iOS avec Swift

J. ROMAGNY

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**

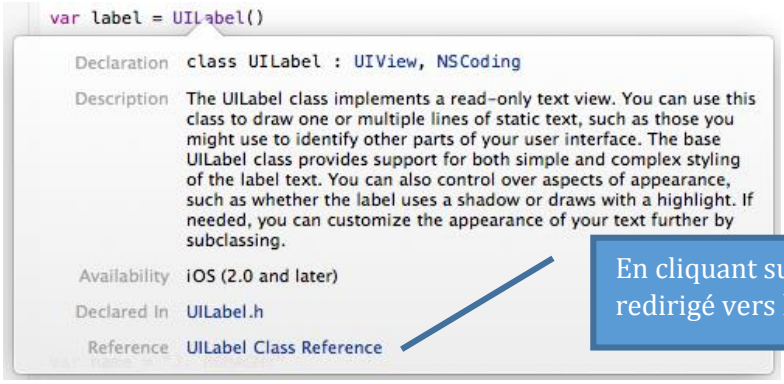
Quelques **particularités** :

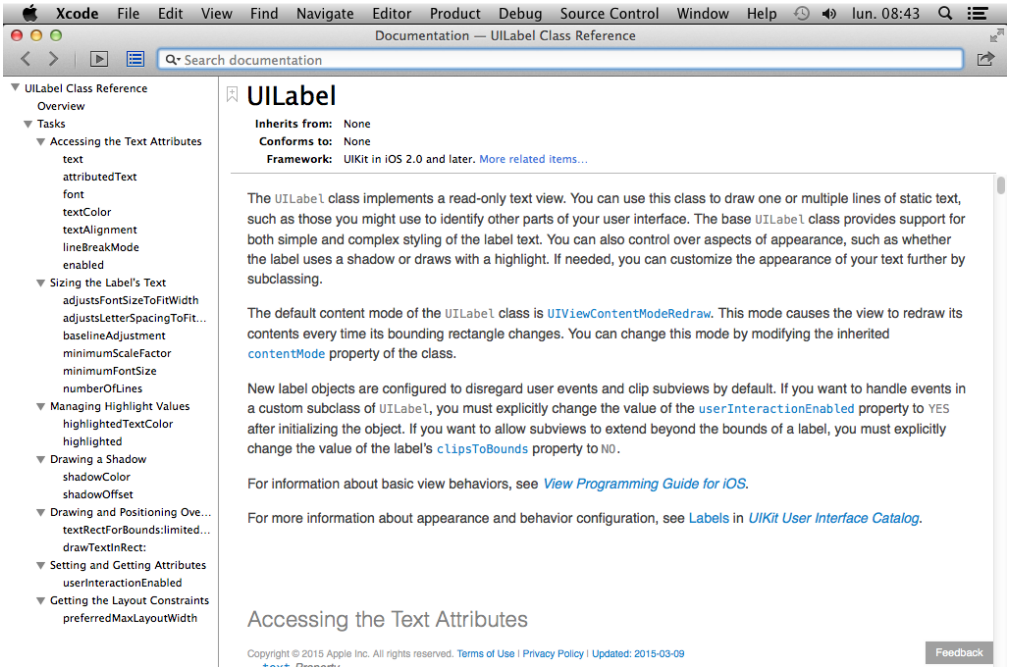
- **Parenthèses** (conditions) et « **;** » en fin d'instructions ne sont **pas obligatoires**
- Attention aux **espaces** qui peuvent être synonymes d'erreur de syntaxe
- Encourage l'utilisation de constantes (mot-clé « **let** »)

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

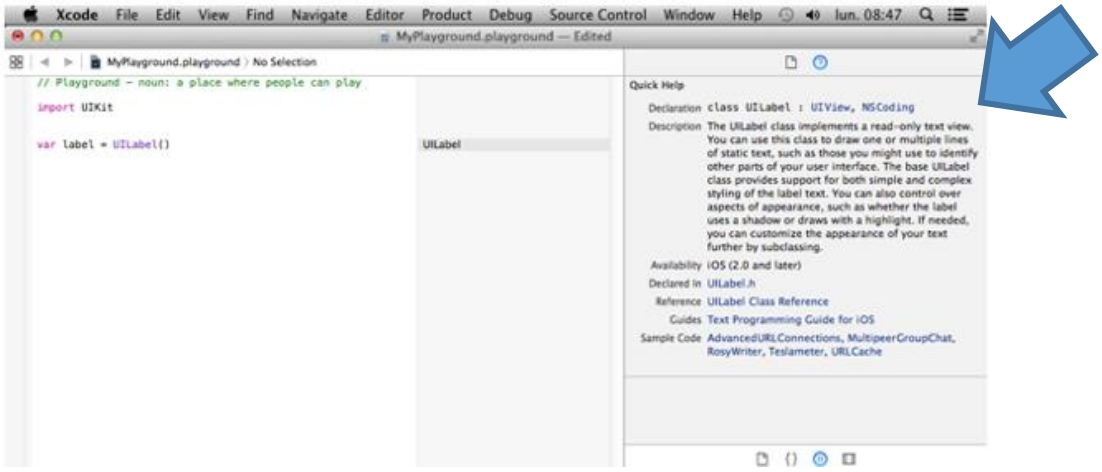
Documentation Swift
UIKit Reference
AppKit reference
Foundation framework reference

Aide depuis **XCode**
Maintenir la touche « options » (à gauche de la touche « command »)





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



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?

if myInt != nil {

}
```

On ajoute « ? » au type

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

Nom du paramètre puis type

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

Avec **retour**

Type de 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

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

Autre possibilité pour **modification**

```
myDictionnary.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 () {

   }
}

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

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

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

```
class Member : User {

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

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

Hérite de « User »

Overrides du constrcuteur 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

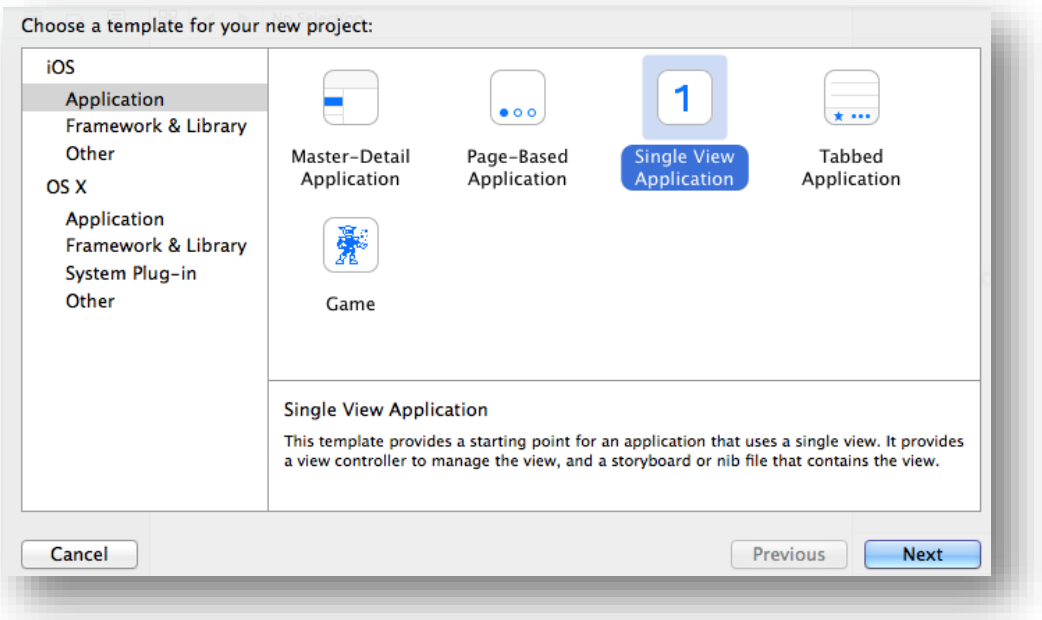
S'inscrire au « Apple Developer Program » pour pouvoir mettre ses applications sur l'App Store.

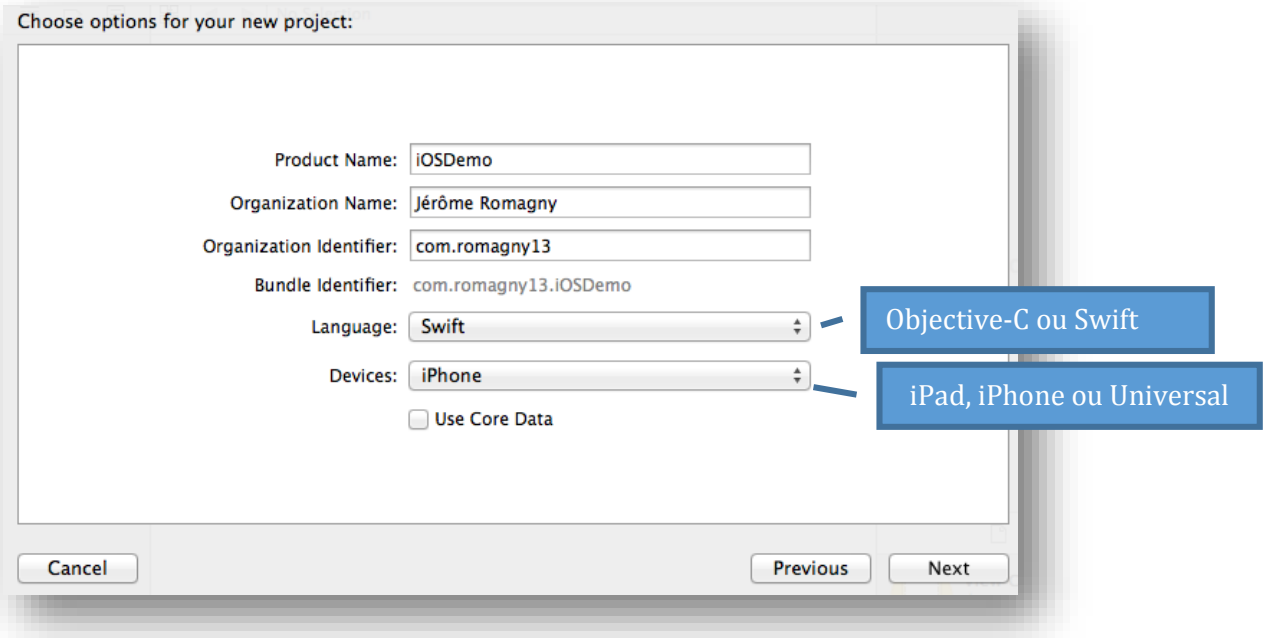
Ressources

2. Créer un nouveau projet

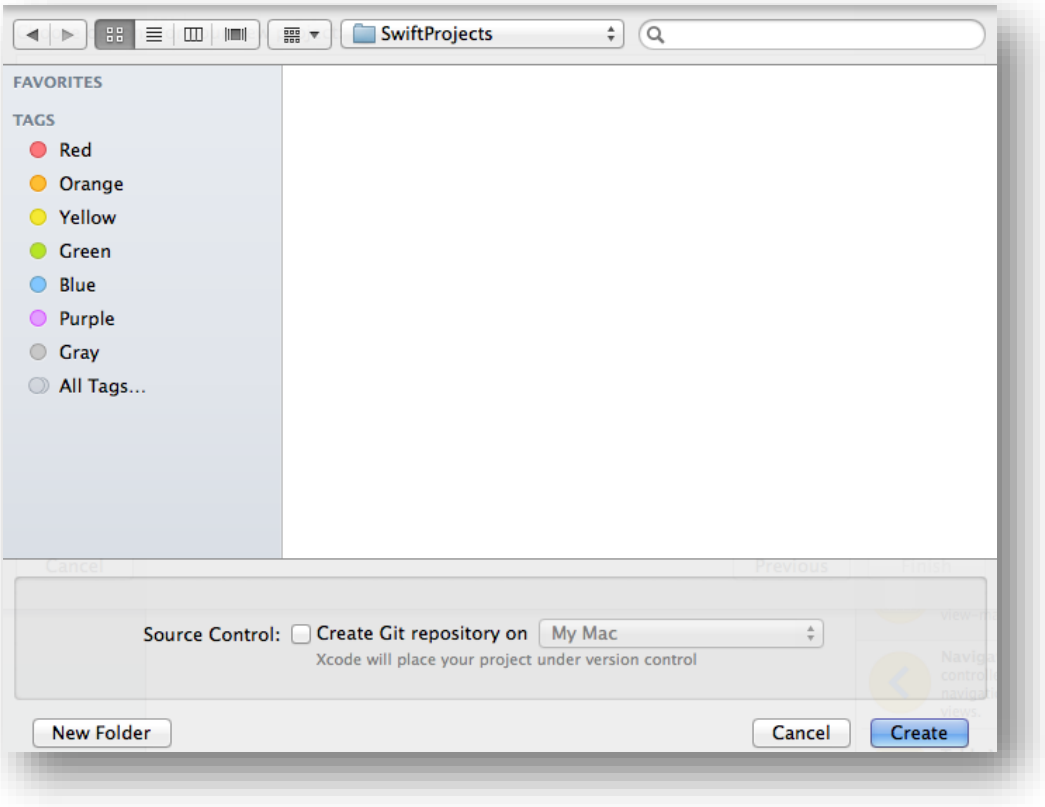
Depuis l'écran de démarrage (« Create a new XCode Project ») ou depuis le menu « File » … « New » .. « Project »

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



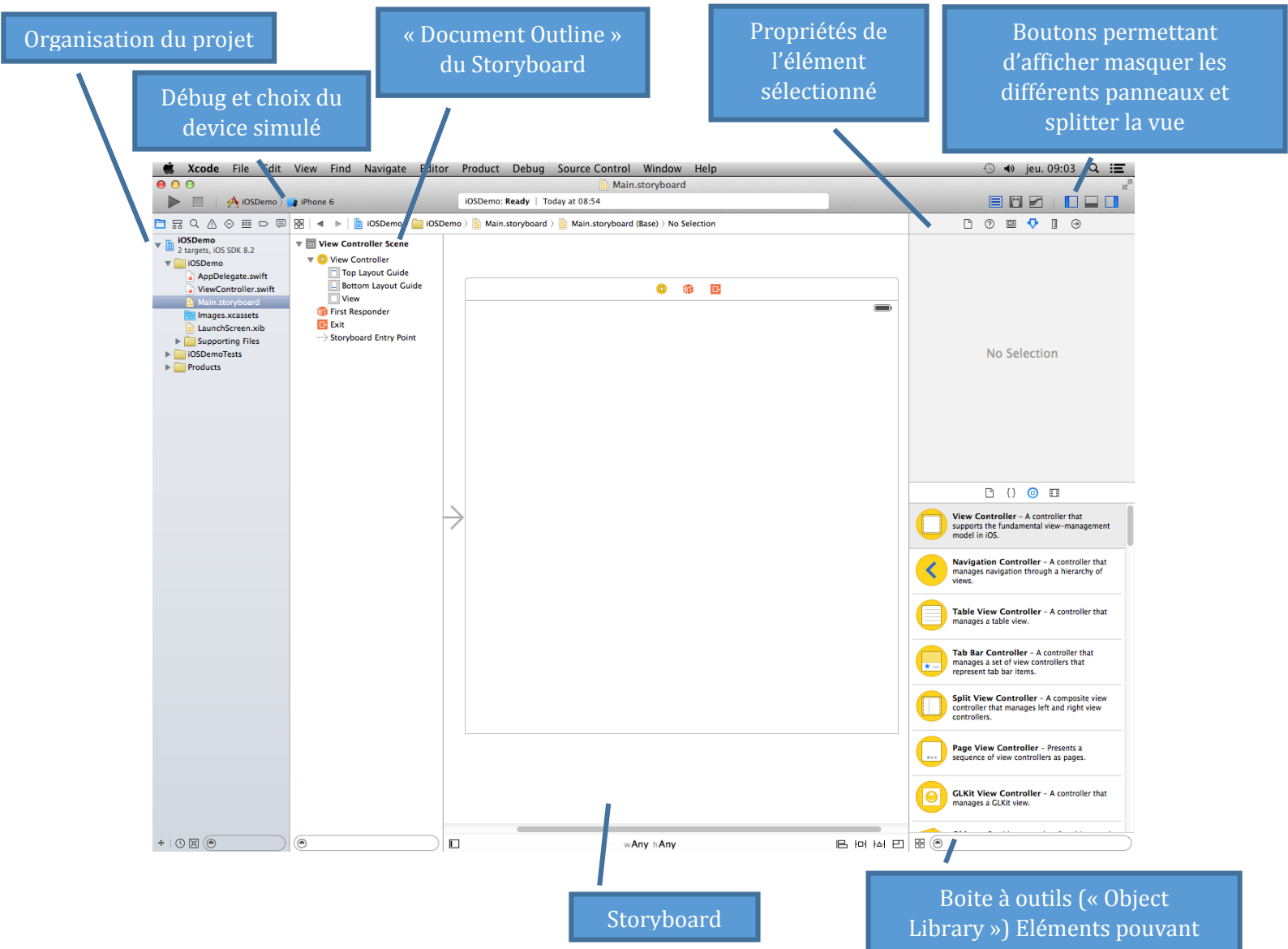


… choix du répertoire du projet …

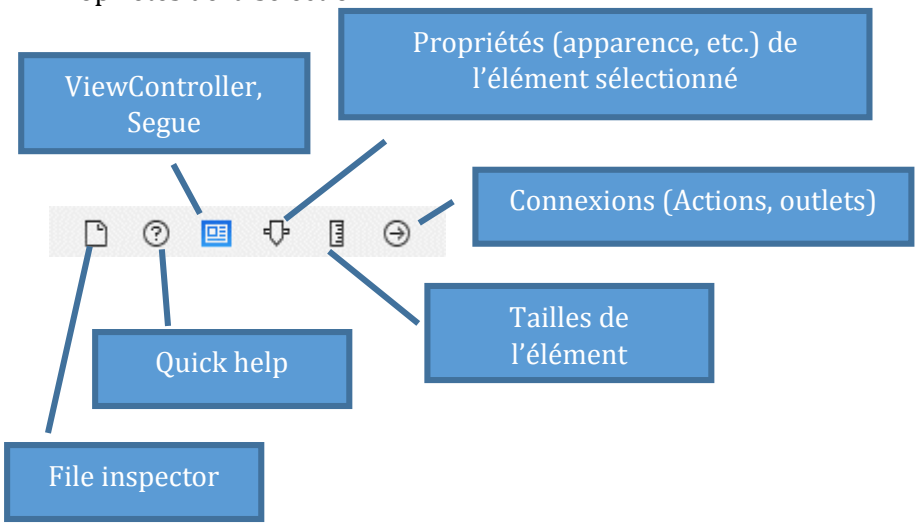


Le Storyboard permet de définir la structure et la navigation entre « ViewControllers ». C'est la partie « Visuelle ».

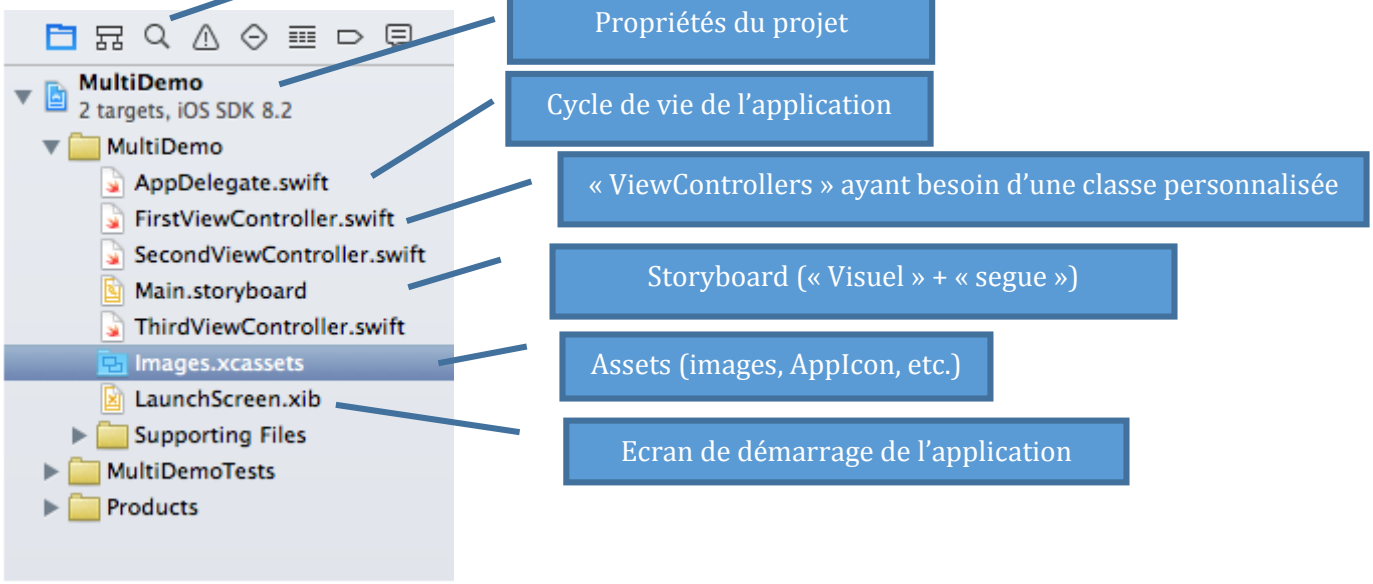
Les ViewControllers utilisent leur classe de base correspondante par défaut, si on veut ajouter son code au « ViewController » il faut créer sa propre classe et remplacer la classe de base du ViewController par celle-ci dans les propriétés du Storyboard



Propriétés de la sélection



Organisation du projet



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

Propriétés du projet

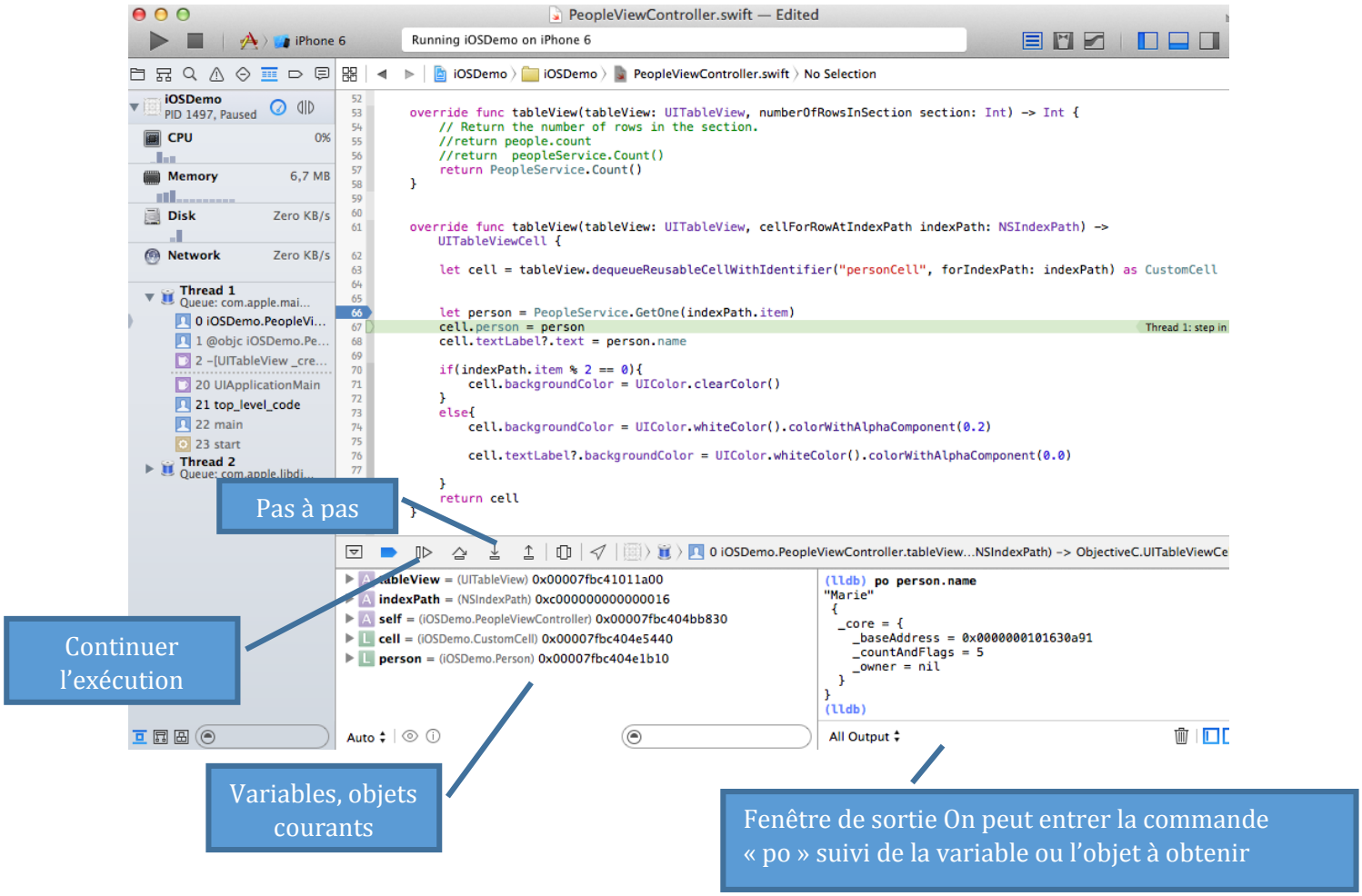
En cliquant sur la racine du projet on a accès aux propriétés du projet (version, device, launchscreen, capabilities, etc.)

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

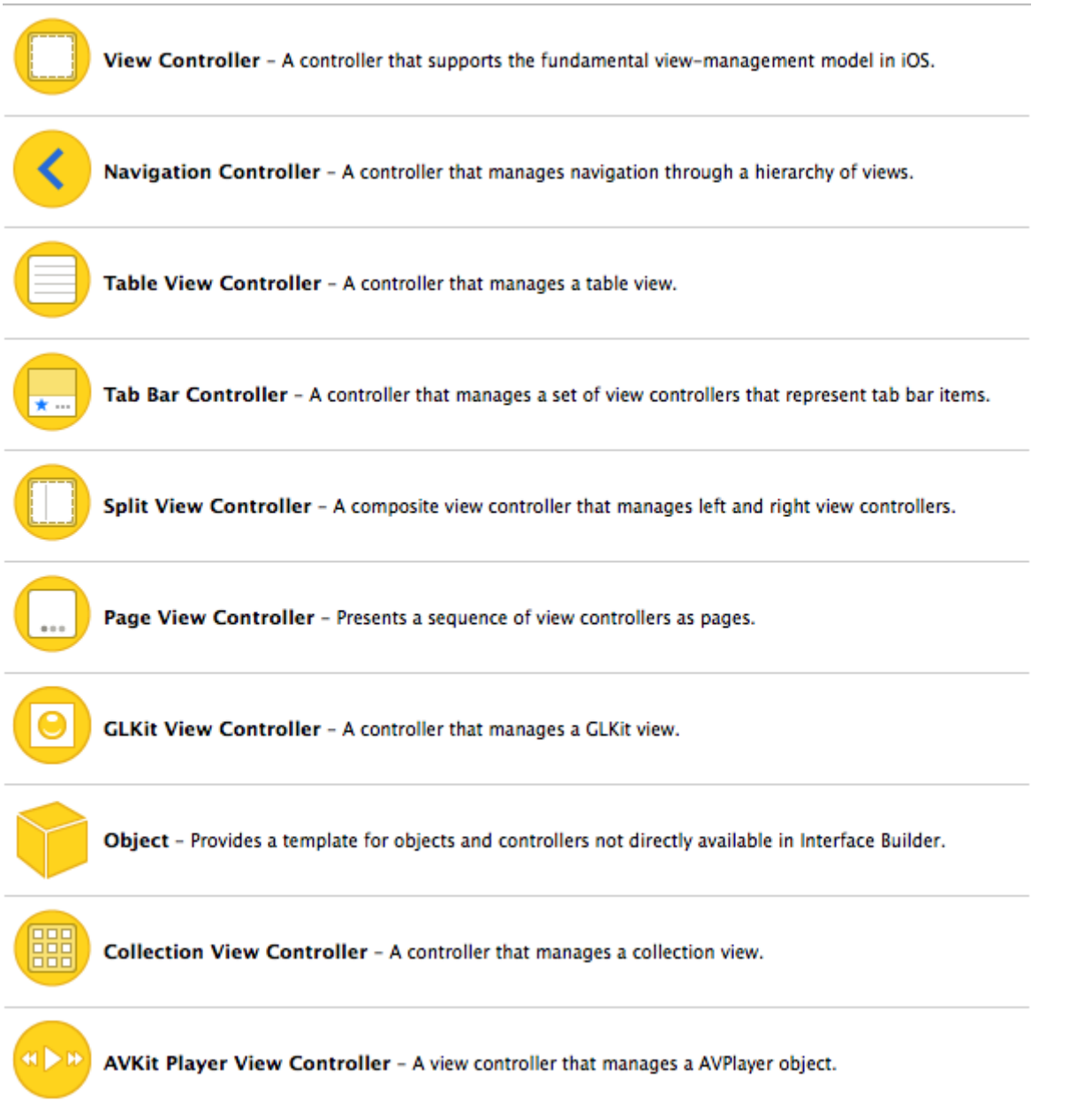
On peut placer des points d'arrêt . Pour supprimer un point d'arrêt… clic droit dessus « Delete Breakpoint »

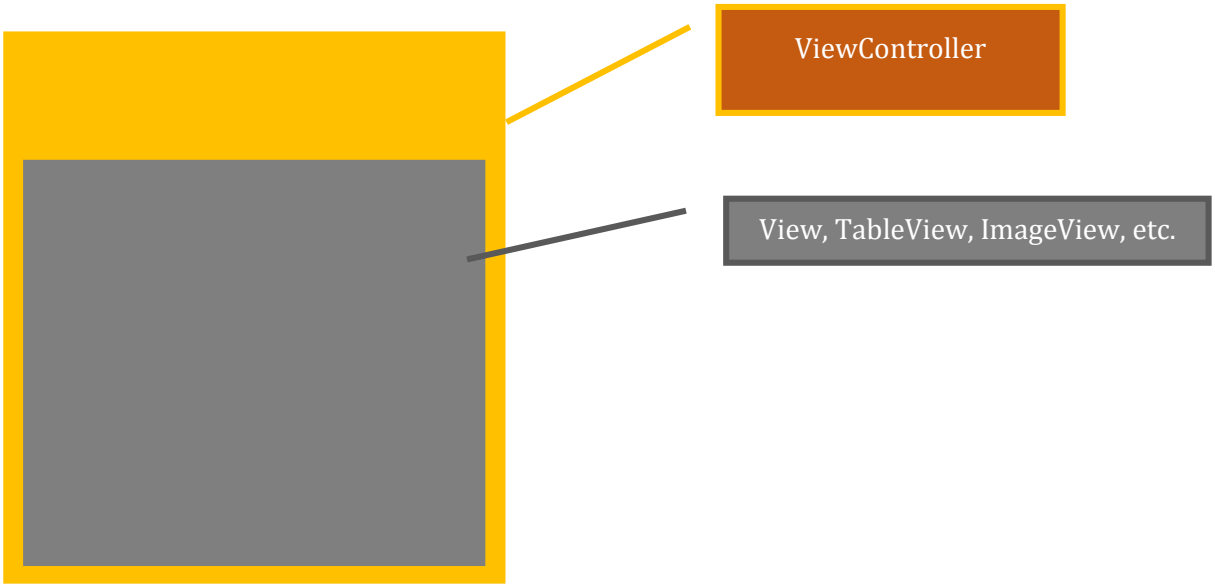


3. « ViewController » côté « Storyboard »

a. Différents « ViewControllers » pouvant être ajoutés au Storyboard

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



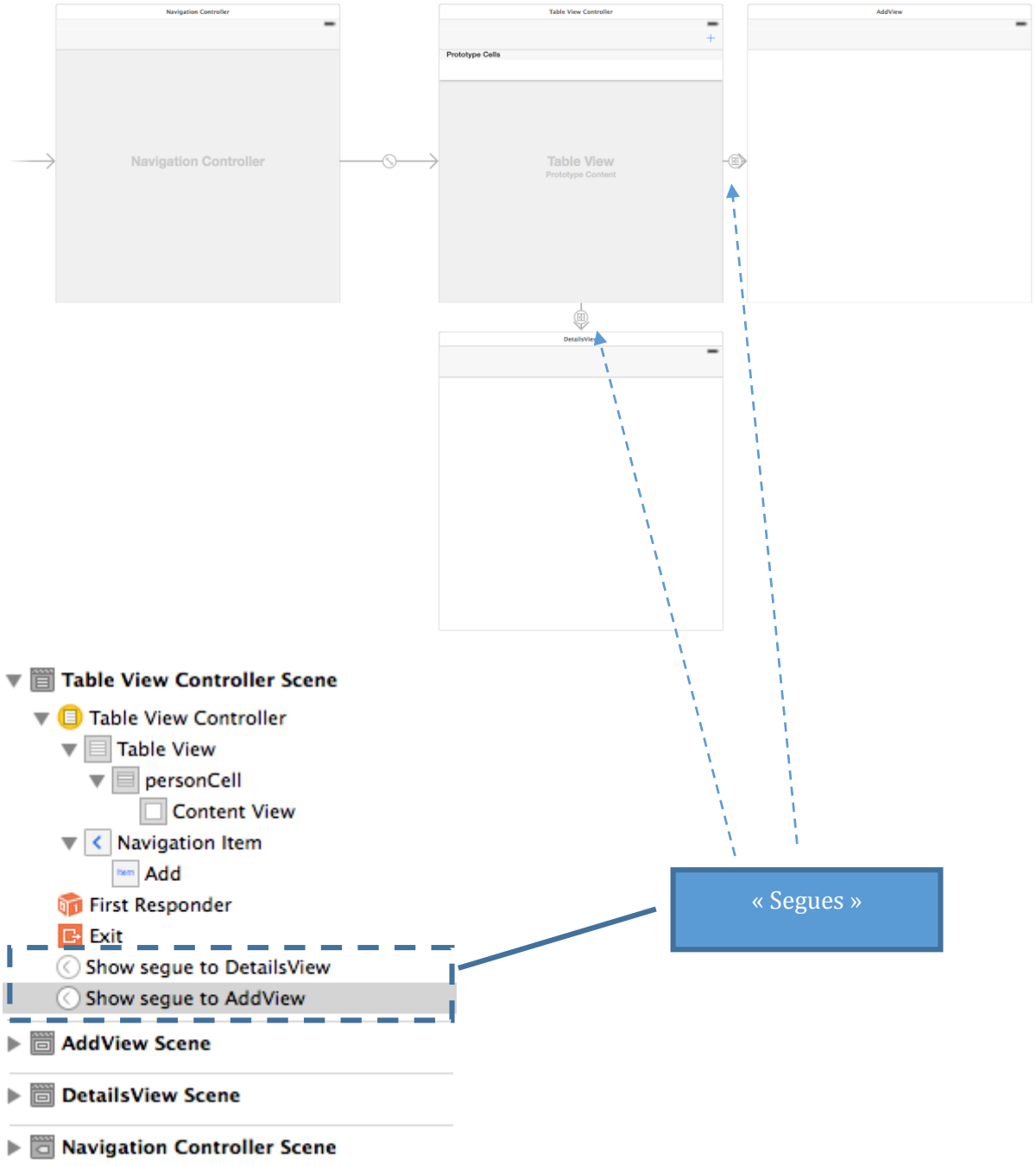


b. « Segues » (Navigation entre « ViewControllers »)

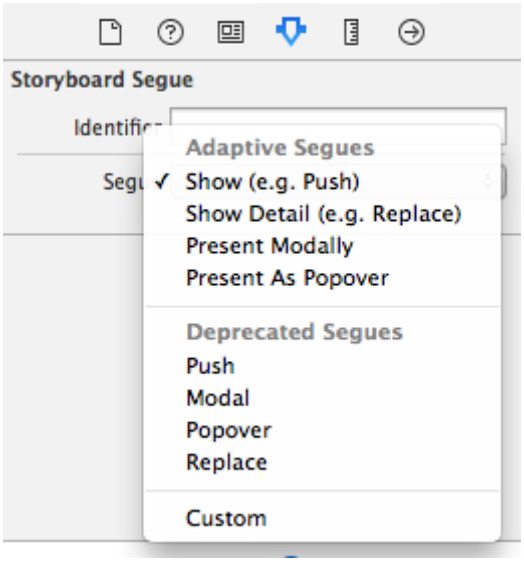
La navigation s'ajoute ainsi :

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



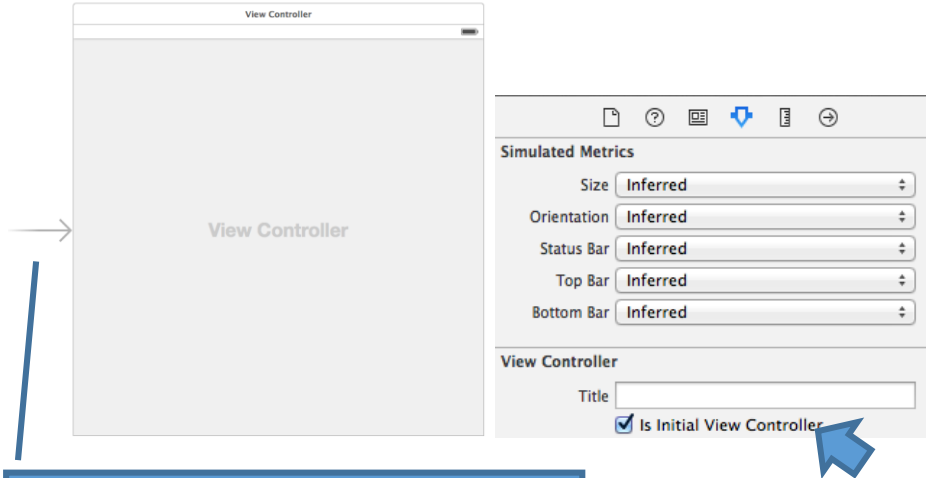
 Show (e.g. Push), Show Detail (e.g. Replace)

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

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

(Les autres sont marqués « Deprecated »)

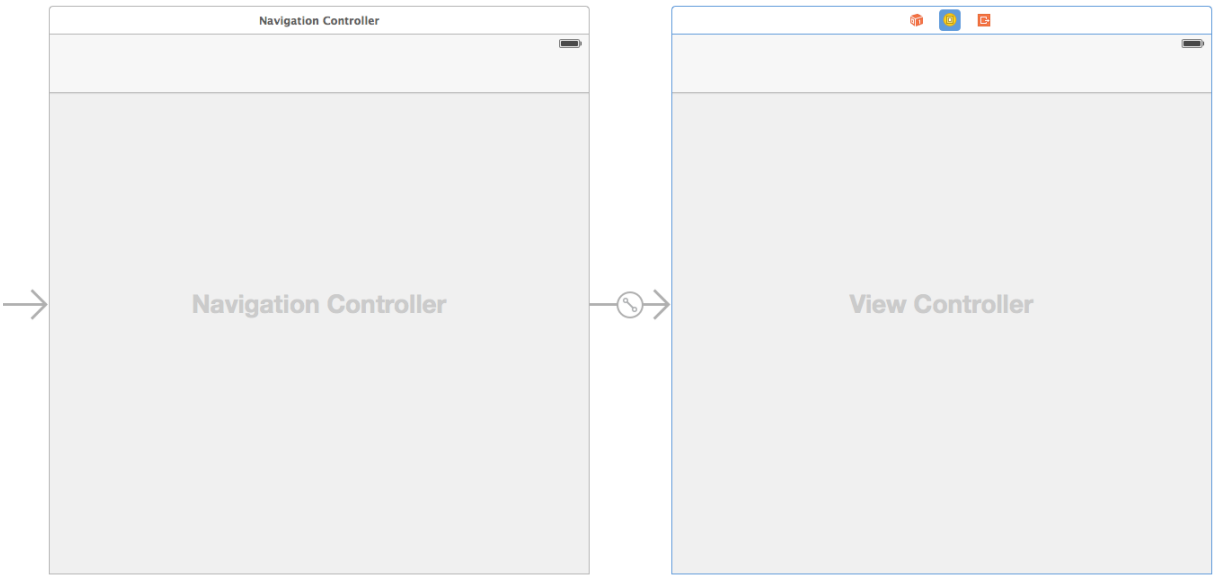
Définir le « ViewController » initial



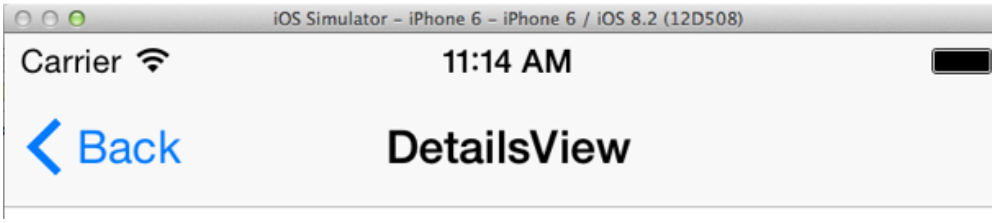
La flèche montre que c'est l'écran de départ de l'application. Cocher « is initial View Controller » dans les propriétés

Ajouter la barre de navigation

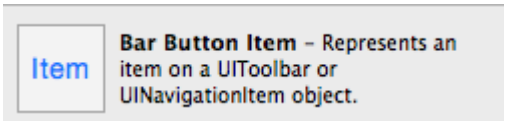
Sélectionner l'écran initial…puis menu « Editor » … « Embed In » … « Navigation Controller »



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)

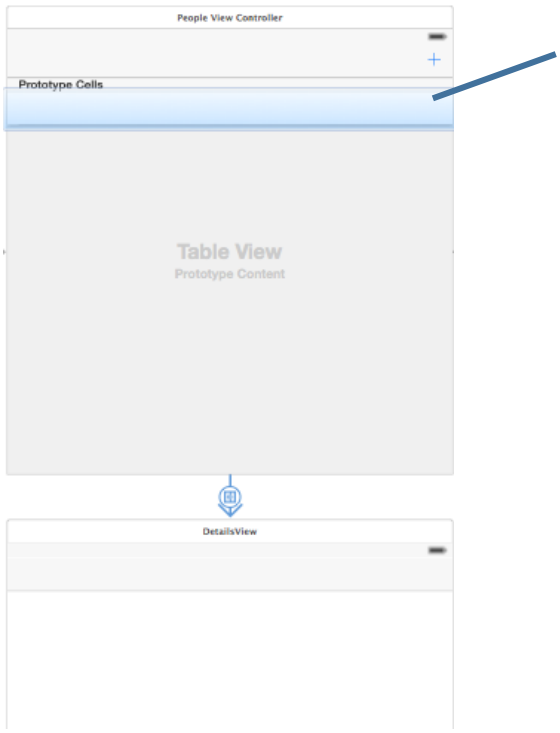


Ajouté à la barre de navigation (« Navigation Controller »). Puis on change la propriété « Identifer » (pour add ici) ce qui change l'apparence du bouton

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

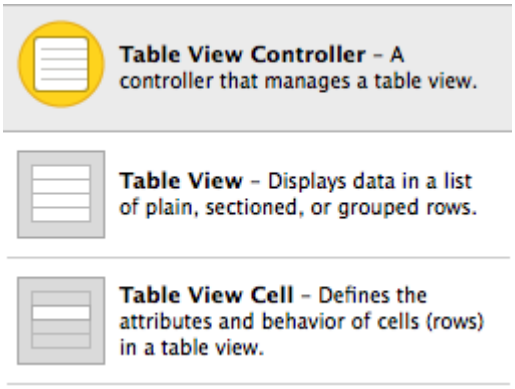
Navigation à partir de cellule de TableViewController

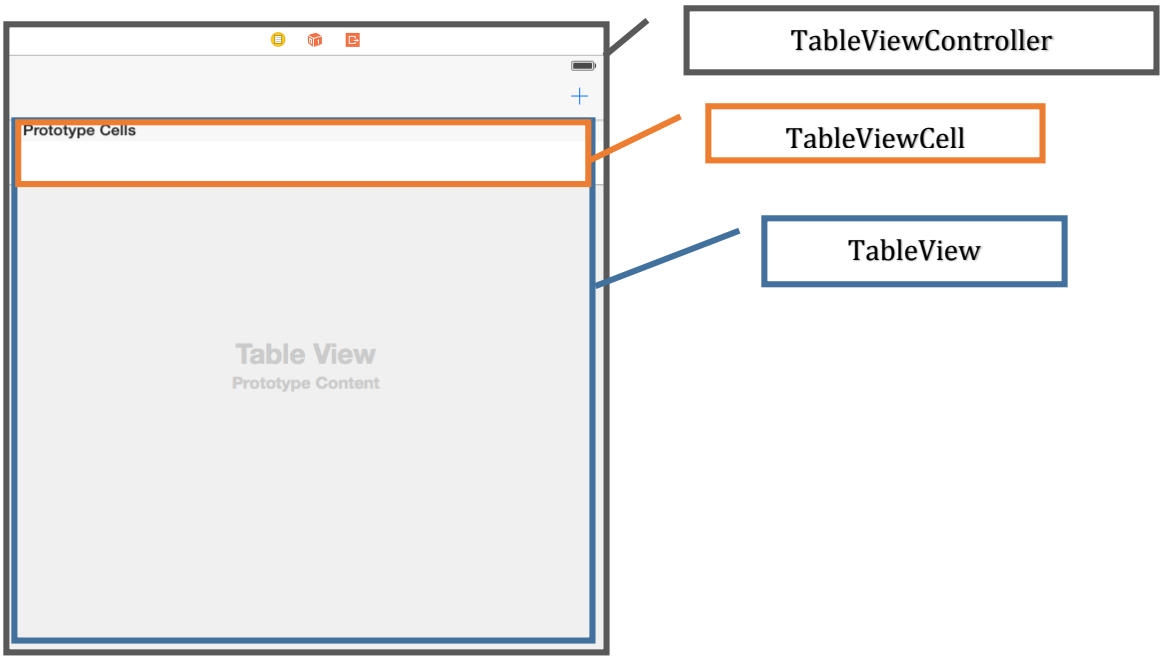
Navigation (« Segues ») de « Prototype Cell » vers un autre « ViewController »



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

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



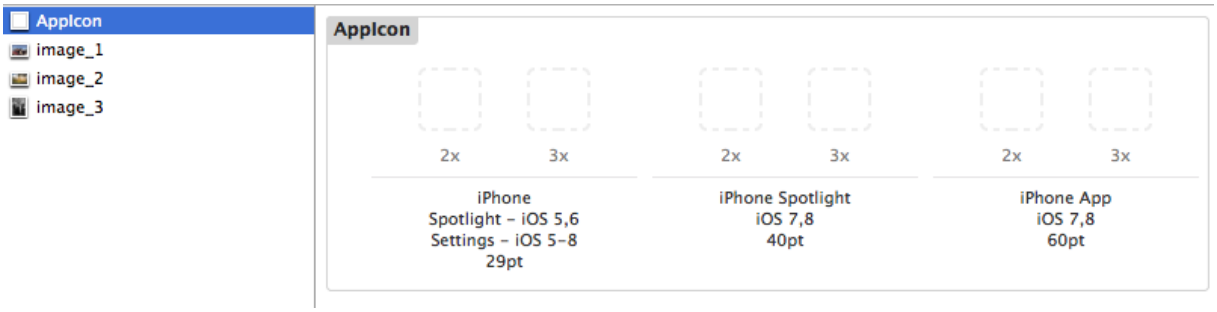


c. Améliorer l'expérience utilisateur

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 :

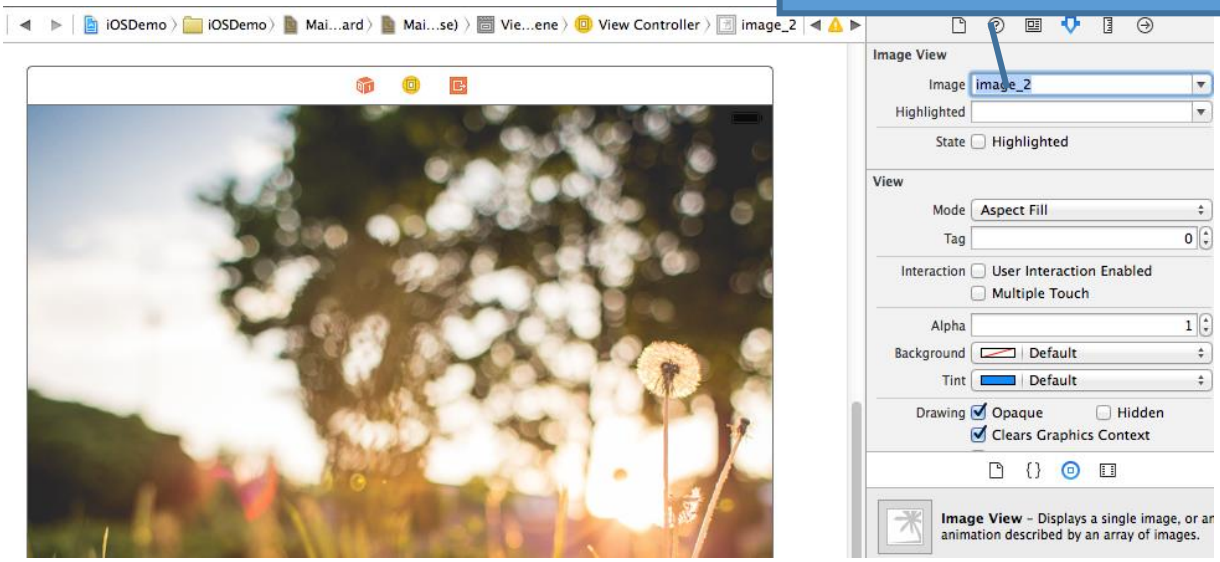
```
var image = UIImage(named :"image_1" )
```

LaunchScreen (.xib)*

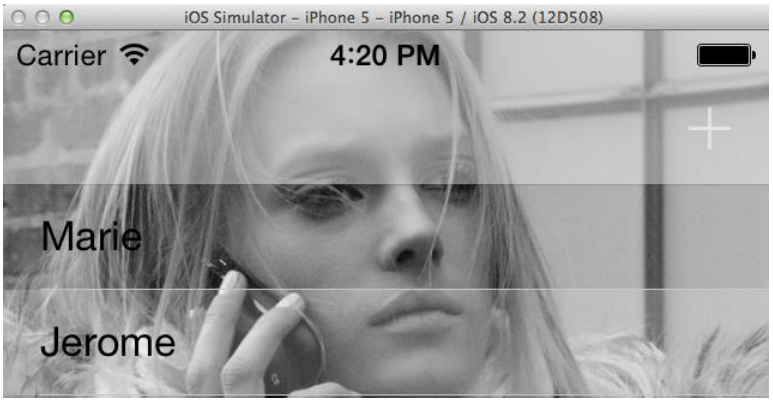
Personnalisable, on peut ajouter des éléments, (exemple une ImageView en fond), la font (couleur, famille, etc.)

ImageView

On peut facilement sélectionner l'image affichée par l' « ImageView » parmis 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.backgroundView = imageView

}

override func viewDidAppear(animated: Bool) {
 super.viewDidAppear(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 boite de texte

```
@IBAction func addPerson(sender: AnyObject) {


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

    nameText.resignFirstResponder()
}
```

Cacher le clavier quand on « click » en dehors

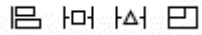
```
override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {
 self.view.endEditing(true)
}
```

Si le clavier n'apparait 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

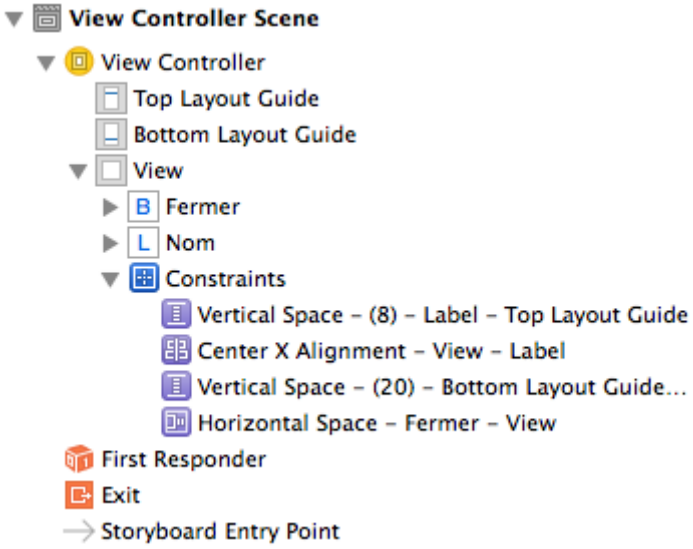


- Alignement
- Tailles
- Pour résoudre les problèmes de contraintes
- « Resizing »

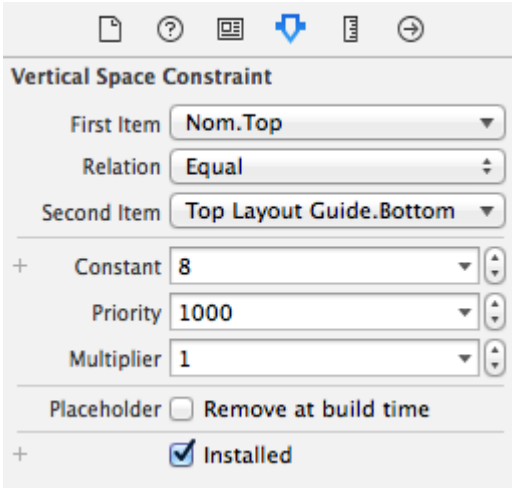
Les repères laissés peuvent avoir plusieurs couleurs :

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

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



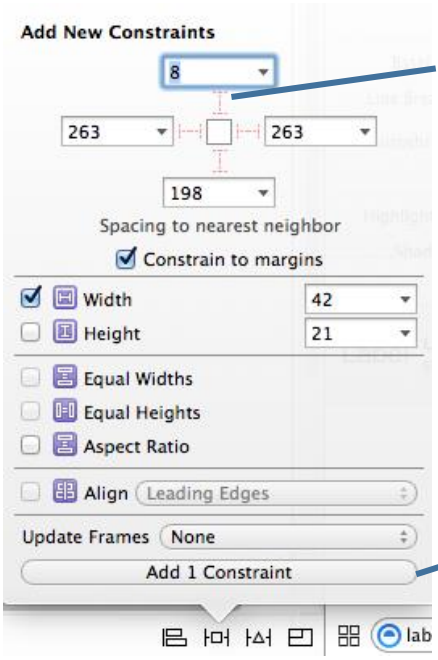
… et leur détail dans le panneau propriétés



Centrer un élément

Par ex pour centrer horizontalement :

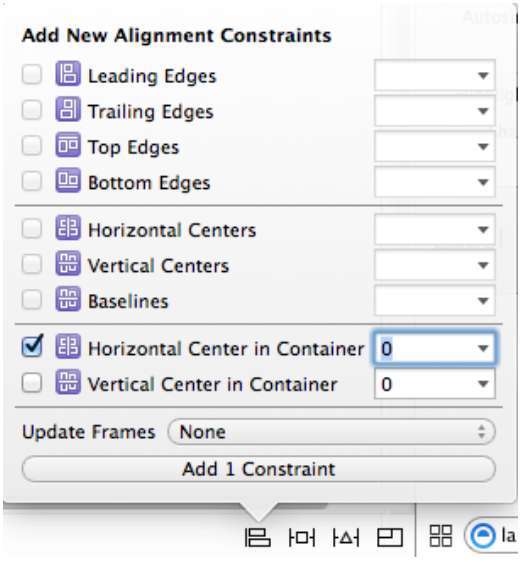
1. Ajouter une contrainte de taille (width)



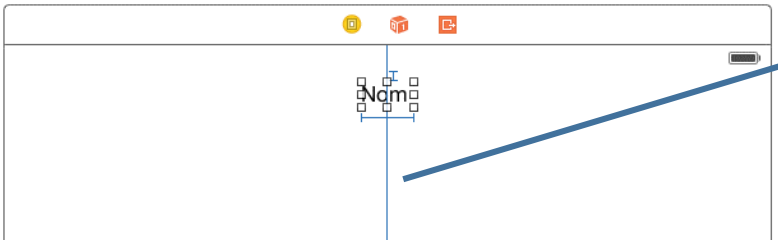
Les traits permettent de définir une marge fixe, ce que l'on fait aussi par « Ctrl + glisser

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 »

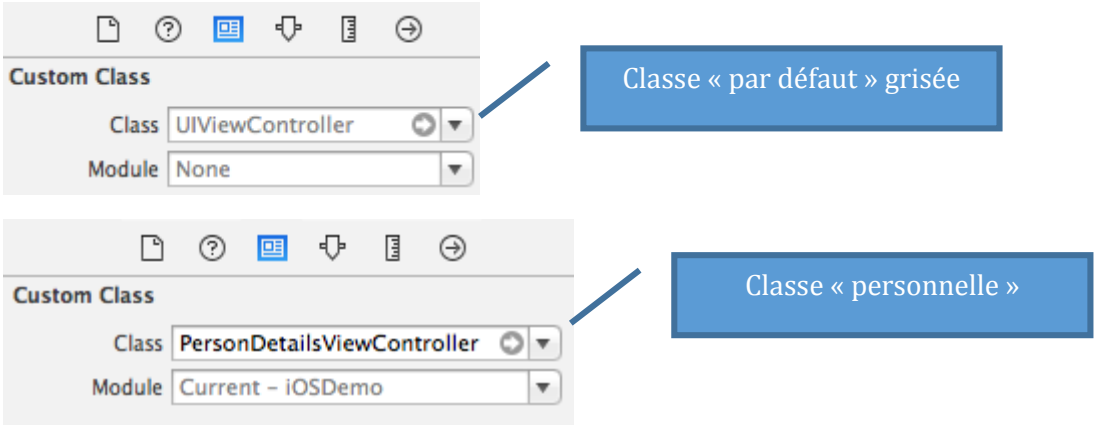
« **Add missing Constraints** » cette option peut être très utile pour ajouter automatiquement pour nous des contraintes. Par exemple on ajoute un label au centre de la vue, puis raccourci bouton « issues » en bas du storyboard (ou menu « Editor » … « Resolve Auto Layout Issues ») et « Add missing constraints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

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.

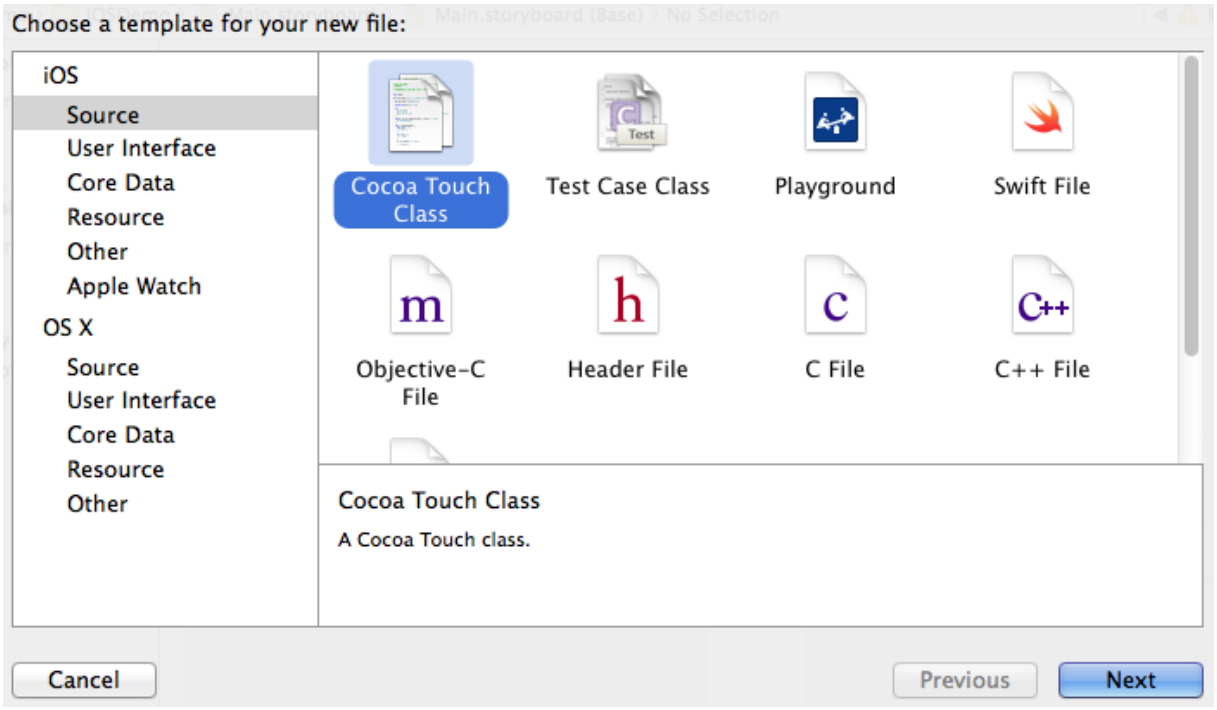
4. « ViewController » côté Code

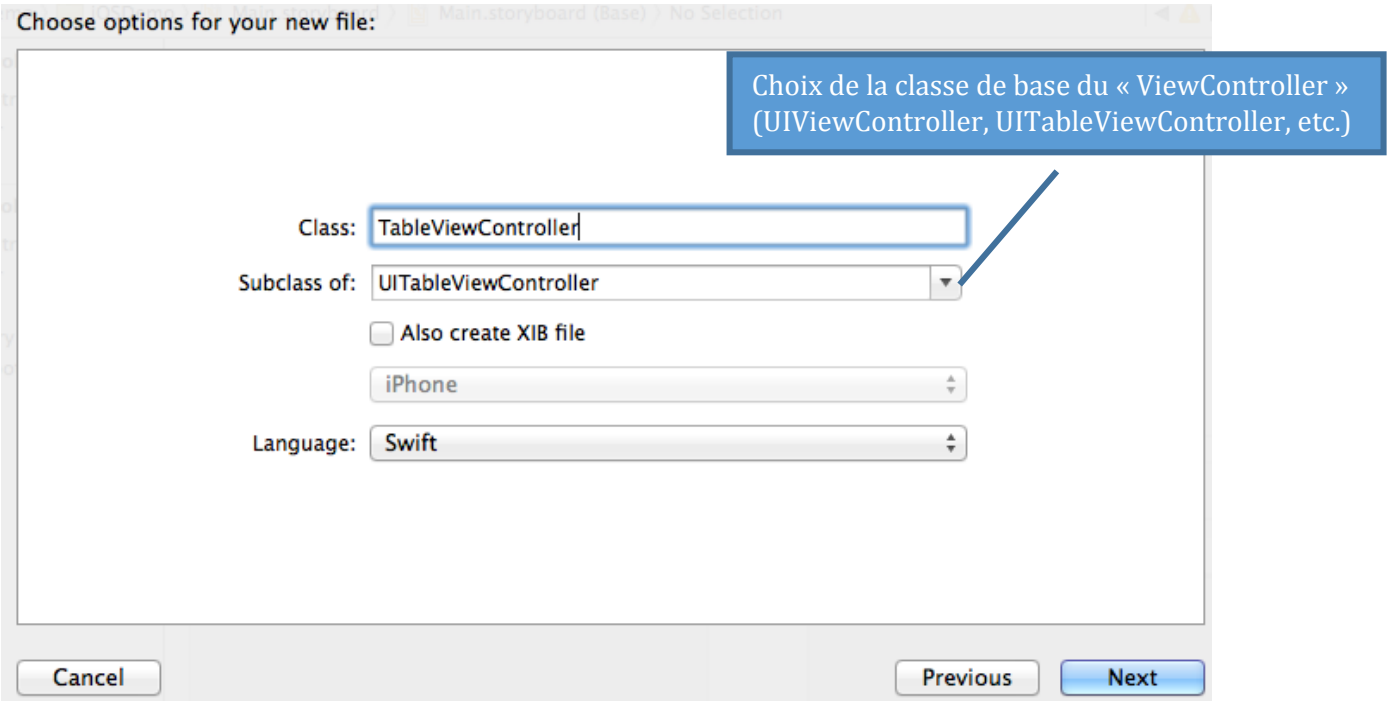
a. Ajout de la classe du « ViewController »

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

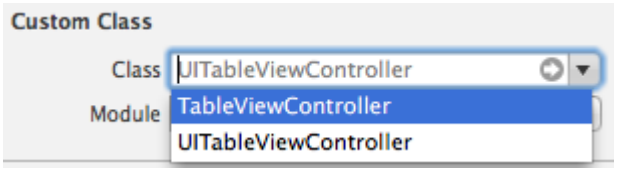


Menu « File » … « New » … « File » (Ou depuis le dossier désiré… clic droit … « New File »)





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. « TableViewController » Affichage de données sous forme de table

```
import UIKit

class TableViewController: 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, numberOfRowsInSection 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

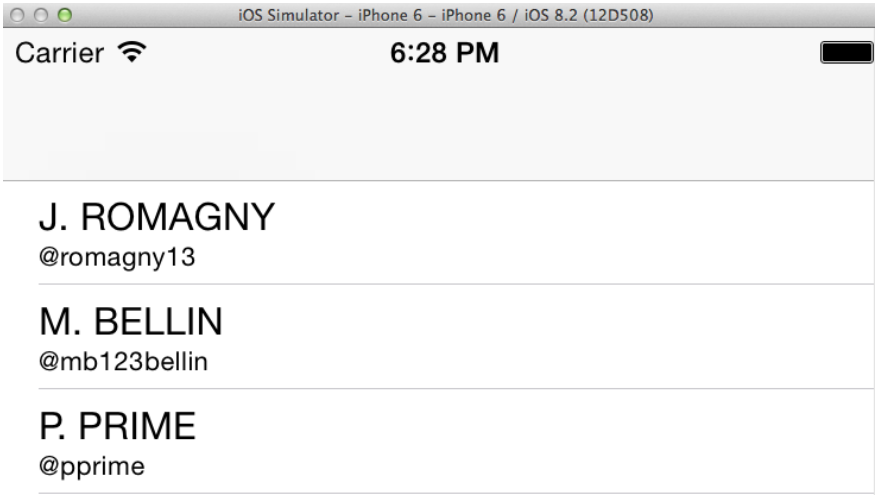
Nb de lignes

Identifier du « TableViewCell »

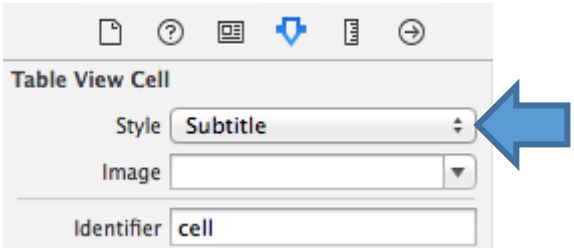
Elément affiché pour chaque ligne

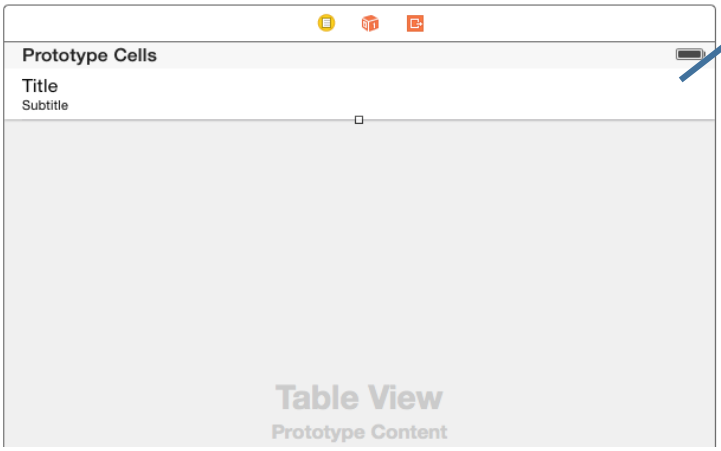
Ajouter un Identifier au « TableViewCell »

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

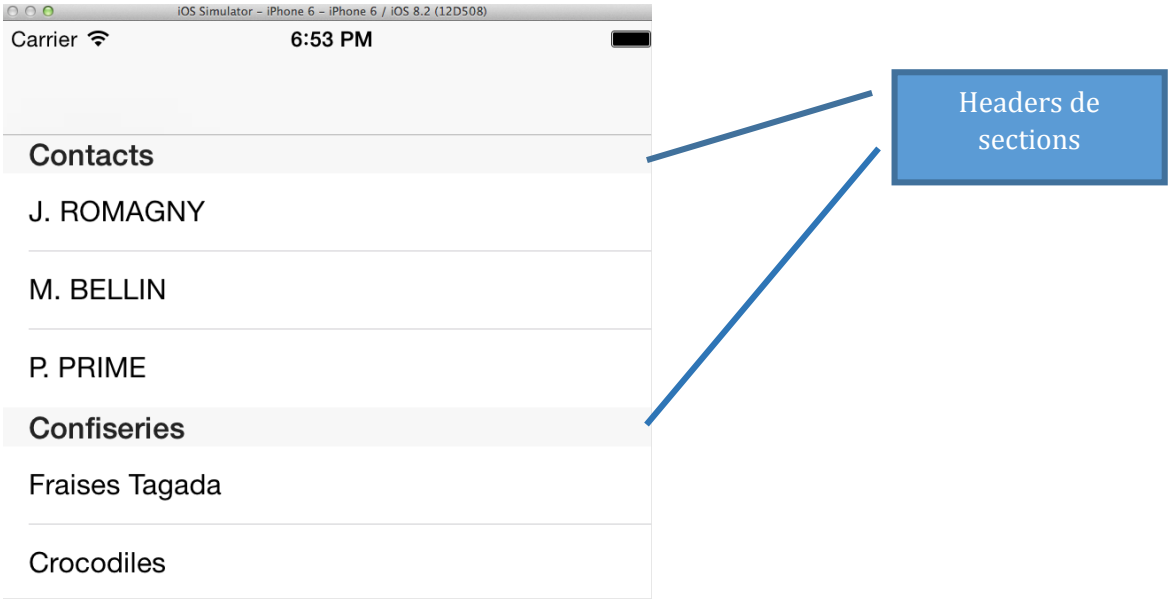
```
override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // Return the number of rows in the section.
    return people.count
}
```

2. Cellule retournée selon la section

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

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



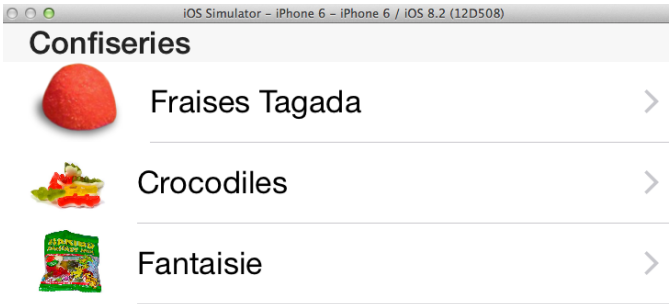
Disclosure

Avoir > à droite des cellules

Régler la propriété Accessory de « TableViewCell »

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

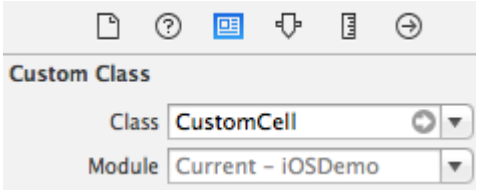
Custom Cell

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'utilser



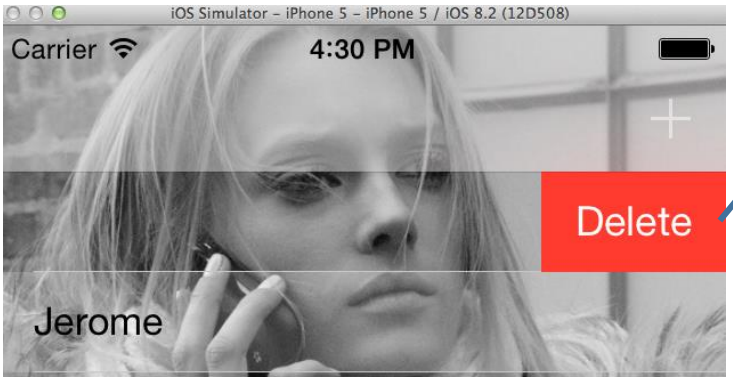
3. Utilisation dans le « TableViewController »

```
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



« Swap »

```
    // 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 {

        }
    }
```

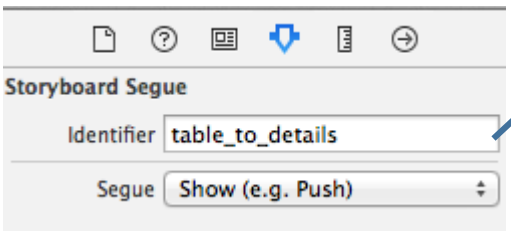
Décommenter ou ajouter le code pour prendre en charge la suppression

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

Dans le « ViewController » émetteur

```
override func 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 e « 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 identifier

Dans le « ViewController » récepteur

```
class PersonDetailsViewController: UIViewController {

   var name : String?

   override func viewDidLoad() {
      super.viewDidLoad()

      self.title = name;
   }
}
```

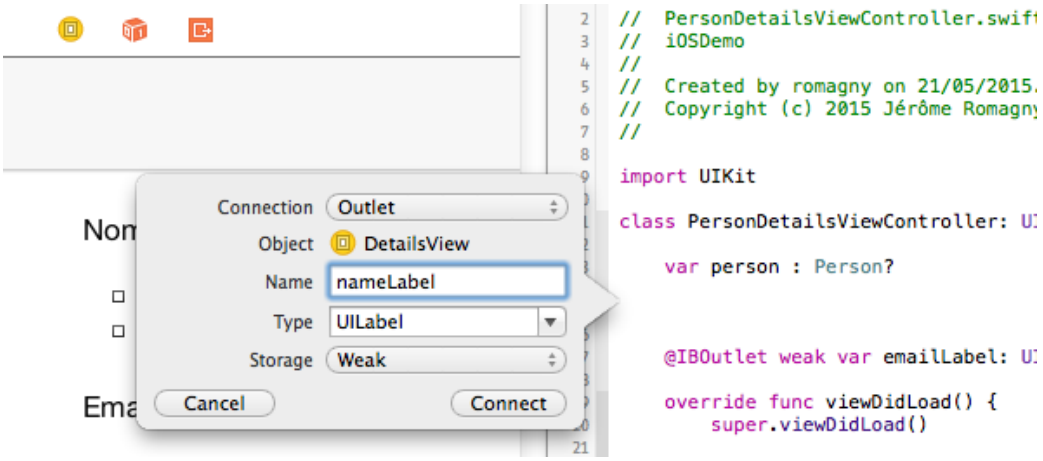
Variable remplie que l'on affiche dans la barre en titre

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…

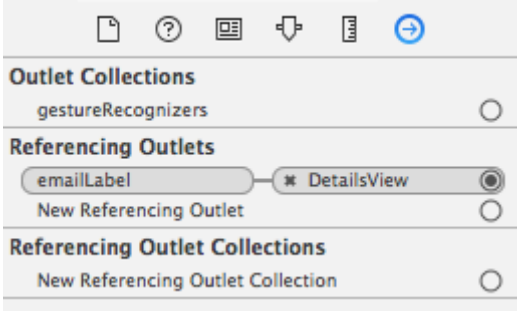


… on fait la même chose pour le label affichant l'email

```
class PersonDetailsViewController: UIViewController {

  var person : Person?

  @IBOutlet weak var nameLabel: UILabel!

  @IBOutlet weak var emailLabel: UILabel!

  override func viewDidLoad() {
    super.viewDidLoad()

    self.title = "Détails";
    nameLabel.text = person?.name
    emailLabel.text = person?.email
  }
}
```

Utilisation des Outlets pour remplir les labels avec les données de la personne

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**

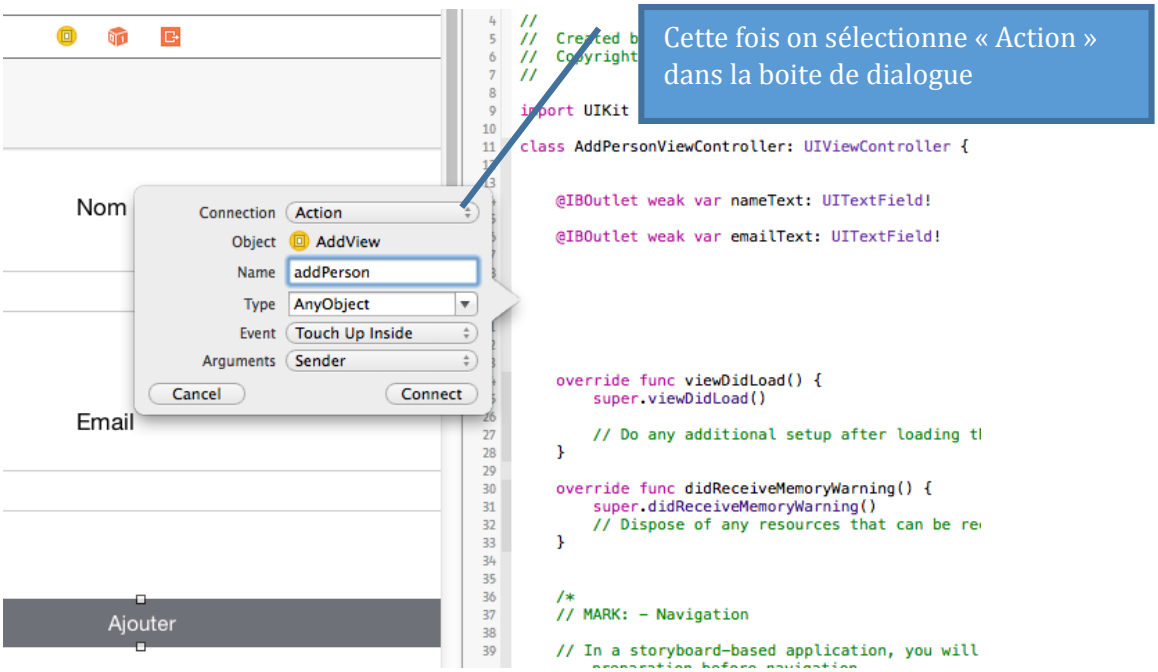
2nde Méthode

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

1ère 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.

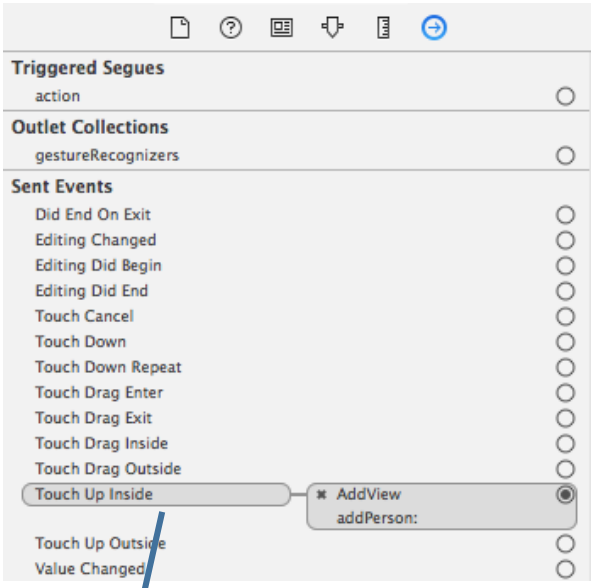


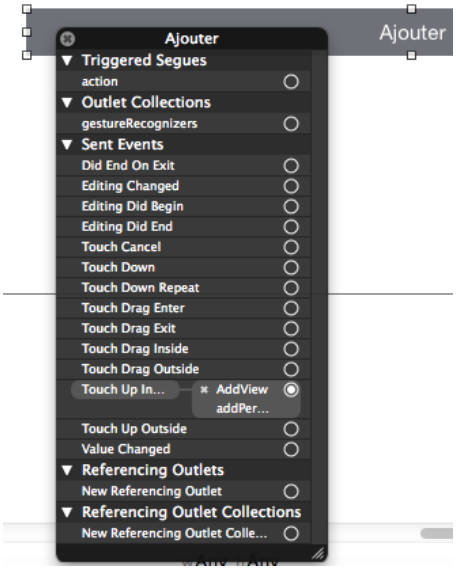
…

```swift
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

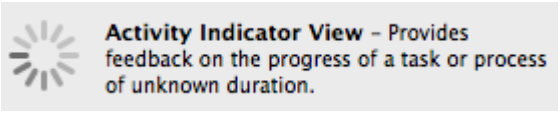
Action non connectée (cercle vide)

```swift
@IBAction func doSomething(sender : AnyObject) {
    resultLabel.text = "OK!"

}
```

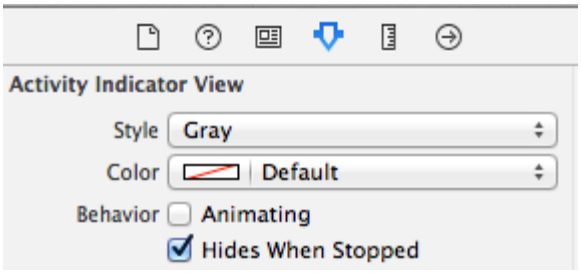
Action connectée

```swift
@IBAction func sayHello(sender: AnyObject) {
    resultLabel.text = "Bonjour \(nameText.text) !"

}
```

g. Activity Indicator



1. Glisser depuis la boite 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 = NSUserDefaults.standardUserDefaults()
      defaults.setObject("ma valeur!", forKey: "mykey")
   }

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

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

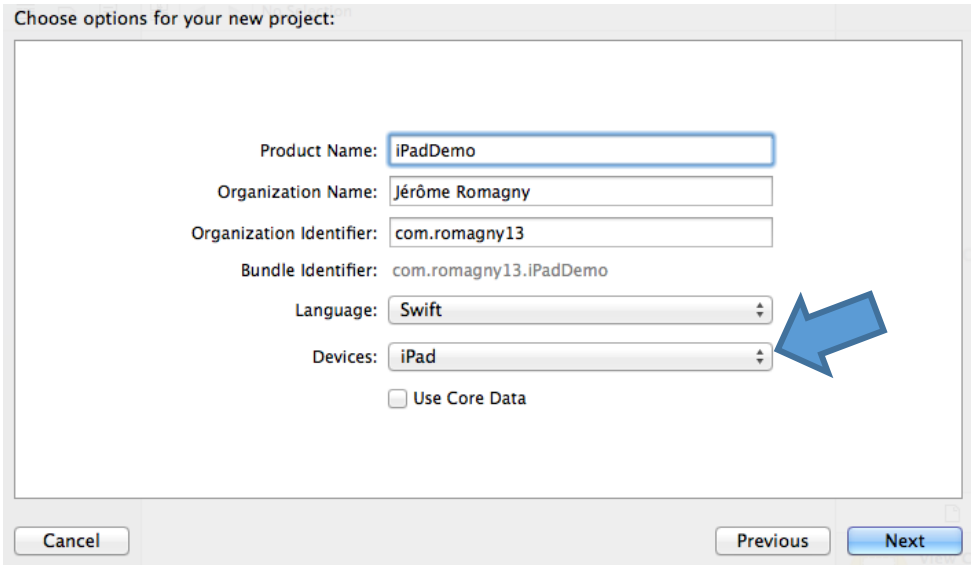
Enregistrement d'une valeur

On essaie de retrouver la valeur

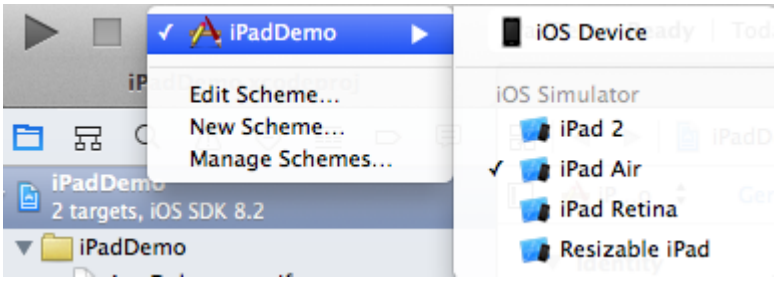
La gestion du cycle de vie de l'application est disponible depuis « AppDelegate.swift »

7. iPad (tablette)

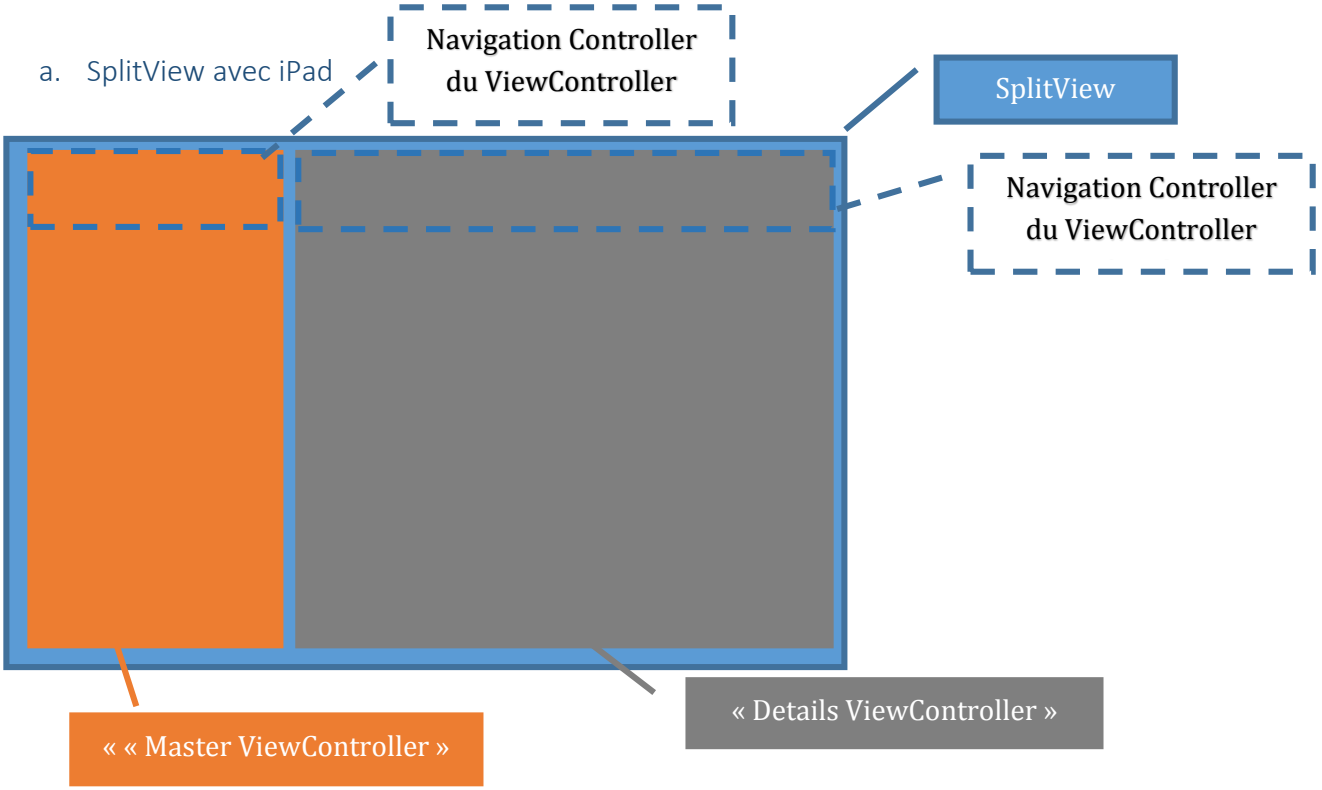
Options lorsque l'on crée un nouveau projet (« iPad » et « Universal »)



Avec le « Simulator »

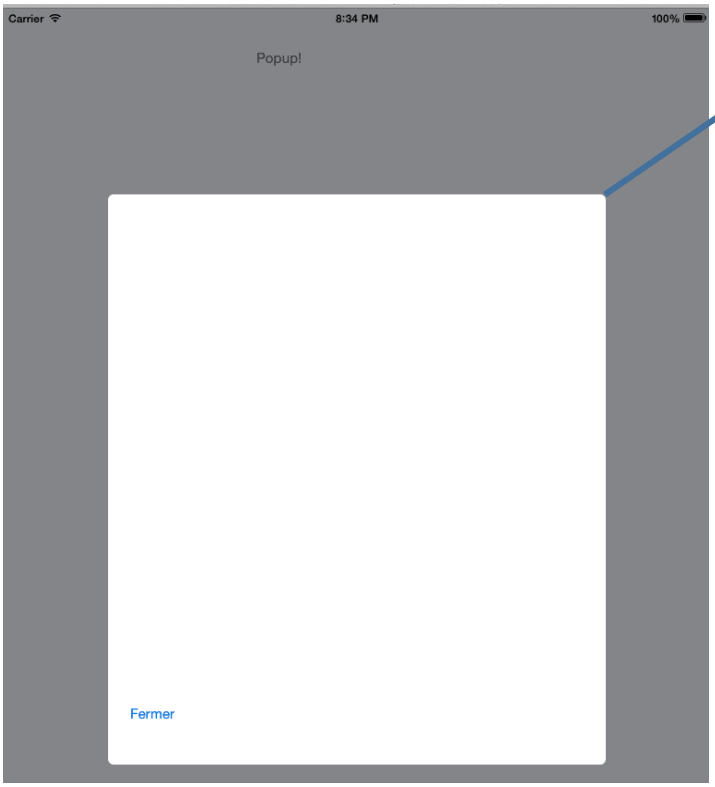


a. SplitView avec iPad



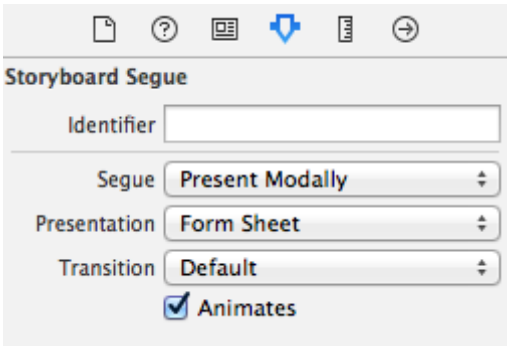
Les « settings » est un exemple de « vue master détails » sur iPad

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 boutonpermettant 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)
}
```