        ・・・
    if let rootViewController = self.window?.rootViewController {
        if rootViewController.respondsToSelector("viewIfLoaded") {
            if rootViewController.viewIfLoaded != nil {
                ↑ Deployment Target を iOS 9 以上にしておかないとエラーになる
                print("rootViewController の view はロード済み ")
            } else {
                print("rootViewController の view は、まだロードされていない ")
            }
        } else {
            print("rootViewController の view はロード済みかどうかはわからない ")
        }
    }
```

注意）Objective-C と互換性のあるクラスのプロパティのゲッター側は、プロパティ名を指定することで -respondsToSelector で調べることができます。

Swift 2 では、これを回避するために、次のように #available を使い、指定バージョン以降でしか利用されないソースエリアを指定します。

```
if let rootViewController = self.window?.rootViewController {
    if rootViewController.respondsToSelector("viewIfLoaded") {
        if #available(iOS 9.0, *) {
            if rootViewController.viewIfLoaded != nil {
                print("rootViewController の view はロード済み ")
            } else {
                print("rootViewController の view は、まだロードされていない ")
            }
        }
    } else {
        print("rootViewController の view はロード済みかどうかはわからない ")
    }
}
```

iOS 9 以降でしか有効にならないエリア

また、viewIfLoaded のようにバージョンによって存在が保証されるプロパティやメソッドなら -respondsToSelector: を使わずに、次のように #available のバージョンによる振り分けをそのまま利用したほうがすっきりします。サンプル: Final/IB-Version では、こちらの記述にしています。

```
if let rootViewController = self.window?.rootViewController {
    if #available(iOS 9.0, *) {   ← iOS 9 なら viewIfLoaded は使える
        if rootViewController.viewIfLoaded != nil {
            print("rootViewController の view はロード済み ")
        } else {
            print("rootViewController の view は、まだロードされていない ")
        }
    } else {
        print("rootViewController の view はロード済みかどうかはわからない ")
    }
}
```

そのほか、guard 〜 else を使った記述も可能です。

詳しくは iBooks:The Swift Programming Language (Swift 2.1) の Control Flow の Checking API Availability を参照してください。

## iOS 9 では、ABAddressBookAddRecord 関数が非推奨となる

これは、ダウンロードできるサンプルの Xcode 7 への対応とは直接関係ありませんが、情報として追記しておきます。

直接 UnmanagedObject の例として、PS/UnmanagedObject で利用している ABAddressBookAddRecord 関数は、iOS Deplyment Target 8.3 では利用可能ですが、iOS Deplyment Target 9.0 からは廃止となります。代わりに CNContactStore や CNSaveRequest を使うようです。

以上が Xcode 7（Swift 2）への対応の説明です。
お疲れさまでした。