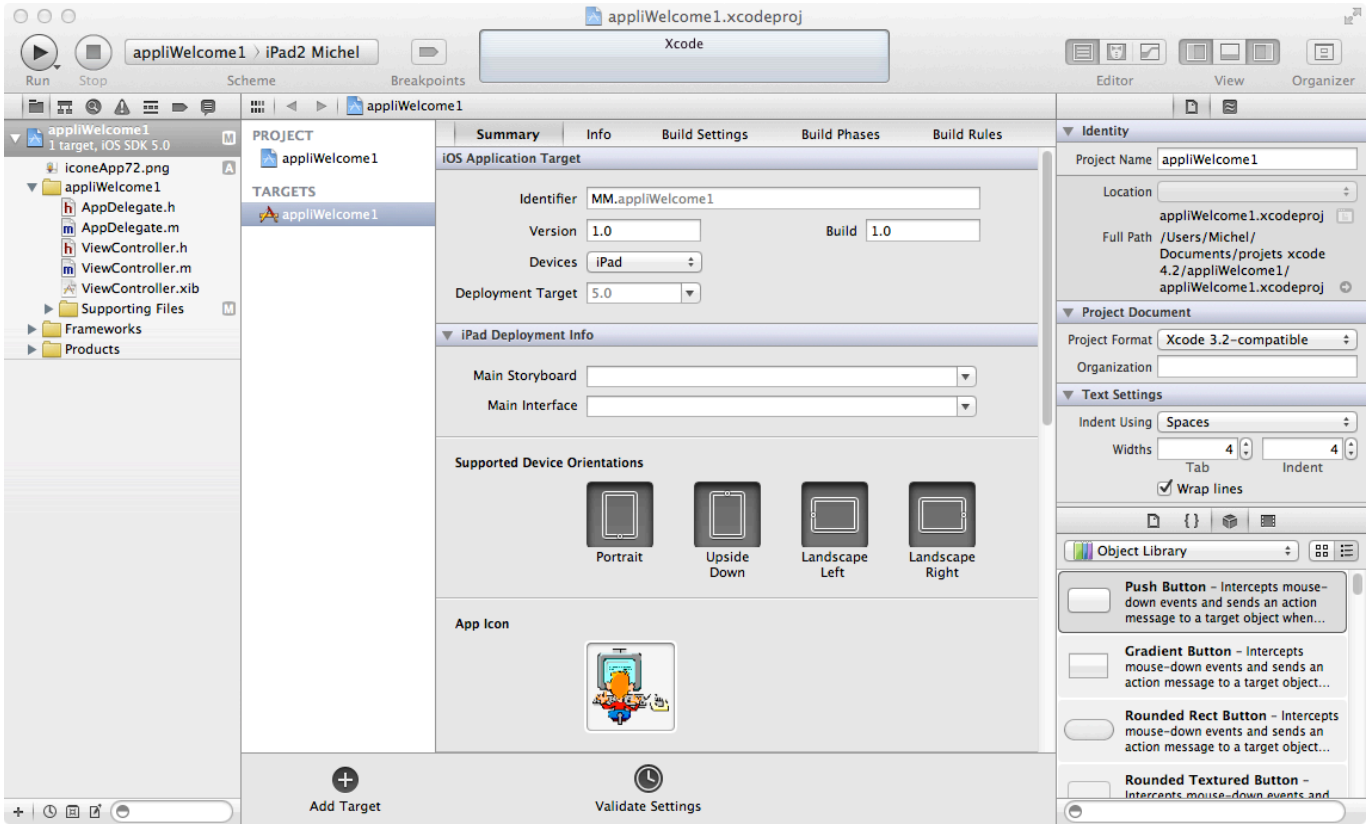
Manipulation 1 : « Welcome on iPad ».

Cette manipulation va consister à créer une application du type « Hello World » affichant un message de bienvenue lors d'un clic sur un bouton.

Création du projet

1. Créer un projet Xcode de type « **Single View Application** » et choisir une cible de type iPad. Par souci de cohérence avec le nommage utilisé dans la présentation, le nommer **appliWelcome1**. Décocher les options « **Use Storyboard** », « **use ARC** » et « **Unit Tests** ».
2. Ajouter au projet un dossier « **ressources** ». Sélectionner la racine du projet dans l'explorateur de fichiers Xcode puis clic droit **New->Group**. Ajouter dans ce dossier ressources le fichier **iconeApp72.png** et configurer cette ressource comme icône de l'application à l'aide de la fenêtre résumé de votre application comme ci-dessous :



Rapide tour des fichiers présents

Les fichiers de la vue et de son contrôleur :

- ✓ **ViewController.h** et **.m** sont les fichiers de définition et de code de la classe **ViewController** qui définit la vue et son contrôleur.
  On remarquera qu'elle hérite de la classe **UIViewController** qui contient les méthodes de base de gestion des événements sur une vue, telles que le chargement de la vue, la rotation selon la position du périphérique, etc…
- ✓ **ViewController.xib** est le fichier d'interface utilisateur de la précédente vue géré par « **Interface Builder** » (IB) qui permet de concevoir sa vue par simples « glisser-déposer » des contrôles sur la vue.

Les fichiers de la classe application :

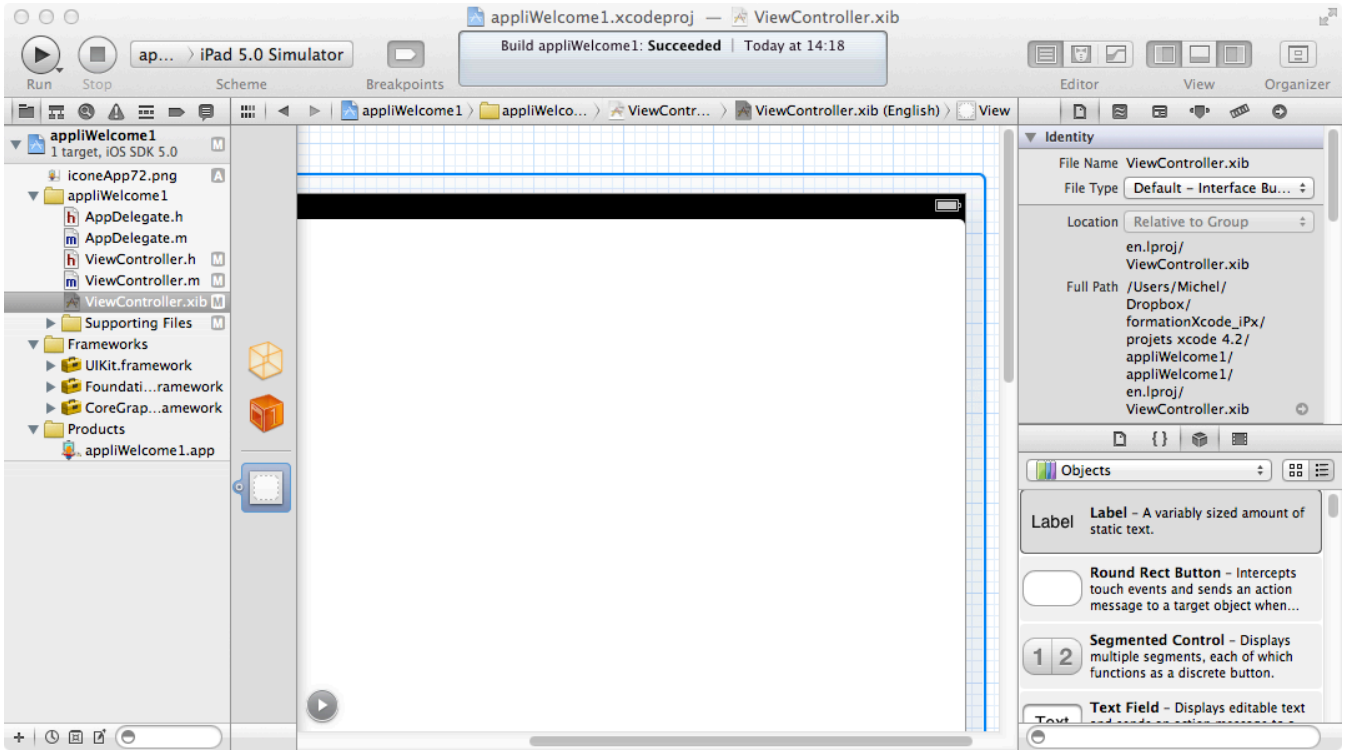
- ✓ **AppDelegate.h** et **.m** sont les fichiers de définition et de code de la classe **AppDelegate** qui gère l'ensemble des événements de la classe application ; démarrage, fin de chargement, etc...
  On remarquera qu'elle hérite de la classe **UIResponder**, classe de base des classes **UIApplication** et **UIView** permettant la prise en compte des « reponses » aux événements de ces classes.
  Elle implémente également le protocole **UIApplicationDelegate** pour la prise en compte des événements généraux de l'application, chargement, passage en background, mémoire insuffisante, etc...

Mise en place de la vue

Comme toutes les applications « Hello World », cette application de prise en main de l'outil Xcode sera simplement constituée d'une vue dans la fenêtre de l'application, cette vue disposant d'un bouton et d'un label, le label affichant un message de bienvenue lors du clic sur le bouton.

3. Faire afficher la vue dans IBuilder simplement en sélectionnant le fichier **ViewController.xib** dans l'explorateur de fichiers de Xcode.
4. Faire apparaître la fenêtre des contrôles graphiques, pour ceci sélectionner la commande **View->Utilities->Object Library**.

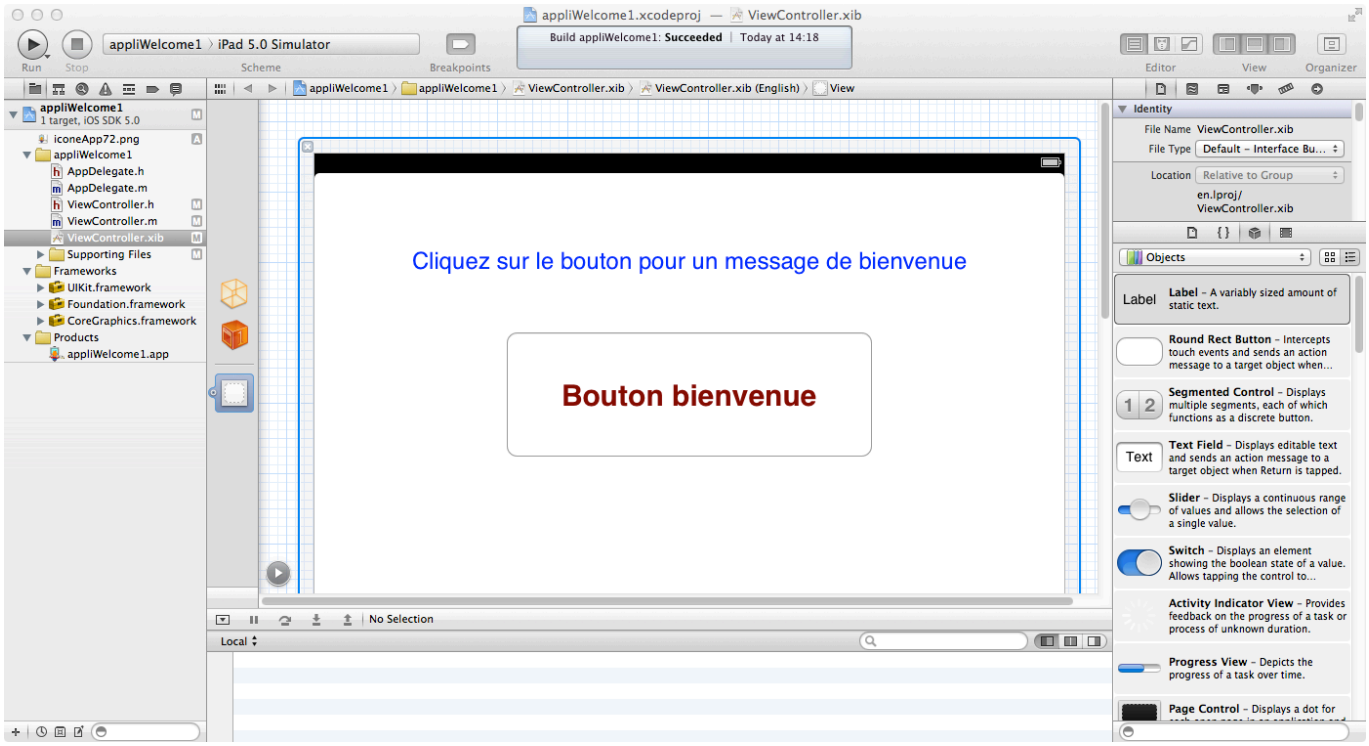
L'espace de travail Xcode devrait ressembler à peu près à ceci :



5. Depuis l'explorateur d'objets **Cocoa** glisser-déposer un contrôle « **label** » et un contrôle « **Round Rect Button** » sur la vue et les agencer à votre guise.

Pour le paramétrage des contrôles l'outil « **Attribute Inspector** » est nécessaire, ouvrir celui-ci à l'aide de la commande **View->Utilities->Attribute Inspector**.

6. Ajuster le label et le bouton comme ci-dessous par exemple (on ressent tout de suite l'âme du designer…) :



7. Ajouter à la définition de la vue (fichier **ViewController.h**) un attribut pour les objets graphiques **IBuilder** que sont **UILabel** et **UIButton** ainsi que les propriétés associées. Les propriétés seront du type « non atomiques » et utiliseront le comptage de référence.
8. Ajouter la définition d'une méthode répondant au délégué « clic sur un bouton » géré par **IBuilder**. La méthode devra avoir le prototype suivant :

   - (IBAction) actionClickSurMonBouton: (id)sender;

   La définition de la classe devrait alors ressembler à ceci :

```
@interface ViewController : UIViewController {
    IBOutlet UILabel *monLabel;
    IBOutlet UIButton *monButton;
}

@property (retain, nonatomic) UILabel *monLabel;
@property (retain, nonatomic) UIButton *monButton;

- (IBAction) actionClickSurMonBouton: (id)sender;

@end
```

9. Ajouter au fichier de code de la vue (fichier **ViewController.m**) le code de la fonction associée au clic sur le bouton. Faire afficher le texte « **Welcome on Ipad** » et éventuellement changer la couleur du texte.

   Le code de cette fonction pourrait ressembler à ceci :

```
- (IBAction) actionClickSurMonBouton: (id)sender{
    monLabel.text = @"Welcome On iPad";
```

```
        monLabel.textColor=[UIColor redColor];
    }
```

10. Compléter la méthode « **dealloc** » avec le code de libération des ressources bouton et label.

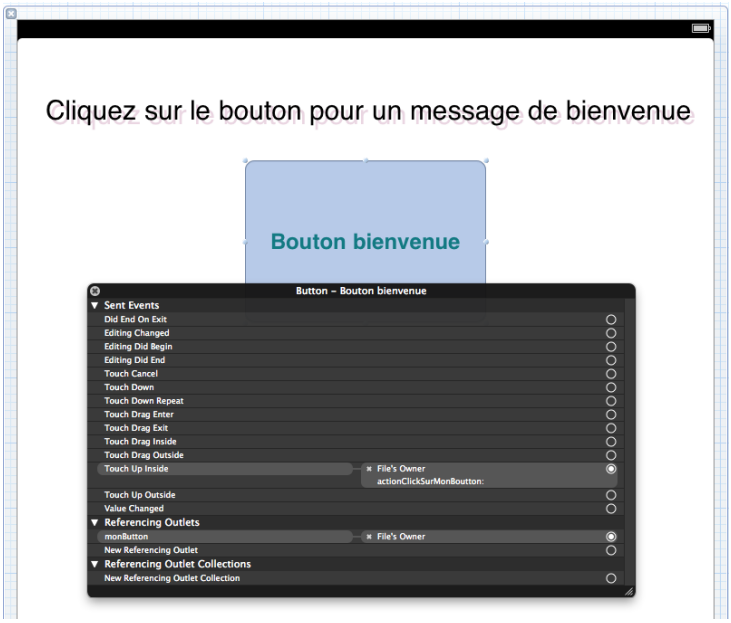
Il faut à présent lier les contrôles graphiques positionnés par **IBuilder** aux attributs définis dans le code, en effet **IBuilder** reste un outil de « design » des objets graphiques mais ne déclare pas d'objet associé dans le code.

11. Sélectionner le fichier **IBuilder** de la vue (fichier **ViewController.xib**). Vous pouvez constater qu'il s'agit d'un fichier XML de définition graphique des contrôles, pour ceci faire un clic droit sur le fichier depuis l'explorateur de fichiers Xcode puis choisir **Open As->Source Code**.
    Revenir ensuite en mode édition.

12. Pour associer l'attribut **monButton** au contrôle bouton **IBuilder**, maintenir appuyée la touche « **ctrl** », sélectionner l'icône « **File's Owner** »[1] à gauche de la vue puis cliquer sur l'icône « **File's Owner** » et sans relâcher le bouton de souris glisser la ligne sur le bouton. Lorsque vous lâchez la souris **IBuilder** vous propose les attributs qui peuvent être associés au bouton, choisissez **monButton** et l'association est réalisée.

13. Il faut également associer la méthode **actionClickSurMonBoutton** à l'action clic sur le bouton. Pour ceci effectuer un clic droit sur le bouton puis à l'aide de la souris sélectionner l'événement « **Touch Up Inside** » et sans relâcher le bouton de la souris glisser la flèche jusque sur l'icône « **File's Owner** », **IBuilder** vous liste alors les méthodes qui peuvent être associées à cet événement, choisir la seule méthode disponible.

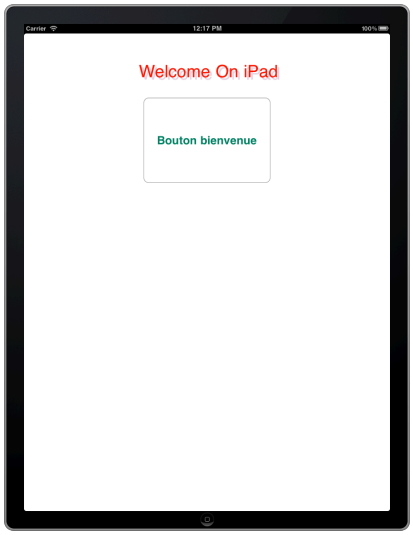
En cliquant à nouveau « bouton droit » sur le bouton vous devriez obtenir la fenêtre ci-dessous signalant les précédentes associations.



14. Associer l'attribut **monLabel** au contrôle label **IBuilder** en utilisant la méthode de la question 12.

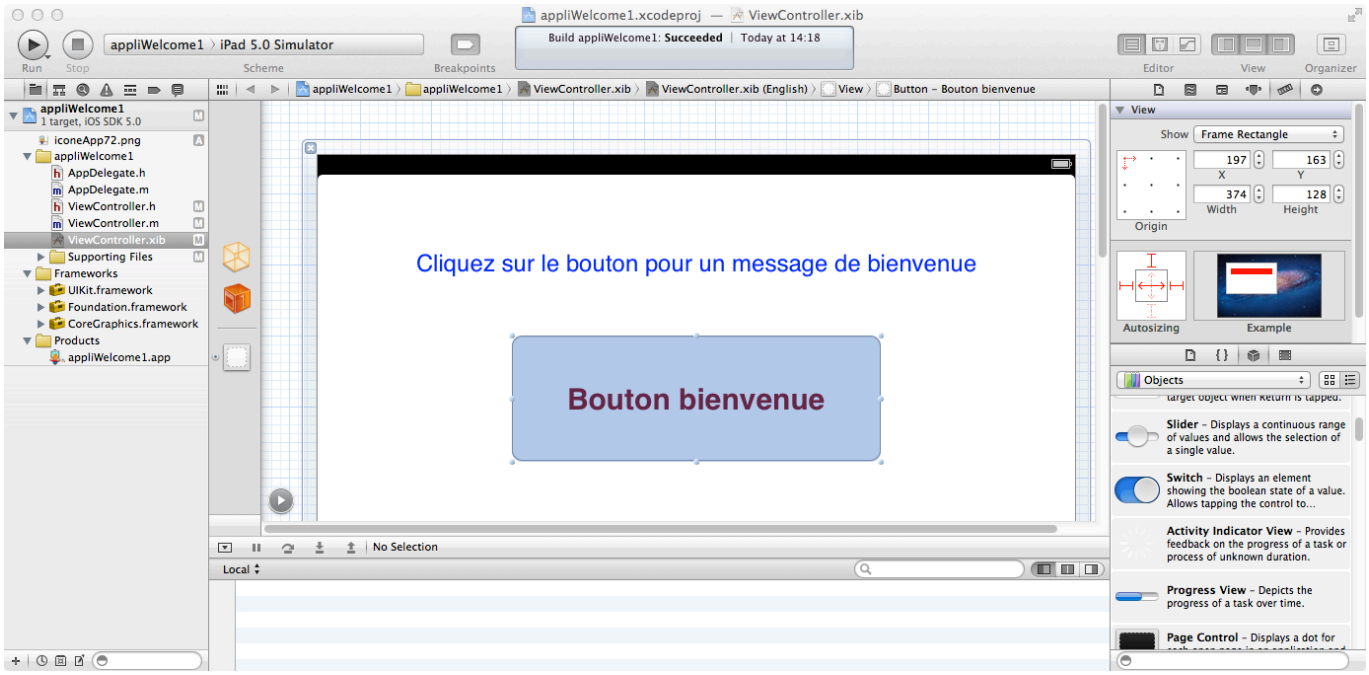
[1] **File's Owner** représente les fichiers de description de la classe contrôleur de la vue contenant les contrôles graphiques à lier. **First Reponder** est le premier objet dans la chaîne des objets qui peuvent répondre à l'événement.

Il ne reste plus qu'à construire et exécuter l'application, le résultat après clic sur le bouton devrait ressembler à l'image suivante :



Que vous utilisiez un périphérique ou un simulateur vous constaterez que la rotation de l'iPad est déjà gérée et que la fenêtre s'adapte, ceci grâce à la prise en compte de la méthode « **shouldAutorotateToInterfaceOrientation:** » de la classe **UIViewController** dont hérite votre vue. Cette méthode doit retourner « **YES** » pour les orientations que le contrôleur doit prendre en compte.
Vous remarquerez cependant qu'en mode paysage vos contrôles ne sont sans doute plus centrés.

15. Pour corriger ce petit manquement, ouvrir l'outil « **Attributes Inspector** », menu **View->Utilities->Attributes Inspector** puis sélectionner l'onglet « **size inspector** ». Faire alors en sorte d'ancrer les contrôles label et bouton en haut de la fenêtre et sur les côtés droit et gauche avec étirement horizontal (voir fenêtre ci-dessous) afin que le redimensionnement automatique s'opère.



16. Tester votre application ainsi modifiée.

Utilisation des outils « Analyse » et « Profile »

Une fois votre projet construit sans erreur ni avertissement vous pouvez approfondir l'analyse de votre code à l'aide de l'outil « **Analyse** » qui réalise une analyse statique de celui-ci et signale les éventuelles imperfections telles que les fuites mémoire pressenties…

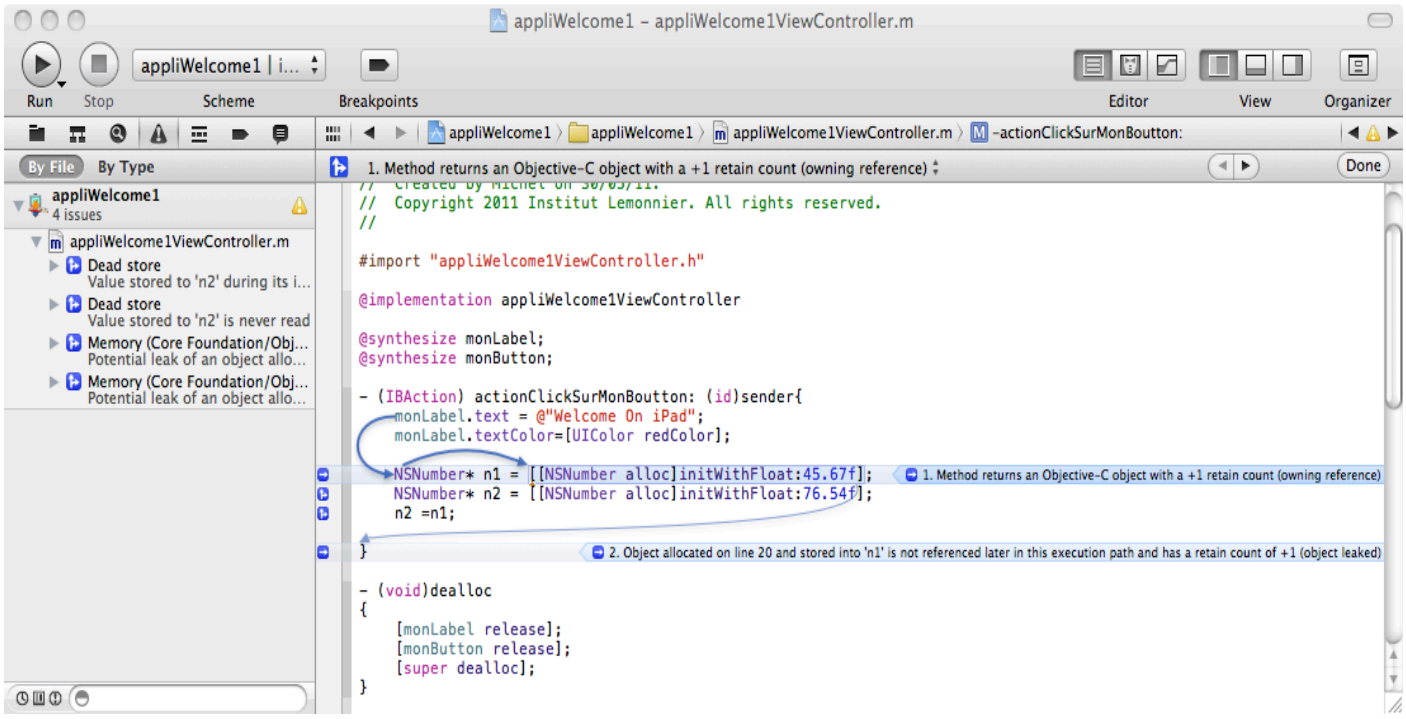
17. Démarrer l'analyse à l'aide du menu « **Product->Analyse** », logiquement aucun message ne devrait s'afficher, signe que les « 3 lignes » de code produites sont correctes…

18. Pour créer une « imperfection » au projet, ajouter un bouton avec le texte «Bouton fuite mémoire » et lui associer une méthode. Ajouter les lignes de code suivantes à la méthode précédente de la classe **ViewController**.

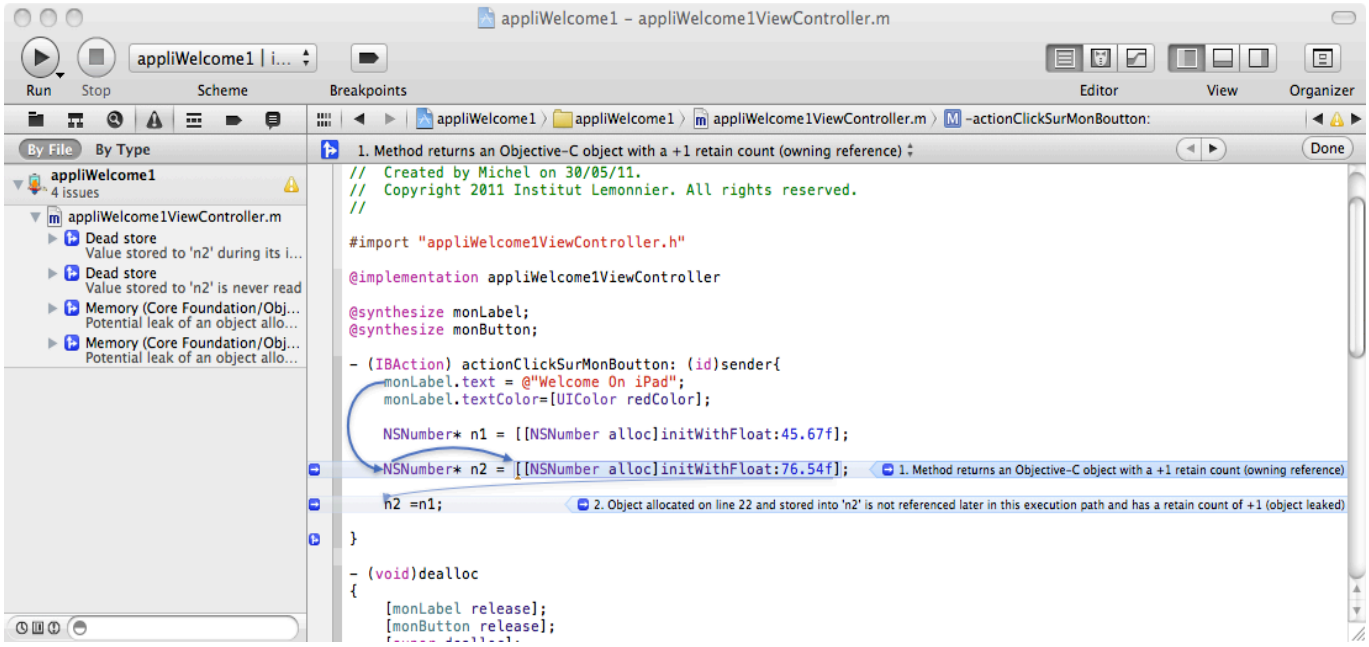
```
NSNumber* n1 = [[NSNumber alloc]initWithFloat:45.67f];
NSNumber* n2 = [[NSNumber alloc]initWithFloat:76.54f];
n2 =n1;
NSLog(@"n1 = %@",n1);
NSLog(@"n2 = %@",n2);
```

19. Démarrer à nouveau l'analyse et constater les problèmes signalés. En cliquant sur le texte des erreurs le détail du problème apparaît avec un fléchage de la séquence problématique.

Ci-dessous la mise en évidence de la fuite mémoire liée à l'instance **n1**.



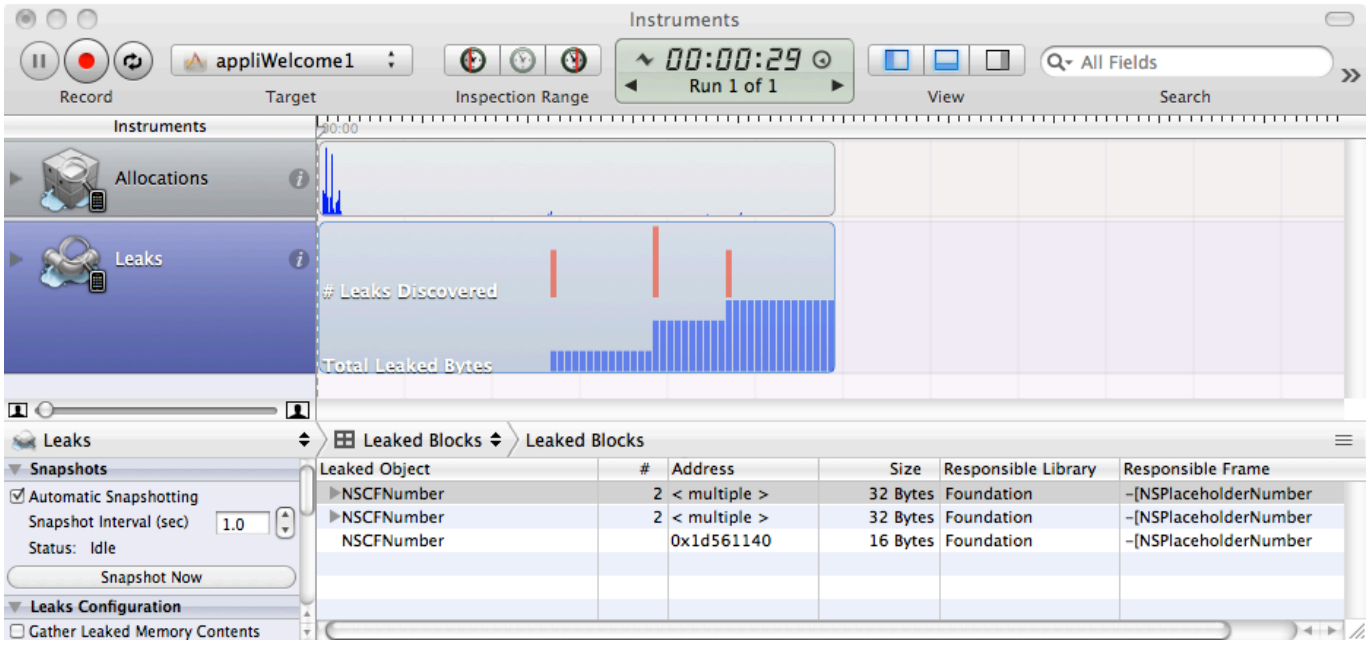
Et ci-dessous la mise en évidence de la référence « perdue » de l'allocation de l'instance **n2** qui crée également une fuite mémoire.



Il est également possible de constater « entre autres » les fuites mémoire et l'usage mémoire d'une application en cours de fonctionnement à l'aide de l'outil dynamique « **Profile** ». Cet outil permet de contrôler bien d'autres éléments comme les accès aux fichiers, l'occupation processeur, etc…

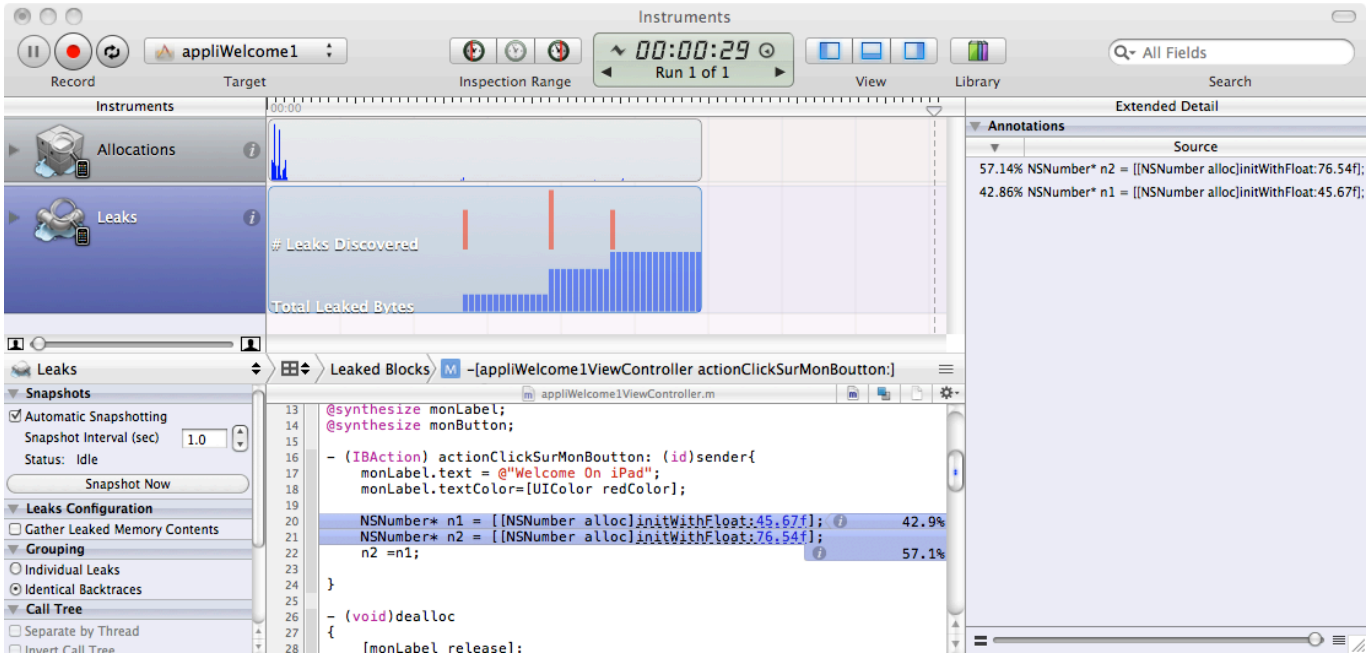
20. Démarrer l'outil « **Profile** » à l'aide du menu « **Product->Profile** » puis choisir le « **template** » « **Leaks** » permettant de contrôler les fuites et l'usage mémoire de l'application. Celui-ci démarre automatiquement l'application ouverte dans Xcode.

En cliquant sur l'instrument **Leaks** il est possible dans l'onglet « **Snapshots** » de choisir la période de rafraîchissement et de la régler à 1s par exemple. A chaque fois que l'on clique sur le bouton « bienvenue » de votre application, une fuite mémoire de deux objets **NSNumber** est créée. Ci-dessous le résultat après trois clics sur le bouton :



Pour avoir le détail de la pile des appels, dans « **Instruments** » activer la vue détaillée à l'aide du menu « **View Extended Details** ». La sélection d'un « **Leaked Objet** » fait alors apparaître la méthode associée.

Un double clic sur la méthode « met alors le doigt » sur les lignes de code responsables de la fuite.



21. Corriger la fuite mémoire et vérifier que votre application et bonne pour l'Apple Store…

Extension possible : ajout de la « localisation »

La localisation va consister à faire en sorte, dans les limites définies par l'application, de prendre en compte la langue définie dans les réglages généraux du matériel.
Nous allons compléter notre application pour quelle gère la langue anglaise (par défaut sur XCode) et la langue française, ainsi notre application devra adapter ses affichages en fonction de la langue définie dans les réglages du matériel.

22. Dans la fenêtre explorateur de projet sélectionner la racine du projet puis l'onglet **Info**. Dans la rubrique « **Localizations** » vous devez disposer pour l'instant du seul langage « **English** », ajouter le langage « **French** ».

Vous pouvez alors constater que le fichier de vue IBuilder « **ViewController** » s'est dédoublé, il existe à présent une version « **English** » et une version « **French** » de votre vue.

23. Recopier les contrôles de la vue initiale dans la nouvelle vue, traduire les messages dans la langue de la vue puis associer les contrôles de la vue aux objets et événements de la classe contrôleur qui reste unique quant à elle.

24. Construire et tester l'application qui doit afficher la vue selon la langue choisie dans les réglages généraux du matériel.

Il reste à faire en sorte que le texte affiché dans le label suite au clic bouton soit également fonction de la langue sélectionnée. Le principe consiste à établir une liste des correspondances « **Clés->Valeurs** » et d'associer à chaque message à afficher une clé qui permettra de retrouver la valeur à afficher selon la langue. On dispose pour cela des fichiers **InfoPlist.strings** (un par langue) qu'il faut compléter des clés à définir et des valeurs associées dans chaque langue.

La syntaxe à employer dans ces fichiers est simple, l'exemple suivant associe à la clé « **monLabelKey** » la valeur « **Welcome On Ipad 2 !!** ».

```
"monLabelKey" = "Welcome On Ipad 2 !!";
```

25. Compléter les fichiers **InfoPlist.strings** avec la clé qui servira à trouver le texte à afficher dans le label suite au clic sur le bouton.

Pour lire la valeur associée à une clé dans un fichier de correspondances **Clés-Valeurs** (.strings) il suffit d'utiliser la méthode **NSLocalizedStringFromTable** du **Foundation Framework**.

26. Modifier le code de la méthode **actionClickSurMonBouton** pour afficher le texte selon la langue sélectionnée.

27. Vérifier le bon fonctionnement de votre application « internationale… ».