

iOS avec Swift

J. ROMAGNY

I. BASES DU LANGAGE SWIFT	3
1. VARIABLES ET CONSTANTES	5
<i>Variables</i>	5
<i>Constantes</i>	5
« <i>Optionals</i> »	5
2. CONDITIONS	6
3. BOUCLES	6
4. FONCTIONS.....	6
5. « COLLECTIONS »	7
a. <i>Tableaux</i>	7
b. <i>Dictionnaire</i>	8
c. « <i>Tuple</i> »	9
6. ENUM.....	9
7. STRUCTURES	9
8. OBJETS	10
a. <i>Héritage</i>	10
b. <i>Protocol</i>	11
9. EXTENSION	11
10. GENERICS.....	11
I. IOS.....	12
1. INSTALLATION	12
2. CREER UN NOUVEAU PROJET	13
3. « VIEWCONTROLLER » COTE « STORYBOARD »	18
a. <i>Différents « ViewControllers » pouvant être ajoutés au Storyboard</i>	18
b. « <i>Segues</i> » (<i>Navigation entre « ViewControllers »</i>)	19
Segue	20
Définir le « ViewController » initial.....	20
Ajouter la barre de navigation	21
Ajouter des boutons à la barre de navigation (Bar Button Item)	21
Navigation à partir de cellule de TableViewController	22
c. <i>Améliorer l'expérience utilisateur</i>	23
Assets (images.xcassets)	23
LaunchScreen (* .xib).....	23
ImageView	24
Définir le background d'une « *View » en code	24
First Responder	25
Auto Layout.....	26
4. « VIEWCONTROLLER » COTE CODE	29
a. <i>Ajout de la classe du « ViewController »</i>	29
b. « <i>UIViewController</i> ».....	30
c. « <i>TableViewCell</i> » <i>Affichage de données sous forme de table</i>	31
Cellules avec titre et sous-titre	32
Plusieurs sections.....	33
Disclosure.....	34
Cellules avec images	34

Custom Cell	35
Suppression de lignes	36
d. <i>Passage de données entre ViewControllers (« PrepareForSegue »)</i>	37
e. <i>Création d'Outlets</i>	38
f. <i>Création d'une Action</i>	39
g. <i>Activity Indicator</i>	41
5. CREATION DE MODELE ET DE SERVICE	41
6. PERSISTANCE DE DONNEES ENTRE SESSIONS	42
7. IPAD (TABLETTE)	43
a. <i>SplitView avec iPad</i>	43
b. <i>« Popup »</i>	44

I. Bases du langage Swift

Pourquoi Swift ?

- Swift se veut **plus simple** à apprendre **qu'Objective-C**... Sans doute que c'était un frein pour le développement sur iOS. Désormais (un peu comme en .NET) on peut choisir son langage : Swift ou Objective-C.
- **Moderne**
- **Productif**

Quelques particularités :

- **Parenthèses** (conditions) et « ; » en fin d'instructions ne sont **pas obligatoires**
- Attention aux **espaces** qui peuvent être synonymes d'erreur de syntaxe
- Encourage l'utilisation de constantes (mot-clé « **let** »)

Pour tester Swift il est possible de créer un Playground (Ecran d'accueil) ou menu « File » ... « New » ... « Playground »



[Documentation Swift](#)

[UIKit Reference](#)

[AppKit reference](#)

[Foundation framework reference](#)

Aide depuis XCode

Maintenir la touche « options » (à gauche de la touche « command »)

var label = UILabel()

Declaration `class UILabel : UIView, NSCoding`

Description The UILabel class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface. The base UILabel class provides support for both simple and complex styling of the label text. You can also control over aspects of appearance, such as whether the label uses a shadow or draws with a highlight. If needed, you can customize the appearance of your text further by subclassing.

Availability iOS (2.0 and later)

Declared In UILabel.h

Reference [UILabel Class Reference](#)

En cliquant sur les liens on est redirigé vers la documentation ...

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help lun. 08:43

Documentation — UILabel Class Reference

UILabel Class Reference

Overview

Tasks

- Accessing the Text Attributes
 - text
 - attributedString
 - font
 - textColor
 - textAlignment
 - lineBreakMode
 - enabled
- Sizing the Label's Text
 - adjustsFontSizeToFitWidth
 - adjustsLetterSpacingToFit...
 - baselineAdjustment
 - minimumScaleFactor
 - minimumFontSize
 - numberOfLines
- Managing Highlight Values
 - highlightedTextColor
 - highlighted
- Drawing a Shadow
 - shadowColor
 - shadowOffset
- Drawing and Positioning Ove...
 - textRectForBounds:limited...
 - drawTextInRect:
- Setting and Getting Attributes
 - userInteractionEnabled
- Getting the Layout Constraints
 - preferredMaxLayoutWidth

UILabel

Inherits from: None
Conforms to: None
Framework: UIKit in iOS 2.0 and later. [More related items...](#)

The UILabel class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface. The base UILabel class provides support for both simple and complex styling of the label text. You can also control over aspects of appearance, such as whether the label uses a shadow or draws with a highlight. If needed, you can customize the appearance of your text further by subclassing.

The default content mode of the UILabel class is `UIViewContentModeRedraw`. This mode causes the view to redraw its contents every time its bounding rectangle changes. You can change this mode by modifying the inherited `contentMode` property of the class.

New label objects are configured to disregard user events and clip subviews by default. If you want to handle events in a custom subclass of UILabel, you must explicitly change the value of the `userInteractionEnabled` property to YES after initializing the object. If you want to allow subviews to extend beyond the bounds of a label, you must explicitly change the value of the label's `clipsToBounds` property to NO.

For information about basic view behaviors, see [View Programming Guide for iOS](#).

For more information about appearance and behavior configuration, see [Labels in UIKit User Interface Catalog](#).

Accessing the Text Attributes

Copyright © 2015 Apple Inc. All rights reserved. [Terms of Use](#) | [Privacy Policy](#) | Updated: 2015-03-09

Feedback

Quick Help Inspector : Menu « View » ... « Utilities » ... « Show Quick Help Inspector »

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help lun. 08:47

MyPlayground.playground — Edited

MyPlayground.playground : No Selection

```
// Playground - noun: a place where people can play
import UIKit

var label = UILabel()
```

UILabel

Quick Help

Declaration `class UILabel : UIView, NSCoding`

Description The UILabel class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface. The base UILabel class provides support for both simple and complex styling of the label text. You can also control over aspects of appearance, such as whether the label uses a shadow or draws with a highlight. If needed, you can customize the appearance of your text further by subclassing.

Availability iOS (2.0 and later)

Declared In UILabel.h

Reference [UILabel Class Reference](#)

Guides [Text Programming Guide for iOS](#)

Sample Code [AdvancedURLConnections](#), [MultipeerGroupChat](#), [RosyWriter](#), [Teslameter](#), [URLCache](#)

1. Variables et constantes

Variables

```
var myString : String = "Bonjour"
var myInt : Int = 10
var myDouble : Double = 9.99
var myBool : Bool = true
```

Ou

```
var myString = "Bonjour"
var myInt = 10
var myDouble = 9.99
var myBool = true
```

Afficher rapidement une sortie avec « **println** »

```
println(myString)
```

Formatage et concaténation

```
var name = "Jérôme"
"Bonjour \"(name)\""
```

Concaténation

```
"Bonjour " + name
```

Conversion du type de données

Méthodes : Int(), Double(), Float(), etc.

```
String(myInt)
```

Constantes

```
let myConst = 100
```

« Optionals »

Ce sont en fait l'équivalent des Nullables, c'est-à-dire des variables pouvant accepter une valeur nulle.

```
var myInt : Int?
```

On ajoute « ? » au type

```
if myInt != nil {
```

Tester si la variable est nulle

```
}
```

2. Conditions

On pourrait ajouter les parenthèses mais elles ne sont pas nécessaires avec Swift

```
if myInt > 10 {
} else if myInt == 10 {
}
else {
}
}
```

+ Utilisation de && et ||

Switch

```
switch myInt {
case 0 :
    break
case 1...10 :
    break
default :
    break
}
```

Opérateur « range » ici de 1 à 10

. Possible également « ..> » et « ..< »

3. Boucles

Parenthèses pas obligatoires

```
while condition {
}
```

Boucle de 0 à 10

```
for index in 0...10 {
    println(index)
}
```

```
var fruits = ["Pomme", "Poire", "Pêche"]
for fruit in fruits {
}
```

4. Fonctions

```
func myFunction (name : String)
    println(name)
}
```

Nom du paramètre puis type

```
myFunction("Jérôme")
```

Type de retour

Avec **retour**

```
func myFunction (name : String) -> String {
    return "Bonjour \(name)"
}
```

```
var result = myFunction("Jérôme")
```

Paramètre avec valeur par défaut

```
func calc (a : Int, b : Int = 10) -> Int{
  return a + b
}

var res1 = calc (5)
var res2 = calc(5,b:20)
```

➔ Forcer à indiquer le nom des paramètres avec #

```
func myFunction (#name : String) {
  println(name)
}

myFunction(name : "Jérôme")
```

5. « Collections »

a. Tableaux

Créer un tableau

```
var fruits = [String]()
var fruits : [String] = []
```

Rempli à l'initialisation

```
var fruits = ["Pomme", "Poire", "Pêche"]
```

Parcours des éléments du tableau

```
for fruit in fruits {
}
```

Nombre d'éléments

```
println(fruits.count)
```

Savoir le tableau est vide

```
if fruits.isEmpty {
}
```

Ajout

```
fruits.append("Banane")
```

.. Ou

```
fruits += ["Kiwi"]
```

Accès aux éléments du tableau (lecture, modification)

```
println(fruits[0])
```

Suppression

```
fruits.removeLast()
fruits.removeAtIndex(0)
```

Tout supprimer

```
fruits.removeAll(keepCapacity: false)
```

.. Ou tableau vide

```
fruits = []
```

b. Dictionnaire

Créer un dictionnaire vide

```
var myDictionary = [Int:String] ()
var myDictionary : [Int:String] = [:]
```

Dictionnaire **rempli à l'initialisation**

```
var myDictionary = ["key1" : "value 1", "key2" : "value 2"]
```

Parcours des éléments

```
for (key,value) in myDictionary {
    println("clé : \{(key) et valeur : \{(value)")
}
```

Savoir si le **dictionnaire est vide**

```
if myDictionary.isEmpty {
}
```

Nombre d'éléments

```
println(myDictionary.count)
```

Ajout ou modification si existante

```
myDictionary[0] = "value 1"
```

Autre possibilité pour **modification**

```
myDictionary.updateValue("new value", forKey: 0)
```

Suppression

```
myDictionary.removeValueForKey(1)
```

Ou

```
myDictionary[1] = nil
```

Tout supprimer

```
myDictionary.removeAll(keepCapacity: false)
```

Ou dictionnaire vide

```
myDictionary = [:]
```

c. « Tuple »

Ce sont des ensembles (groupés entre parenthèses). « Tuple » pour quintuple, sextuple par exemple.

```
var name = "J. ROMAGNY"
var twitter = "@romagny13"

var myTuple = (name,twitter)
```

6. Enum

```
enum myEnum {
    case One
    case two
}
```

Utilisation

```
var useMyEnum : myEnum = myEnum.One
```

Ou

```
var useMyEnum = myEnum.One
```

Facilité d'écriture, utile pour les switch également

```
var useMyEnum : myEnum
useMyEnum = .One
```

7. Structures

```
struct User {
    var name : String
    var email : String
    static func myFunc () {
    }
}
```

Méthode « statique » déclarée avec
« static » dans une structure

```
let marie = User(name: "Marie Bellin", email: "mb3@hotmail.com")
```

8. Objets

Classe

```
class User {
  var name : String
  var email : String

  init() {
    name = ""
    email = ""
  }
  init(name : String, email : String) {
    self.name = name
    self.email = email
  }

  deinit {

  }

  func sayHello() -> String {
    return ("Bonjour \ \(name)")
  }
  class func doSomething() -> String {
    return ("...")
  }
}
```

Variables à initialiser soit dans le constructeur soit à la déclaration. Marquer les membres « private » pour les rendre inaccessible en dehors du fichier

Constructeurs. Accès aux membres avec « self »

Destructeur

Méthode « statique » accessible par le nom de la classe

Propriété readonly

```
private(set) var email : String = ""
```

Création d'un objet

```
var marie : User = User()
```

..Ou

```
var marie = User()
```

Avec constructeur

```
var marie = User (name : "Marie Bellin", email : "mb3@htomail.com")
```

Accès aux membres

```
var result = marie.sayHello()
```

Pas d'import de « headers » à faire avec Swift (se fait de manière « transparente »)

a. Héritage

Hérite de « User »

```
class Member : User {

  override init() {
    //
    super.init()
  }

  override func sayHello() -> String {
    return ("Bonjour cher membre!")
  }
}
```

Overrides du constructeur et d'une méthode, accès aux membres de la classe avec « super »

Utilisation. Ex

```
var alex = Member ()
alex.name = "Alex Prime"
var result = alex.sayHello()
```

b. Protocol

Définit les membres que la classe devra implémenter

```
protocol UserProtocol {
    var name : String { get }
    var email : String { get set }
    func sayHello()
}
```

```
class User : UserProtocol{
    var name : String = ""
    var email : String = ""

    func sayHello() {

    }
}
```

9. Extension

Exemple on crée une méthode d'extension pour la classe User

```
extension User {
    func sayBye() -> String{
        return ("Aurevoir \{(name)")
    }
}
```

```
var marie = User ()
marie.name = "marie"
marie.sayBye()
```

10. Generics

Un peu comme avec C#, on retrouve les « Generics »

[Documentation](#)

I. iOS

1. Installation

Besoins :

- **Mac OS X** (Maverick minimum) (ou [création d'une image virtuelle](#) avec VM Ware par exemple)
- **XCode 6.*** (free)

Aller sur l'App Store  , chercher « xcode » et l'installer



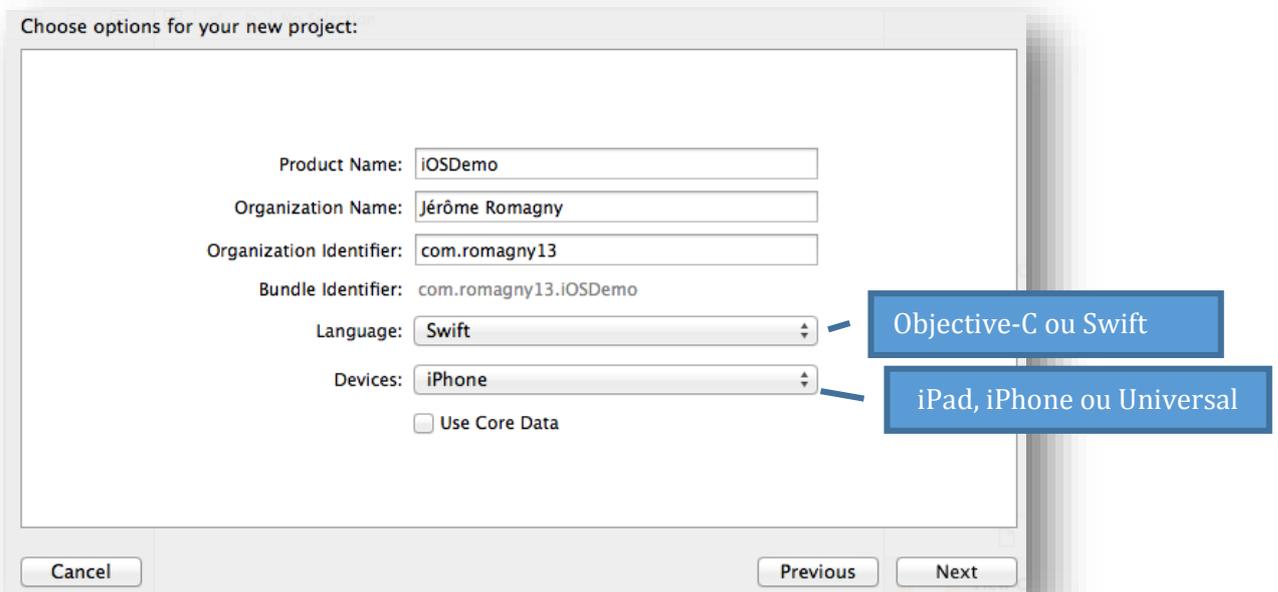
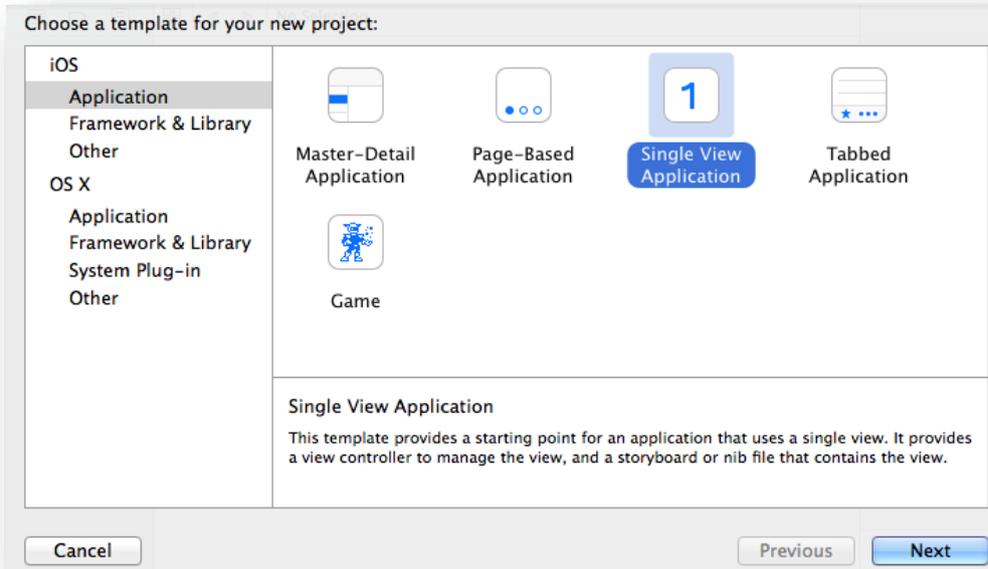
[S'inscrire au « Apple Developer Program »](#) pour pouvoir mettre ses applications sur l'App Store.

[Ressources](#)

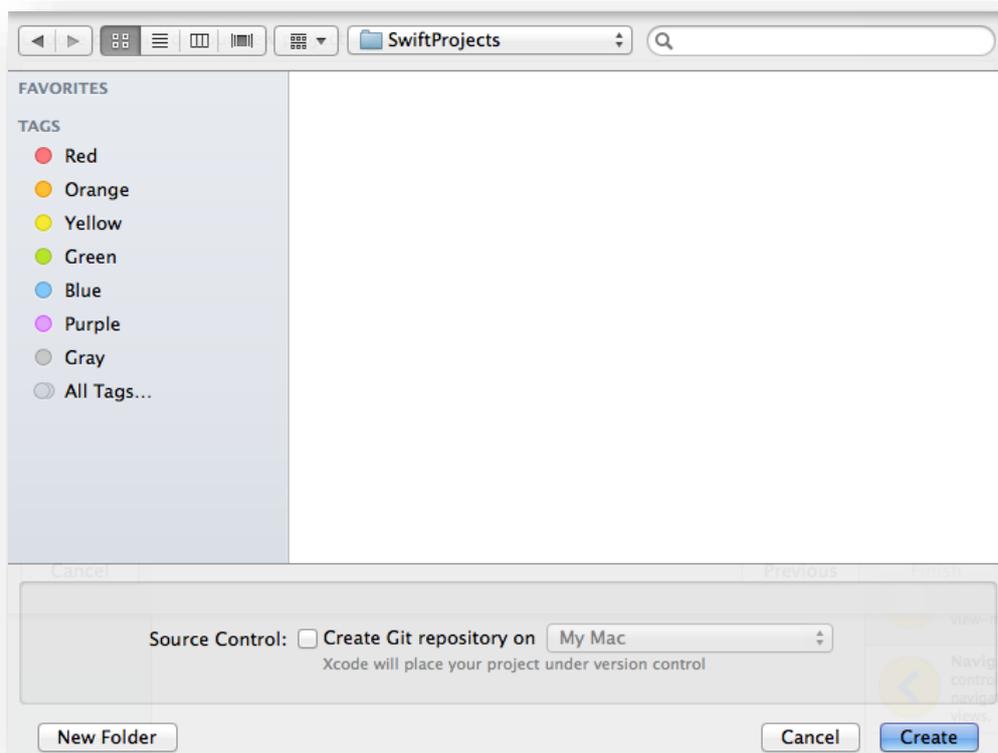
2. Créer un nouveau projet

Depuis l'écran de démarrage (« Create a new XCode Project ») ou depuis le menu « File » ... « New » .. « Project »

- « Single View Application » représente l'application de départ de base.



... choix du répertoire du projet ...



Le Storyboard permet de définir la structure et la navigation entre « ViewControllers ». C'est la partie « Visuelle ».

Les ViewControllers utilisent leur classe de base correspondante par défaut, si on veut ajouter son code au « ViewController » il faut créer sa propre classe et remplacer la classe de base du ViewController par celle-ci dans les propriétés du Storyboard

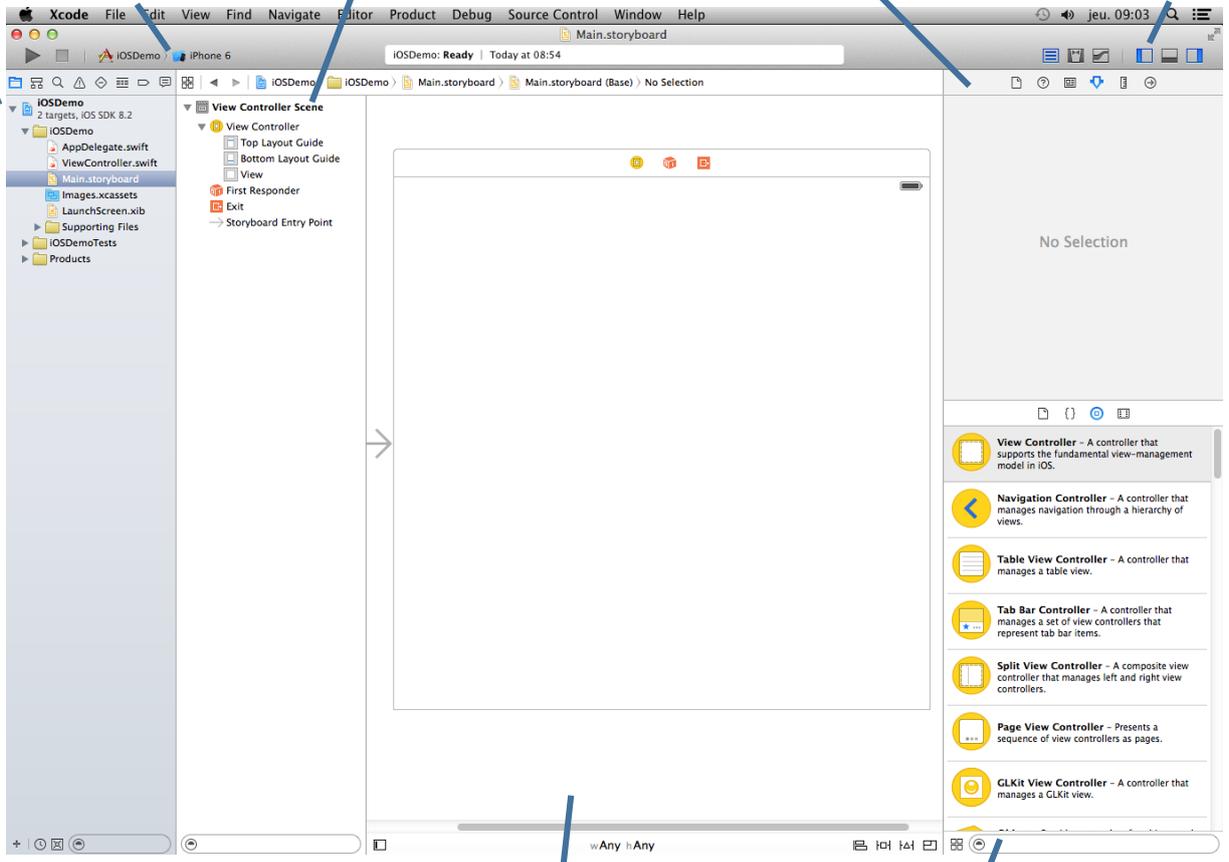
Organisation du projet

Débug et choix du device simulé

« Document Outline » du Storyboard

Propriétés de l'élément sélectionné

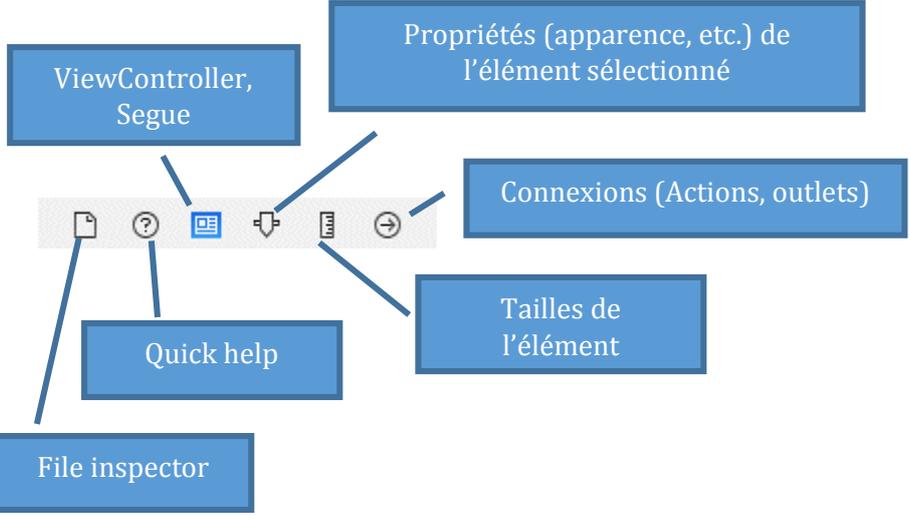
Boutons permettant d'afficher/masquer les différents panneaux et splitter la vue



Storyboard

Boîte à outils (« Object Library ») Eléments pouvant être ajoutés au Storyboard

Propriétés de la sélection



View Controller, Segue

Propriétés (apparence, etc.) de l'élément sélectionné

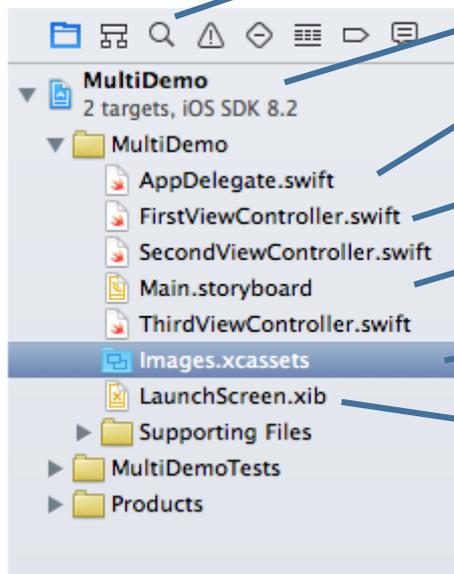
Connexions (Actions, outlets)

Quick help

Tailles de l'élément

File inspector

Organisation du projet



Vues projet et Debug

Propriétés du projet

Cycle de vie de l'application

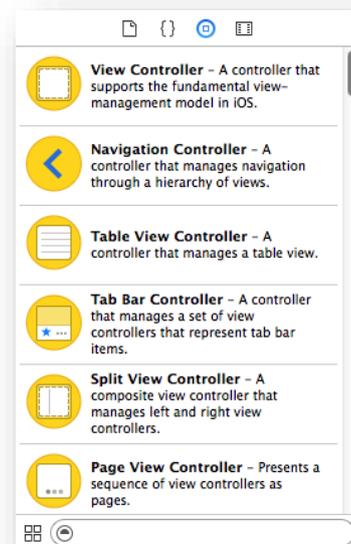
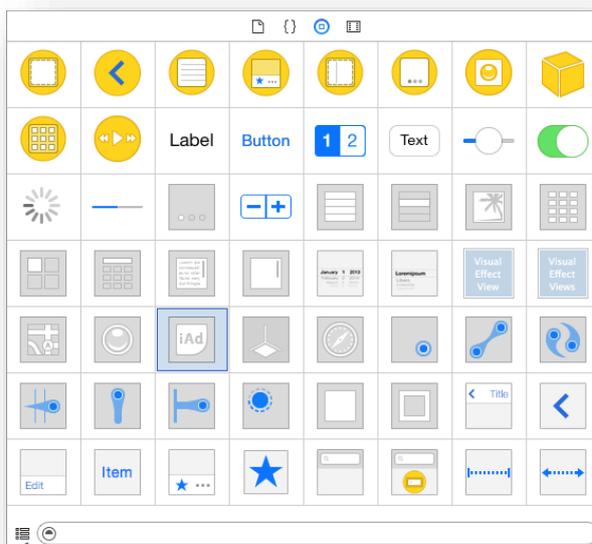
« ViewControllers » ayant besoin d'une classe personnalisée

Storyboard (« Visuel » + « segue »)

Assets (images, AppIcon, etc.)

Ecran de démarrage de l'application

Boite à outils (« Object Library ») : éléments à ajouter au Storyboard



Le bouton permet de basculer de la vue icônes à la vue liste

Propriétés du projet

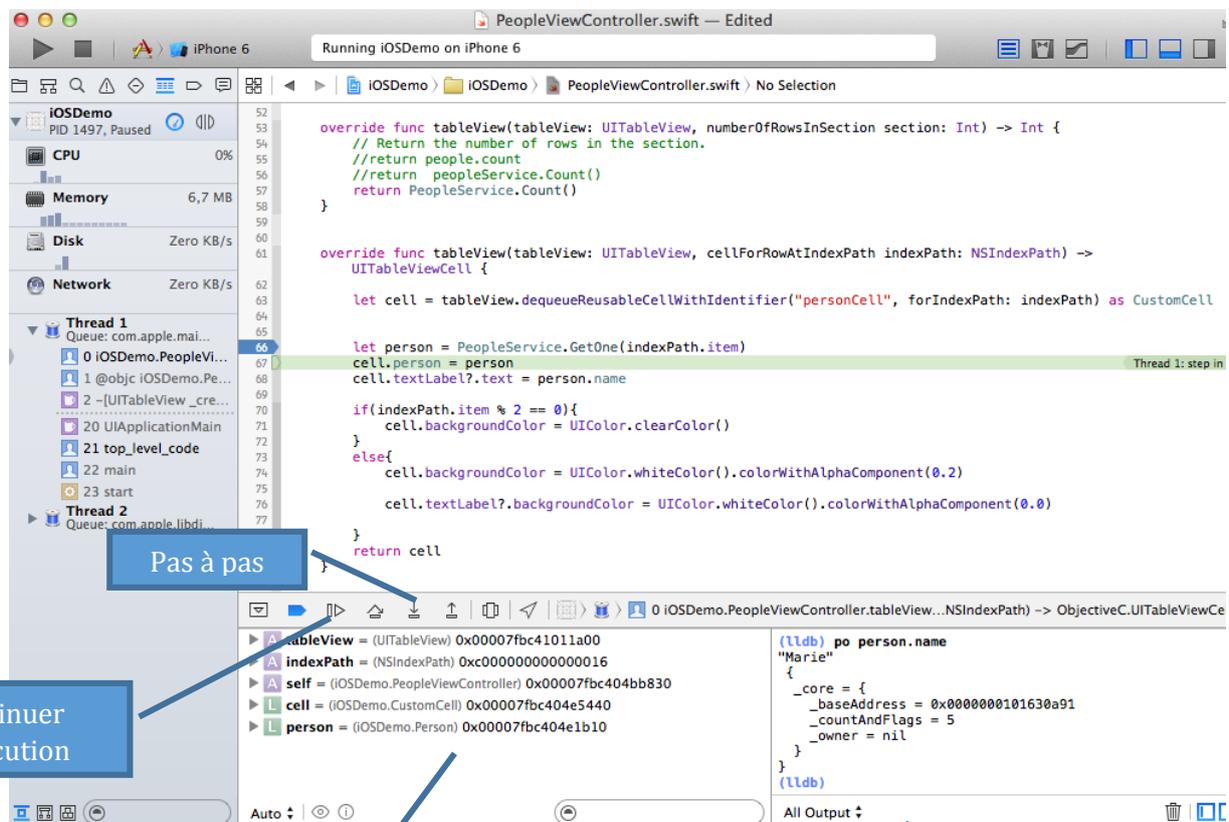
En cliquant sur la racine du projet on a accès aux propriétés du projet (version, device, launchscreen, capabilities, etc.)

iOS Simulator

Lorsque le simulator est lancé on a accès à un menu permettant par exemple de zoomer, scale (menu « Window »). Le menu Hardware permet de changer la version émulée, gérer le keyboard, etc.

Debug

On peut placer des points d'arrêt . Pour supprimer un point d'arrêt... clic droit dessus « Delete Breakpoint »



The screenshot shows the Xcode IDE with a Swift file named `PeopleViewController.swift` open. The code is paused at line 66, which is highlighted in green. A breakpoint is set at this line. The console shows the current state of variables:

```

tableView = (UITableView) 0x00007fbc41011a00
indexPath = (NSIndexPath) 0xc000000000000016
self = (iOSDemo.PeopleViewController) 0x00007fbc404bb830
cell = (iOSDemo.CustomCell) 0x00007fbc404e5440
person = (iOSDemo.Person) 0x00007fbc404e1b10
  
```

Thelldb console shows the command `(lldb) po person.name` and its output:

```

(lldb) po person.name
"Marie"
{
  _core = {
    _baseAddress = 0x0000000101630a91
    _countAndFlags = 5
    _owner = nil
  }
}
(lldb)
  
```

Annotations in blue boxes point to various elements:

- Pas à pas**: Points to the step-through button in the debugger toolbar.
- Continuer l'exécution**: Points to the continue button in the debugger toolbar.
- Variables, objets courants**: Points to the console window showing the current state of variables.
- Fenêtre de sortie On peut entrer la commande « po » suivi de la variable ou l'objet à obtenir**: Points to thelldb console window.

3. « ViewController » côté « Storyboard »

a. Différents « ViewControllers » pouvant être ajoutés au Storyboard

Repérables avec leur couleur « jaune » dans l'« Object Library »



View Controller – A controller that supports the fundamental view-management model in iOS.



Navigation Controller – A controller that manages navigation through a hierarchy of views.



Table View Controller – A controller that manages a table view.



Tab Bar Controller – A controller that manages a set of view controllers that represent tab bar items.



Split View Controller – A composite view controller that manages left and right view controllers.



Page View Controller – Presents a sequence of view controllers as pages.



GLKit View Controller – A controller that manages a GLKit view.



Object – Provides a template for objects and controllers not directly available in Interface Builder.



Collection View Controller – A controller that manages a collection view.



AVKit Player View Controller – A view controller that manages an AVPlayer object.



ViewController

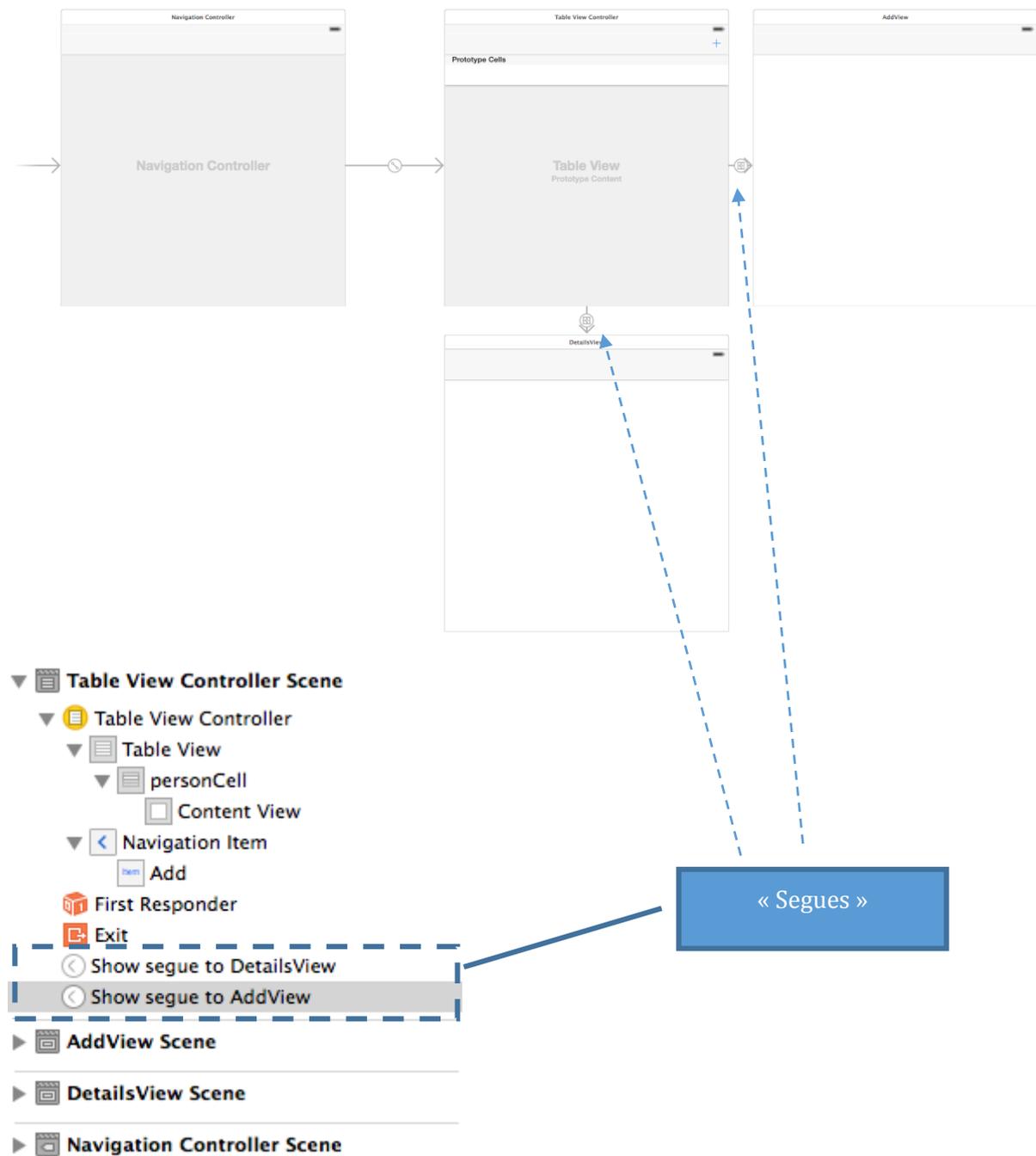
View, TableView, UIImageView, etc.

b. « Segues » (Navigation entre « ViewControllers »)

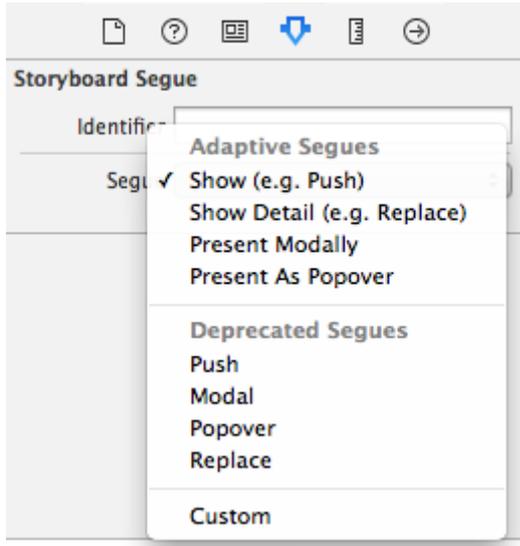
La navigation s'ajoute ainsi :

1. On maintient « ctrl » sur le contrôle (bouton par exemple)
2. on glisse la souris vers le « ViewController » vers lequel naviguer.

Une flèche avec un symbole est ajoutée entre les 2 « ViewControllers ». Il est possible également de réaliser cette opération depuis le panneau « Document Outline »



Segue



Show (e.g. Push), Show Detail (e.g. Replace)



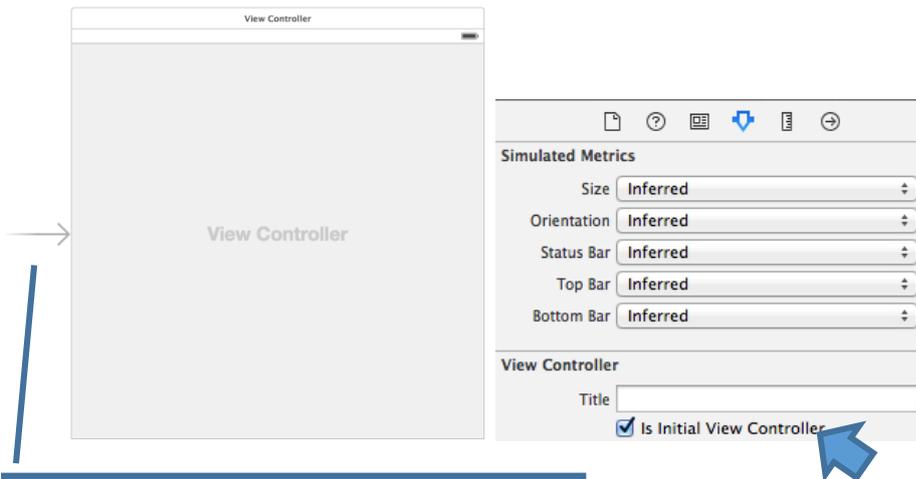
« Present Modally » Options : Presentation (Full Screen, etc.), Transition (effet)



« Present as Popover » Options Directions (up, left, etc.), Anchor

(Les autres sont marqués « Deprecated »)

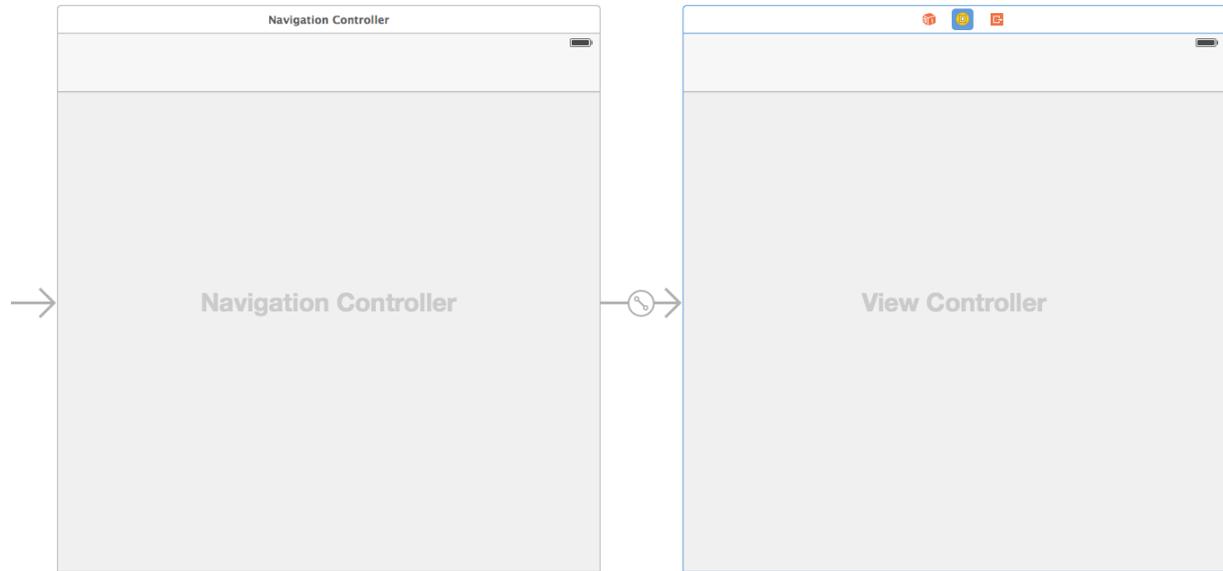
Définir le « ViewController » initial



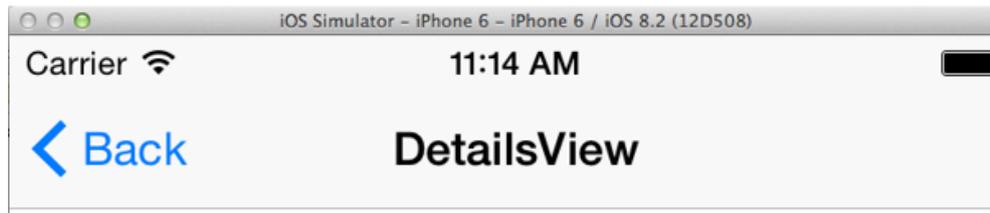
La flèche montre que c'est l'écran de départ de l'application. Cocher « is initial View Controller » dans les propriétés

Ajouter la barre de navigation

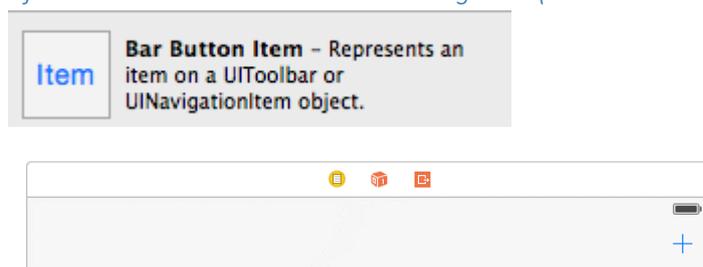
Sélectionner l'écran initial...puis menu « Editor » ... « Embed In » ... « Navigation Controller »



Actuellement la barre n'affichera qu'un bouton « retour » lorsque la navigation de retour sera possible.



Ajouter des boutons à la barre de navigation (Bar Button Item)

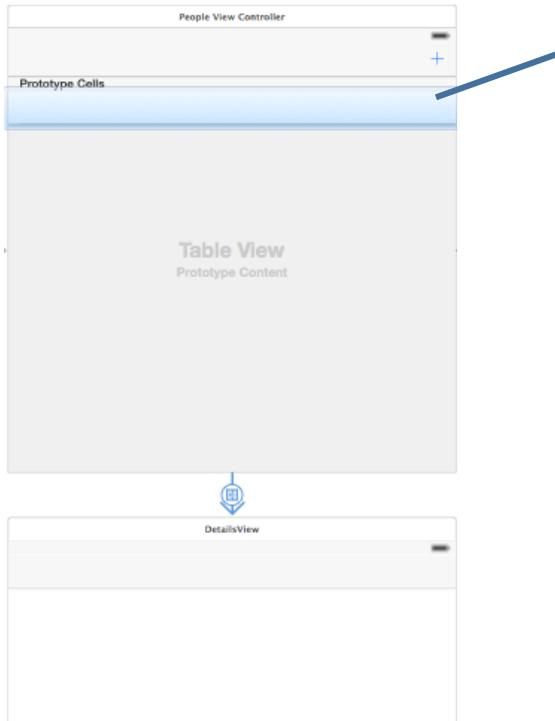


Ajouté à la barre de navigation (« Navigation Controller »). Puis on change la propriété « Identifier » (pour add ici) ce qui change l'apparence du bouton

Puis ajout de « Segues » glisser depuis le bouton vers le nouveau « ViewController » à afficher

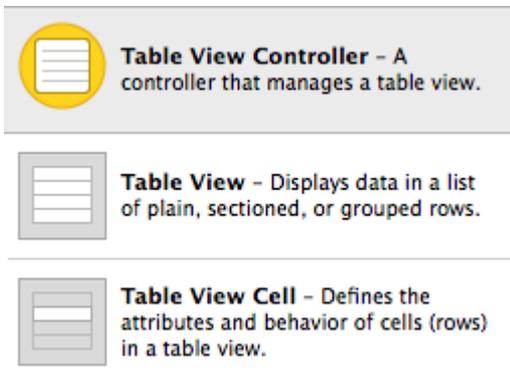
Navigation à partir de cellule de *TableViewController*

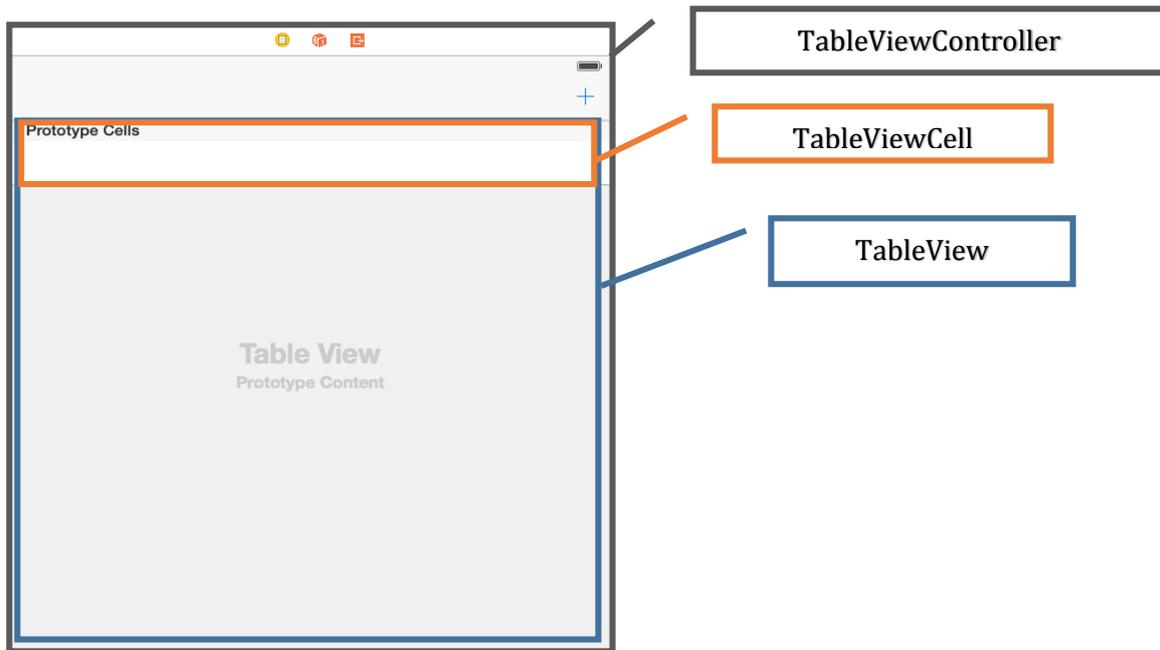
Navigation (« Segues ») de « Prototype Cell » vers un autre « ViewController »



Element « TableViewCell » ajouté par défaut avec un « TableViewController » mais pouvant être ajouté depuis la boîte à outils (« Object Library »)

Les 3 éléments importants de la boîte à outils pour les « TableViewControllers »



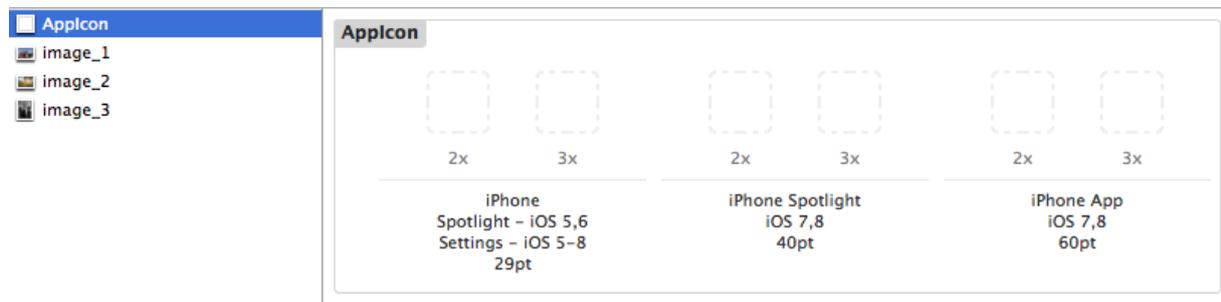


c. Améliorer l'expérience utilisateur

UI guidelines

Assets (images.xcassets)

- Il suffit de glisser déposer les images
- [AppIcon](#) : glisser une icône pour chaque emplacement selon les tailles



Retrouver et utiliser une image des assets :

```
var image = UIImage(named : "image_1")
```

LaunchScreen (.xib)*

Personnalisable, on peut ajouter des éléments, (exemple une UIImageView en fond), la font (couleur, famille, etc.)

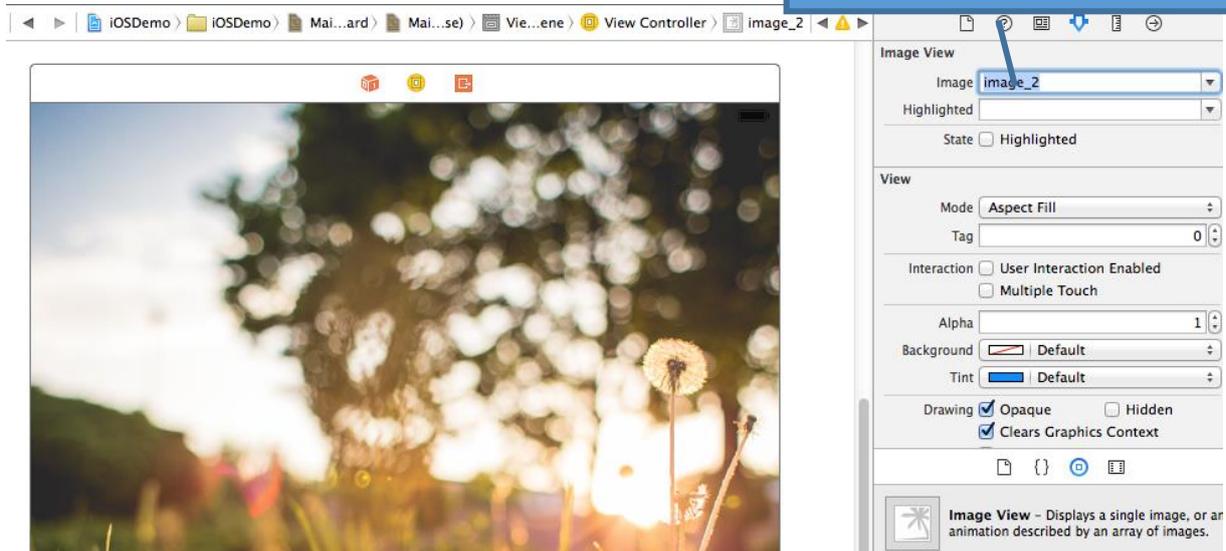


UIImageView



Image View – Displays a single image, or an animation described by an array of images.

On peut facilement sélectionner l'image affichée par l'« UIImageView » parmi les assets



Définir le background d'une « *View » en code



```

override func viewDidLoad() {
    super.viewDidLoad()

    var imageView = UIImageView(image : UIImage(named:"image_3"))
    imageView.contentMode = UIViewContentMode.ScaleAspectFill
    tableView.backgroundColor = imageView
}

override func viewWillAppear(animated: Bool) {
    super.viewWillAppear(animated)
    tableView.reloadData()
    navigationController?.navigationBar.alpha = 0.5
}

```

Image en fond et
Mode

Transparence Barre
de navigation

Cellules pour « UITableViewControllers »

```

override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath:
NSIndexPath) -> UITableViewCell {

    let cell = tableView.dequeueReusableCellWithIdentifier("personCell", forIndexPath:
indexPath) as CustomCell
    let person = PeopleService.GetOne(indexPath.item)
    cell.person = person
    cell.textLabel?.text = person.name

    if(indexPath.item % 2 == 0){
        cell.backgroundColor = UIColor.clearColor()
    }
    else{
        cell.backgroundColor = UIColor.whiteColor().colorWithAlphaComponent(0.2)

        cell.textLabel?.backgroundColor =
UIColor.whiteColor().colorWithAlphaComponent(0.0)
    }
    return cell
}

```

Cellules avec une cellule sur deux plus transparente que l'autre

First Responder

Faire perdre le focus à un contrôle. Exemple quand on valide (clic sur bouton) on fait prendre le focus à une boîte de texte

```

@IBAction func addPerson(sender: AnyObject) {

    PeopleService.Add(nameText.text, email: emailText.text)
    nameText.resignFirstResponder()
}

```

Cacher le clavier quand on « click » en dehors

```

override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {
    self.view.endEditing(true)
}

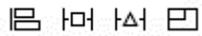
```

Si le clavier n'apparaît pas avec le Simulator → « **command + k** » (ou menu hardware du « Simulator »)

Auto Layout

Un peu comme un « site Web Responsive » ... On définit des « contraintes » pour que les éléments restent positionnés « correctement » selon les différentes résolutions et orientations de devices.

2 méthodes pour ajouter des contraintes : soit par le « Ctrl + glisser de la souris » soit depuis les boutons de raccourcis en bas du Storyboard

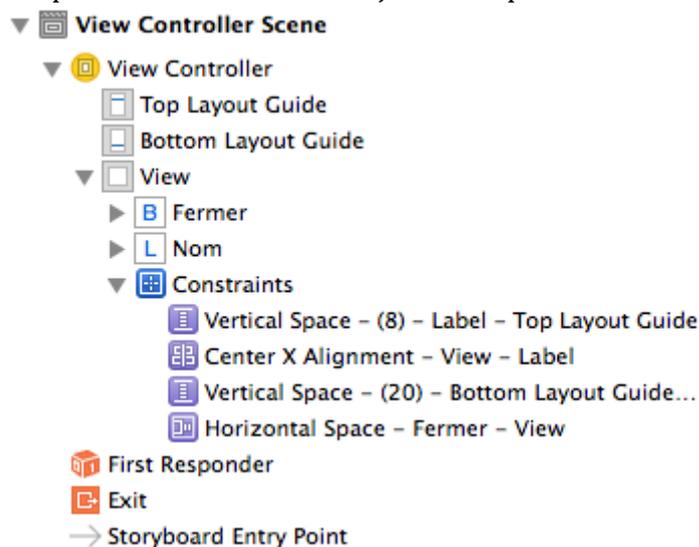


- Alignement
- Tailles
- Pour résoudre les problèmes de contraintes
- « Resizing »

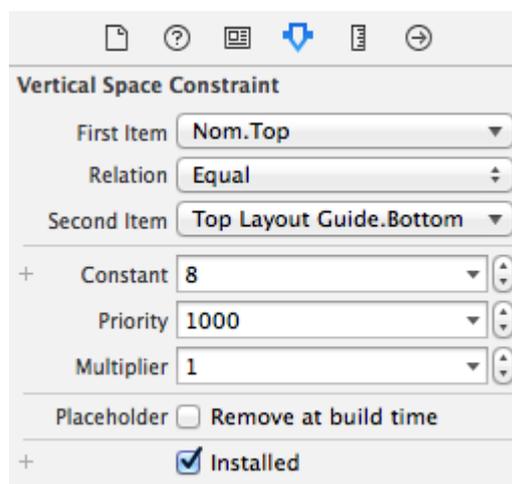
Les repères laissés peuvent avoir plusieurs couleurs :

- Bleu : ok
- Orange : avertissement, problème de contrainte
- Rouge : erreur

On peut voir les contraintes ajoutées depuis le « Document Outline »



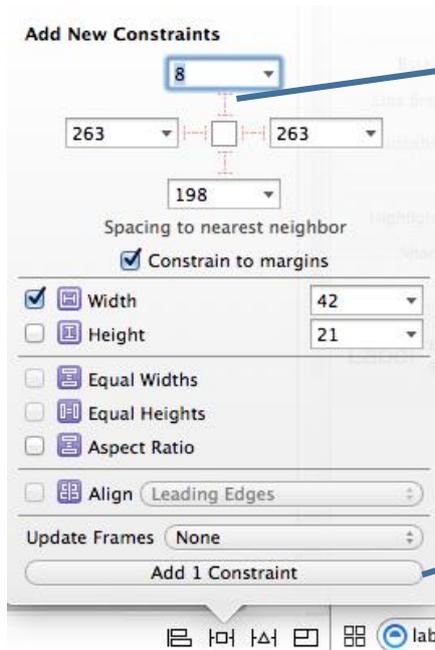
... et leur détail dans le panneau propriétés



Centrer un élément

Par ex pour centrer horizontalement :

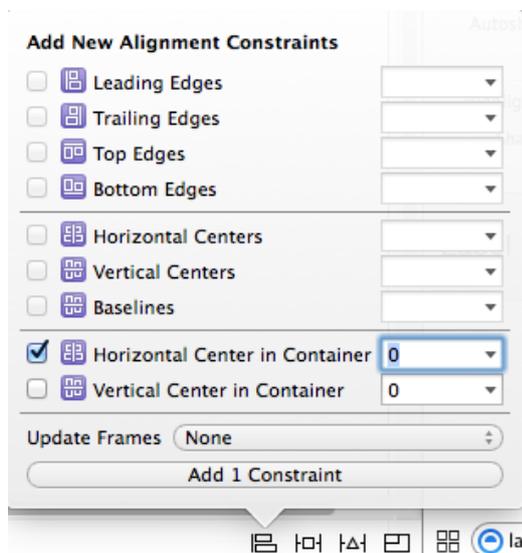
1. Ajouter une contrainte de taille (width)



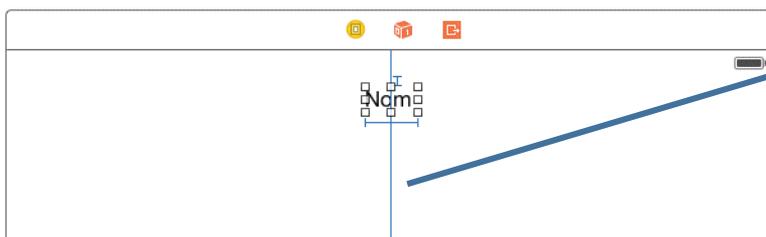
Les traits permettent de définir une marge fixe, ce que l'on fait aussi par « Ctrl + glisser »

Cliquer pour ajouter les contraintes

2. et une contrainte de marge (horizontal)



3. Et enfin contrainte par rapport au top. Maintenir « Ctrl » et glisser de la souris vers le top choisir « Top space to top layout guide »)



Les traits sont bleus, tout est ok. En cas de problème ils pourraient être orange (avertissement) voir rouge (erreur)

Ancrer un élément

Exemple un bouton en bas à droite



1. Ajouter une contrainte de taille (vu comme précédemment)
2. Ajouter une contrainte par rapport au bas (bottom). Toujours pareil maintenir « Ctrl » et glisser depuis le contrôle vers le bord. Choisir « Bottom space to bottom layout guide »
3. Ajouter une contrainte par rapport à droite. « Ctrl » + glisser vers le bord droit et choisir « trailing space to container margin »

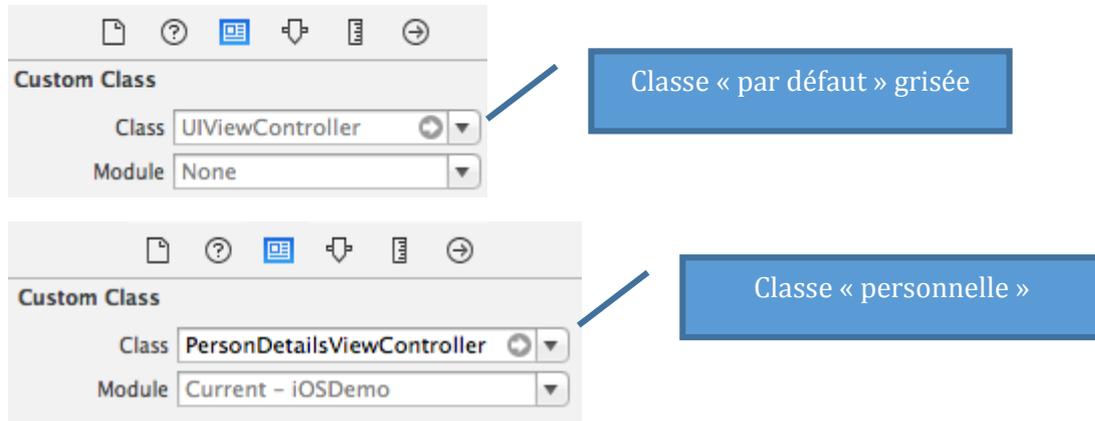
« **Add missing Constraints** » cette option peut être très utile pour ajouter automatiquement pour nous des contraintes. Par exemple on ajoute un label au centre de la vue, puis raccourci bouton « issues » en bas du storyboard (ou menu « Editor » ... « Resolve Auto Layout Issues ») et « Add missing constraints ».

Pour **tester** utiliser le menu « Hardware » avec le Simulator : tourner (« rotate right » ou « command + → ») pour voir avec un changement d'orientation (portrait/paysage) et essayer sous plusieurs devices.

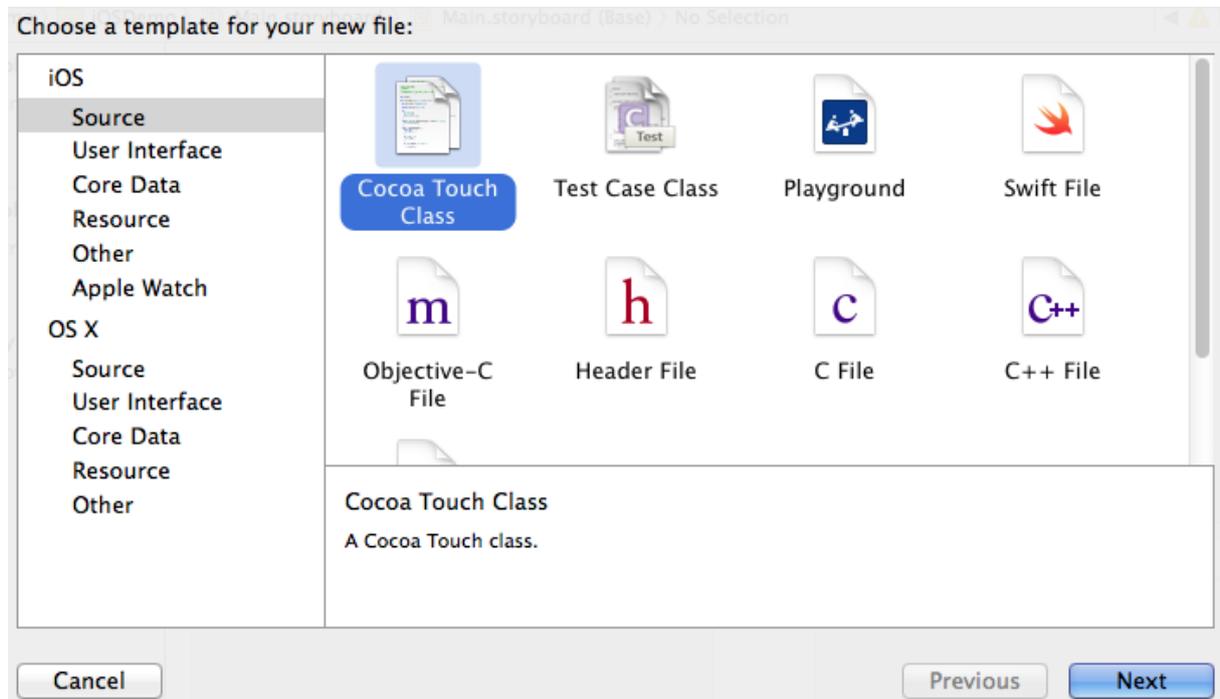
4. « ViewController » côté Code

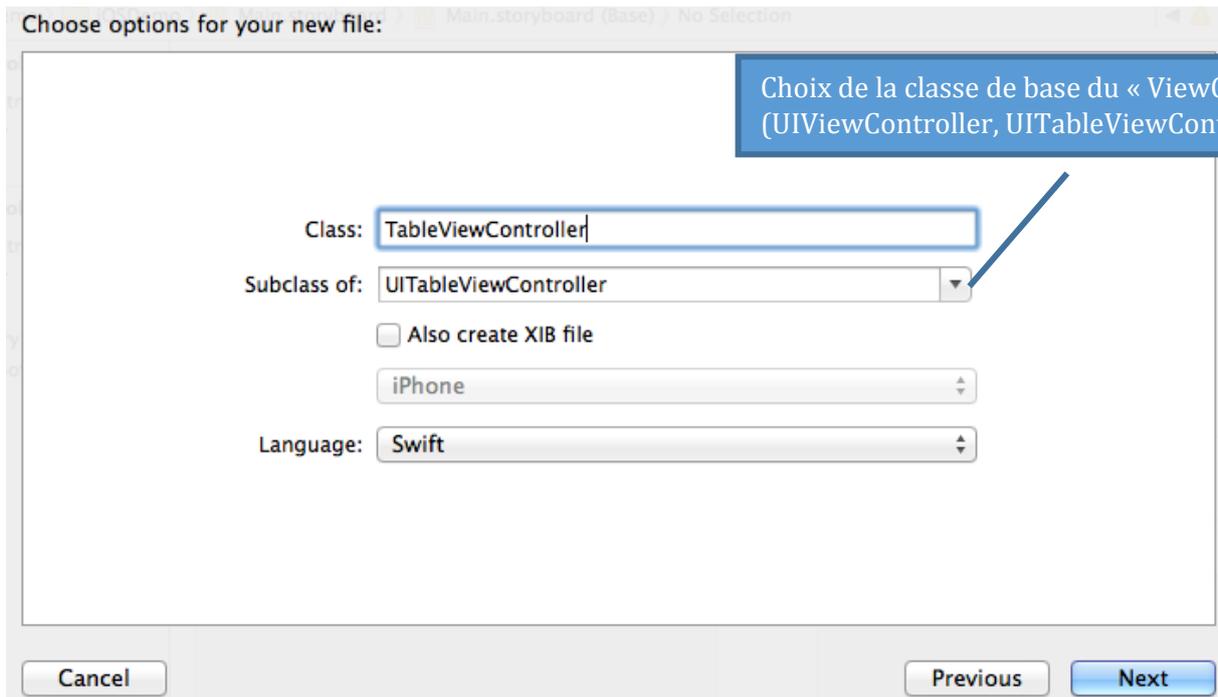
a. Ajout de la classe du « ViewController »

Une classe de base est affectée au « ViewController ». Si toutefois on a besoin de personnaliser le code il faut créer sa propre classe.

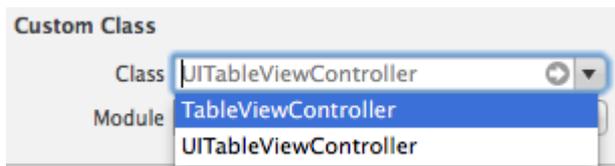


Menu « File » ... « New » ... « File » (Ou depuis le dossier désiré... clic droit ... « New File »)





Choix du « ViewController » dans les propriétés du « ViewController » sélectionné dans le Storyboard



b. « UIViewController »

C'est un peu le « ViewController » de base à tout faire.

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
    /*
    // MARK: - Navigation
    // In a storyboard-based application, you will often want to do a little preparation before navigation
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        // Get the new view controller using segue.destinationViewController.
        // Pass the selected object to the new view controller.
    }
    */
}
```

c. « UITableViewController » Affichage de données sous forme de table

```
import UIKit

class UITableViewController {

    let people = ["Marie", "Jerome"]

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {
        return people.count
    }

    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("personCell", forIndexPath: indexPath) as UITableViewCell

        cell.textLabel?.text = people[indexPath.item]
        return cell
    }
}
```

Exemple avec un tableau

1 section

Nb de lignes

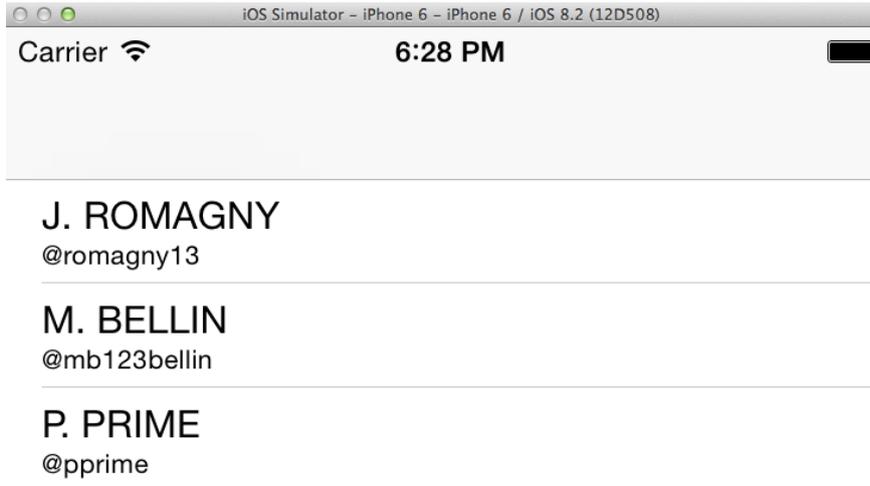
Identifiant du « TableViewCell »

Élément affiché pour chaque ligne

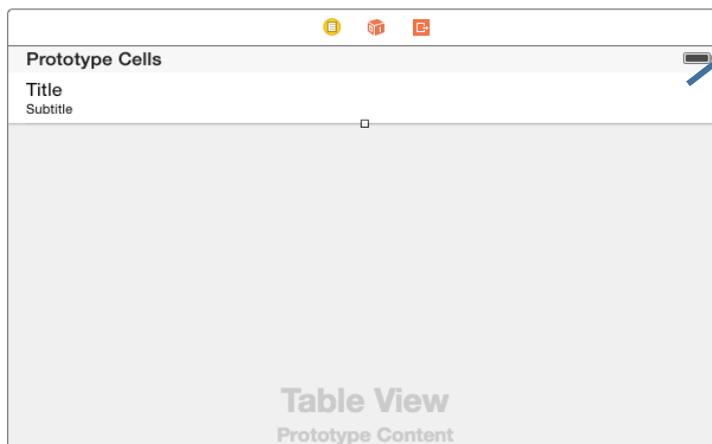
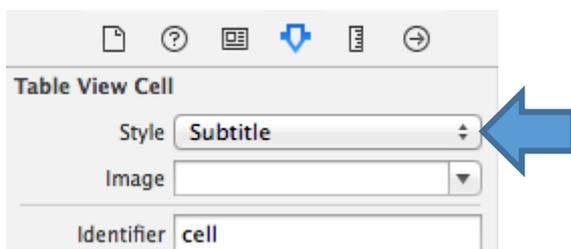
Ajouter un Identifiant au « TableViewCell »



Cellules avec titre et sous-titre



1. Changer le style de « TableViewCell »



Le « style » est modifié

2. On définit la valeur du sous-titre (« detailTextLabel »)

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
    UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("cell", forIndexPath: indexPath) as
        UITableViewCell

    let (name,twitter) = people[indexPath.row]
    cell.textLabel?.text = name
    cell.detailTextLabel?.text = twitter

    return cell
}
```

(On utilise ici un tableau de « Tuples »)

```
let people = [("J. ROMAGNY", "@romagny13"), ("M. BELLIN", "@mb123bellin"), ("P. PRIME", "@pprime")]
```

Plusieurs sections

1. Changer le nombre de sections

```
override fun tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // Return the number of rows in the section.
    return people.count
}
```

2. Cellule retournée selon la section

```
override fun tableView(tableView: UITableView, cellForRowAtIndexPath indexPath:
NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("cell", forIndexPath: indexPath)
    as UITableViewCell

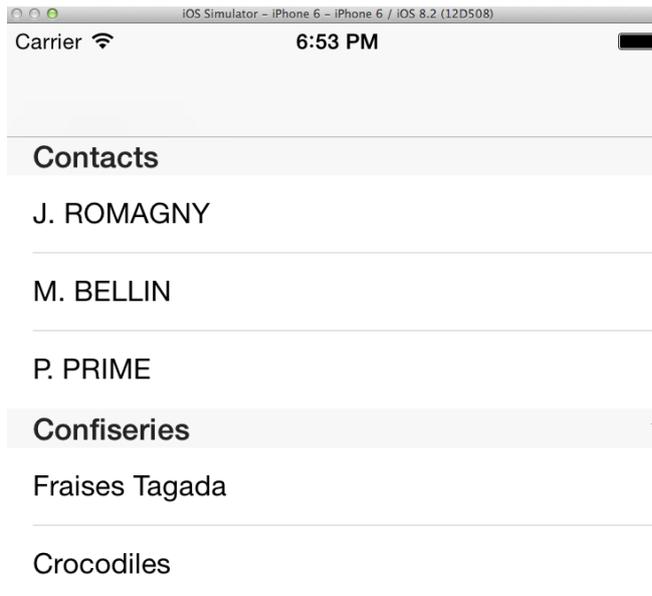
    if indexPath.section == 0 {
        let (name,twitter) = people[indexPath.row]
        cell.textLabel?.text = name
        cell.detailTextLabel?.text = twitter
    }
    else{
        let candy = candies[indexPath.row]
        cell.textLabel?.text = candy
    }

    return cell
}
```

3. Header de sections

```
override fun tableView(tableView: UITableView, titleForHeaderInSection section: Int) ->
String? {

    if section == 0 {
        return "Contacts"
    }
    else{
        return "Confiseries"
    }
}
```



Disclosure

Avoir > à droite des cellules

Régler la propriété Accessory de « UITableViewCell »



D'autres options sont possibles, par exemple avoir un texte « details »

Cellules avec images



On ajoute les images aux assets du projet, puis on utilise un tableau de « tuples »

```
let candies = [("Fraises Tagada", "tagada"), ("Crocodiles", "croco"), ("Fantaisie", "fantasy")]
```

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath:
NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("cell", forIndexPath: indexPath)
    as UITableViewCell
```

```
    let (name, imageName) = candies[indexPath.row]
    var image = UIImage(named: imageName)
    cell.textLabel?.text = name
    cell.imageView?.image = image
```

```
    return cell
}
```

On récupère l'image correspondante au nom dans le tableau

Custom Cell

1. Créer une nouvelle classe héritant de « UITableViewCell »

```
import UIKit

class CustomCell: UITableViewCell {

    var person : Person?

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

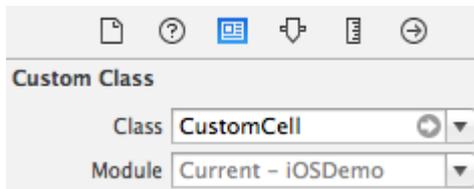
    override func setSelected(selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state
    }

}
```

Par exemple cette cellule de tableau comportera un objet « Person »

2. Changer le type de la cellule « TableViewCell » du « TableViewController » allant l'utiliser



3. Utilisation dans le « TableViewController »

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath:
NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("personCell", forIndexPath:
indexPath) as CustomCell

    let person = peopleService.GetOne(indexPath.item)
    cell.person = person
    cell.textLabel?.text = person.name
    return cell
}
```

Et récupération

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if(segue.identifier == "table_to_details"){

        let cell = sender as CustomCell
        let destinationController = segue.destinationViewController as
PersonDetailsViewController

        destinationController.person = cell.person
    }
}
```

Suppression de lignes



```
// Override to support editing the table view.  
override func tableView(tableView: UITableView, commitEditingStyle editingStyle:  
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {  
    if editingStyle == .Delete {  
  
        PeopleService.Remove(indexPath.item)  
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)  
    } else if editingStyle == .Insert {  
  
    }  
}
```

Décommenter ou
ajouter le code pour
prendre en charge
la suppression

d. Passage de données entre ViewControllers (« PrepareForSegue »)

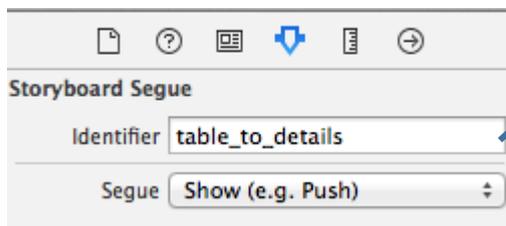
Dans le « ViewController » émetteur

```
override fun prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if(segue.identifier == "table_to_details"){

        let destinationController = segue.destinationViewController as
        PersonDetailsViewController

        let cell = sender as UITableViewCell
        destinationController.name = cell.textLabel!.text
    }
}
```

On vérifie que c'est bien la navigation vers le « ViewController » désiré, puis on récupère le « ViewController » destination pour lequel on renseigne la variable (dans l'exemple nommée « name ») avec la valeur de la cellule



On sélectionne le « Segue » entre les 2 « ViewControllers » et lui donne un identifiant

Dans le « ViewController » récepteur

```
class PersonDetailsViewController: UIViewController {
    var name : String?
    override fun viewDidLoad() {
        super.viewDidLoad()
        self.title = name;
    }
}
```

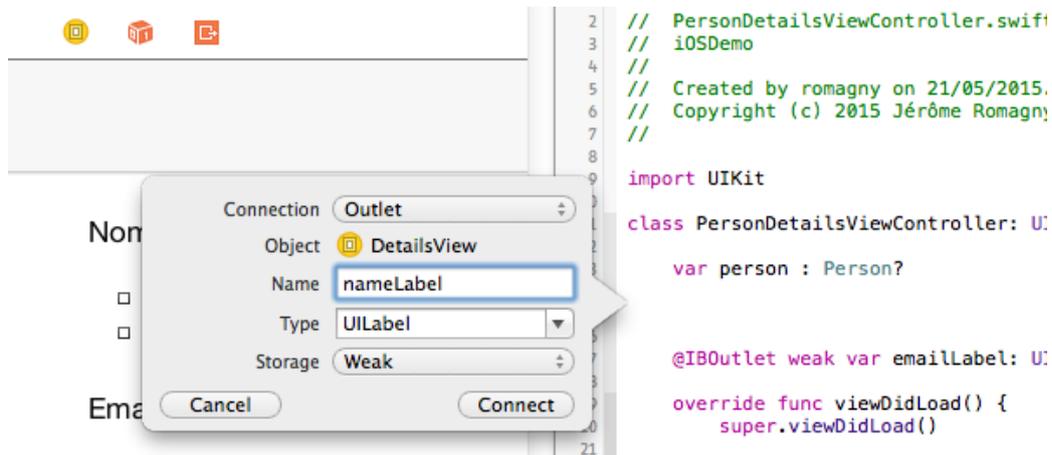
Variable remplie que l'on affiche dans la barre en titre

e. Création d'Outlets

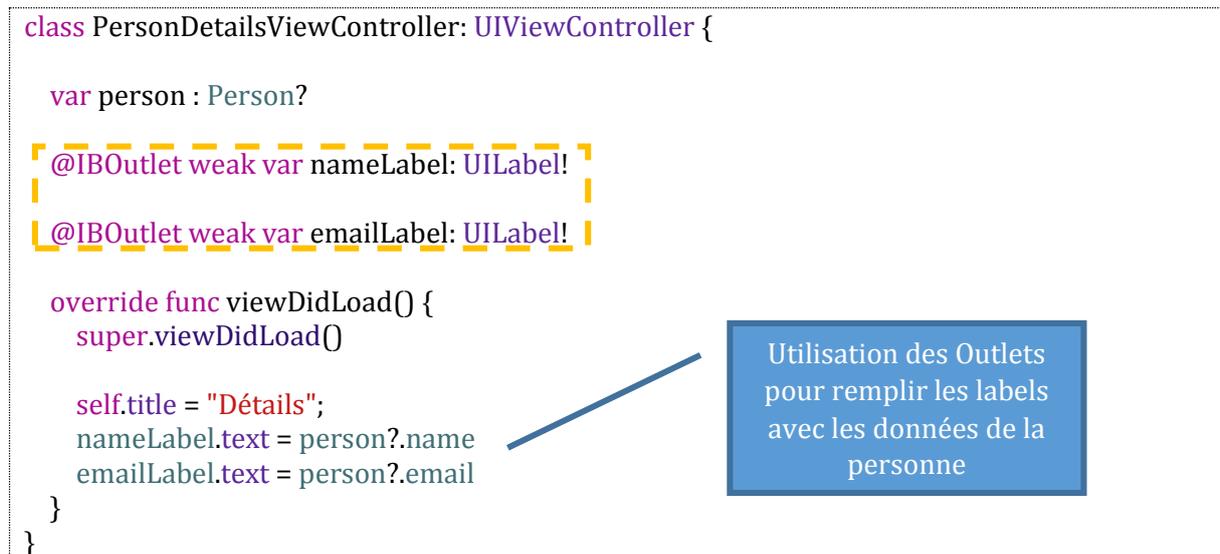
1ère Méthode

Exemple on crée deux « Outlets » correspondants à deux labels devant afficher le détail de la personne (nom et email)

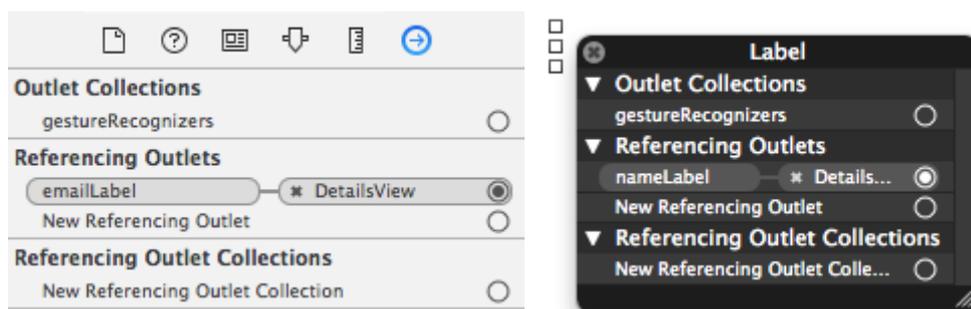
1. Utiliser la vue splittée ...
2. Se positionner sur le contrôle (exemple ici un label) maintenir « Ctrl »
3. et faire glisser la souris vers l'emplacement dans le code où sera inséré celle-ci...



... on fait la même chose pour le label affichant l'email



Avec cette méthode les labels du Storyboard sont automatiquement connectés aux Outlets. Panneau Propriétés (onglet « Connections Connector ») ou depuis le menu contextuel du contrôle



Attention toutefois, si on renomme ou supprime l'Outlet dans le code, la connexion ne sera pas supprimée et peut être source d'exception. Ne pas oublier donc dans ce cas d'aller dans les propriétés des contrôles depuis le Storyboard pour supprimer les connexions.

Note Avec **Objective-C** les **Outlets** sont placées dans l'**@interface**

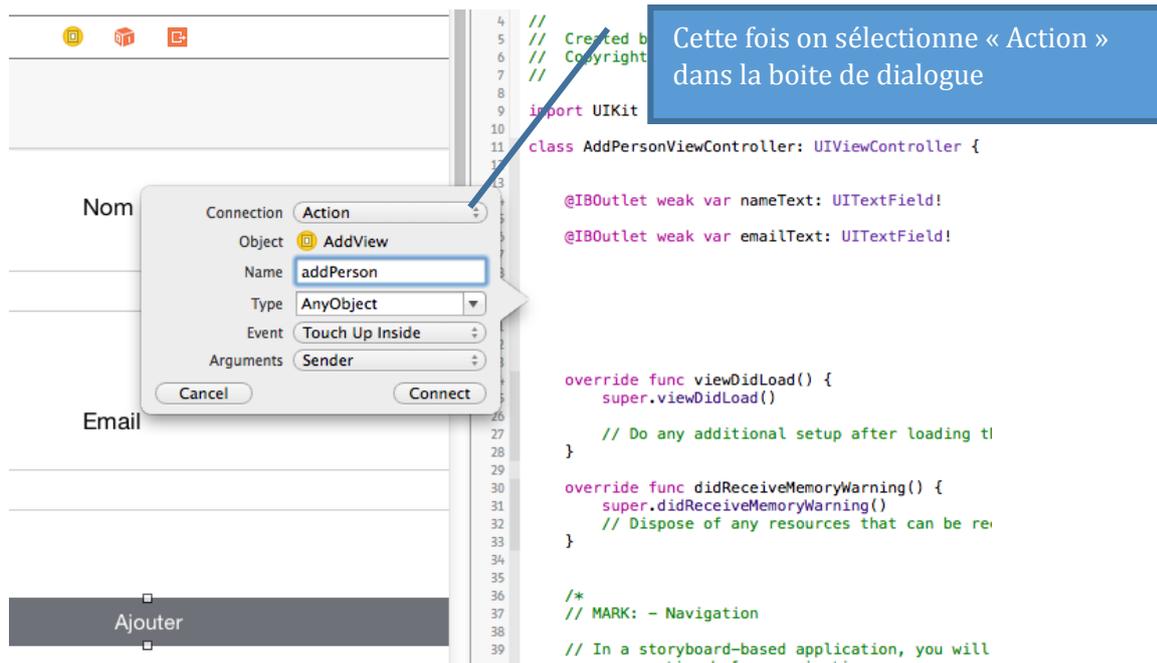
2^{de} Méthode

1. Définir les Outlets en code
2. Aller ensuite dans le panneau de propriétés (onglet « Connections Connector ») des contrôles afin de les connecter.

f. Création d'une Action

1^{ère} méthode

Pareil que pour les Outlets, en vue splittée maintenir « Ctrl » depuis le contrôle désiré (ici un bouton) et glisser jusqu'à l'endroit où insérer l'action dans le code.



...

```
import UIKit

class AddPersonViewController: UIViewController {

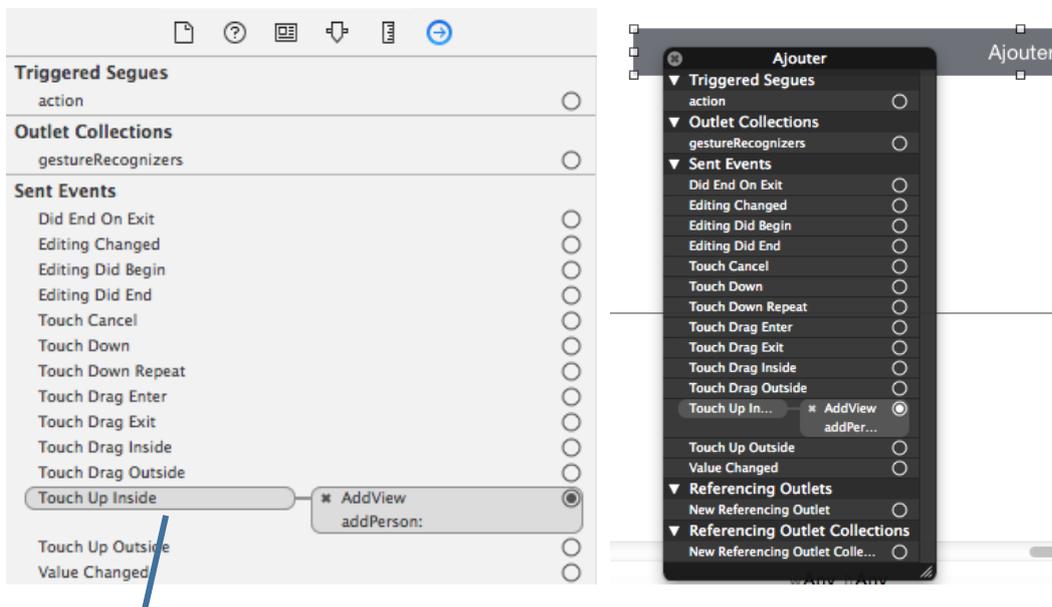
    @IBOutlet weak var nameText: UITextField!

    @IBOutlet weak var emailText: UITextField!

    @IBAction func addPerson(sender: AnyObject) {
        PeopleService.Add(nameText.text, email: emailText.text)
    }
}
```

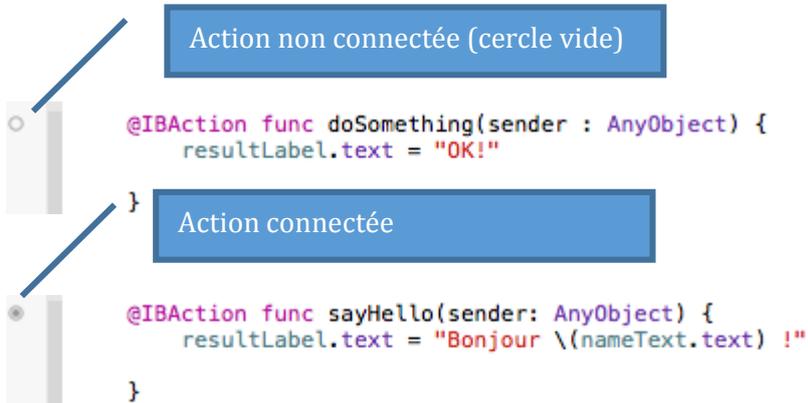
2ème méthode

En code puis on va dans les propriétés (onglet « Connections Connector ») ou depuis le menu contextuel du contrôle (bouton ici) afin de la connecter à l'action.



Affiche les éléments connectés. Si on supprime une connexion on ne supprime pas le code du « ViewController »

Dans le code

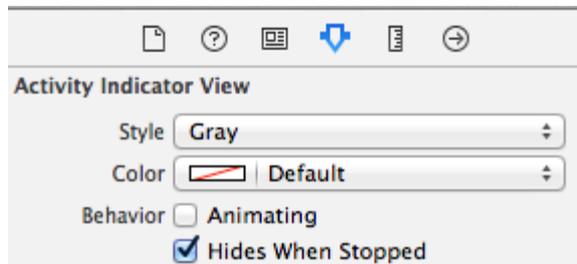


g. Activity Indicator



Activity Indicator View - Provides feedback on the progress of a task or process of unknown duration.

1. Glisser depuis la boîte à outils sur le « ViewController » désiré
2. Propriétés « Hides When Stopped » pour le cacher



3. Ajout d'un Outlet au « ViewController » puis utilisation

```
@IBOutlet weak var activity:
    UIActivityIndicatorView!

@IBAction func addPerson(sender: AnyObject) {
    activity.startAnimating()

    PeopleService.Add(nameText.text, email:
        emailText.text)

    activity.stopAnimating()
}
```

5. Création de modèle et de service

Modèle

```
import UIKit

class Person: NSObject {
    var id : String?
    var name : String?
    var email : String?

    override init () {
        super.init()
    }

    init (id: String ,name : String,email : String){
        self.id = id
        self.name = name
        self.email = email
    }
}
```

Service

```

class PeopleService: NSObject {
    struct people{
        static var array =
[Person(id:NSUUID().UUIDString,name:"Marie",email:"mb3@laposte.net"),Person(id:NSUUID
().UUIDString, name:"Jerome",email:"romagny13@gmail.com")]
    }

    class func Count() -> Int{
        return people.array.count
    }
    class func GetOne(id :Int) -> Person{
        if(people.array.count>0){
            return people.array[id]
        }
        return Person()
    }
    class func Add(name : String, email : String) {
        var person = Person(id:NSUUID().UUIDString,name: name, email: email)
        people.array.append(person)
    }
    class func Remove(id :Int) {
        people.array.removeAtIndex(id)
    }
}

```

6. Persistance de données entre sessions

```

class ViewController: UIViewController {

    @IBOutlet weak var output: UILabel!

    @IBAction func Addvalue(sender: AnyObject) {
        // add
        var defaults = UserDefaults.standardUserDefaults()
        defaults.setObject("ma valeur!", forKey: "mykey")
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // load data
        var defaults = UserDefaults.standardUserDefaults()
        var myvalue = defaults.objectForKey("mykey") as? String

        if myvalue != nil {
            output.text = myvalue
        }
        else {
            output.text = "Pas de valeur retrouvée!"
        }
    }
}

```

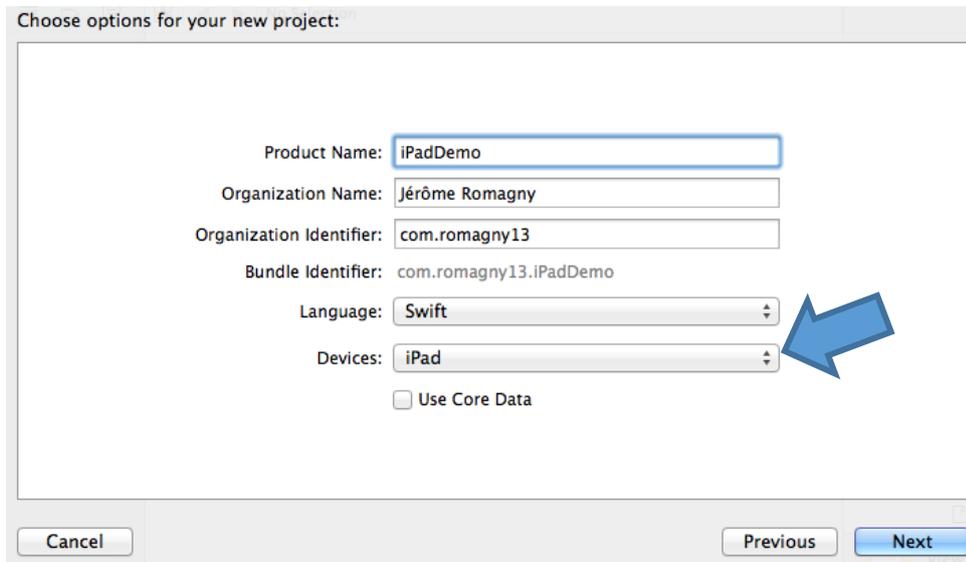
Enregistrement d'une valeur

On essaie de retrouver la valeur

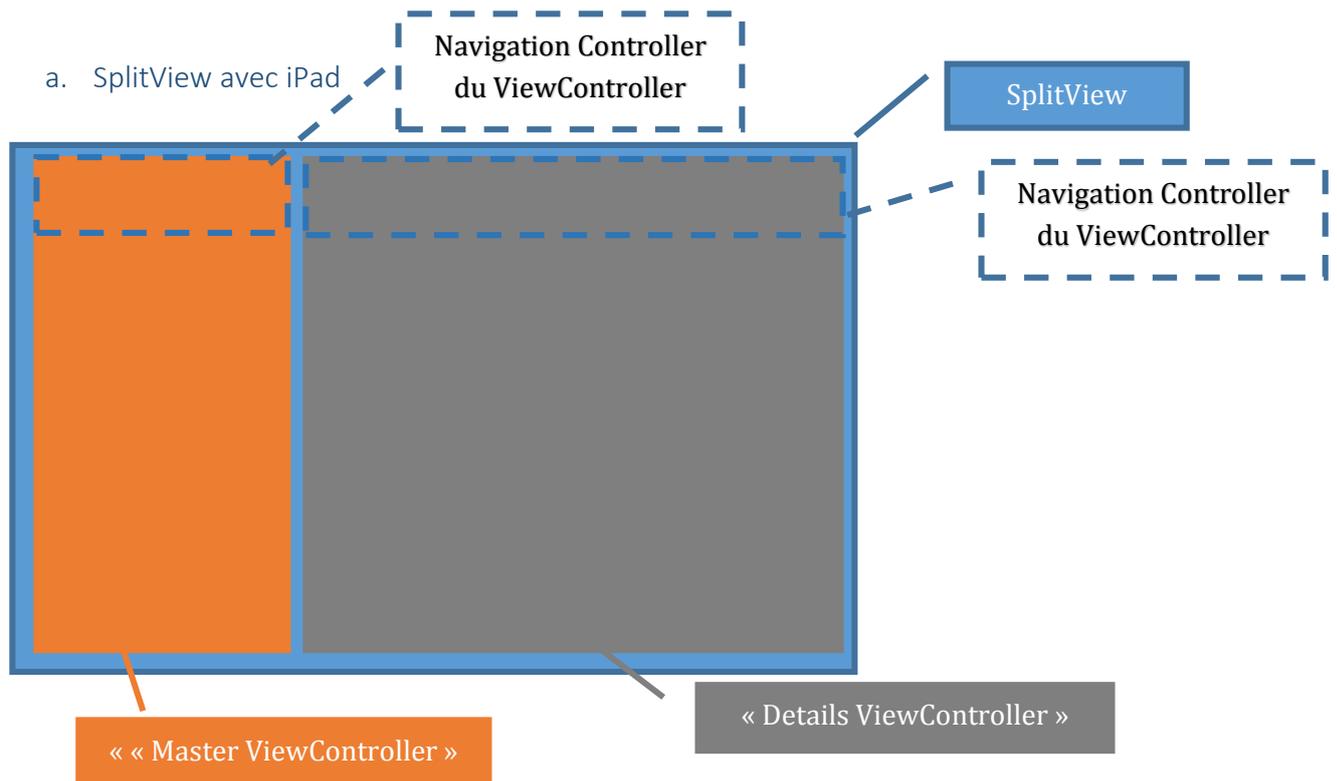
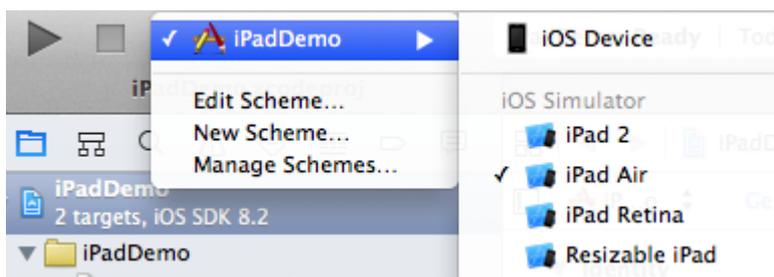
La gestion du cycle de vie de l'application est disponible depuis « AppDelegate.swift »

7. iPad (tablette)

Options lorsque l'on crée un nouveau projet (« iPad » et « Universal »)



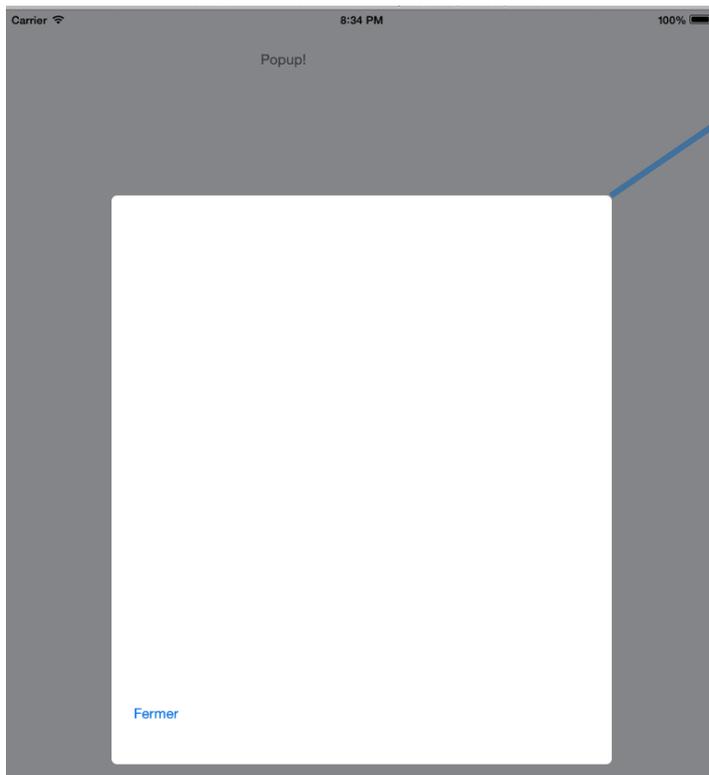
Avec le « Simulator »



Les « settings » est un exemple de « vue master détails » sur iPad

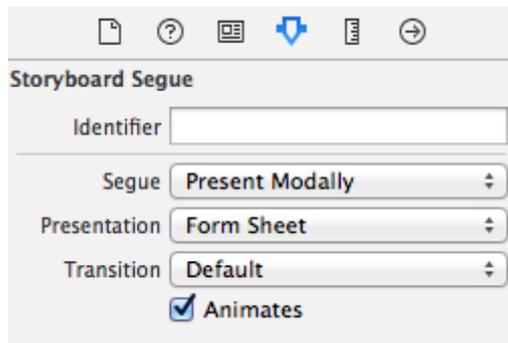


b. « Popup »



Sur iPad la popup peut ne pas occuper toute la « page » comme sur iPhone

- On a 2 « ViewControllers », un ViewController initial avec un bouton permettant d'afficher le second.
- Le « Segue » est réglé sur « Present Modally » avec Presentation « Form Sheet »



- Pour fermer la popup on ajoute un bouton (« Fermer ») avec une action

```
@IBAction func dismiss(sender: AnyObject) {  
    self.dismissViewControllerAnimated(true, completion: nil)  
}
```