Conception d'interfaces, le « Storyboard »
Vues et contrôleurs de vues

Brève présentation

Le « storyboard » est une fonctionnalité intégrée dans Xcode depuis la version 4.2 qui permet d'obtenir une représentation graphique des vues qui composent une application iOS ainsi que les transitions de navigation entre ces vues (enchaînements). Xcode génère tout le code nécessaire pour implémenter le comportement défini dans le storyboard. En outre, les enchaînements peuvent également être déclenchés par programmation dans des situations où le comportement ne peut pas être défini graphiquement à l'aide d'Interface Builder « **IBuilder** ».

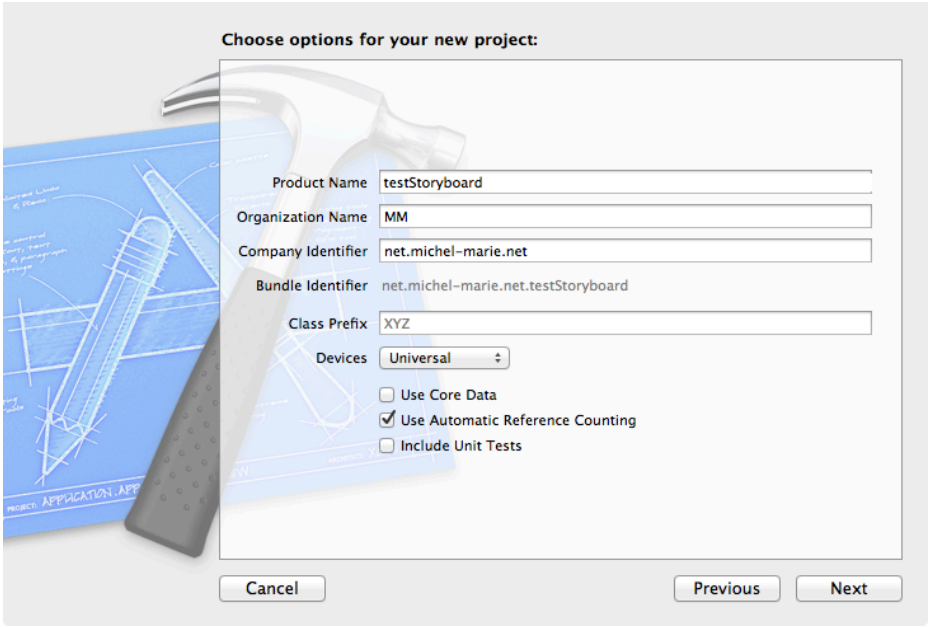
Une fois le storyboard défini, il reste à utiliser **Interface Builder** afin de concevoir l'interface utilisateur de chaque vue de la manière habituelle.

Le « design » ainsi conçu est sauvegardé dans un fichier d'extension « **.storyboard** » unique par plateforme et par localisation.

Application vide avec Storyboard

Lors de la création d'un nouveau projet avec un patron spécifique (**Single View**, **tabbed Application**, etc…) il suffit de cocher la case « **Use storyboard** ». Un storyboard avec la vue correspondante au patron et les fichiers du contrôleur de vue associés sont alors générés. Pour cet exemple j'ai choisi un projet vierge dans lequel je vais ajouter le fichier storyBoard et les fichiers contrôleurs de vue associés.

Ci-dessous la fenêtre de création d'un projet « vierge » :



Il faut ensuite lui ajouter un fichier de storyboard, commande **File->New->File->User Interface->StoryBoard->iPhone** ou **iPad**. Si l'application doit être universelle il faut ajouter un fichier pour chacun des deux types de matériel.

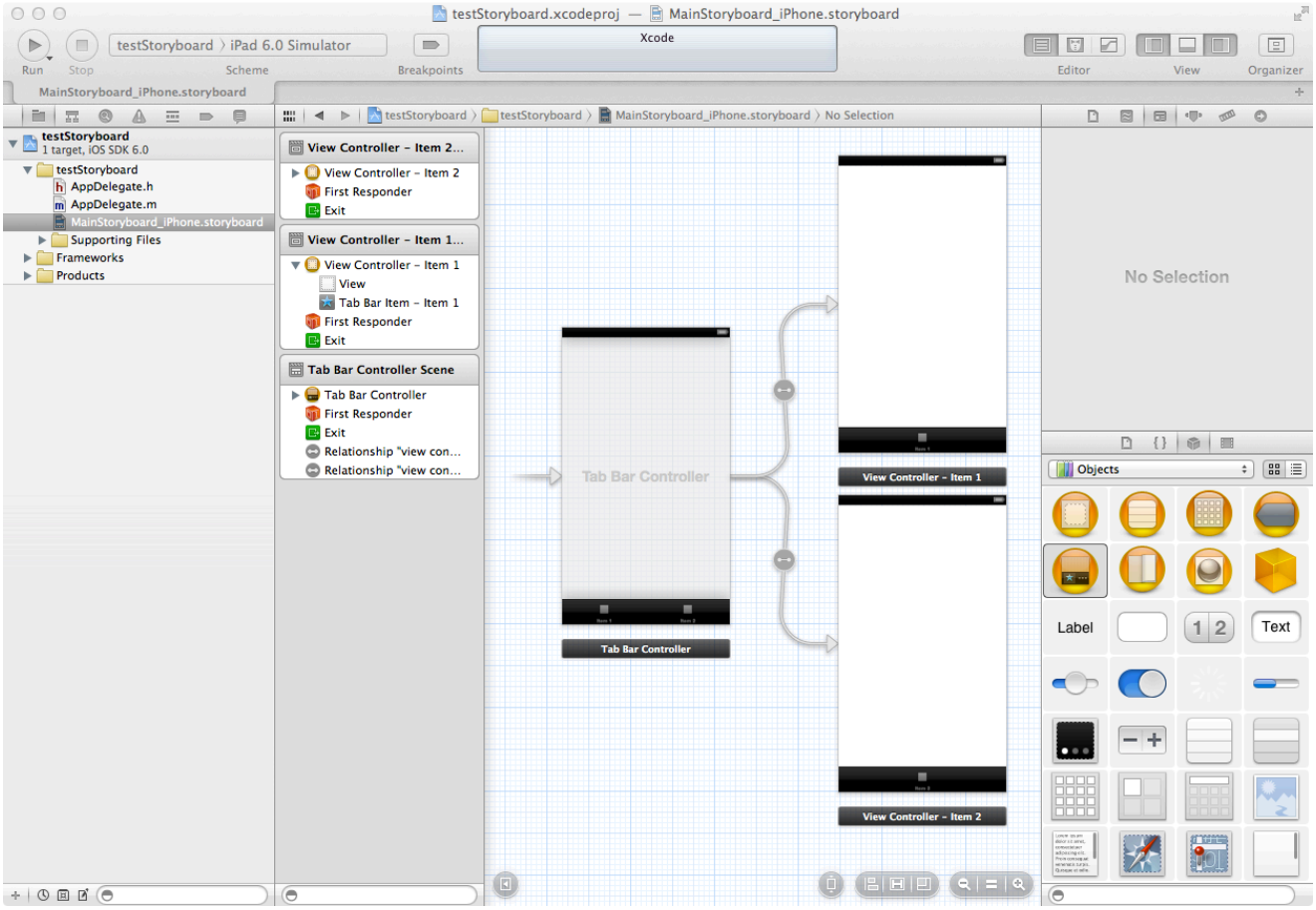
La vue suivante présente le storyboard de l'application pour la cible « **iPhone** » de localisation « **en** ».

L'éditeur storyboard fait partie de l'outil « **Interface Builder** ». Vous pouvez faire glisser de nouveaux contrôles de la bibliothèque d'objets (en bas à droite) dans votre contrôleur de vue afin de concevoir son aménagement comme précédemment dans les vues **XIB**.

La barre latérale sur la gauche (la scène) montre la structure du document. Il existe une version miniature de cette structure du document située en-dessous de la scène, nommée le « **dock** ».

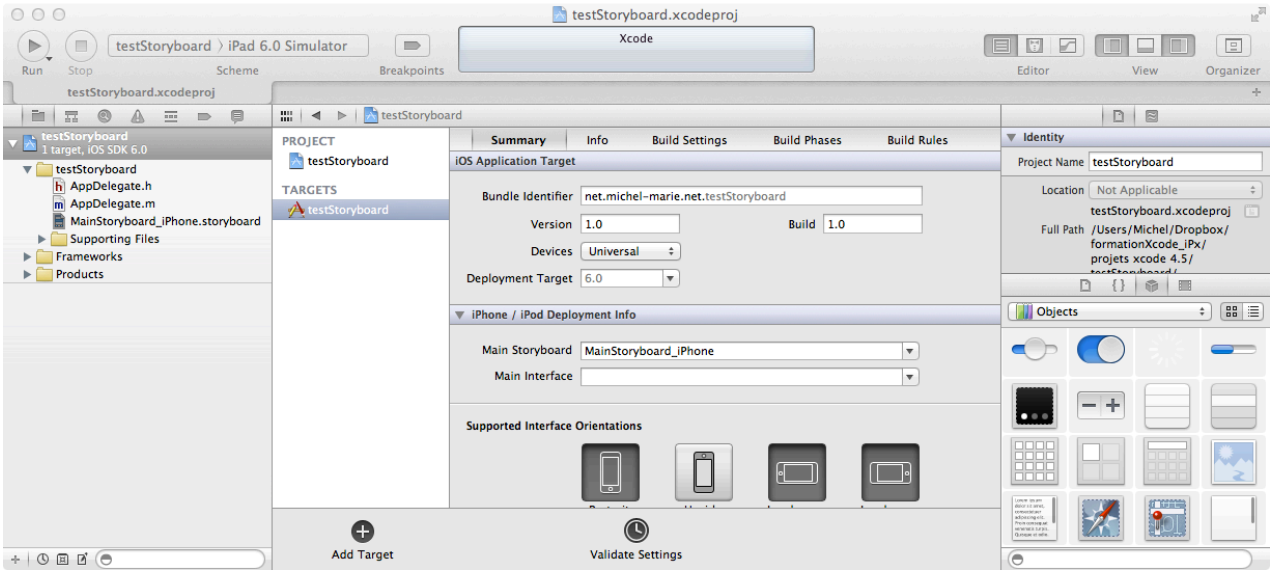
Un contrôleur « **Tab Bar Controller** » a été glissé sur la page du storyboard depuis la bibliothèque d'objets. Ce contrôleur gère un ensemble de contrôleurs de vue, chaque contrôleur étant associé à un des éléments d'une « **Tab Bar** ». Chaque contrôleur de vue fournit des informations sur son élément de la « **Tab Bar** » ou « barre d'onglets » et fournit la vue à afficher lorsque l'élément est sélectionné.

Notez que l'ajout du « **Tab Bar Controller** » ajoute 3 éléments au storyboard, un contrôleur pour la navigation entre les vues associées aux onglets (**Tab Bar Controller**) et un contrôleur pour chacune des vues associées. La flèche située à gauche de la vue signale qu'il s'agit de la vue chargée au démarrage de l'application, pour changer de vue de démarrage il suffit de déplacer la flèche sur la nouvelle vue « principale ».



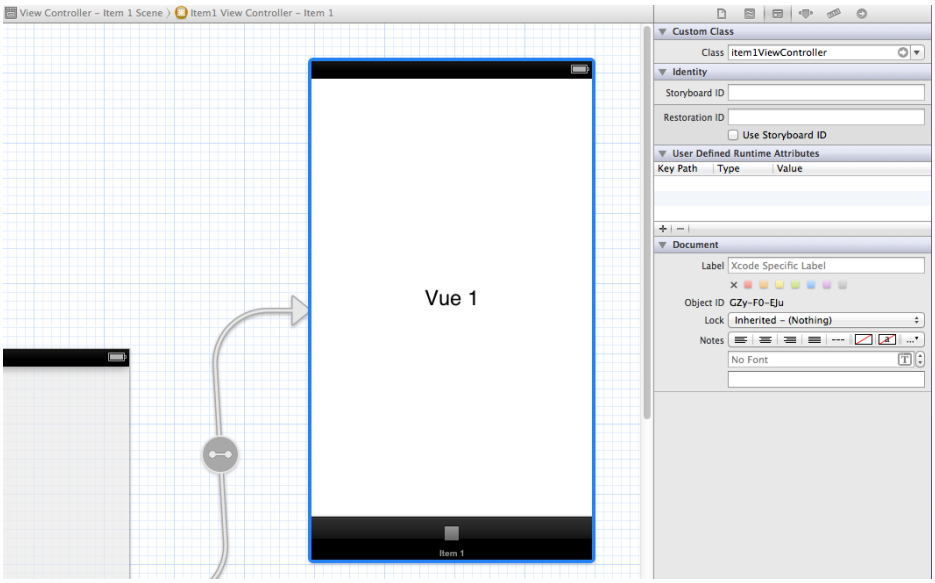
Il faut spécifier dans la fenêtre des propriétés générales du projet, onglet « Summary », le nom du fichier de storyboard à utiliser comme « **Main storyboard** », ou storyboard de démarrage de l'application.

Par défaut le projet vide charge une vue « codée » dans **AppDelegate** méthode **didFinishLauchingWithOptions**. Il faut donc supprimer ce code pour ne laisser que le return **YES** la vue étant gérée par le storyboard.



Il faut spécifier les classes contrôleur associées aux vues **IBuilder** du storyboard. Pour ceci il faut ajouter une classe du type **UIViewContrôleur** par vue et l'associer à la vue IBuilder depuis l'outil « **Identity Inspector** » rubrique **Class**.

La fenêtre ci-dessous montre l'association entre la classe item1ViewController (classe héritant de **UIViewController**) et la vue « **Item 1** ». Il faut faire de même pour la seconde vue.



Ci-dessous le résultat obtenu sur le simulateur iPhone :

Si l'on veux remplacer la scène de la vue 1 pour afficher une liste (vue table) ceci nécessite de remplacer le contrôleur de type **ViewController** par un contrôleur de type **UITableViewController**.

Par cela il faut le sélectionner puis le supprimer depuis le menu Edit et enfin depuis la bibliothèque d'objets glisser un nouveau contrôleur du type recherché « **Table View Controller** » sur le storyboard. Si l'on ne veut pas perdre le storyboard initial il est possible d'en ajouter un nouveau sur lequel on réalisera l'opération et que l'on prendra soin d'associer comme storyboard principal de l'application.

Les transitions entre vues « Segue » ou « Segue way »

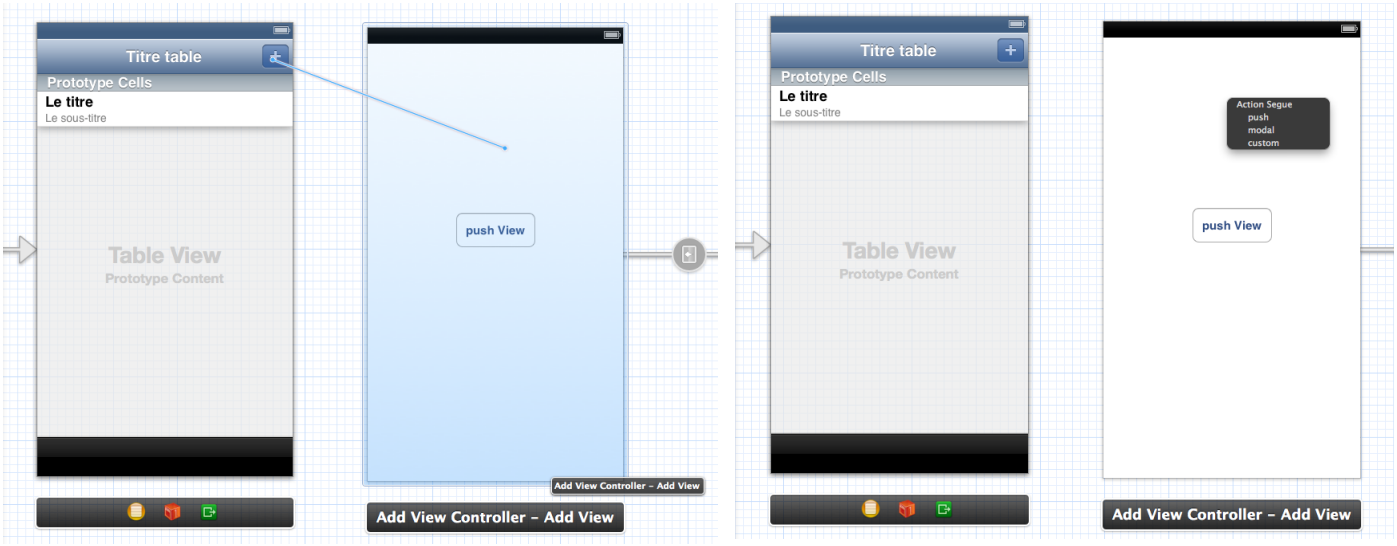
On l'aura compris, l'objectif du storyboard est de faciliter la conception des vues et du scénario de transition entre les vues en regroupant l'ensemble dans un unique fichier storyboard et de permettre une conception en mode « tout à la souris » !

Pour les vues cela ne change pas grand chose à ce qu'il était possible de faire avec **IBuilder** si ce n'est le fait que l'on passe d'un fichier .**XIB** par vue à un unique fichier .**storyboard** pour l'ensemble des vues.

Par contre pour ce qui est des transitions entre vues « **segue** » qui étaient codées, elles sont à présent, tout du moins en partie, réalisées à la souris en tirant une ligne (tout en maintenant la touche CTRL appuyée) entre l'objet déclencheur sur la vue source et la vue de destination de la transition.

L'exemple ci-dessous montre la création d'une transition entre le bouton « **+** » de la barre de navigation de la vue source et la vue de destination. Au moment où l'on relâche la souris une fenêtre de sélection demande de choisir un mode de transition « **Action segue** » parmi 3 possibles :

- ✓ transition de type « **push** » : la vue destination remplace la vue source avec gestion du retour à la vue source par un bouton ajouté à la barre de navigation. Ceci est automatiquement géré par le « **Navigation Controller** » qui gère et stocke l'ensemble des vues dont il a la charge,
- ✓ transition de type « **modal** » : la vue destination remplace la vue source sans retour possible, tout du moins géré par le storyboard,
- ✓ transition de type « **custom** » : la vue destination remplace la vue source avec moyen de personnaliser la transition, mais cette fois il va falloir écrire du code à cette fin.



Le symbole qui apparaît sur la ligne de transition est spécifique au mode de transition.

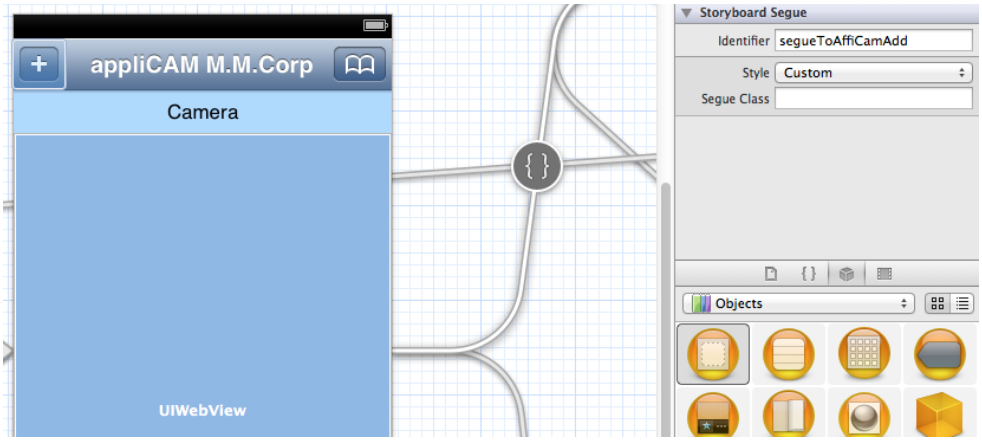
La transition « Custom »

La transition de type « **custom** » permet de notifier le contrôleur de vue source qu'une transition va être opérée. Il suffit pour cela de surcharger la méthode **prepareForSegue** dans le contrôleur de vue source.

Cette méthode reçoit un argument objet de la classe **UIStoryboardSegue** qui permet entre autres d'obtenir une référence au contrôleur de vue destination de la transition afin, par exemple, de lui passer des paramètre via des attributs prévus à cet effet.

Dans le cas où plusieurs transitions « custom » sont gérées par une même vue il faut pouvoir les distinguer, à cet effet chaque transition « segue » doit être identifiée par un « **segue identifier** » configuré via l'outil « **Attribute Inspector** ».

L'exemple ci-dessous montre une transition « **custom** » d'identificateur « **segueToAffiCamAdd** » et le code de la fonction **prepareForSegue** appelée au niveau du contrôleur source préalablement à la transition.



La méthode **prepareForSegue** récupère une référence au contrôleur destination afin d'affecter son attribut **editionMode** à **NO** avant que la transition vers cette vue ne s'opère.

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if([segue.identifier isEqualToString:@"segueToAffiCamAdd"])
    {
        addCamViewController *vueAddCamera =
                    (addCamViewController*)segue.destinationViewController;
        vueAddCamera.editionMode = NO;
    }
}
```

La classe **UIViewController**, classe de base des contrôleurs de vue dispose outre du délégué précédent :

- ✓ d'une méthode **performSegueWithIdentifier** qui permet de forcer l'activation d'une transition à partir du code,
- ✓ d'une propriété **storyboard**, objet de la classe **UIStoryboard** contenant les contrôleurs gérés par le storyboard.

La « segue Class »

On aura remarqué sur la copie d'écran précédente que la transition « **custom** » fait apparaître en plus d'un identificateur une « **segue class** » optionnelle. Il est possible dans « **Attribute Inspector** » de préciser une classe héritant de la classe de base **UIStoryboardSegue**. Cette « **segue class** » est responsable de la transition visuelle entre les vues. Elle agira dans un second temps, après la méthode **prepareForSegue**.

Elle dispose entre autres :

- ✓ d'une propriété **sourceViewController**, référence au contrôleur de la vue source,
- ✓ d'une propriété **destinationViewController**, référence au contrôleur de la vue destination,
- ✓ d'une propriété **identifier**, identificateur de transition,
- ✓ d'une méthode **perform** qu'il faut surcharger pour personnaliser la transition entre les vues.

En général, les modes de transition proposés par **IBuilder** satisfont aux besoins des applications, sinon il faut coder son mode de transition via la méthode **perform**. Le choix d'un style de transition géré par **IBuilder** se fait au niveau du « **Navigation Controller** » dans l'outil « **Attribute Inspector** ».

Exemple de classe surchargeant la méthode **perform** :

```
#import <UIKit/UIKit.h>

@interface customSegueTransition : UIStoryboardSegue
@end

@implementation customSegueTransition

-(void)perform {
    UIViewController *sourceViewController = (UIViewController*)[self
                                                sourceViewController];
    UIViewController *destinationController = (UIViewController*)[self
                                                destinationViewController];

    CATransition* transition = [CATransition animation];
    transition.duration = 5;
    transition.timingFunction = [CAMediaTimingFunction

     functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
    transition.type = kCATransitionReveal;
    transition.subtype = kCATransitionFromTop;
    [sourceViewController.navigationController.view.layer
addAnimation:transition
                                                    forKey:kCATransition];
    [sourceViewController.navigationController
pushViewController:destinationController
                                                    animated:YES];
}
@end
```

La classe **CATransition** permet de configurer le type de transition souhaité pour le basculement entre les vues source et destination.
Sa propriété **timingFunction** attend une fonction normalisée sur l'intervalle {0 , 1} en entrée et en sortie qui définit la courbe d'avancement de l'animation (mappage temps en entrée/temps en sortie).
La classe **CAMediaTimingFonction** permet de définir cette courbe à partir de profils prédéfinis tels que :

- ✓ `kCAMediaTimingFunctionEaseInEaseOut` : transition rapide au début puis lente puis rapide en fin de transition,
- ✓ `kCAMediaTimingFunctionDefault` : transition suivant une courbe de Bésier,
- ✓ `etc…`

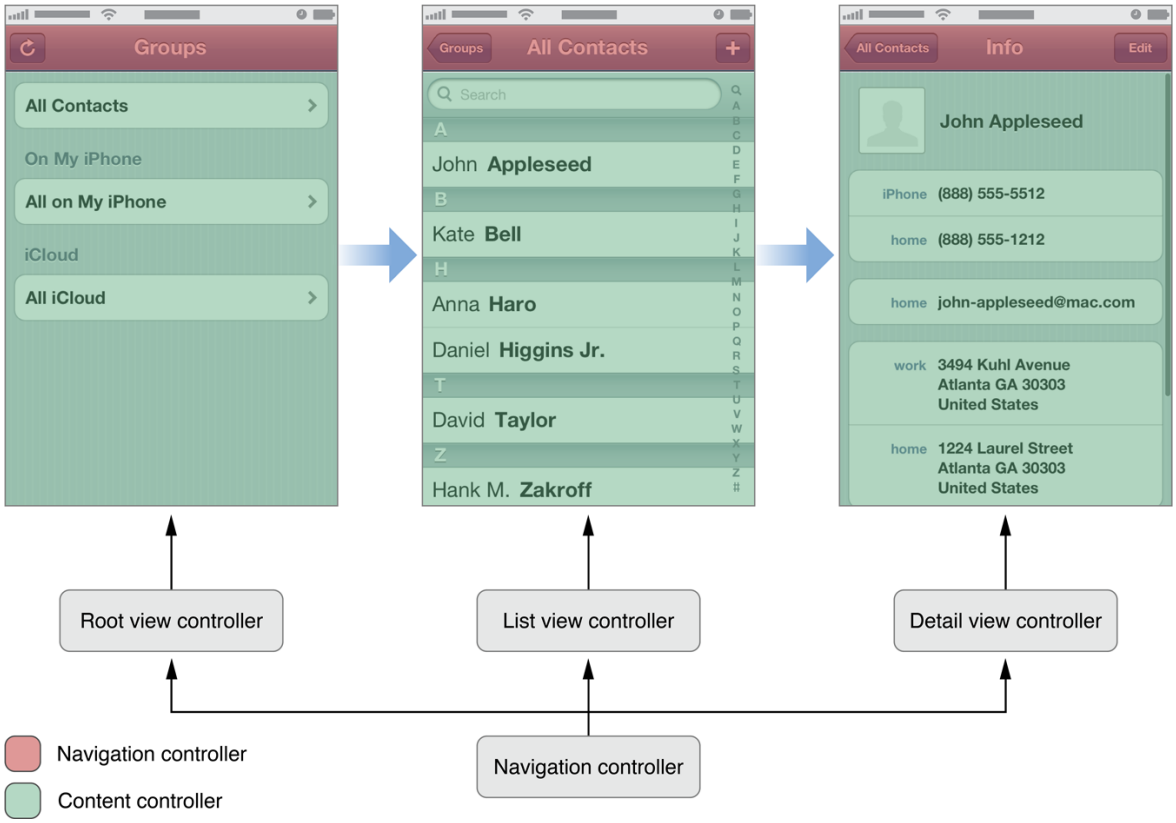
Le type puis sous-type de la transition sont ici définis comme graduelle depuis le haut.
Enfin le contrôleur de vue de destination est activé sur le contrôleur de navigation.

Le « Navigation Controller »

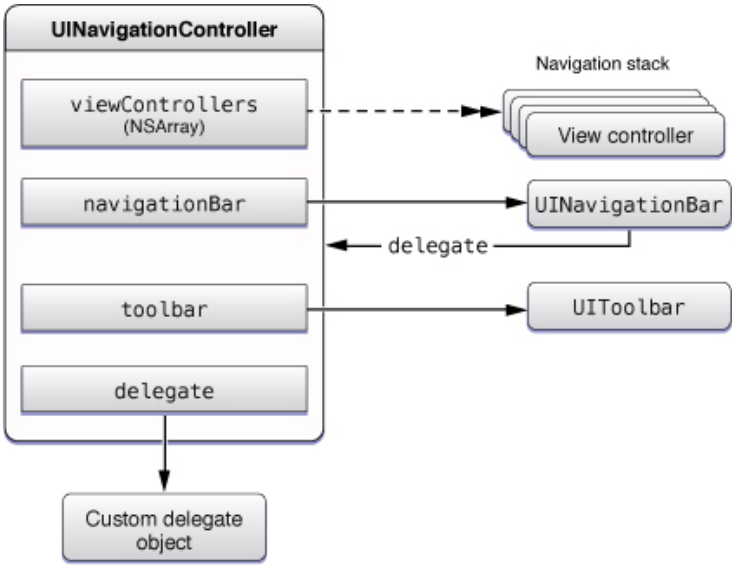
Magasin hiérarchique des contrôleurs de vues

La classe **UINavigationController** implémente un contrôleur de vue spécialisé qui gère la navigation de contenu hiérarchique. Cette interface de navigation permet de naviguer entre les vues de l'application. A chaque vue est associé un contrôleur spécialisé.

Exemple d'interface réalisant la navigation entre 3 vues :



Il est nécessaire de fournir au « Navigation Controller » les contrôleurs de vue du contenu que l'on souhaite présenter. Le contrôleur de navigation crée la vue, gère la barre de navigation et barre d'outils ajoutés à l'interface de navigation, et il est responsable de la gestion de ces vues.



La gestion de la pile de navigation :

L'opération la plus courante consiste à « pousser » les nouveaux contrôleurs de vue sur la pile en utilisant la méthode **pushViewController:**, ceci provoque l'affichage da la vue associée à de ce contrôleur de vue.
La suppression de la vue pour revenir à la vue précédente se fait à l'aide de la méthode **popViewController:**.
Bien que cette opération puisse être réalisée dans le code, le contrôleur de navigation intègre automatiquement un bouton de retour qui apparaît dans la barre de navigation en haut contrôleur de vue afin de répondre à cette commande.

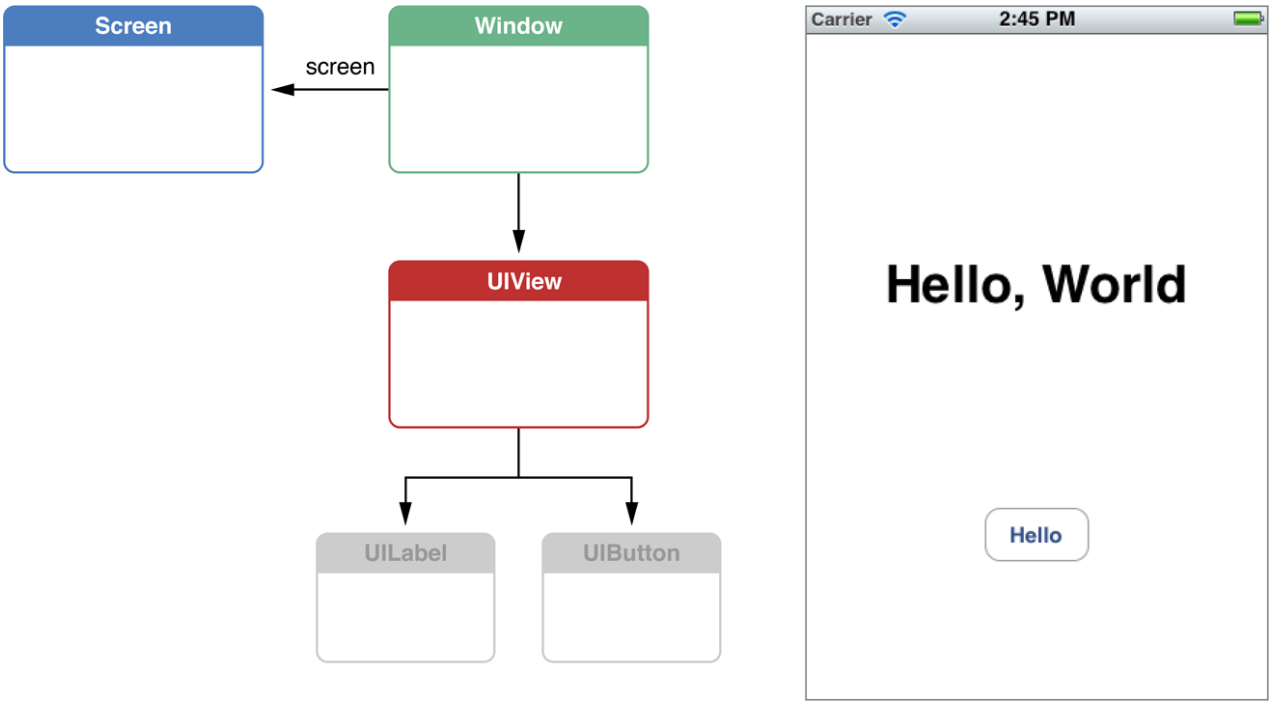
Exemple :

```
        MasterViewController*                mVC=                (MasterViewController
*)[[self.navigationController
                                                    viewControllers] objectAtIndex:0];
        [self.navigationController pushViewController:mVC animated:YES];

        …

        [self.navigationController popViewController:mVC animated:YES];
```

Un objet de commande de navigation peut être associé à un objet délégué personnalisé conforme au protocole **UINavigationControllerDelegate**. Ce protocole permet de répondre aux changements qui surviennent lors de la navigation et d'effectuer des traitements supplémentaires ou des tâches de nettoyage.

Le contrôleur de vue « UIViewController »

La classe **UIViewController** fournit le modèle de gestion de la vue. Cette classe est spécialisée pour produire le contrôleur adapté aux besoins de la vue pour cette partie de l'application.
Un contrôleur de vue gère un ensemble de vues qui composent une partie de l'interface utilisateur de votre application.
La définition d'une sous-classe de **UIViewController** impose de spécifier les différentes vues imbriquées qui seront gérés par le contrôleur (propriété **view** de la classe **UIViewController**).

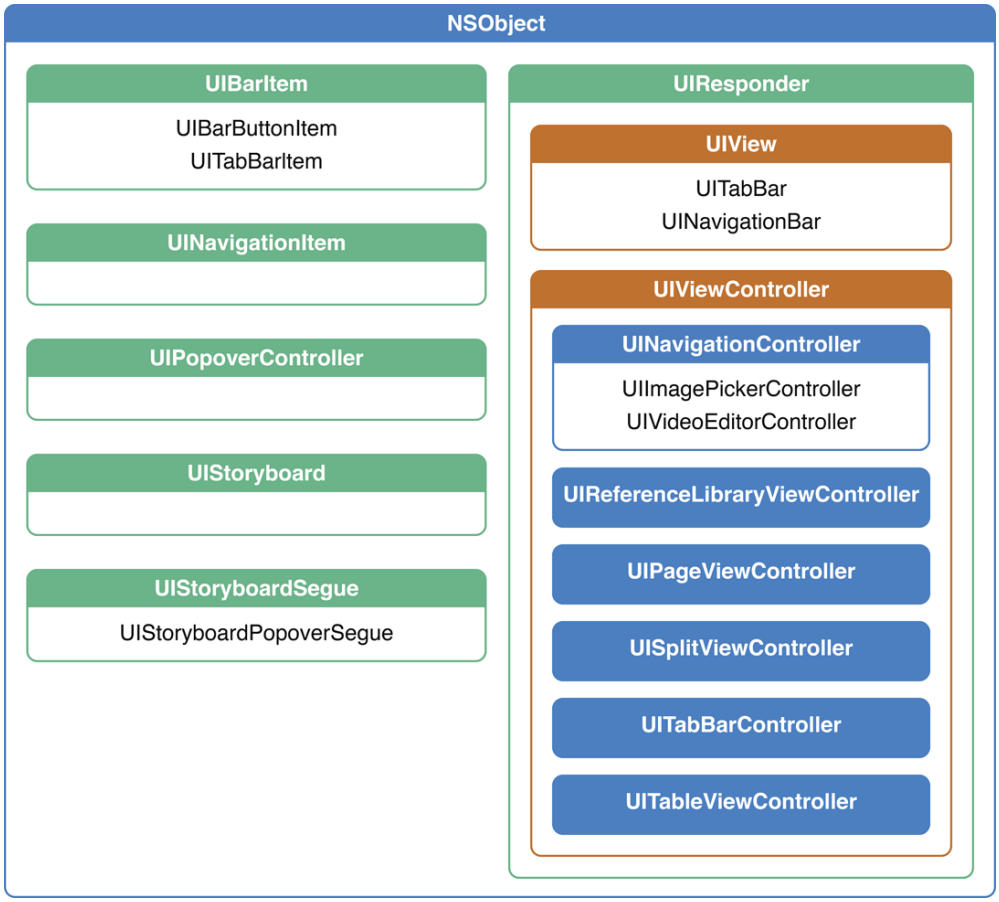


Il y a trois principaux objets à l'œuvre ici :

- ✓ Un objet **UIScreen** qui identifie l'écran physique connecté à l'appareil.
- ✓ Un objet **UIWindow** qui apporte les outils de dessin à l'écran.
- ✓ Un ensemble d'objets **UIView** pour effectuer le dessin. Ces objets sont attachés à la fenêtre et affichent leur contenu lorsque l'objet fenêtre en fait la demande.

Chaque contrôleur de vue organise et contrôle une vue, cette vue est la vue racine d'une hiérarchie de vues.

Les classes contrôleur de vue font partie du framework UIKit :



La vue dans la fenêtre

La hiérarchie des vues et la gestion des sous-vues

En plus de fournir son propre contenu, une vue agit généralement comme conteneur pour d'autres vues. Quand une vue en contient une autre, une relation parent-enfant est créée entre les deux vues.

- ✓ Les vues enfant sont accessibles par la propriété **subviews**.
- ✓ La vue parent est accessible par la propriété **superview**.

La création de ce type de relation a des implications à la fois sur l'aspect visuel et sur le comportement de l'application.

Visuellement, le contenu d'une sous-vue occulte tout ou partie du contenu de sa vue parent. La relation influe également sur les comportements de la vue :

- ✓ La modification de la taille d'une vue parent a un effet d'entraînement qui peut causer la modification de la taille et de la position de toutes les sous-vues.
- ✓ La hiérarchie des vues détermine aussi la façon dont l'application répond aux événements. Quand un événement se produit à l'intérieur d'une vue spécifique, le système transmet un

objet événement à cette vue. Si la vue ne gère pas cet événement tactile particulier, il sera transmis à sa « supervue » et ainsi de suite…

Les propriétés suivantes de la classe **UIView** permettent de manipuler la mise en forme des vues :

- ✓ **frame** : pour animer les changements de position et la taille de la vue,
- ✓ **bounds** : pour animer les changements de la taille de la vue,
- ✓ **center** : pour animer la position de la vue,
- ✓ **transform** : pour faire pivoter ou mettre à l'échelle la vue.
- ✓ **alpha** : pour modifier la transparence de la vue,
- ✓ **backgroundColor** : pour changer la couleur de fond de la vue,
- ✓ **contentStretch** : pour choisir la façon dont le contenu de la vue s'ajuste à la vue.

Le système de coordonnées de la vue

Le système de coordonnées par défaut dans UIKit a son origine dans le coin supérieur gauche comme décrit sur la figure suivante. Les valeurs de coordonnées sont représentées en utilisant des nombres à virgule flottante.