


Approche RAD pour l'iPhone SDK avec Interface Builder

par [Morvan Mikaël \(Morvan Mikaël\)](#) (Blog)



Date de publication : 08/05/2008

Dernière mise à jour : 08/05/2008


Cet article est un tutoriel sur l'iPhone SDK et notamment l'approche RAD avec l'utilisation d'Interface Builder Je vais montrer pas à pas comment réaliser une  **IHM** avec le minimum de programmation et en utilisant le très peu documenté Interface Builder

I - Introduction.....	3
II - Description de l'application.....	5
III - Xcode.....	5
IV - Interface Builder.....	8
V - Retour à Xcode.....	16
VI - Retour à Interface Builder.....	18
VII - On a terminé.....	18
VIII - Conclusion.....	19

I - Introduction

J'ai éprouvé beaucoup de difficulté pour réaliser une  IHM avec l'iPhone SDK (beta 4) ! Manque de documentation, manque d'exemple, et même sur le net, j'ai eu beaucoup de mal à trouver des exemples concrets de réalisation d' IHM.

J'ai donc décidé de réaliser ce tutorial pour que le maximum de personnes puisse accéder au SDK de l'iPhone.

L'objectif de ce tutorial sera de présenter la programmation d' IHM sur l'iPhone SDK mais avec une approche RAD. Pour mener à bien cette approche, j'ai utilisé Xcode bien entendu mais surtout Interface Builder avec son approche bien particulière et surtout : ses bugs !

Soyons clair, un gros travail est nécessaire afin de comprendre l'Objective-C et pour manipuler les concepts bizarres du genre la gestion de l'appel des méthodes avec les crochets. Mon tutoriel ne sera utile que si vous prenez le temps d'étudier la doc « iPhone OS Programming Guide ».

Cette documentation est disponible sur <http://developer.apple.com/iphone/program/>. Une fois que vous vous êtes inscrits en créant votre Apple Id, vous pourrez télécharger le SDK et la documentation.

Un lien qui peut être utile pour débiter en Objective-C: [Pierre Chatelier: De C++ à Objective-C](#)



II - Description de l'application

L'application sera très très simple, il s'agit de réaliser une application contenant 3 vues différentes avec une TabBar permettant de passer d'une vue à l'autre. Un screenshot de l'application est présenté ci-dessus. Pour bien comprendre le fonctionnement, j'ai décomposé les objets utilisés par l'application :

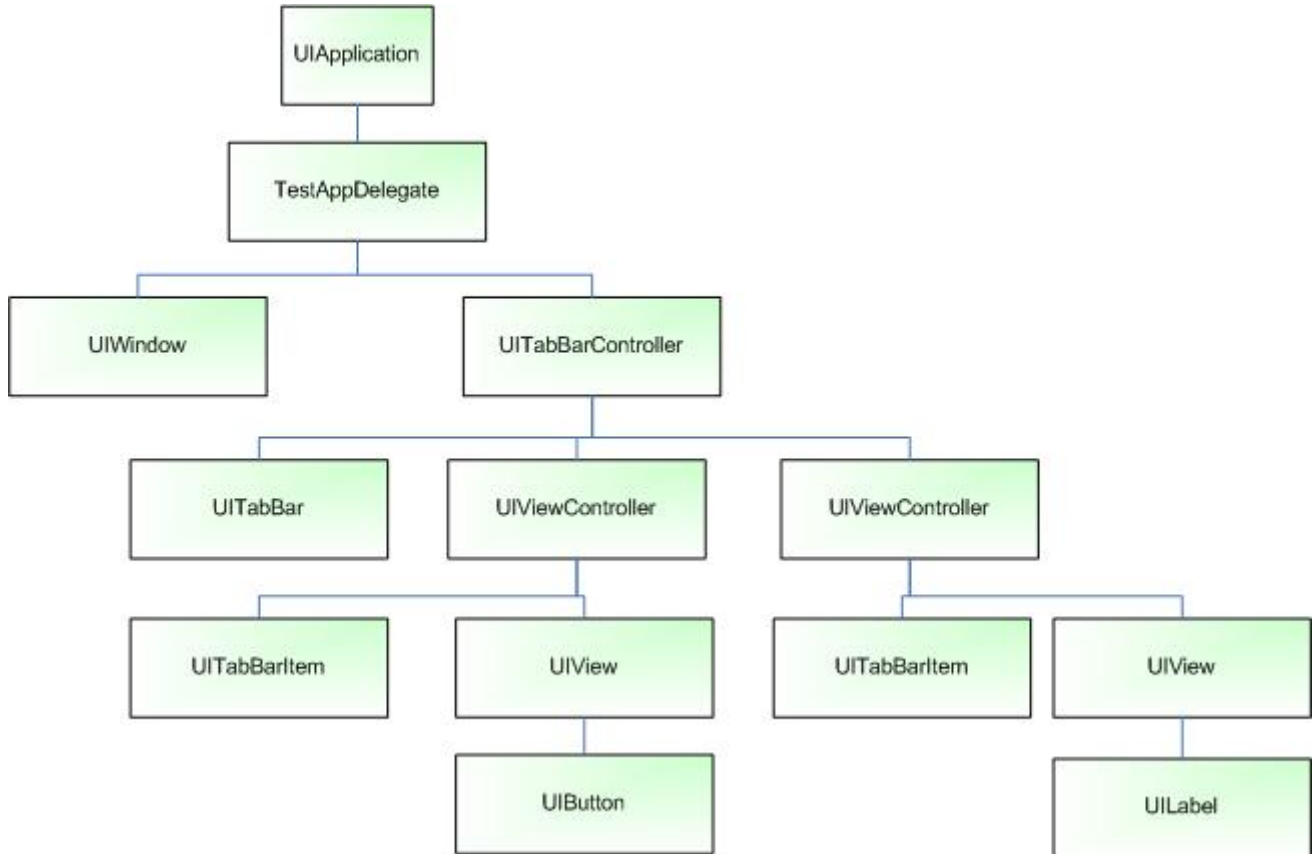
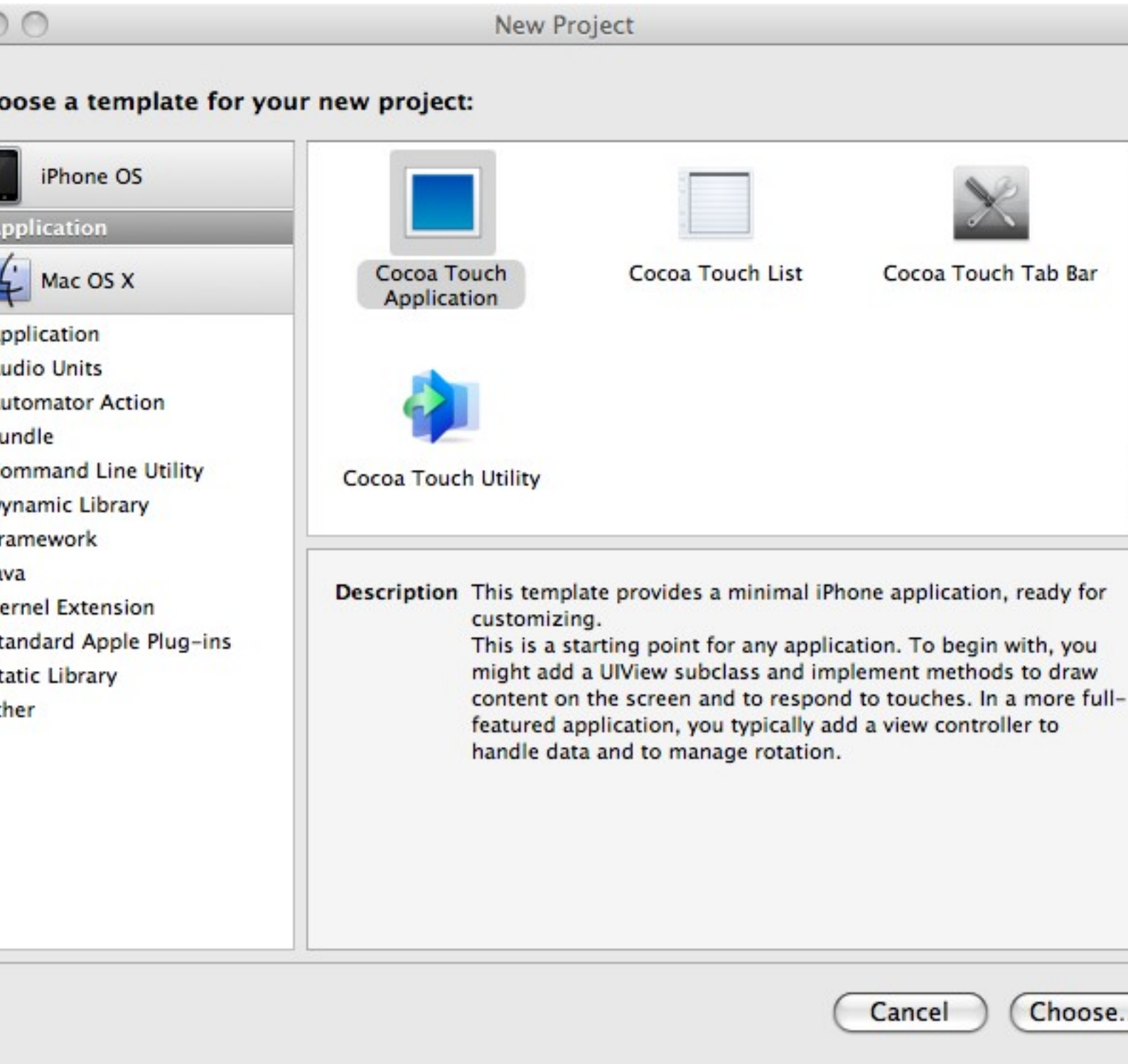


Schéma de l'application

Maintenant, nous allons commencer à réaliser le squelette de notre application avec les assistants de Xcode.

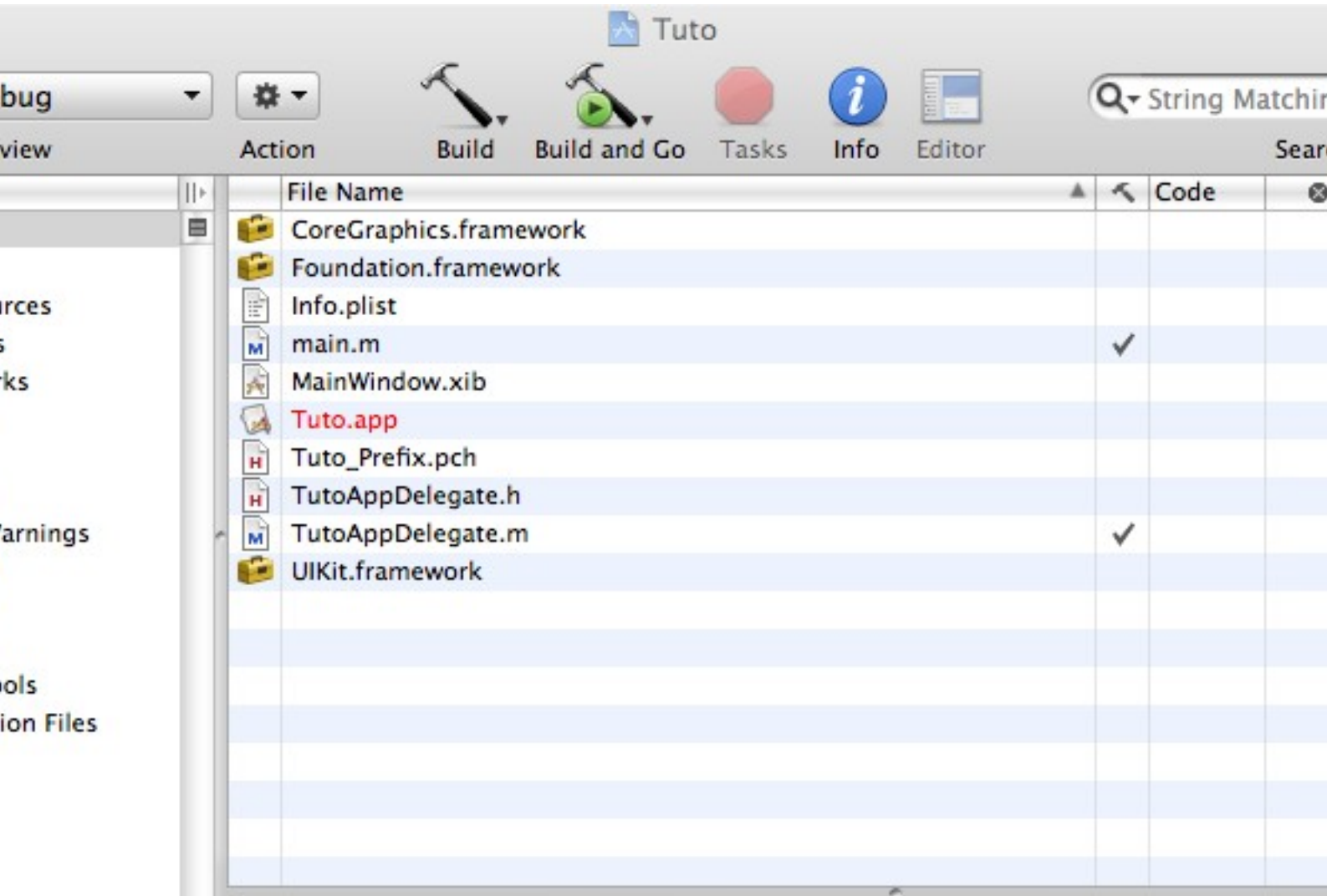
III - Xcode

Lancez Xcode, sélectionnez « Cocoa Touch Application » puis cliquez sur « Choose... » Saisissez « Tuto » comme nom pour votre application.



Nouveau projet iPhone SDK

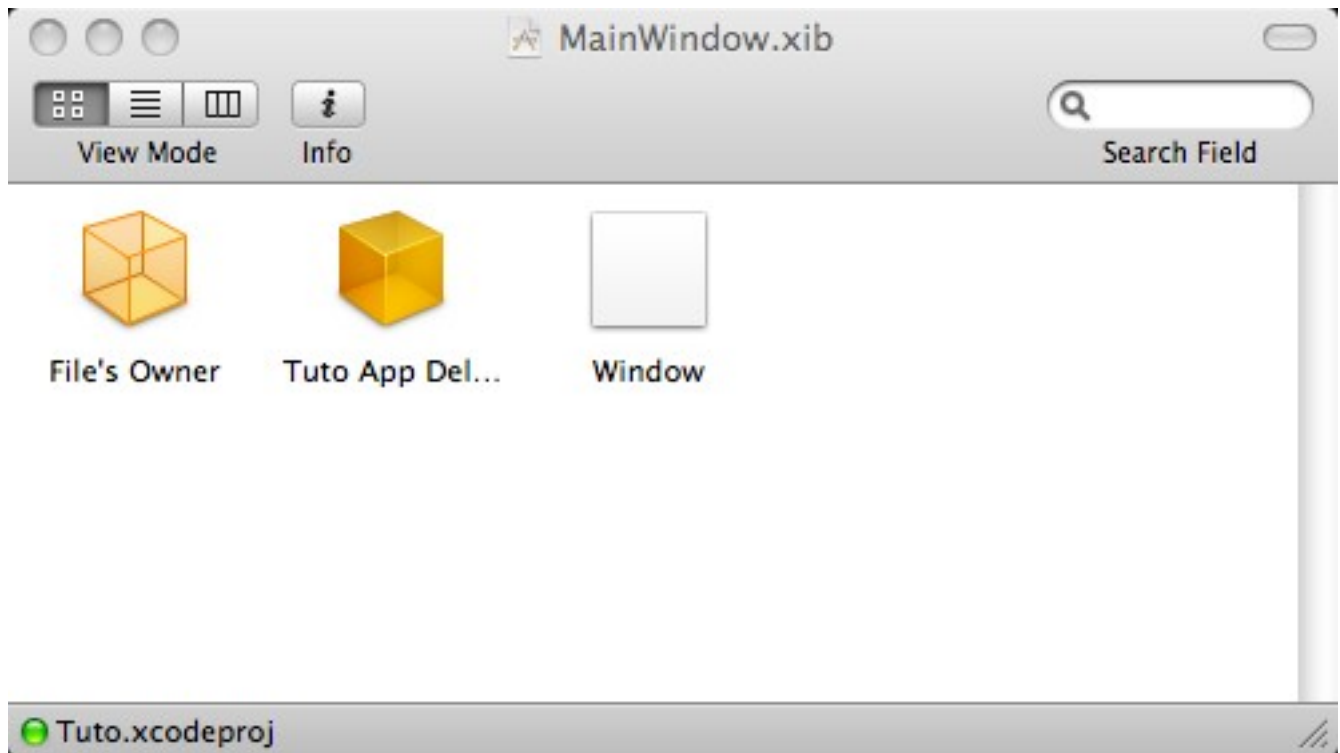
Vous arrivez sur la fenêtre suivante:



Fenêtre d'un nouveau projet

Double-cliquez sur le fichier « MainWindow.xib » et vous devez arriver sous Interface Builder. Ce fichier représente en quelque sorte les ressources de l'application (un peu comme un .res en C++).

IV - Interface Builder



Interface Builder

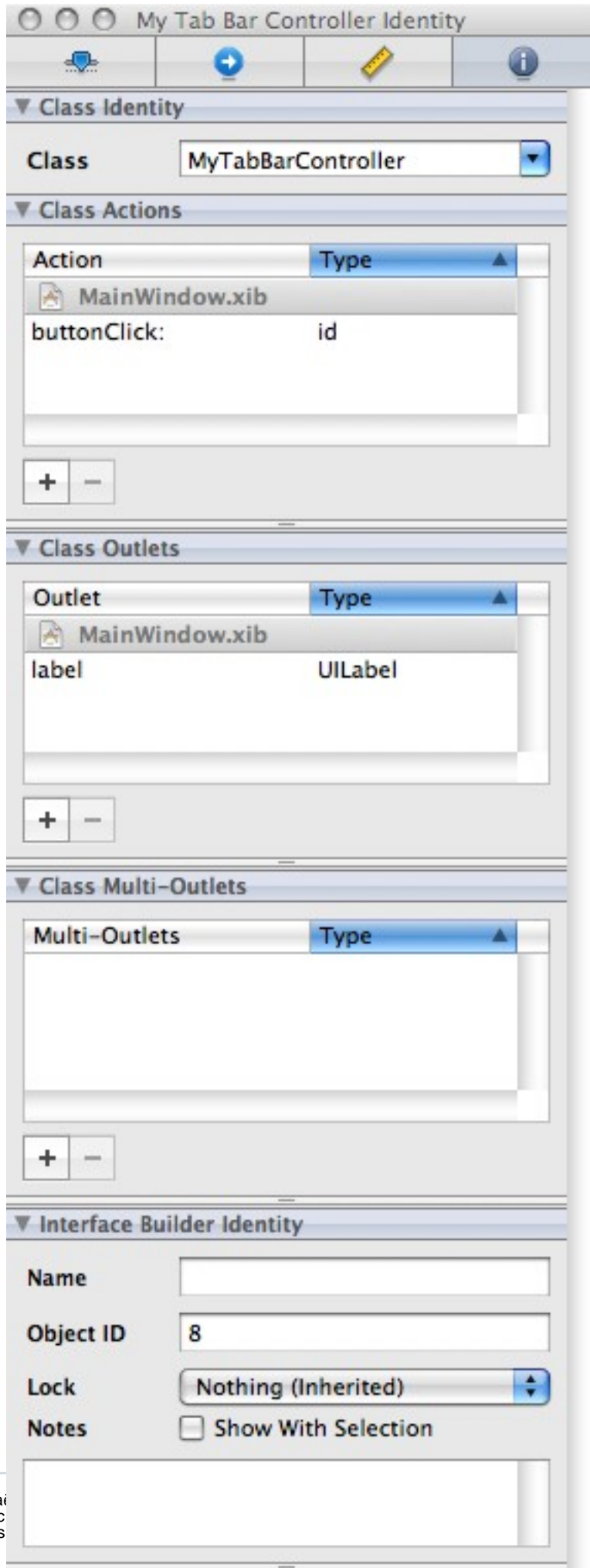
Allez dans le menu Tools et cliquez sur Library et Inspector.
Dans la fenêtre Library, sélectionnez Tab Bar Controller puis « Drag Drop » sur la fenêtre.



TabBarController

Maintenant, nous allons ajouter la gestion des événements ainsi que la gestion du label à notre TabBarController.

En premier, sélectionnez l'item Tab Bar Controller dans la fenêtre principale d'Interface Builder. Dans la fenêtre gérant les attributs, sélectionnez l'onglet information (i). Saisissez « MyTabBarController » comme nom de classe. Ajoutez une action nommée « buttonClick : » (attention aux « : ») de type « id » et un Outlet nommé « label » de type « UILabel ». Vous devez obtenir quelque chose comme ça :



On va maintenant compléter l'IHM de notre TabBarController.
Double-cliquez sur le Tab Bar Controller pour obtenir la fenêtre représentant votre interface :



Première fenêtre du TabBarController

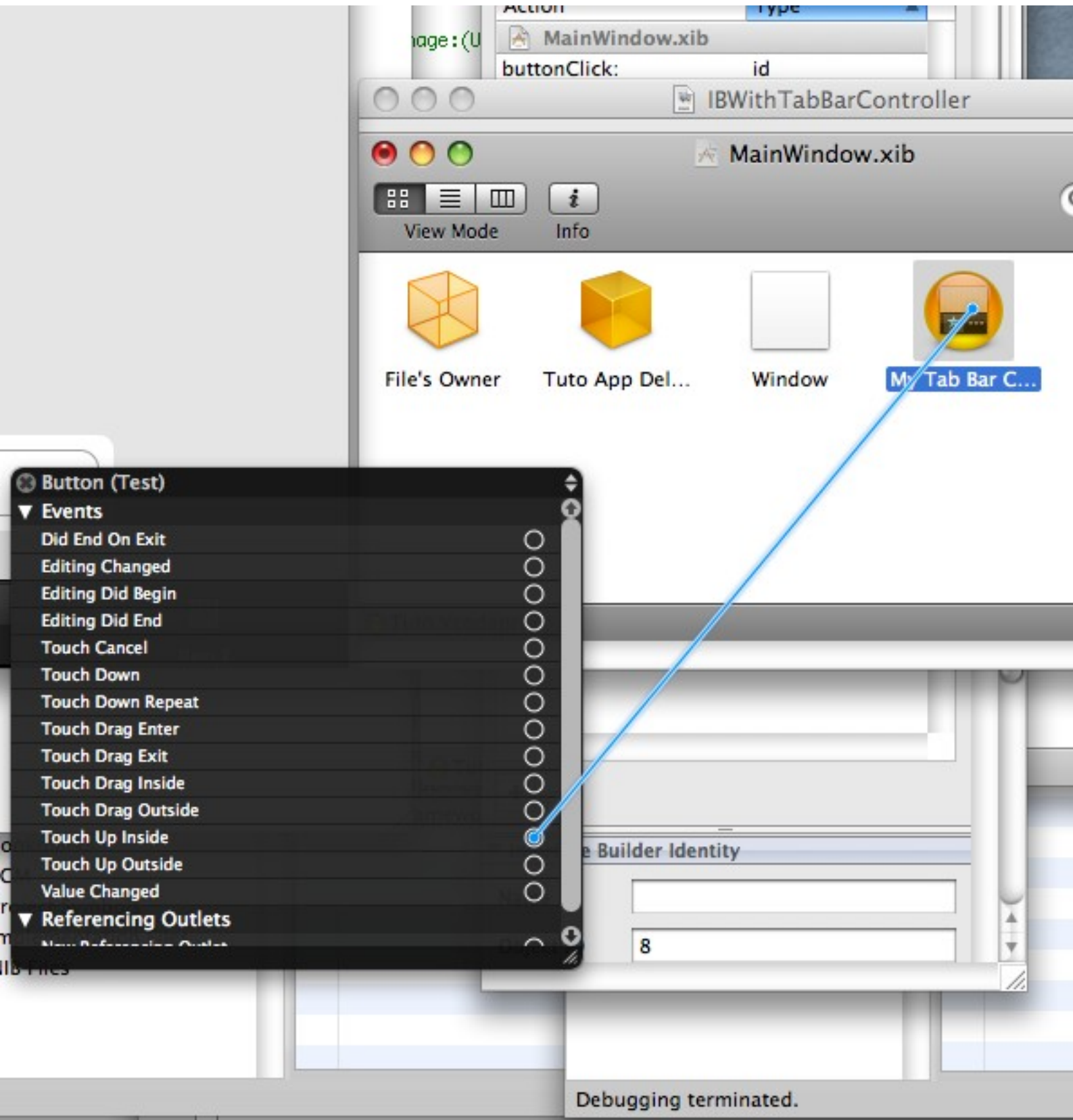
Cliquez sur le premier bouton Item1 en bas à gauche de la fenêtre, sélectionnez un contrôle UIView dans la library et « Drag drop » sur la fenêtre.

Ajoutez également un « Round Rect Button » sur la View.



Le controle UIView

Maintenant, on va relier l'événement Touch up Inside de notre bouton à notre TabBarController :
Faites un clic droit sur le bouton: une fenêtre s'ouvre. Dans la liste des événements proposés, reliez à l'aide de la souris l'événement au TabBarController. Une fenêtre buttonClick vous sera proposée. Acceptez-la.



Événement Touch up Inside

Ajoutez une view pour le deuxième Item et ajoutez un label.

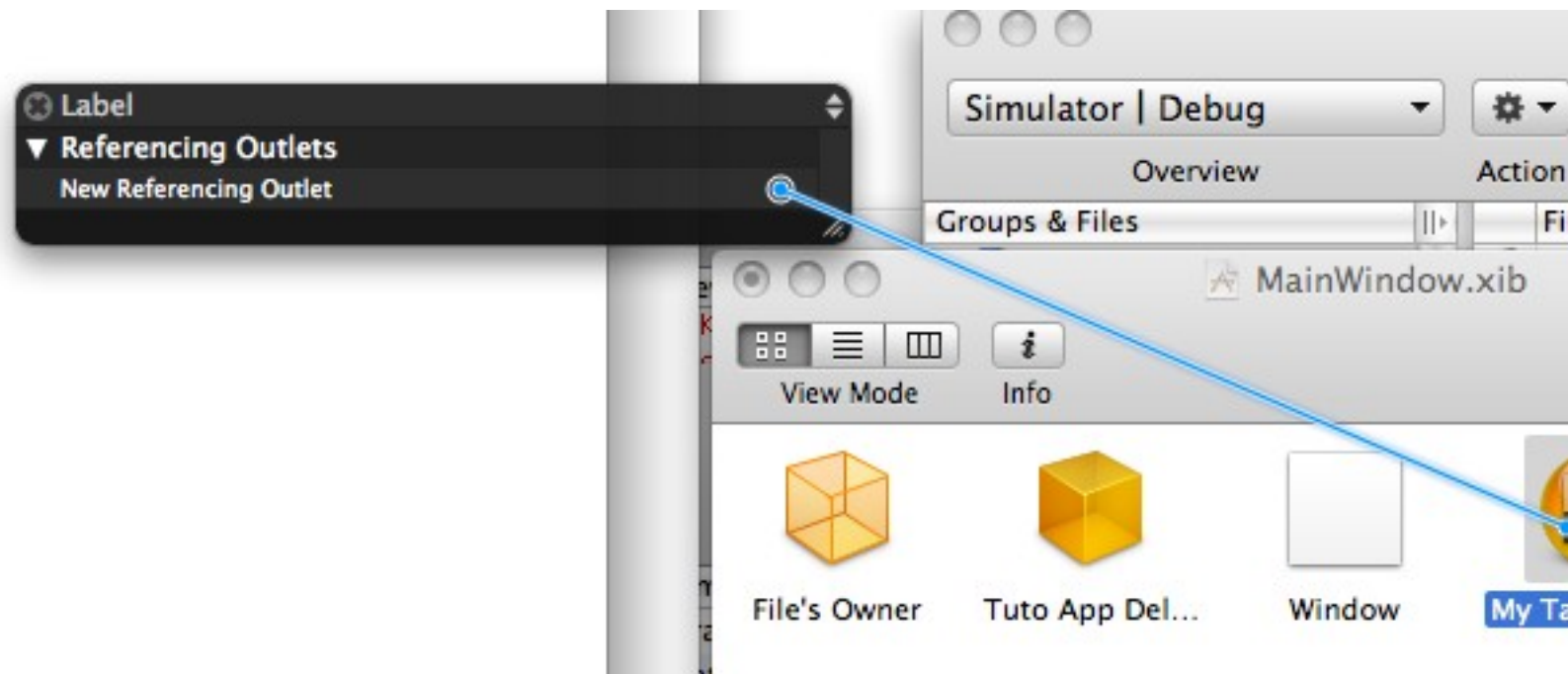


Label



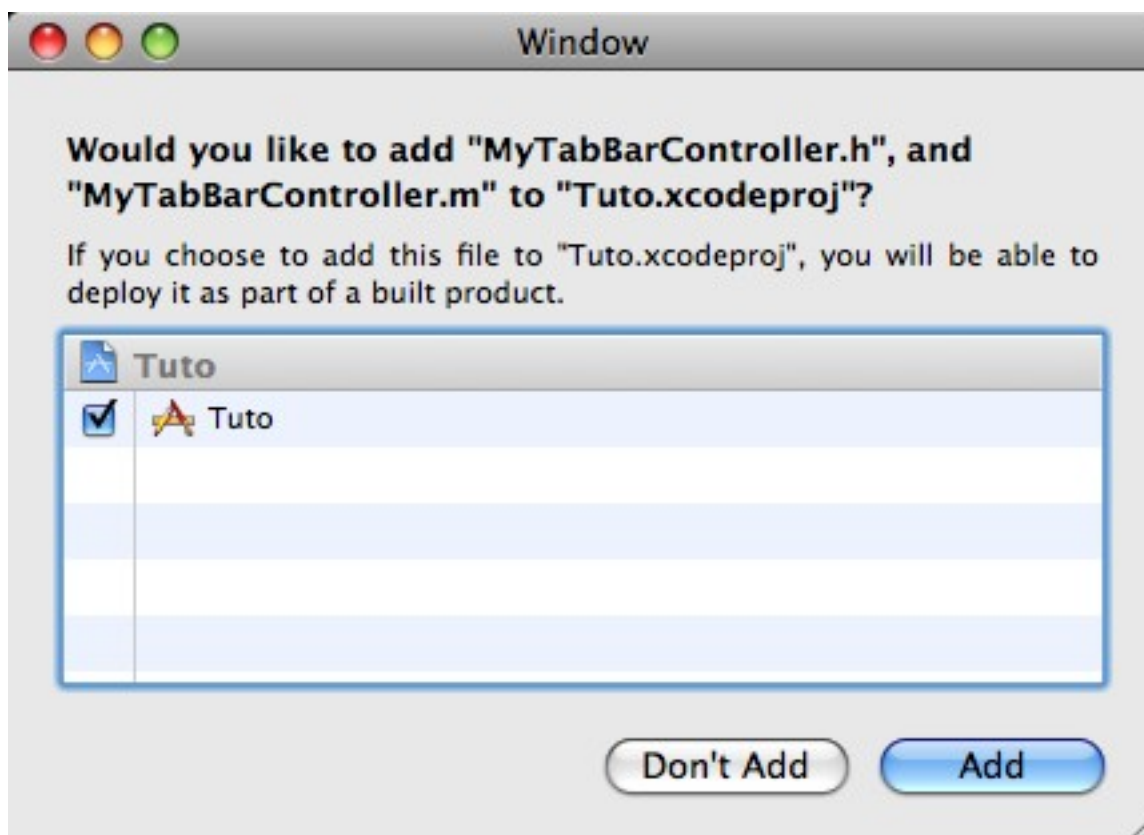
UIView pour le deuxième onglet

Reliez également la propriété « New Referencing Outlet » du label au TabBarController :



Propriété New Referencing Outlet

Retournez dans la fenêtre principale d'Interface Builder et sélectionnez l'item My Tab Bar Controller. Dans le menu File, cliquez sur « Write Class File? », entrez le nom « MyTabBarController » et cliquez sur le bouton « Enregistrer ». Sélectionnez l'item « tuto » et cliquez sur « Add » dans la fenêtre qui apparaît :



Sauvegarde

V - Retour à Xcode

Dans la fenêtre listant les fichiers du projet, vous pouvez constater que deux nouveaux fichiers ont été ajoutés : MyTabBarController.h et MyTabBarController.m. Ces fichiers représentent le code lié à notre nouveau MyTabBarController.

```
#import "UIKit/UIKit.h"
#import "Foundation/Foundation.h"

@interface MyTabBarController : /* Specify a superclass (eg: NSObject or NSView) */ {
    IBOutlet UILabel *label;
}
- (IBAction)buttonClick:(id)sender;
@end
```

```
#import "MyTabBarController.h"

@implementation MyTabBarController
- (IBAction)buttonClick:(id)sender {

}
@end
```

On va coder un peu quand même :)

1) dans le .h modifiez la classe mère pour le MyTabBarController : UITabBarController. En fait on doit donner lui spécifier sa classe.

2) dans le .m on va mettre un petit message dans le label suite à l'événement click. Regardez bien la belle notation pointée dans Objective-C :)

Voici ce que vous devez avoir une fois les modifications effectuées :

```
#import "UIKit/UIKit.h"
#import "Foundation/Foundation.h"

@interface MyTabBarController : UITabBarController {
    IBOutlet UILabel *label;
}
- (IBAction)buttonClick:(id)sender;
@end
```

```
#import "MyTabBarController.h"

@implementation MyTabBarController
- (IBAction)buttonClick:(id)sender {
    [label setText:@"C'est gagné"];
}
@end
```

Maintenant, il nous reste à relier notre AppDelegate à notre TabBarViewController. On doit donc déclarer une nouvelle propriété de type UITabBarController. Ouvrez le fichier « TutoAppDelegate.h » :

```
#import "UIKit/UIKit.h"

@interface TutoAppDelegate : NSObject <UIApplicationDelegate> {
    IBOutlet UIWindow *window;
}
```

```
@property (nonatomic, retain) UIWindow *window;

@end
```

Et modifiez-le comme suit :

```
#import "UIKit/UIKit.h"

@interface TutoAppDelegate : NSObject <UIApplicationDelegate> {
    IBOutlet UIWindow *window;
    IBOutlet UITabBarController *tabBarController;
}

@end
```

Et puis il faut également dire à notre application d'utiliser le TabBarViewController. Ouvrez le fichier « TutoAppDelegate.m » :

```
#import "TutoAppDelegate.h"

@implementation TutoAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Override point for customization after app launch
}

- (void)dealloc {
    [window release];
    [super dealloc];
}

@end
```

Et modifiez-le comme suit :

```
#import "TutoAppDelegate.h"

@implementation TutoAppDelegate

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Override point for customization after app launch
    [window addSubview:tabBarController.view];
    [window makeKeyAndVisible];
}

- (void)dealloc {
    [window release];
    [tabBarController release];
    [super dealloc];
}

@end
```

Vous remarquerez qu'il ne faut pas oublier de désalouer la mémoire de l'objet précédemment créé.

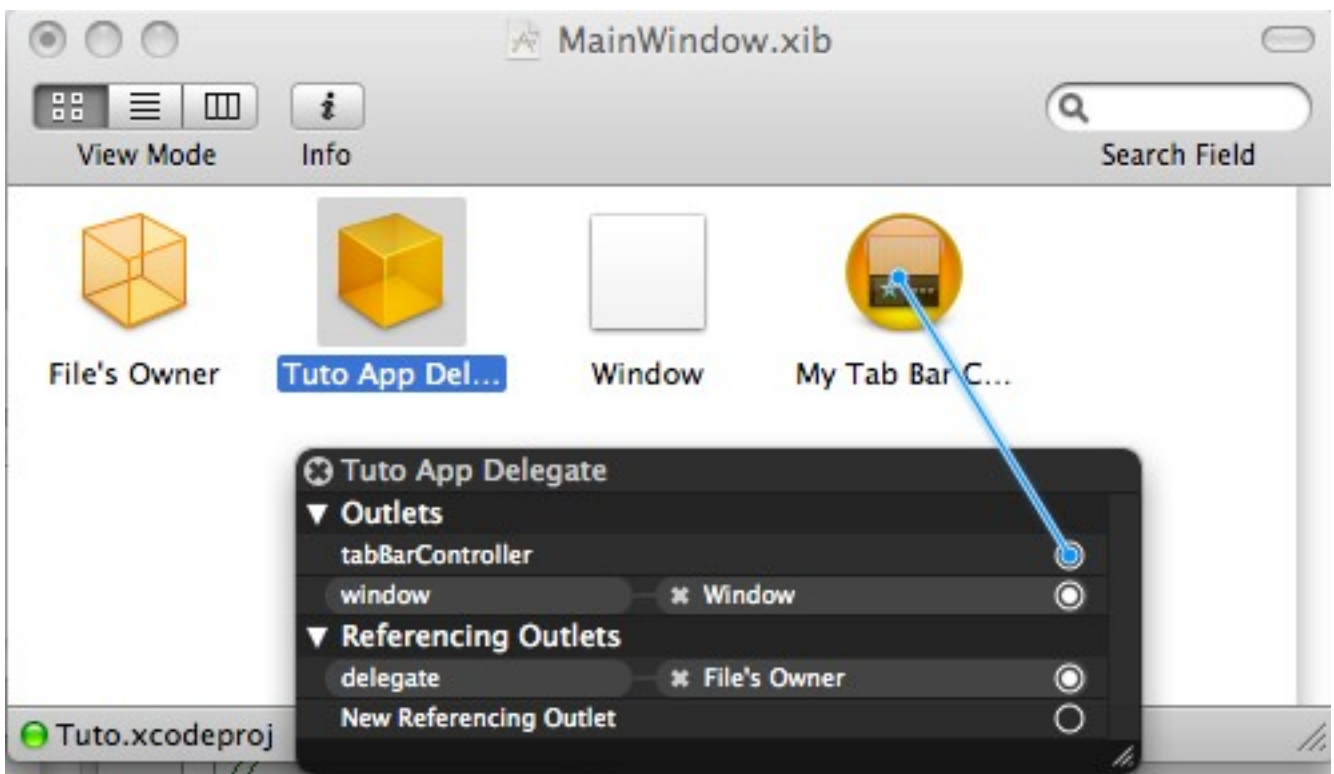
N'oubliez pas d'enregistre le tout :)

VI - Retour à Interface Builder

Maintenant, il nous reste à relier le tout :

Pour l'instant, nous avons créé d'un côté des objets visuels avec Interface Builder et d'un autre côté des nouvelles propriétés dans Xcode. Nous allons maintenant relier les deux mondes.

Cliquez sur l'item App Delegate et reliez la propriété « tabBarController » à notre « MyTabBarController » :



Liason AppDelegate TabBarController

On enregistre et ...

VII - On a terminé

... en fait, on a terminé!

On build et on lance



Build And Go

Et on doit obtenir l'application du début

Le code source de l'application: [Code Source](#)

VIII - Conclusion

Il est bien évident que ce petit tutoriel n'a aucune application pratique.

Par contre, il montre, par l'exemple, la manière de développer en mode RAD avec le SDK de l'iPhone.

Bien entendu, il faudra pratiquer encore énormément afin de réussir à réaliser de vraies applications, mais je pense que la prise en main est assez rapide.

Pour ma part, je n'ai pris en main mon MacBookPro et le SDK de l'iPhone que depuis quatre jours.

Ce que j'ai trouvé le plus difficile est la prise en main de l'Objective-C.

Je développe sur différents langages (Delphi, Java, C#, Php,...) mais je ne m'habitue pas au manque de notation pointée pour appeler les méthodes d'un objet :(

De plus, je trouve que la prise en main est assez fastidieuse par rapport aux autres langages. Mais bon, tout est une question de goût et d'habitude...

