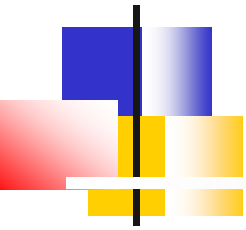
Flex 4.5

Développements pour Mobiles



Références

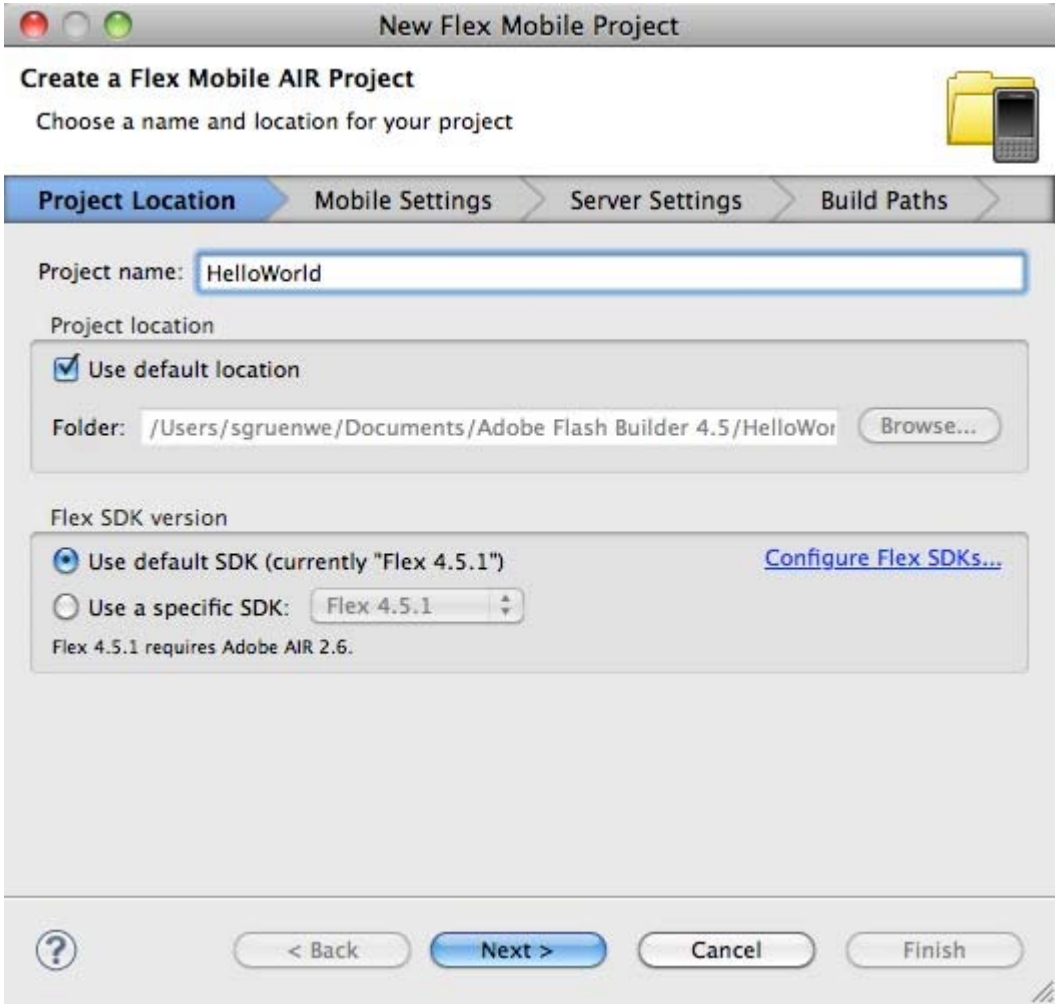
- http://www.adobe.com/devnet/flash-builder/articles/hello-world.html
- developing_mobile_apps_flex dans le dossier formation Flex
- http://help.adobe.com/fr_FR/flex/mobileapps/

Introduction

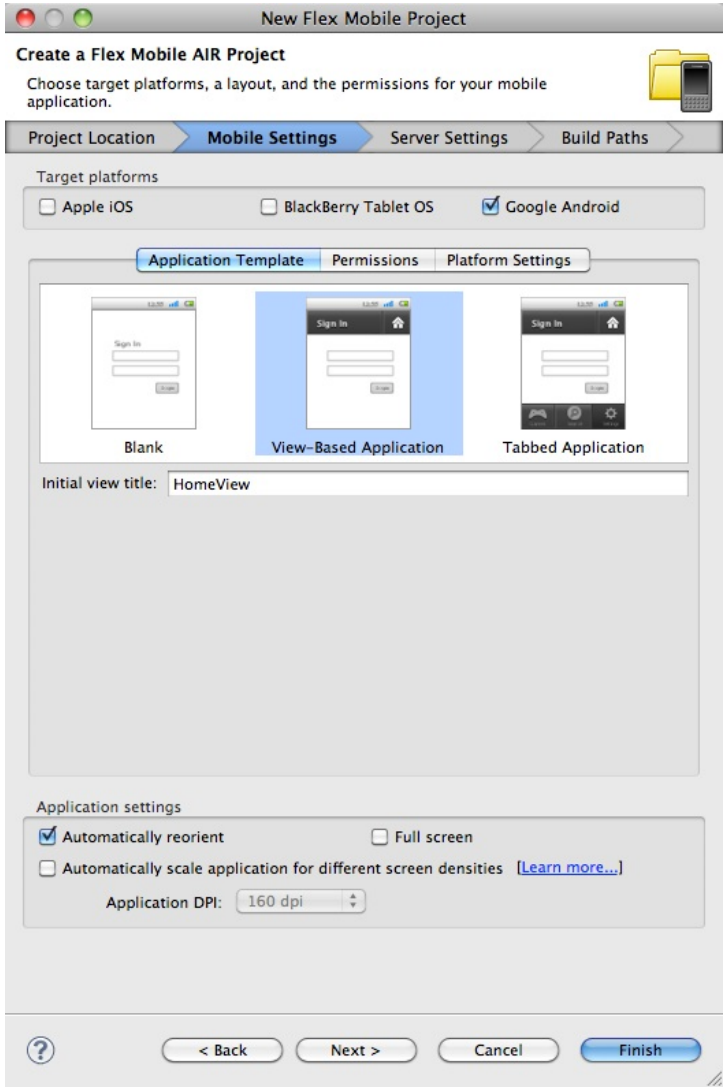
- Généralités
  - Flash Builder 4.5 permet de construire des applications mobiles pour une variété de téléphones en utilisant la même plateforme que pour le bureau ou le web
  - On peut désormais créer des applications pour une exécution sur iOS Apple, Google Android et tablettes BlackBerry
  - Le but de ce cours est de montrer comment créer un projet Mobile à simuler sur ordinateur ou à mettre directement sur téléphone

Introduction

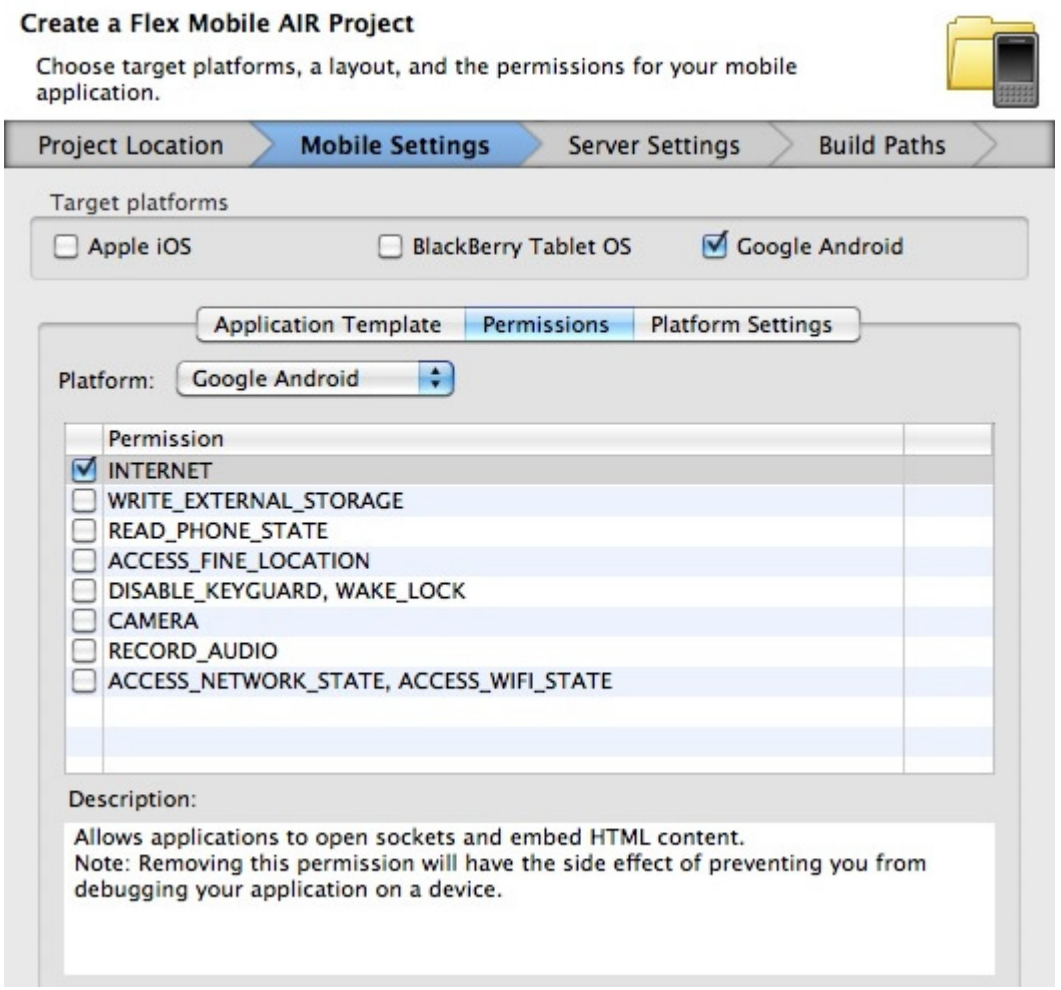
- Créer un projet mobile
  1. Sélectionnez Fichier> Nouveau> Projet Flex mobile pour ouvrir l'Assistant de projet
  2. Sur l'Assistant du projet, indiquez le nom du projet : HelloWorld
  3. Gardez les autres paramètres à leurs valeurs par défaut, et cliquez sur Suivant


New Flex Mobile Project
Create a Flex Mobile AIR Project
Choose a name and location for your project
Project Location
Mobile Settings
Server Settings
Build Paths
Project name:
HelloWorld
Project location
Use default location
Folder:
/Users/sgruenwe/Documents/Adobe Flash Builder 4.5/HelloWor
Browse...
Flex SDK version
Use default SDK (currently "Flex 4.5.1")
Configure Flex SDKs...
Use a specific SDK:
Flex 4.5.1
Flex 4.5.1 requires Adobe AIR 2.6.
< Back
Next >
Cancel
Finish

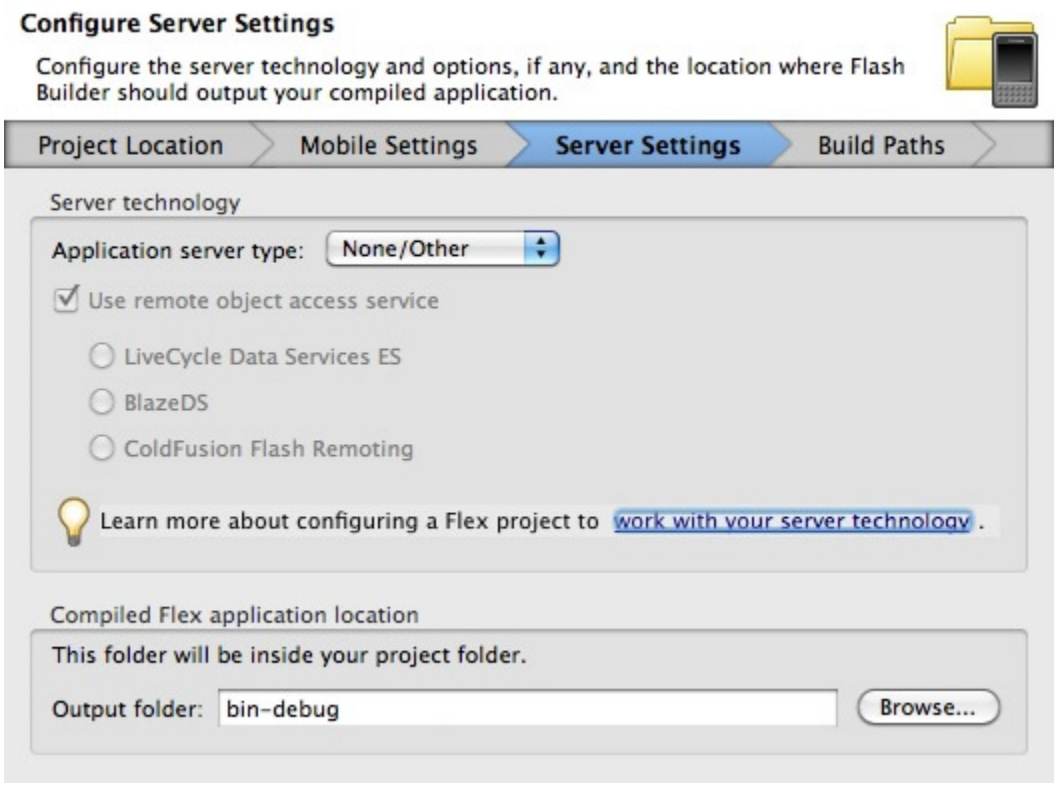
4. s'assurer que Google Android est coché comme plateforme cible (Notez que vous pouvez également cocher BlackBerry Tablet OS et iOS d'Apple)
   - Il existe trois modèles de conception *d'application : choisissez l'application basée sur la vue. Laissez les autres paramètres par défaut*



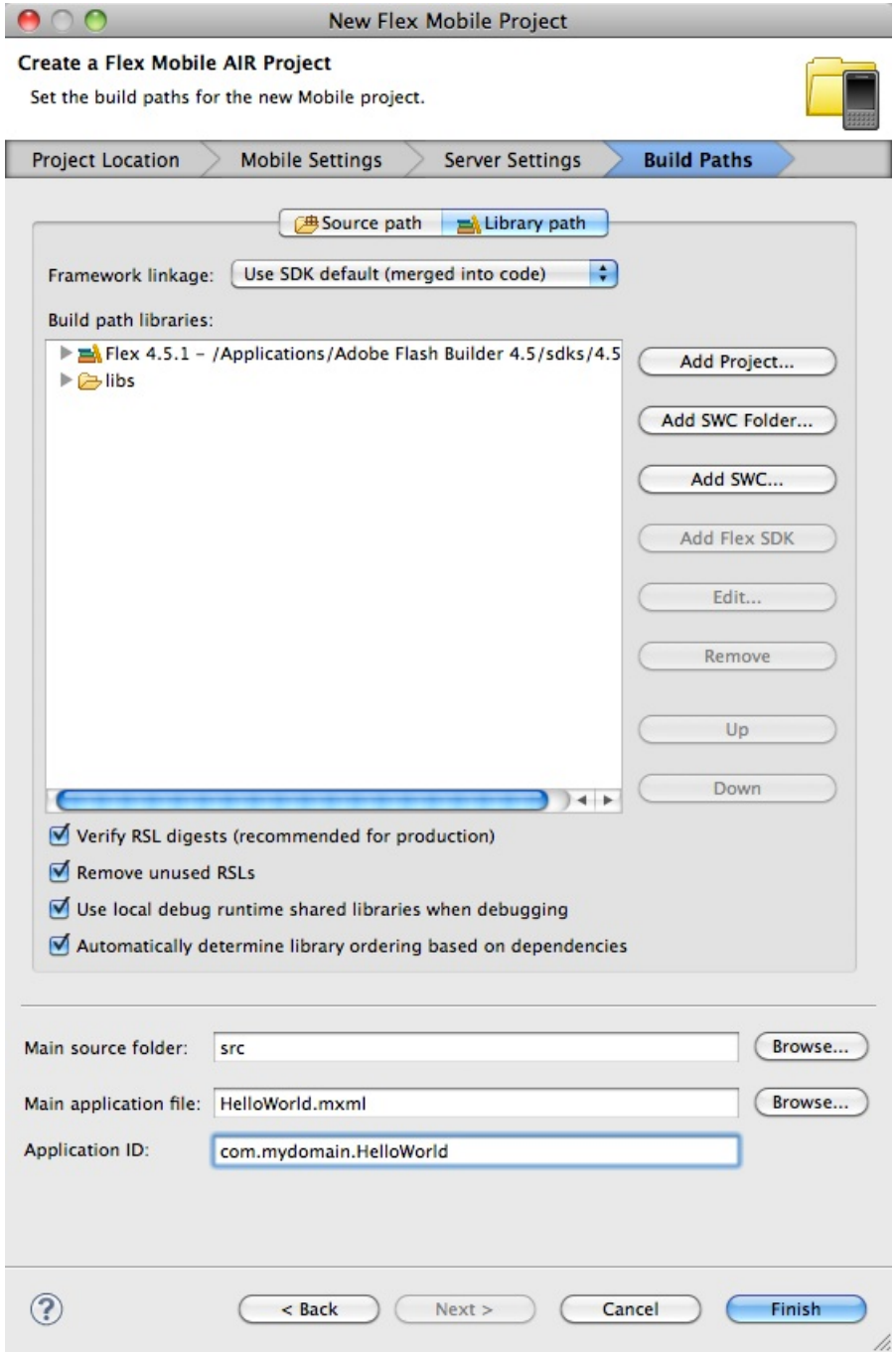
5. Cette étape est optionnelle. On peut cliquer sur « Permissions » pour voir quelles sont les permissions allouées par l'appli au device à l'exécution. Cocher sur Internet car on veut simuler le device sur Internet



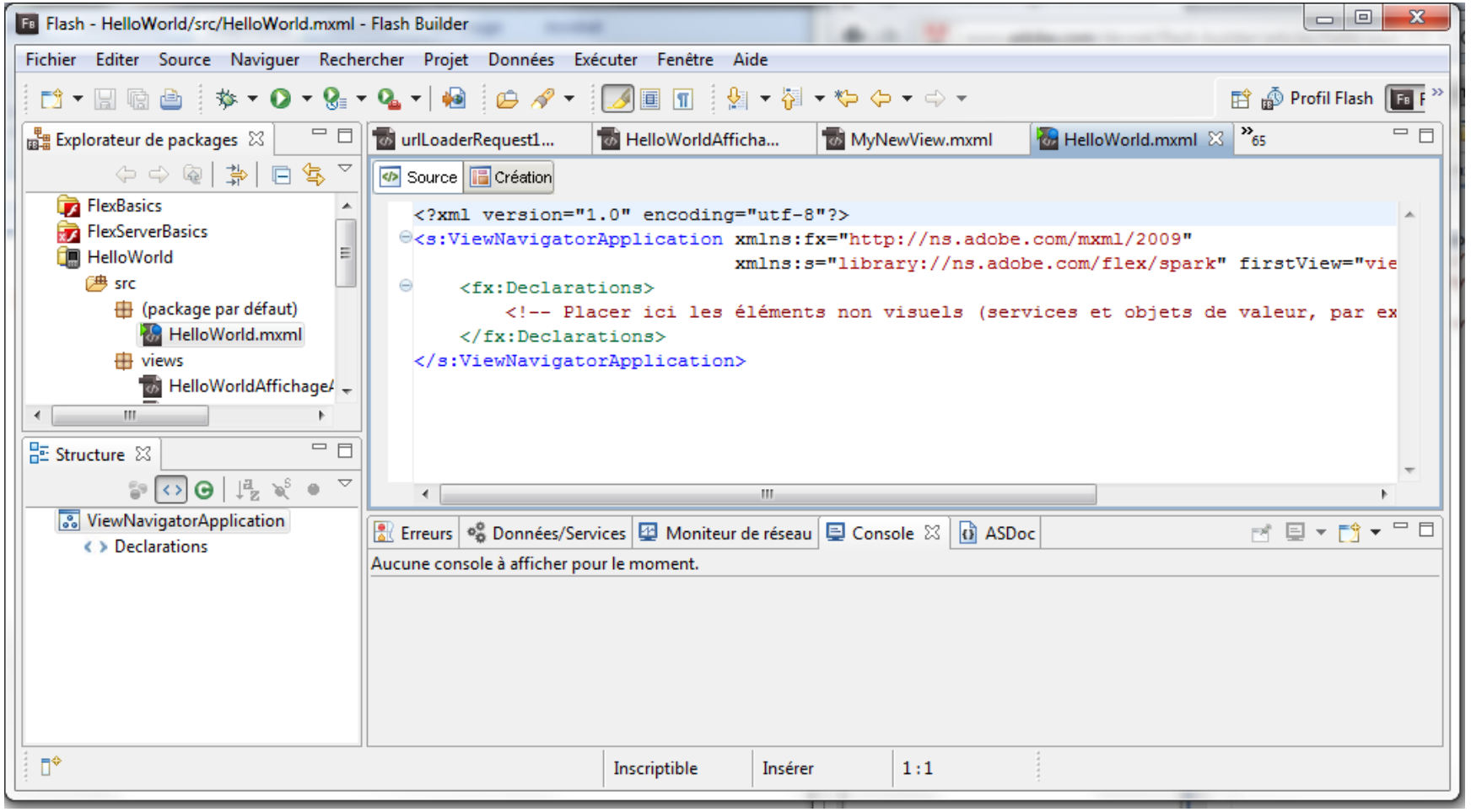
6. Cliquez sur Suivant pour passer à l'écran des paramètres du serveur de l'Assistant. Laissez les paramètres tels quels (voir Figure), puis cliquez à nouveau sur Suivant.



7. Flash Builder crée maintenant le projet et met à jour les paramètres de chemin
   - Vous pouvez ajouter des bibliothèques si nécessaire pour l’application
   - Vous pouvez aussi modifier certains noms de fichiers et de paramètres
   - Pour vos besoins ici, vous avez seulement besoin de changer l'ID d'application, qui est une chaîne unique nécessaire pour aider à identifier votre application
   - Pour l’ID, il est conseillé de mettre un nom avec un domaine : commydomain.MyApp. Mettez ici : **com.mydomain.HelloWorld**


New Flex Mobile Project
Create a Flex Mobile AIR Project
Set the build paths for the new Mobile project.
Project Location
Mobile Settings
Server Settings
Build Paths
Source path
Library path
Framework linkage:
Use SDK default (merged into code)
Build path libraries:
Flex 4.5.1 - /Applications/Adobe Flash Builder 4.5/sdks/4.5
libs
Add Project...
Add SWC Folder...
Add SWC...
Add Flex SDK
Edit...
Remove
Up
Down
Verify RSL digests (recommended for production)
Remove unused RSLs
Use local debug runtime shared libraries when debugging
Automatically determine library ordering based on dependencies
Main source folder:
src
Browse...
Main application file:
HelloWorld.mxml
Browse...
Application ID:
com.mydomain.HelloWorld
< Back
Next >
Cancel
Finish

- A ce point, Flash Builder crée un projet comme suit :



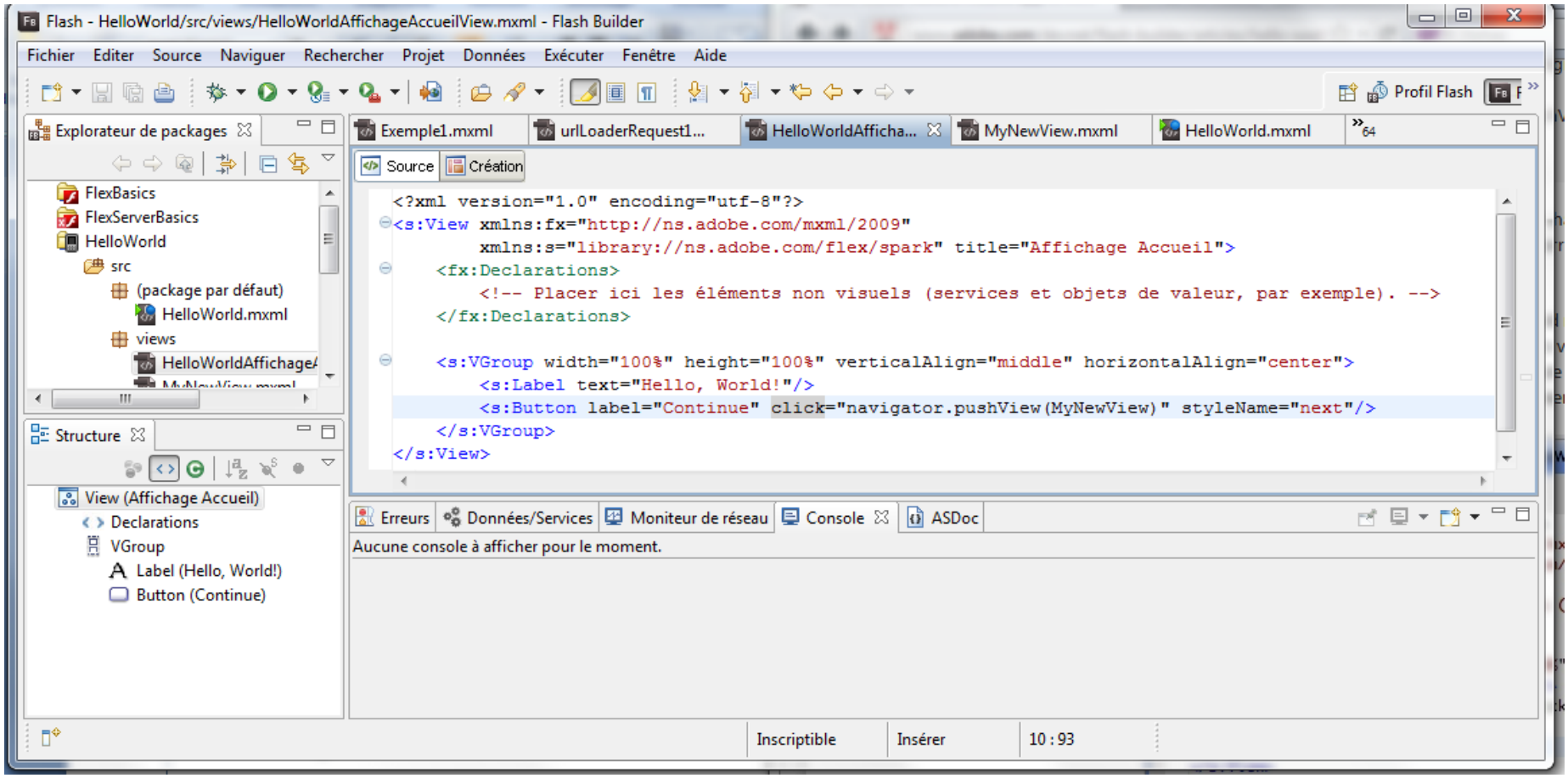
- Flash Builder génère les fichiers suivants :
  - ProjectName.mxml: ici HelloWorld.mxml. C'est le fichier d'application pour le projet. Généralement, on n'a pas à lui ajouter un contenu
  - ProjectNameHomeView.mxml: ici HelloWorldHomeView.mxml.
  - Ce fichier représente l'écran initial (ou vue) du projet
  - Flash Builder place le fichier dans un package appelé «vues»

- Jusqu’à maintenant, l’appli est juste un conteneur vide, à part quelques infos sur la configuration par défaut
- Nous allons ajouter du texte sur l'écran du mobile et un bouton en dessous qui fait quelque chose lorsqu'il est pressé
- Cliquez sur HelloWorldHomeView.mxml pour le rendre visible à l’éditeur et lui ajouter ce code avant </s:View>

```
<s:VGroup width="100%" height="100%"
  verticalAlign="middle" horizontalAlign="center">
  <s:Label text="Hello, World!"/> <s:Button
  label="Continue"
  click="navigator.pushView(MyNewView)"
  styleName="next"/>
</s:VGroup>
```

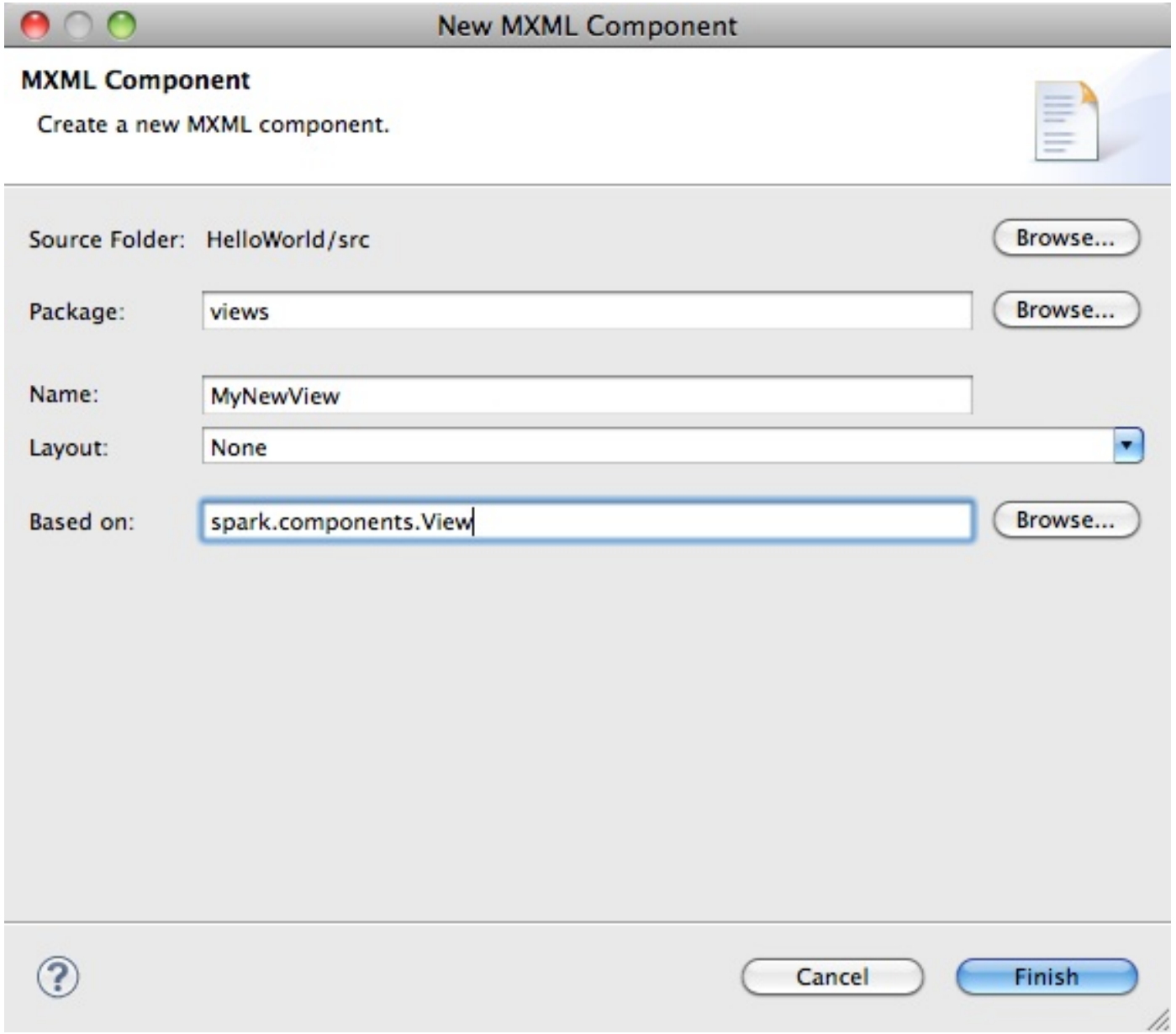
- Sauvegarder le fichier

- Le code ressemble à ceci



- Créer MyNewView que ce script est censé lancer

1. Dans le menu principal, sélectionnez Fichier> Nouveau> MXML Component
2. Assurez-vous que "HelloWorld / src" apparaît comme le dossier source. Si elle n'est pas, cliquez sur le bouton Parcourir pour sélectionner le projet HelloWorld
3. Pour le package, tapez « views » si ce n'est pas déjà là
4. Pour le Nom, tapez le nom de l'écran (voir) qui est référencé dans le script : MyNewView
5. Gardez le layout à vide. Assurez-vous que le composant est "basé sur" spark.components.View (voir figure)
6. Cliquer sur « Terminer »


New MXML Component
MXML Component
Create a new MXML component.
Source Folder: HelloWorld/src
Browse...
Package: views
Browse...
Name: MyNewView
Layout: None
Based on: spark.components.View
Browse...
Cancel
Finish

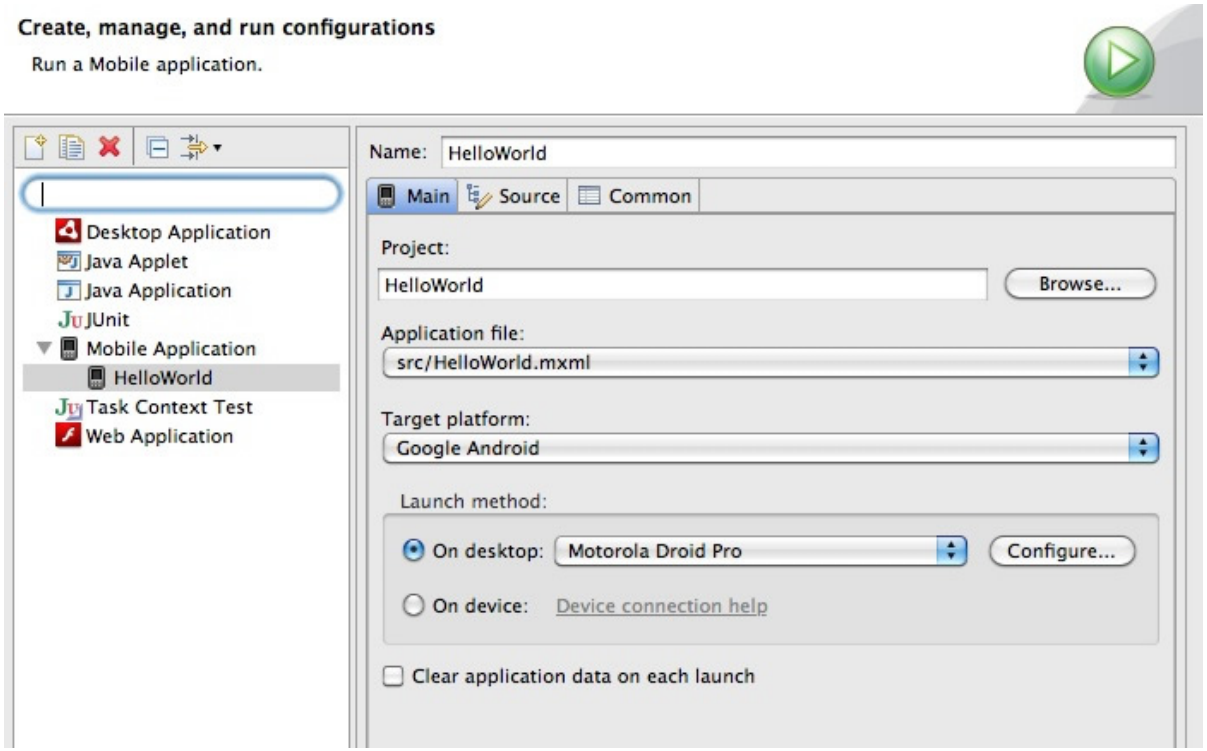
- Cliquez sur le composant MyNewView pour le rendre visible
- Ajoutez un peu de texte et un bouton de ce point de vue
- Tapez le code suivant :

```
<s:VGroup width="100%" height="100%"
   verticalAlign="middle" horizontalAlign="center">
   <s:Label text="Success!" />
   <s:Button label="Back" click="navigator.popView()"
   styleName="back" />
</s:VGroup>
```

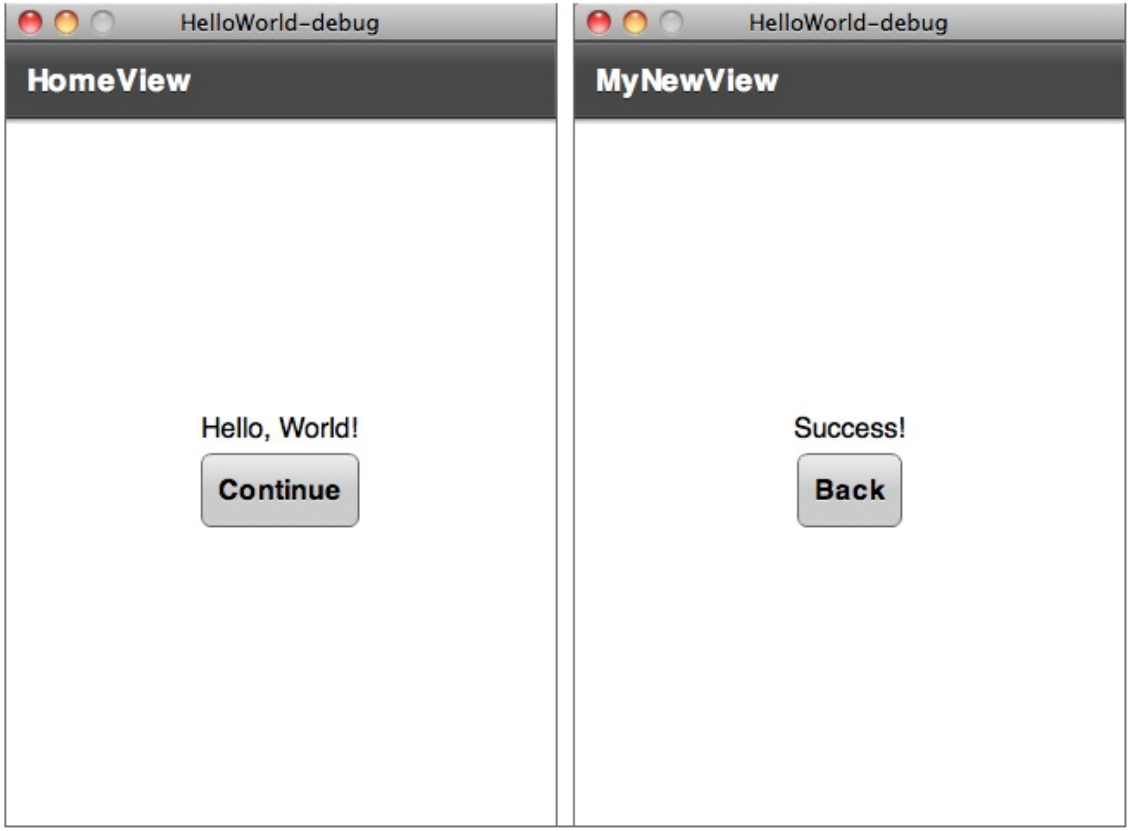
Testez l’appli sur le bureau

- Vous n'avez pas besoin d'un appareil mobile pour tester votre application
- Flash Builder permet d’exécuter et déboguer des applications mobiles sur le bureau
- Assurez-vous que le fichier principal MXML pour le projet (son nom devrait être HelloWorld.mxml) soit ouvert et actif dans l'éditeur

- Assurez-vous que la plateforme cible soit assignée à Google Android
- Pour la méthode de lancement, sélectionnez Bureau, puis choisissez un téléphone pour simuler, par exemple, Pro Motorola Droid
- Vos paramètres doivent ressembler à ceux présentées dans la figure



- Cliquez sur Exécuter pour accepter les paramètres et lancer l'application sur le bureau
- Un lanceur de programme appelé ADL fonctionne maintenant, en montrant une fenêtre (appelée HelloWorld-debug) qui affiche "Bonjour le monde!" sur l'écran et un HomeView bouton Continuer en dessous



- Cliquez sur le bouton Continuer pour passer à l'écran suivant, MyNewView, qui affiche "Success!" sur l'écran et un bouton Retour en dessous
- En cliquant sur le bouton Continuer est l'équivalent d'appuyer sur le bouton avec le doigt sur le périphérique réel
- Cliquer soit sur le bouton Précédent ou appuyez sur Ctrl-B (Commande-B) pour revenir à l'écran précédent

Test sur Android

- Fonctionnement
  - Testez votre application sur un périphérique simulé est un bon moyen pour voir à quoi il ressemble, mais l'exécution et le débogage app sur un périphérique réel, c'est encore mieux
  - Étapes à suivre :
    1. Assurez-vous que le mode de débogage USB soit activé sur votre appareil
       Sur Android 2.2, vous faites cela en lançant l'application des paramètres du système, en tapant Applications> Développement, puis assurez-vous que Débogage USB soit cochée. (Appuyez sur OK pour permettre un débogage USB)

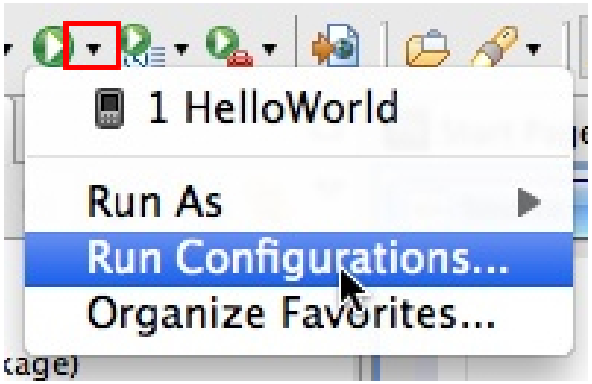
Test sur Android

2. Connectez votre téléphone à l'ordinateur via un câble USB. Votre appareil peut vous demander de choisir un type de connexion. Choisissez de le monter comme un disque dur

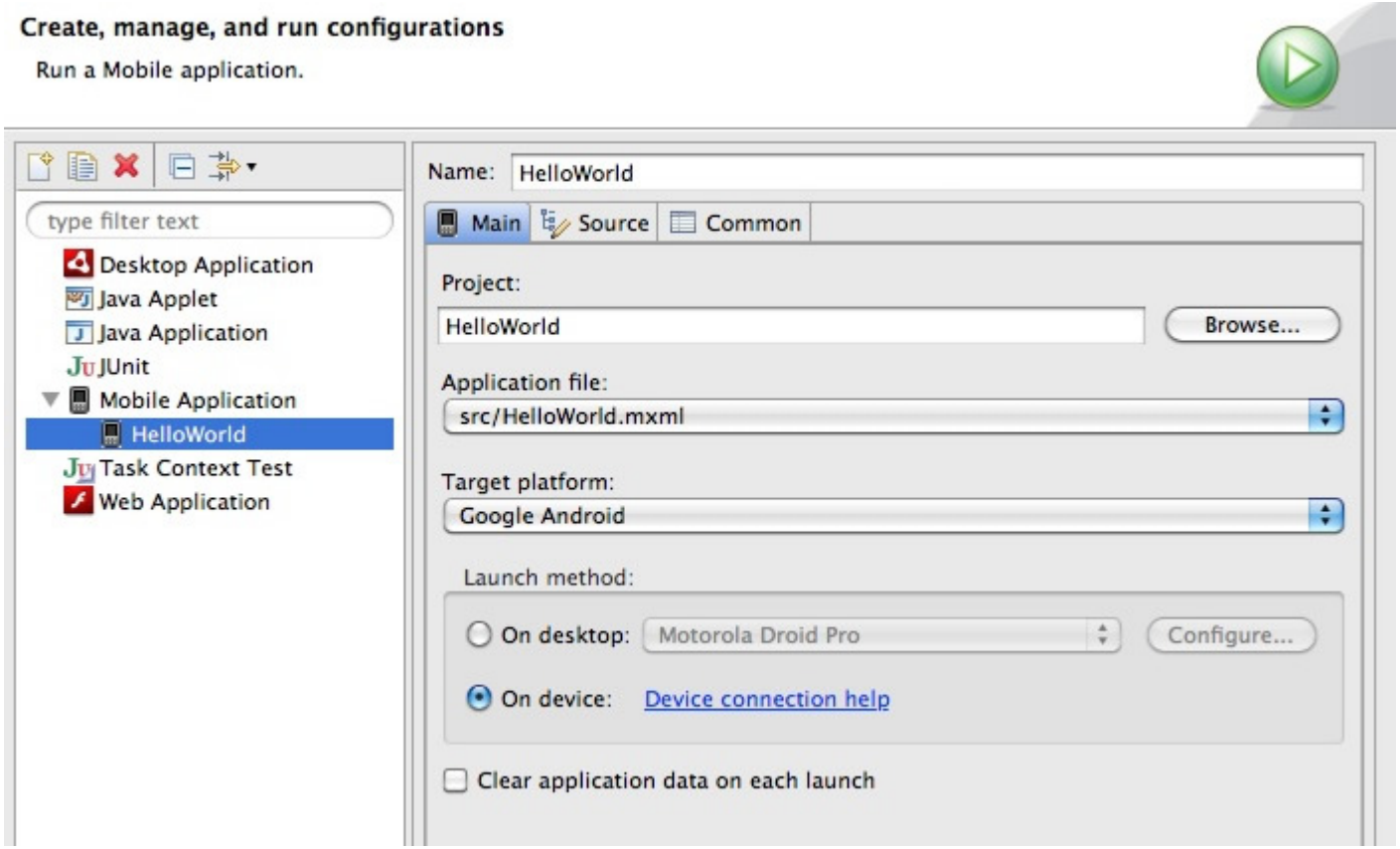
   Si vous voyez une boîte de dialogue qui vous demande de choisir votre appareil, et ne pas voir le vôtre dans la liste, cliquez sur le lien "je ne vois pas mon périphérique connecté à la liste". Cela vous mènera à la page Connecter Google Android dans la documentation en ligne

Test sur Android

3. Comme vous avez exécuté une configuration de test de HelloWorld sur le bureau de la section précédente, cliquez cette fois sur la flèche noire vers le bas à côté du bouton Exécuter dans la barre d'outils en haut de l'espace de travail Flash Builder et sélectionnez configurations Exécuter (voir figure) pour afficher la boîte de dialogue

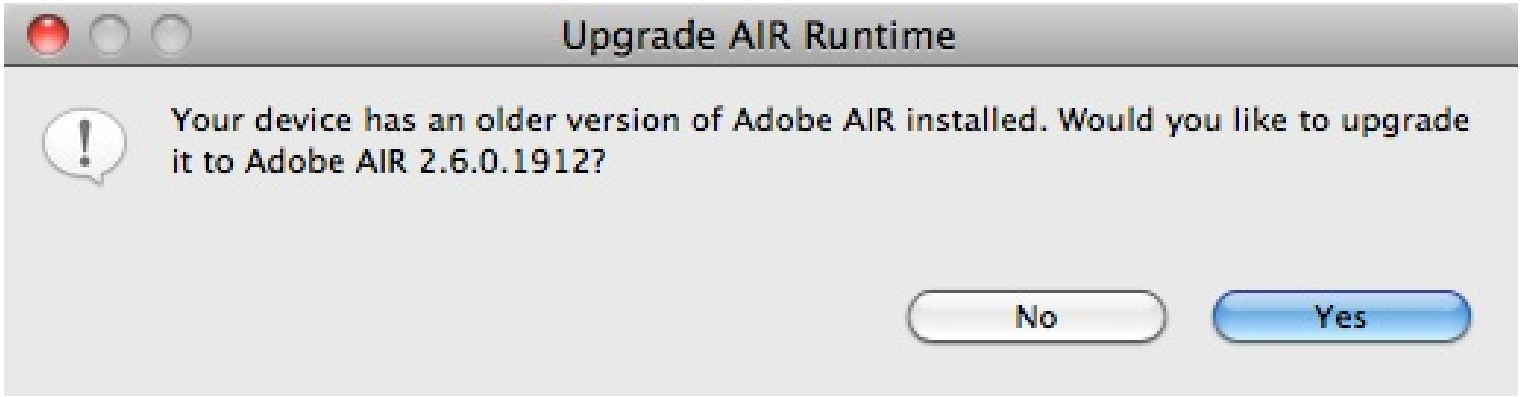


4. Dans la boîte de dialogue Run Configurations, assurez vous que le champ Project affiche "HelloWorld" (sinon, cliquer sur Browse pour le sélectionner), le fichier application est mis à "src/HelloWorld.mxml", Google Android est la plateforme cible, et sur On Device la méthode lancée sélectionnée



- Sélectionner la plateforme cible approriée pour votre téléphone , si ce n’est pas un Android

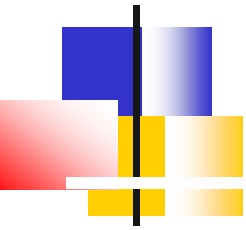
5. Cliquer Run pour permettre à Flash Builder d’envoyer l’appli au télpéhone. Votre téléphone doit répondre et votre appli doit se lancer automatiquement
   Note: si votre ordinateur vous demande d’upgrader la dernière version d’AIR, dites Ok.



Empaquetage de l’application pour sa publication

- Une fois que vous êtes prêt à publier la dernière version de votre application, par exemple sur un Android du marché, vous pouvez l'exporter à partir de Flash Builder

1. Avec votre projet ouvert dans Flash Builder, sélectionnez Projet Release Exporter> Build
2. Dans la boîte de dialogue, assurez-vous que votre projet HelloWorld apparaît dans le sélecteur de projet. HelloWorld.mxml devrait apparaître dans le sélecteur d'applications
3. Connectez votre appareil à l’ordinateur si ce n'est déjà fait … Continuer



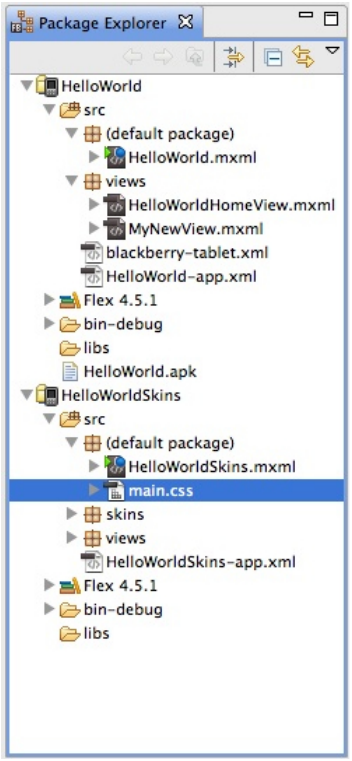
Flex 4.5

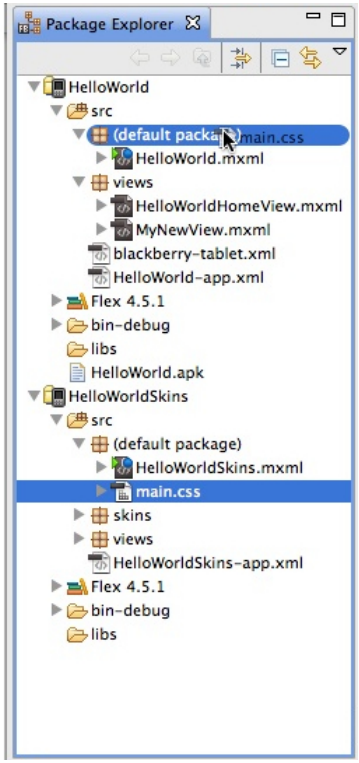
Retour sur les applis bureau

Réhabillage des boutons

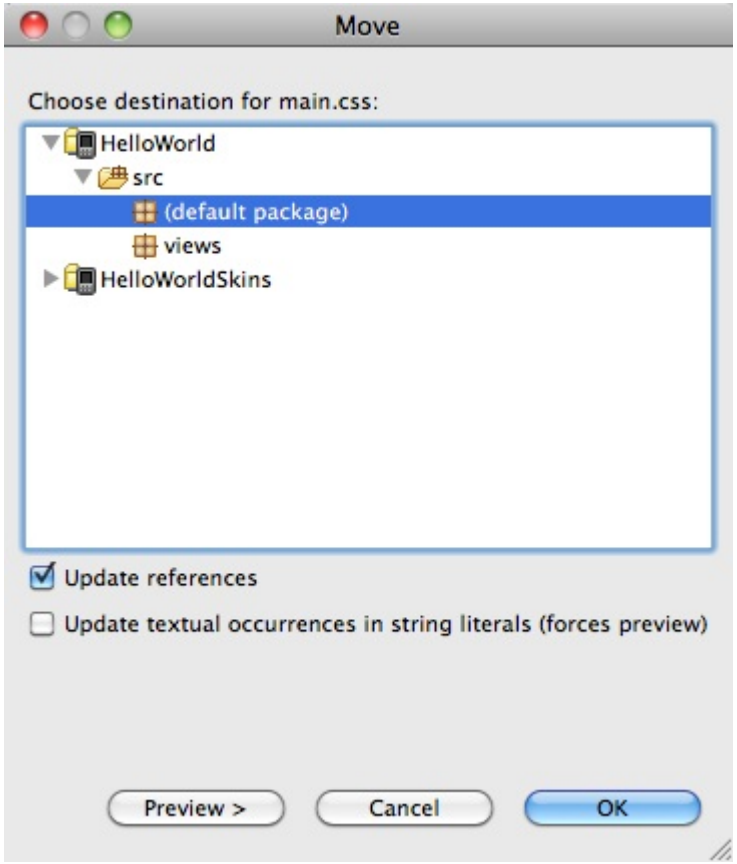
- Utilisation d'une feuille de style
  - Flash Builder permet de changer l'apparence ("peau") d'éléments visuels dans votre projet
  - Étapes
    - Téléchargez le fichier exemple de projet (HelloWorld-source.zip) du haut de l'article et extraire son contenu dans un dossier de votre choix (par exemple, sur le bureau)
    - Sélectionnez Fichier> Importer> Flash Builder > Flash Builder Project
    - Importez le fichier : HelloWorldSkins.fxp et cliquez : Open

- Cliquez sur Terminer
- Noter l'ajout de HelloWorldSkins project au Package Explorer
- Ouvrir HelloWorldSkins pour voir le fichier main.css dans HelloWorldSkins/src/(default package)
- Déposez main.css du projet HelloWorldSkins et le glisser dans le répertoire HelloWorld/src/(default package)

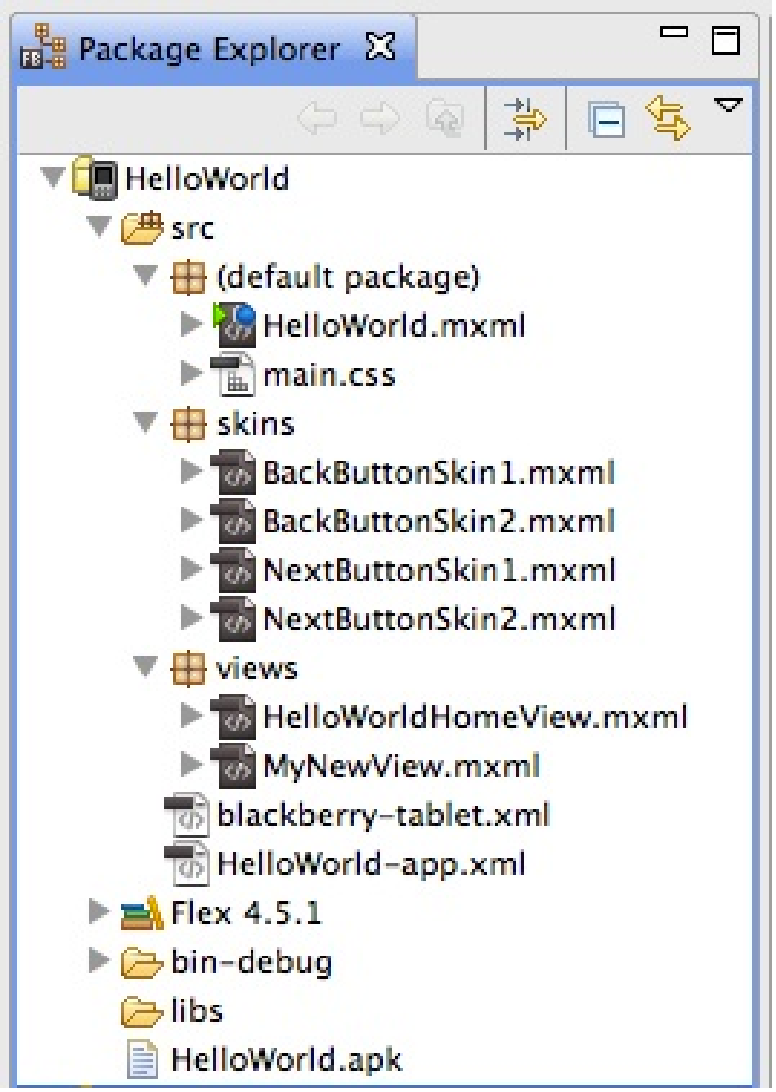




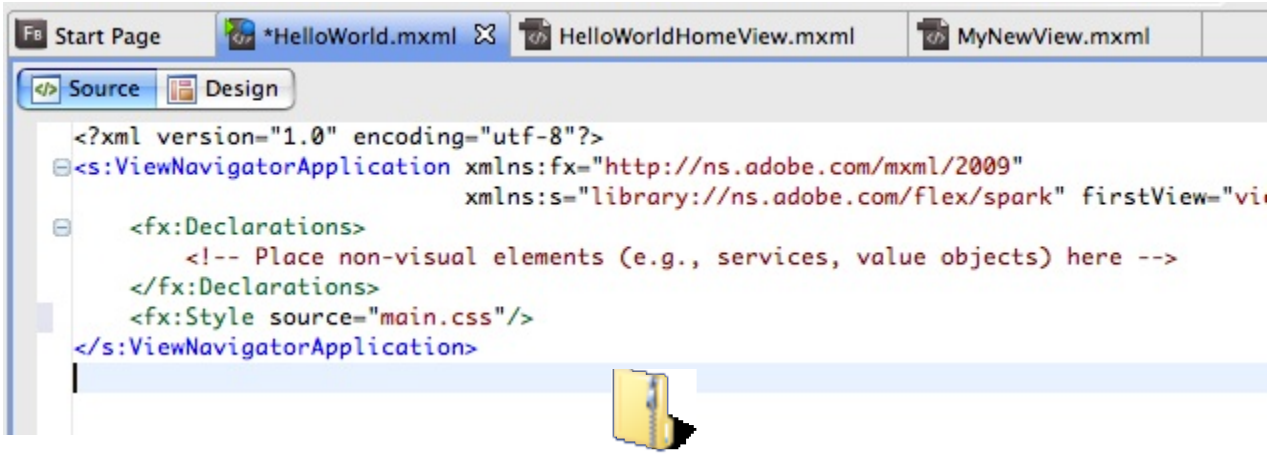
- Dans la boîte de dialogue Déplacer, confirmez que vous voulez HelloWorld / src / (package par défaut) comme dossier de destination pour le déplacement
- Si une autre destination est en surbrillance, cliquez simplement sur "(package par défaut)" pour le sélectionner
- Cliquez sur OK



- Selectionner File > New > Package
- Vérifiez que "HelloWorld/src" apparait comme chemin et entrer **skins** comme nom de package
- Cliquer Finir
- Ouvrir HelloWorldSkins/src/skins pour découvrir 4 fichiers MXML
  - BackButtonSkin1.mxml
  - BackButtonSkin2.mxml
  - NextButtonSkin1.mxml
  - NextButtonSkin2.mxml
- Déposez les 4 fichiers dans le répertoire HelloWorld/src/skins que vous venez juste de créer
- Assurez vous que lors du transfert que la boite de dialogue demande de confirmer la destination des 4 fichiers
- Assurez vous que "skins" soit sélectionné et cliquer OK pour réaliser le transfert
- La structure asset de HelloWorld doit ressembler à ceci

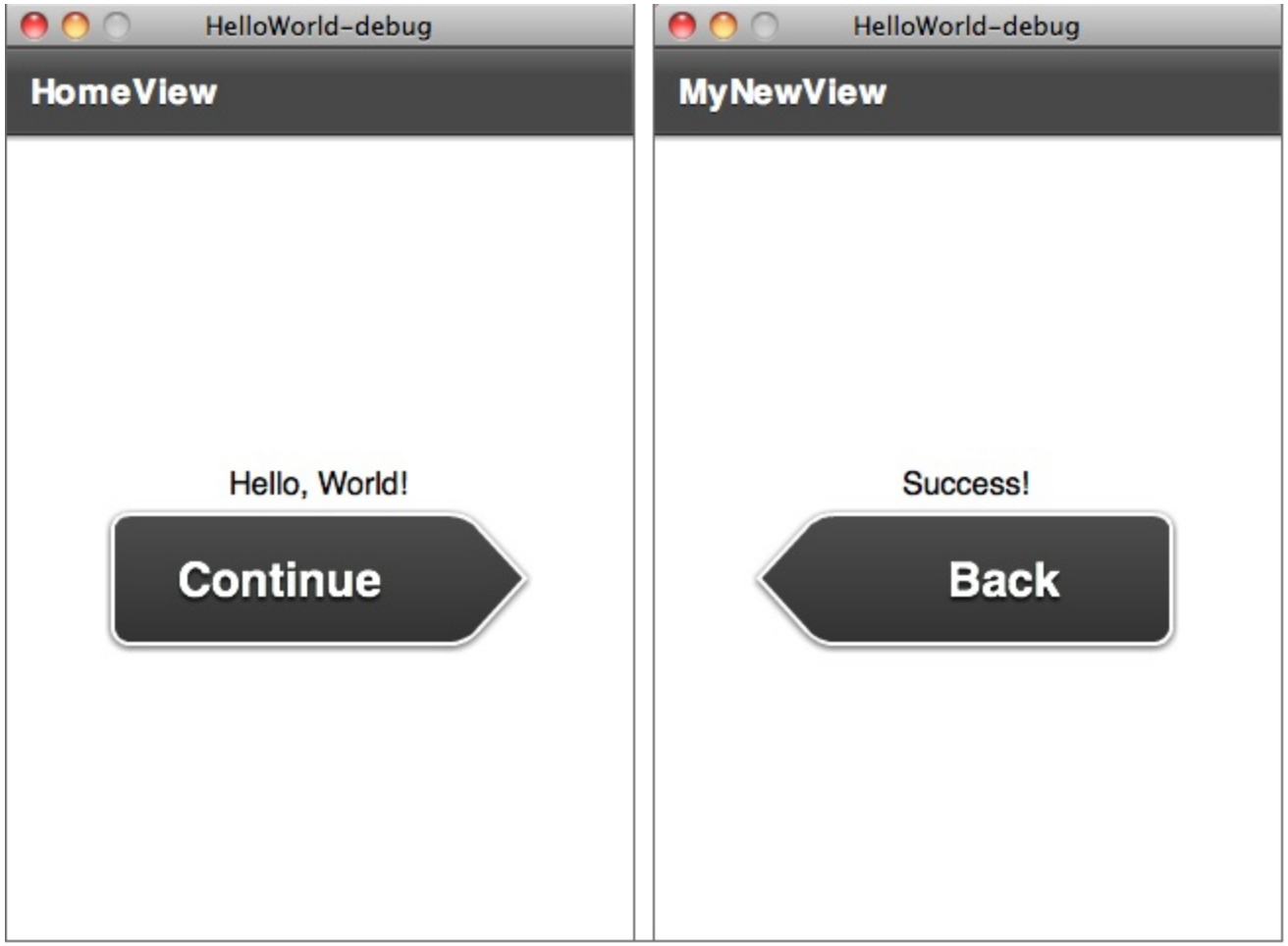

Package Explorer
HelloWorld
src
(default package)
HelloWorld.mxml
main.css
skins
BackButtonSkin1.mxml
BackButtonSkin2.mxml
NextButtonSkin1.mxml
NextButtonSkin2.mxml
views
HelloWorldHomeView.mxml
MyNewView.mxml
blackberry-tablet.xml
HelloWorld-app.xml
Flex 4.5.1
bin-debug
libs
HelloWorld.apk

- Ajouter dans le fichier HelloWorld.mxml un lien vers la .css : <fx:Style source="main.css"/>

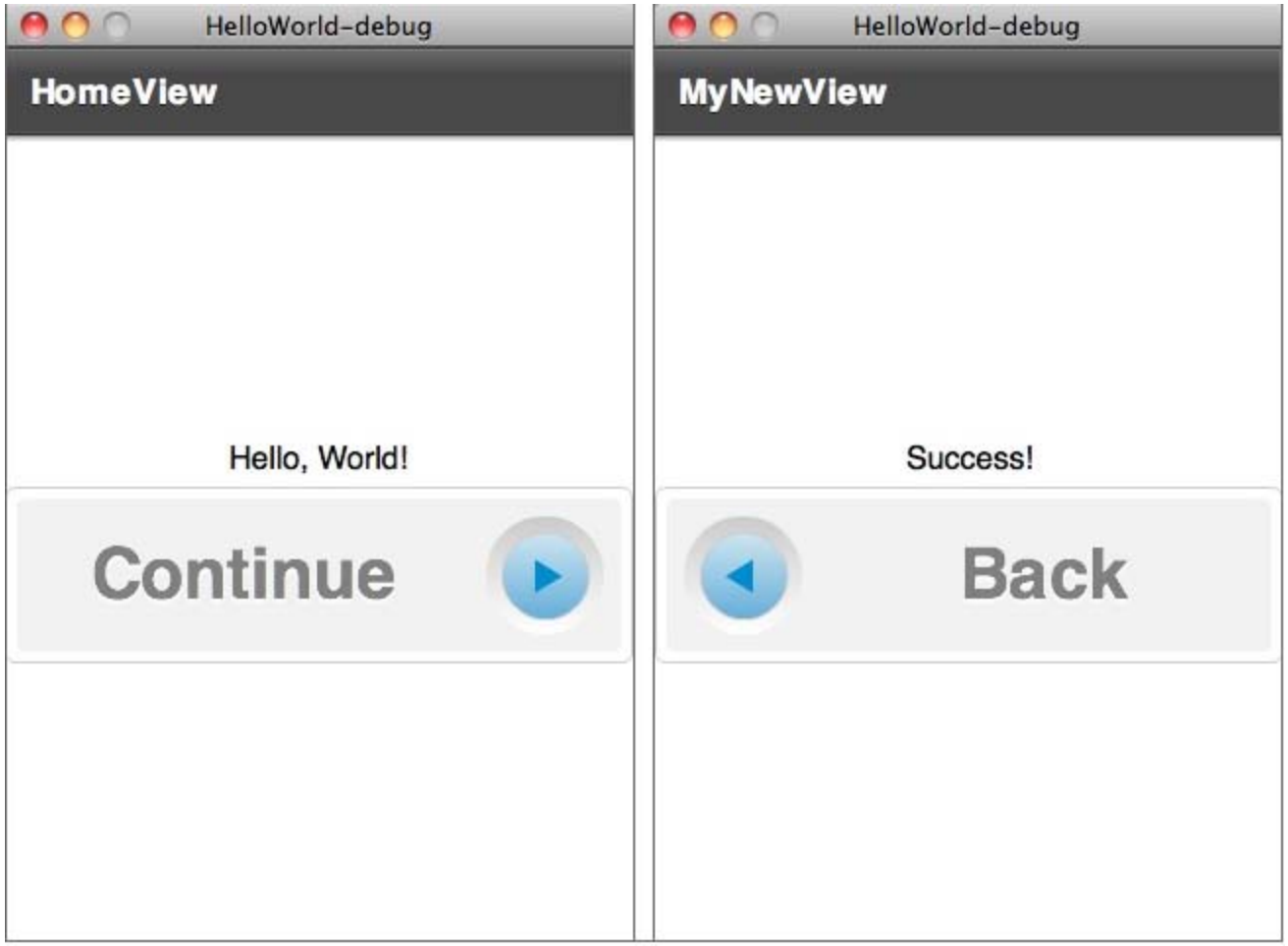


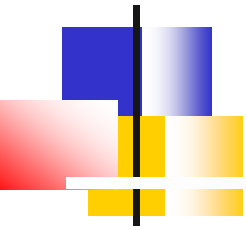
HelloWorld-source.zip

- Lancer l'application : vous devez obtenir :


HelloWorld-debug
HomeView
Hello, World!
Continue
HelloWorld-debug
MyNewView
Success!
Back

- La figure précédente montre un ensemble d’habillages sur les quatre transférés
- Un simple changement de code peut modifier l’apparence des boutons
- D’abord, fermer la fenêtre ADL montrant la demo. Ensuite, double-cliquer sur main.css dans HelloWorld/src/(default package) pour l’ouvrir
- Changer l’habillage de "1" to "2":
  - skins.NextButtonSkin1 ➔ skins.NextButtonSkin2


HelloWorld-debug
HomeView
Hello, World!
Continue
HelloWorld-debug
MyNewView
Success!
Back



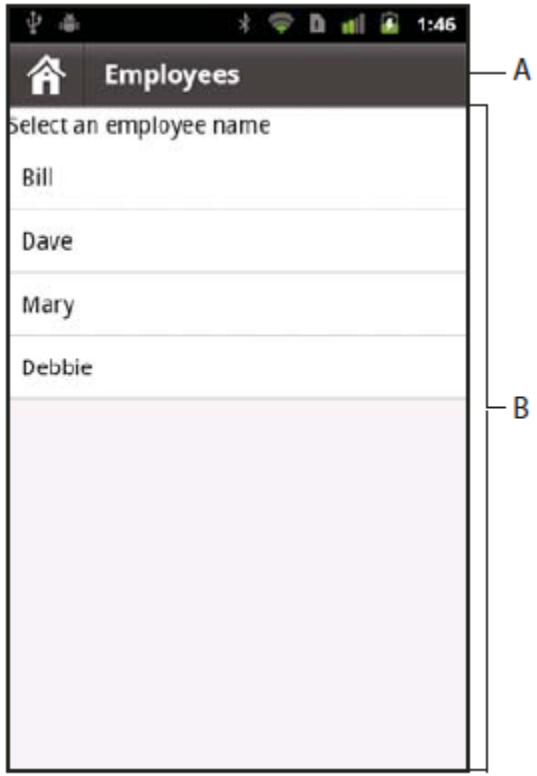
Flex 4.5

Agencement d’une application Mobile

Interface utilisateur et agencement

Agencer une application mobile

- Utiliser des sections et des vues
  - Une application mobile est définie par un ou plusieurs écrans, ou vues
  - Exemple : liste de contacts ayant 3 vues
    1. La vue principale (Home) pour ajouter des informations sur les contacts
    2. La vue des contacts contenant une liste de contacts
    3. La vue de recherche pour rechercher les contacts (Maison)

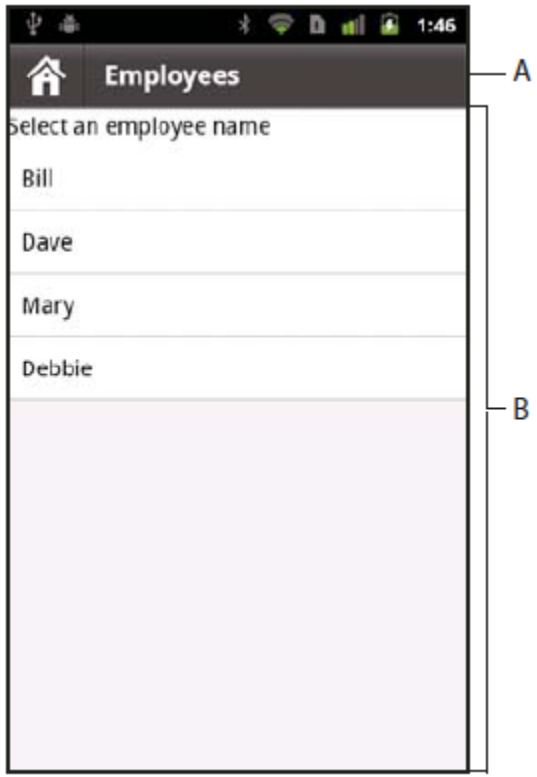


A. ActionBar control B. Content area

Interface utilisateur et agencement

Agencer une application mobile

- Les contrôles associés
  - Chaque zone utilise un contrôle
  - A : ActionBar control
    - Permet l'affichage contextuel à propos de l'état courant des contacts
    - Comprend
      - Une zone de titre, une zone de navigation vers les contacts
  - B : Content Area
    - Affiche les écrans individuels

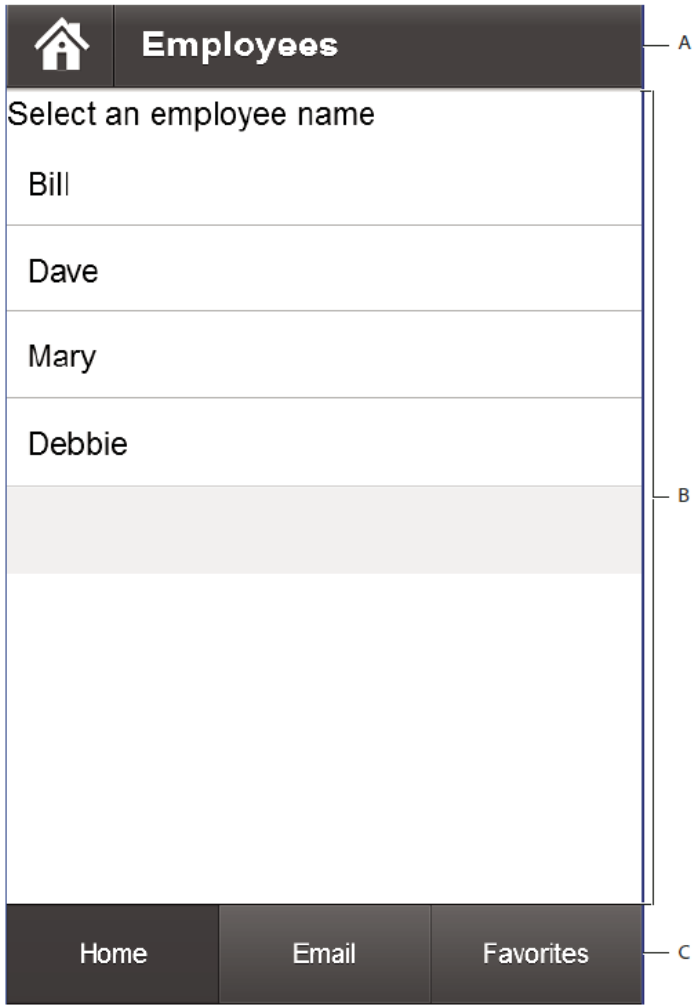


A. ActionBar control B. Content area

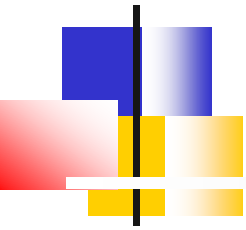
Interface utilisateur et agencement

Agencer une application mobile

- Une application mobile avec des sections
  - Une application mobile plus complexe peut définir plusieurs zones ou sections
  - Par exemple,
    - Une section Home
    - Une section Email
    - Une section Favorites
  - Chaque section a une ou plusieurs vues

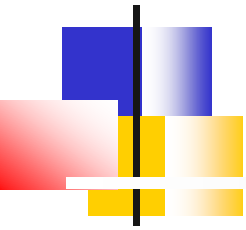


A. ActionBar control B. Content area C. Tab bar



Définir des sections

- Les contrôles
  - Pour gérer les vues dans les sections, Flex utilise la barre tab
    - Chaque bouton de la barre tab correspond à une section
    - Flex utilise le contrôle ButtonBarBase pour implémenter la barre tab
  - Chaque section définit son ActionBar
  - Ainsi la barre tab est globale et ActionBar est spécifique à chaque section

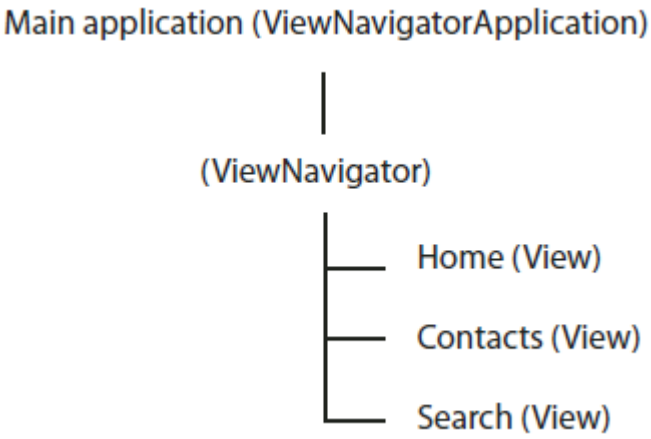


Flex 4.5

Agencer une application simple

Interface utilisateur et agencement

- Principe
  - La figure suivante montre l'architecture d'une application mobile simple, composée de 4 fichiers :
    - Un fichier du main et un fichier pour chaque vue
    - Il n'y a pas de fichier distinct pour ViewNavigator
    - Le conteneur ViewNavigatorApplication le crée



- Les classes utilisées dans une application mobile simple

Class	Description
ViewNavigatorApplication	Defines the main application file. The ViewNavigatorApplication container does not take any children.
ViewNavigator	Controls navigation among the views of an application. The ViewNavigator also creates the ActionBar control. The ViewNavigatorApplication container automatically creates a single ViewNavigator container for the entire application. Use methods of the ViewNavigator container to switch between the different views.
View	Defines the views of the application, where each view is defined in a separate MXML or ActionScript file. An instance of the View container represents each view of the application. Define each view in a separate MXML or ActionScript file.

Les étapes

1. Utilisez le conteneur : ViewNavigatorApplication pour définir le fichier de l'application principale : ExempleMobile1 et SparkSingleSection

```
<?xml version="1.0" encoding="utf-8"?>
<s:ViewNavigatorApplication
   xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   firstView="views.HomeView">
</s:ViewNavigatorApplication>
```

- Le conteneur ViewNavigatorApplication crée automatiquement un simple objet ViewNavigator qui
  - définit l'ActionBar
  - permet de naviguer dans les vues de l'application

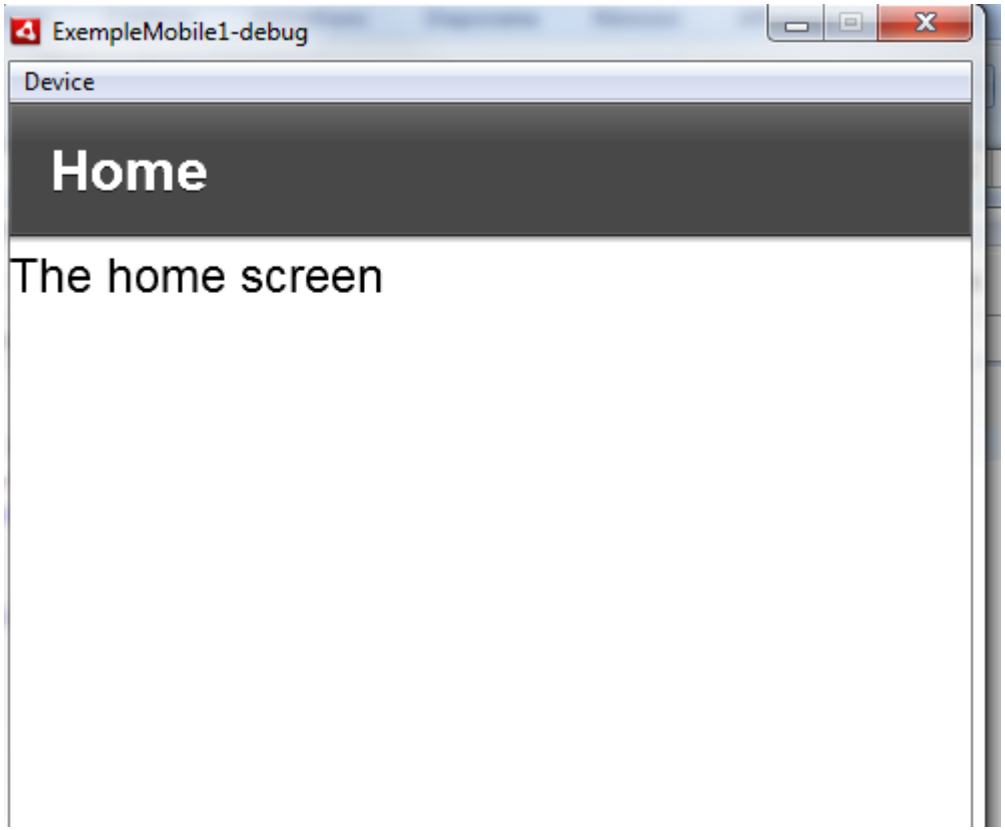
2. Ajouter un conteneur View à l'application
   - Chaque application mobile a au moins une vue
   - Quand l'application principale crée ViewNavigator, elle ne définit aucune vue
   - Chaque vue peut être décrite en MXML ou ActionScript
   - Chaque vue contient une propriété data qui spécifie la donnée associée à la vue
   - Les vues peuvent utiliser cette propriété pour passer les données d'une vue à l'autre quand l'utilisateur navigue entre elles

- ViewNavigatorApplication.firstView = propriété à utiliser pour spécifier le fichier qui définit la première vue
- Dans l'exemple précédent, la propriété firstView spécifie views.HomeView

- La vue : HomeView.mxml :

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark" title="Home">
   <s:layout>
      <s:VerticalLayout paddingTop="10"/>
   </s:layout>
   <s:Label text="The home screen"/>
</s:View>
```

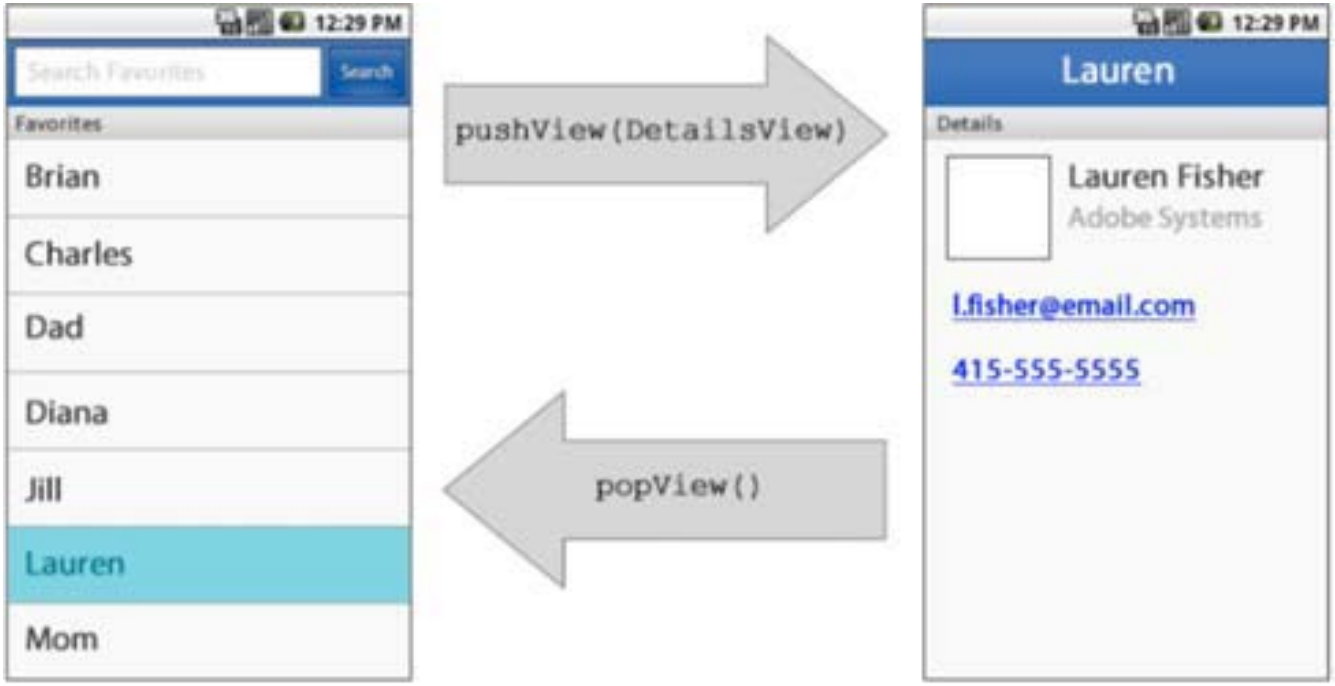
- Résultat : ExempleMobile1



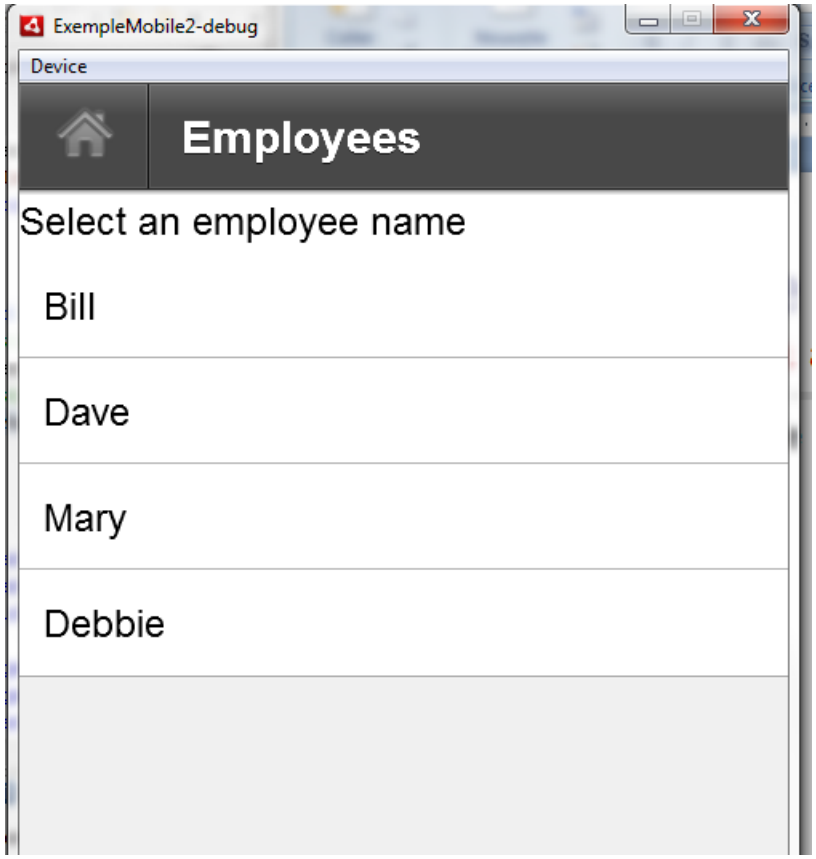
- Naviguez dans les vues d'une application mobile simple : ExempleMobile2
  - Une pile d'objets vues contrôle la navigation dans une application mobile
  - L'objet en haut de la pile définit la vue visible
  - Le conteneur ViewNavigator gère la pile
  - Pour changer de vue, il faut empiler un objet view ou dépiler pour revenir à la précédente vue
  - Pour économiser la mémoire, par défaut le ViewNavigator garantit qu'un seul point de vue est en mémoire à la fois
  - Cependant, il conserve les données pour les vues antérieures sur la pile
  - Par conséquent, lorsque l'utilisateur revient à la vue précédente, la vue peut être réinitialisée avec les données appropriées

- Les méthodes de la classe ViewNavigator pour contrôler la navigation
  - pushView ()
    - empiler une vue. La Vue passée comme argument à pushView () devient la vue courante
  - popView ()
    - Dépiler l'objet View courant. L'objet précédent devient la vue courante
  - popToFirstView ()
    - Dépiler tous les objets View et les détruire, sauf le premier. L'objet Voir le premier sur la pile devient la vue courante

- popAll ()
  - Vide la pile de ViewNavigator et détruit tous les objets View. Votre application affiche une vue blanche
- Exemple



- Les vues


ExempleMobile2-debug
Device
Employees
Select an employee name
Bill
Dave
Mary
Debbie


ExempleMobile2-debug
Device
Employee View
Bill
Smith
11233

- Démarche pour créer des vues :

1. Créer un écran d'accueil (de démarrage)
   - Exemple : ExempleMobile2
   - Vous pouvez charger un écran de démarrage directement à partir d'un fichier image
   - Pour configurer l'écran de démarrage, utilisez les propriétés splashScreenImage, splashScreenScaleMode et splashScreenMinimumDisplayTime de la classe d'application
   - Par exemple, l'exemple suivant charge un écran de démarrage à partir d'un fichier JPG à l'aide du format letterbox :

```
<?xml version="1.0" encoding="utf-8"?>
<s:ViewNavigatorApplication
    xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView"
    splashScreenImage="@Embed('assets/logo.jpg')"
    splashScreenScaleMode="letterbox">
</s:ViewNavigatorApplication>
```

2. Définir des vues

- Une application mobile définit typiquement des écrans multiples, ou des vues
- Comme les utilisateurs naviguent à travers l'application, ils passent d’une vue à l’autre
- Pour définir le point de vue d'une application mobile, utilisez le conteneur View
- Pour contrôler la navigation entre les vues, utilisez le conteneur ViewNavigator

- Utiliser pushView() pour changer de vue
  - Utiliser la méthode ViewNavigator.pushView() pour empiler une nouvelle vue
  - Accéder à ViewNavigator la propriété ViewNavigatorApplication.navigator
  - L’empilement d’une vue change l’affichage de l’application qui se cale sur la vue

Syntaxe de pushView()

- pushView(viewClass:Class, data:Object = null, context:Object = null, transition:spark.transitions:ViewTransitionBase = null):void
- Où :
  - viewClass spécifie le nom de classe de la vue
    - Cette classe correspond généralement au fichier MXML qui définit la vue
  - data : spécifie les données transmises à la vue
    - Cet objet est écrit à la propriété View.data de la nouvelle vue

- Context : spécifie un objet arbitraire écrit à la propriété ViewNavigator.context
  - Lorsque la nouvelle vue est créée, elle peut faire référence à cette propriété et effectuer une action basée sur cette valeur
  - Par exemple, la vue peut afficher les données de différentes manières sur la base de leur contexte
- Transition : spécifie la transition à jouer lorsque la vue change

- Utiliser data pour passer un objet simple
  - On peut utiliser data pour passer des données utilisées par la vue
  - La vue peut ensuite accéder à ces données en utilisant la propriété View.data

- Exemple : EmployeeView.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   title="Employee View">
   <s:layout>
      <s:VerticalLayout paddingTop="10"/>
   </s:layout>
   <s:VGroup>
      <s:Label text="{data.firstName}"/>
      <s:Label text="{data.lastName}"/>
      <s:Label text="{data.companyID}"/>
   </s:VGroup>
</s:View>
```

- Exemple
  - Dans cet exemple, EmployeeView est défini dans le fichier EmployeeView.mxml
  - Cette vue utilise la propriété des données pour accéder aux nom, prénom et ID d'un employé de l'objet qui lui est transmis
  - La propriété View.data est garantie d'être valide au moment de l'événement pour ajouter l'objet View

3. Passer des données à la première vue de l’application
   - Les propriétés ViewNavigatorApplication.firstView et ViewNavigator.firstView définissent la première vue de l’application
   - Pour passer des données à la première vue, utilisez la propriété ViewNavigatorApplication.firstViewData, ou la propriété ViewNavigator.firstViewData

Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSection.mxml -->
<s:ViewNavigatorApplication
    xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView">
<fx:Script>
    <![CDATA[
      protected function
      button1_clickHandler(event:MouseEvent):void {
      // Switch to the first view in the section.
      navigator.popToFirstView(); }
 ]]>
</fx:Script>
    <s:navigationContent>
      <s:Button icon="@Embed(source='assets/Home.png')"
      click="button1_clickHandler(event)"/>
    </s:navigationContent>
</s:ViewNavigatorApplication>
```

Commentaires

- Dans l'exemple, on a défini une application mobile en utilisant le conteneur ViewNavigatorApplication
- Le conteneur ViewNavigatorApplication crée automatiquement une seule instance de la classe ViewNavigator que vous utilisez pour naviguer dans les vues définies par l'application
- Cet exemple définit un bouton Accueil dans la zone de navigation du contrôle ActionBar
- La sélection du bouton Home dépile toutes les vues avant la première vue

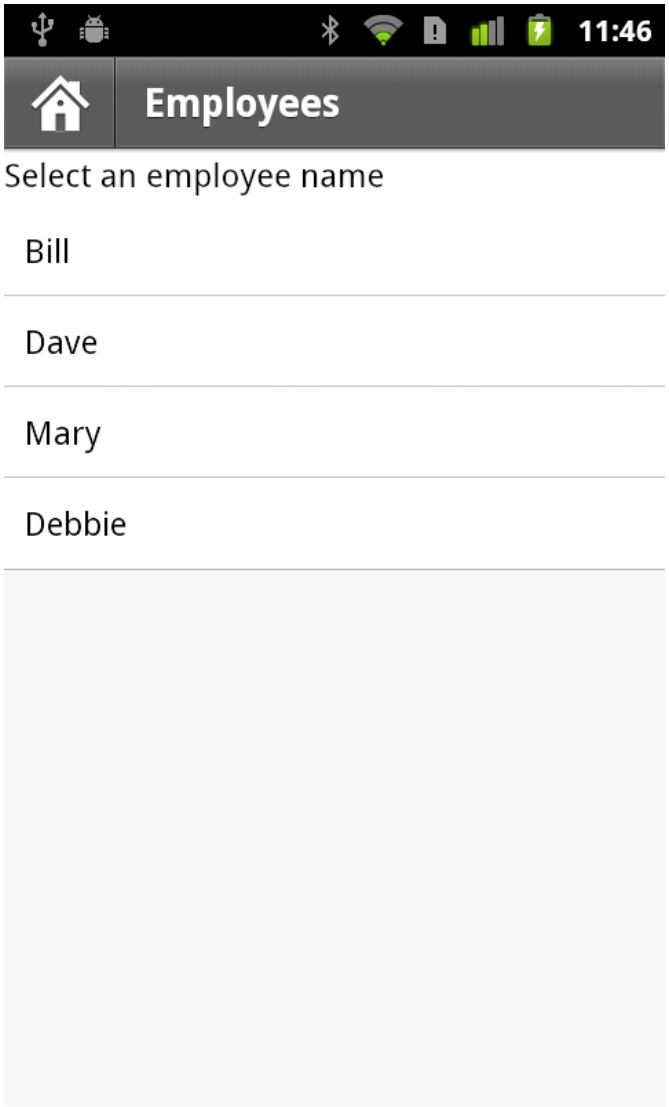
La vue : EmployeeMainView.mxml

- Définit la première vue de l'application

```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
title="Employees">
<s:layout>
    <s:VerticalLayout paddingTop="10"/>
</s:layout>
<fx:Script>
<![CDATA[
import spark.events.IndexChangeEvent;
protected function
    myList_changeHandler(event:IndexChangeEvent):void {
    navigator.pushView(views.EmployeeView,myList.selectedItem);
}
]]>
</fx:Script>
```

```
<s:Label text="Select an employee name"/>
<s:List id="myList" width="100%" height="100%" labelField="firstName"
   change="myList_changeHandler(event)">
<s:ArrayCollection>
   <fx:Object firstName="Bill" lastName="Smith" companyID="11233"/>
   <fx:Object firstName="Dave" lastName="Jones" companyID="13455"/>
   <fx:Object firstName="Mary" lastName="Davis" companyID="11543"/>
   <fx:Object firstName="Debbie" lastName="Cooper" companyID="14266"/>
</s:ArrayCollection>
</s:List>
</s:View>
```

Résultat


11:46
Employees
Select an employee name
Bill
Dave
Mary
Debbie

Résultat

- Cette vue définit un contrôle List qui permet à l'utilisateur de sélectionner un nom d'employé
- La sélection d'un nom provoque le gestionnaire d'événements d'empiler une instance de vue différente, nommé EmployeeView
- Empiler une instance de EmployeeView provoque la demande de changement à la vue EmployeeView
- Le pushView () dans cet exemple prend deux arguments
  - la nouvelle vue
  - et un objet qui définit les données à transmettre à la nouvelle vue
  - Dans cet exemple, vous passez l'objet de données correspondant à l'élément actuellement sélectionné dans le contrôle List

4. Retourner une donnée depuis une vue
   - ViewNavigator.popView () redonne le contrôle de la vue courante à la vue précédente sur la pile
   - Lorsque popView () s'exécute, la vue courante est détruite et la vue précédente sur la pile est restaurée
   - Restaurer la vue précédente inclut la réinitialisation des propriétés des données de la pile

- La nouvelle vue est rétablie avec l'objet de données d'origine au moment où elle a été désactivée
- Par conséquent, vous n'avez pas besoin d’utiliser l'utilisation l'objet d’origine pour transmettre les données de l'ancienne vue vers la nouvelle
- Au lieu de cela, vous remplacez createReturnObject () de l'ancienne vue
- createReturnObject () retourne un objet unique

5. Retourner un type d'objet

- L'objet retourné par createReturnObject() est affecté à la propriété ViewNavigator.poppedViewReturnedObject
- Le type de donnée de poppedViewReturnedObject est ViewReturnObject.
- ViewReturnObject definit deux propertiés, context and object
  - object
    - contient l'objet retourné par createReturnObject()
  - context
    - contient la valeur de l'argument context qui était passée à la vue quand la vue était empilée en utilisant pushView()

- La propriété poppedViewReturnedObject est garantie pour se mettre dans la nouvelle vue avant que la vue reçoive l'événement add
- Si la propriété poppedViewReturnedObject.object est nulle, aucune donnée n'est retournée

- Exemple : passage de la donnée à la vue : le projet SelectFont
  - L'exemple suivant montre une vue qui vous permet de définir une taille de police
  - L'exécution de la méthode createReturnObject () renvoie la valeur comme un nombre
  - Le champ de la propriété fontSize transmis à partir de la vue précédente définit la valeur initiale du contrôle TextInput :
    - <s:TextInput id="ns" text="{data.fontSize}"/>

Le programme principal

```
<?xml version="1.0" encoding="utf-8"?>
<s:ViewNavigatorApplication
    xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.MainFontView">
</s:ViewNavigatorApplication>
```

- MainFontView.mxml

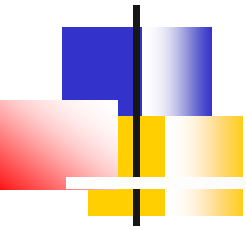
```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Font Size"
    add="addHandler(event);">
    <s:layout>
      <s:VerticalLayout paddingTop="10"/>
    </s:layout>
    <fx:Script>
      <![CDATA[
      import mx.events.FlexEvent;
       //Récupérer la valeur de la fonte dans une autre vue
      protected function clickHandler(event:MouseEvent):void
    {
            navigator.pushView(views.SelectFont, data);
      }
```

```
// Set the font size in the event handler for the add event.
protected function addHandler(event:FlexEvent):void {
if (data == null) {
    data = new Object();
    data.fontSize = getStyle('fontSize');
return;
}
// Otherwise, set data.fontSize to the retured value, and set the font size.
data.fontSize = navigator.poppedViewReturnedObject.object;
setStyle('fontSize', data.fontSize);
}
]]>
</fx:Script>
<s:actionContent>
    <s:Button label="Set Font&gt;" click="clickHandler(event);"/>
</s:actionContent>
<s:Label text="Text to size."/>
</s:View>
```

SelectFont.mxml : la vue

```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Select Font Size" add="addHandler(event);">
    <s:layout>
      <s:VerticalLayout paddingTop="10" paddingLeft="10"
      paddingRight="10"/>
    </s:layout>
<fx:Script>
<![CDATA[
import mx.events.FlexEvent;
// Define return Number object.
protected var fontSize:Number;
// Initialize the return object with the passed in font size.
// If you do not set a value,
// return this value for the font size.
```

```
protected function addHandler(event:FlexEvent):void {
fontSize = data.fontSize;
}
// Save the value of the specified font.
protected function changeHandler(event:Event):void {
    fontSize=Number(ns.text);
    navigator.popView();
}
// Override createReturnObject() to return the new font size.
override public function createReturnObject():Object {
return fontSize;
}
]]>
</fx:Script>
<s:Label text="Select Font Size"/>
<!-- Set the initlial value of the TextInput to the passed fontSize -->
<s:TextInput id="ns" text="{data.fontSize}"/>
<s:Button label="Save" click="changeHandler(event);"/>
</s:View>
```



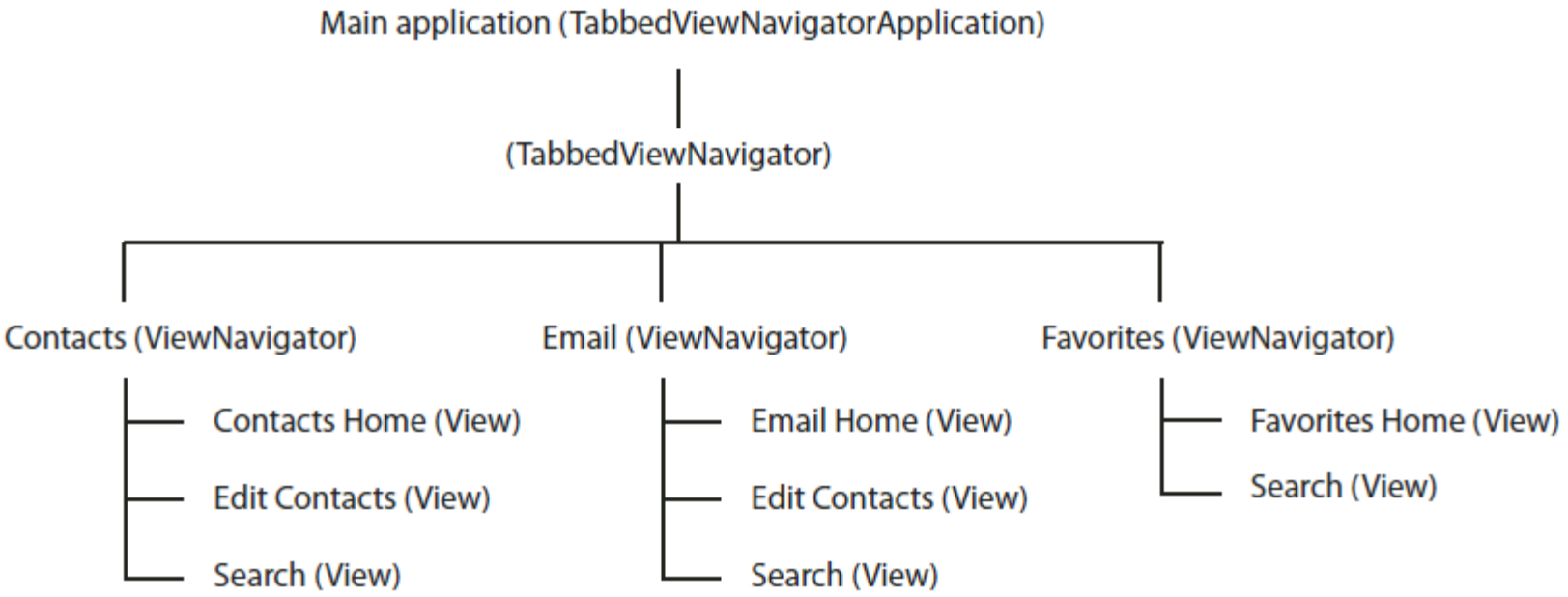
Flex 4.5

Agencer une application mobile avec plusieurs sections

Interface utilisateur et agencement

Agencer une application mobile

- Principe
  - Une appli Mobile peut recueillir des vues dans chaque section
  - A l'exécution, un seul point de vue est actif et résident en mémoire
  - Exemple



Classes utilisées

Class	Description
TabbedViewNavigatorApplication	Defines the main application file. The only allowable child of the TabbedViewNavigatorApplication container is ViewNavigator. Define one ViewNavigator for each section of the application.
TabbedViewNavigator	Controls navigation among the sections that make up the application. The TabbedViewNavigatorApplication container automatically creates a single TabbedViewNavigator container for the entire application. The TabbedViewNavigator container creates the tab bar that you use to navigate among the sections.
ViewNavigator	Define one ViewNavigator container for each section. The ViewNavigator controls navigation among the views that make up the section. It also creates the ActionBar control for the section.
View	Defines the views of the application. An instance of the View container represents each view of the application. Define each view in a separate MXML or ActionScript file.

Exemple : SparkMultipleSectionAB

- Utilisez le conteneur TabbedViewNavigatorApplication pour définir l'application principale, comme suit :

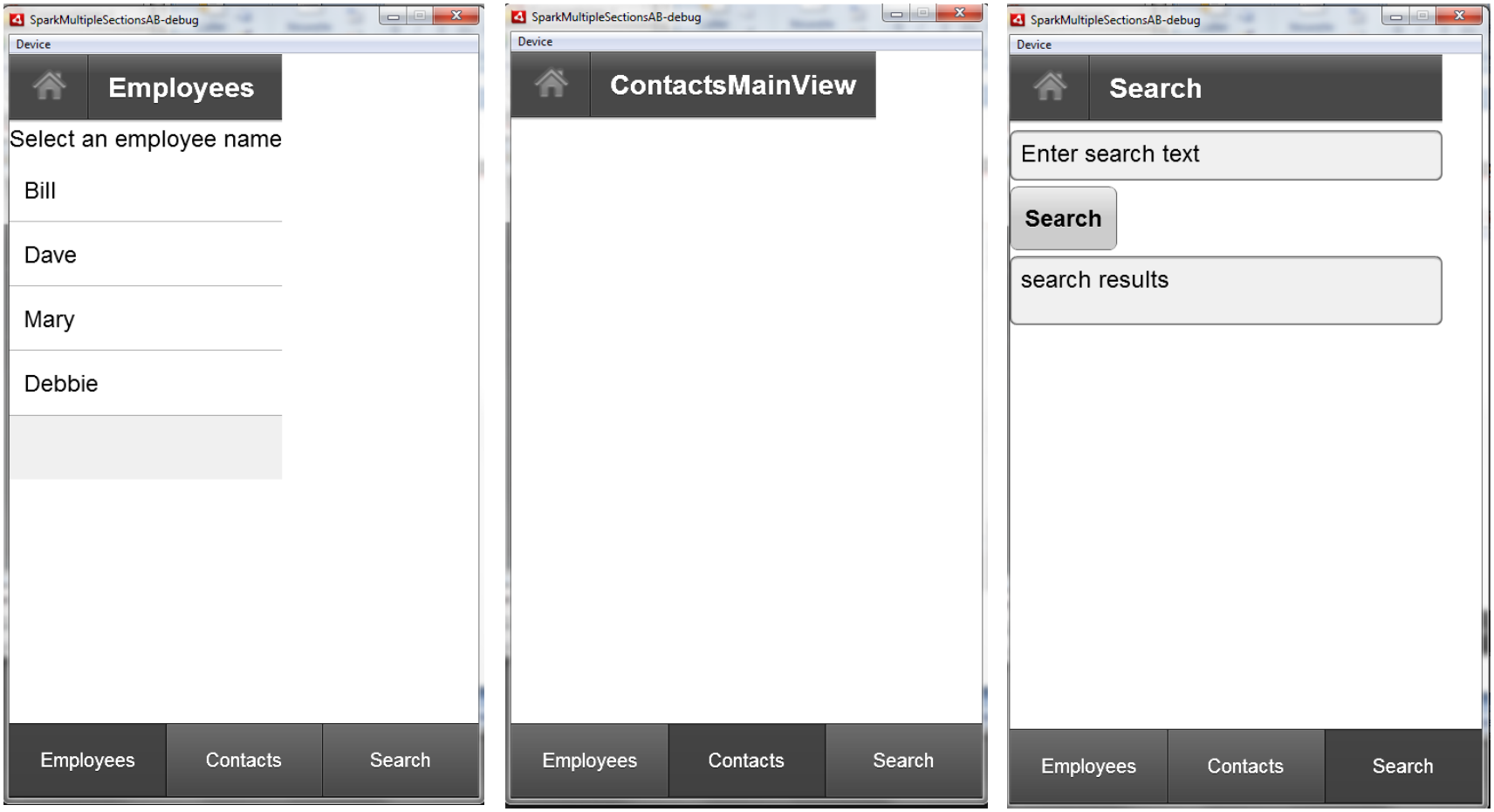
```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMultipleSectionsSimple.mxml -->
<s:TabbedViewNavigatorApplication
   xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark">
<s:ViewNavigator label="Contacts" firstView="views.ContactsHome"/>
<s:ViewNavigator label="Email" firstView="views.EmailHome"/>
<s:ViewNavigator label="Favorites" firstView="views.FavoritesHome"/>
</s:TabbedViewNavigatorApplication>
```

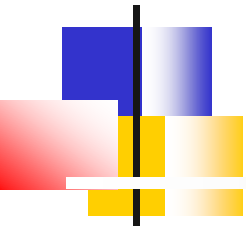
- Le seul élément enfant admissible du conteneur TabbedViewNavigatorApplication est ViewNavigator
- Chaque section correspond à un conteneur ViewNavigator différent
- Utilisez le conteneur ViewNavigator pour naviguer dans les vues de chaque section, et pour définir le contrôle ActionBar pour la section
- Utilisez la propriété ViewNavigator.firstView pour spécifier le fichier qui définit la première vue dans la section

- Le conteneur TabbedViewNavigatorApplication crée automatiquement un seul conteneur de type TabbedViewNavigator
- Le conteneur TabbedViewNavigator crée alors une barre d'onglets au bas de l'application
- Vous n'avez pas à ajouter une logique à l'application pour naviguer parmi les sections

- Pour changer l'affichage dans la section courante, qui correspond à la ViewNavigator courant, utilisez :
  - pushView () et popView ()
- Pour changer le dossier courant, utilisez la barre d'onglets
- Lorsque vous switchez de section, vous switchez au conteneur ViewNavigator de la nouvelle section
- L'affichage change pour afficher l'objet View placé au sommet de la pile pour la nouvelle ViewNavigator
- Vous pouvez également modifier des sections par programme en utilisant la propriété tabbedViewNavigator.selectedIndex
- Cette propriété contient l'index 0-base du navigateur de vue sélectionnée

- Exemple : SparkMultipleSectionsAB

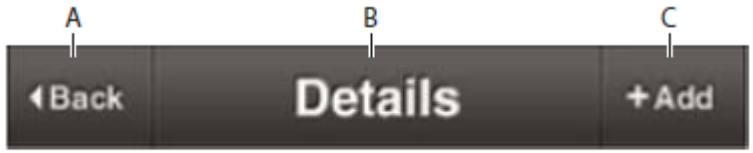




Interface utilisateur et agencement

- Définir la navigation, le titre, et l'action de contrôle
  - Configurez le contrôle ActionBar
    - Le conteneur ViewNavigator définit le contrôle ActionBar
    - Le contrôle ActionBar fournit une zone standard pour un titre, et pour les contrôles de navigation et d'action
    - Il vous permet de définir des contrôles globaux que les utilisateurs peuvent accéder de n'importe où dans l'application, ou dans une vue spécifique
    - Par exemple, vous pouvez utiliser le contrôle ActionBar pour ajouter un bouton Home, un bouton de recherche et d’autres options

- Configurez le contrôle ActionBar (suite)
  - Pour une application mobile avec une seule section, ce qui signifie un récipient ViewNavigator unique, toutes les vues partagent la barre de la même action
  - Pour une application mobile avec plusieurs sections, ce qui signifie une avec des conteneurs ViewNavigator multiples, chaque section définit sa barre d'action propre
  - Utilisez le contrôle ActionBar pour définir la zone de la barre d'action. Le contrôle ActionBar définit trois zones distinctes, comme suit :



A. Navigation area B. Title area C. Action area

Configuration des zones de l'ActionBar

- Zone de navigation
  - Contient des composants qui permettent à l'utilisateur de naviguer dans la section
  - Par exemple, vous pouvez définir un bouton Accueil dans la zone de navigation
  - Utilisez la propriété navigationContent pour définir les éléments qui apparaissent dans la zone de navigation
  - Utilisez la propriété navigationLayout pour définir la disposition de la zone de navigation

Configuration des zones de l'ActionBar

- Zone de titre
  - Contient soit une chaîne contenant le texte du titre, ou des composants
  - Si vous spécifiez les composants, vous ne pouvez pas spécifier une chaîne de titre
  - Utilisez la propriété du titre pour spécifier la chaîne à apparaître dans la zone de titre
  - Utilisez la propriété titleContent pour définir les éléments qui apparaissent dans la zone de titre
  - Utilisez la propriété titleLayout pour définir la disposition de la zone de titre
  - Si vous spécifiez une valeur pour la propriété titleContent, l'habillage ActionBar ignore la propriété du titre

Configuration des zones de l’ActionBar

- Zone d’action
  - Contient des composants qui définissent les actions que l'utilisateur peut prendre dans une vue. Par exemple, vous un bouton de recherche ou un bouton d’actualisation
  - Utilisez la propriété actionContent pour définir les éléments qui apparaissent dans la zone d'action
  - Utilisez la propriété actionLayout pour définir la disposition de la zone d'action

Exemple : SparkActionBarSimple.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
firstView="views.MobileViewHome">
<fx:Script>
<![CDATA[
protected function button1_clickHandler(event:MouseEvent):void {
// Perform a refresh
}
]]>
</fx:Script>
<s:navigationContent>
<s:Button label="Home" click="navigator.popToFirstView();"/>
</s:navigationContent>
<s:actionContent>
<s:Button label="Refresh" click="button1_clickHandler(event);"/>
</s:actionContent>
</s:ViewNavigatorApplication>
```

Exemple : MobileViewHome.mxml

```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
title="Home View">
<s:layout>
    <s:VerticalLayout paddingTop="10"/>
</s:layout>
<s:Label text="Home View"/>
<s:Button label="Submit"/>

</s:View>
```

Autre exemple :

- Cet exemple utilise un fichier d'application principal avec un seul article qui définit un bouton Accueil dans la zone de navigation du conteneur ViewNavigatorApplication et un bouton de recherche dans le domaine d'action:

Autre exemple : SparkActionBarOverride.mxml

```
<s:ViewNavigatorApplication  …
    firstView="views.MobileViewHomeOverride">
<fx:Script>
<![CDATA[
protected function button1_clickHandler(event:MouseEvent):void {
navigator.popToFirstView();}
protected function button2_clickHandler(event:MouseEvent):void {
// Handle search}
]]>
</fx:Script>
<s:navigationContent>
    <s:Button icon="@Embed(source='assets/Home.png')"
      click="button1_clickHandler(event);"/>
</s:navigationContent>
<s:actionContent>
    <s:Button icon="@Embed(source='assets/Search.png')"
     click="button2_clickHandler(event);"/>
</s:actionContent>
</s:ViewNavigatorApplication>
```

SparkActionBarOverride.mxml

- La première vue de cette application est MobileViewHomeOverride view
- La vue MobileViewHomeOverride définit un bouton de contrôle pour naviguer dans une seconde vue qui définit la page de recherche :

MobileViewHomeOverride.mxml

```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
title="Home View">
<s:layout>
    <s:VerticalLayout paddingTop="10"/>
</s:layout>
<fx:Script>
<![CDATA[
// Navigate to the Search view.
protected function
    button1_clickHandler(event:MouseEvent):void {
    navigator.pushView(SearchViewOverride);
}
]]>
</fx:Script>
<s:Label text="Home View"/>
<s:Button label="Search" click="button1_clickHandler(event)"/>
</s:View>
```

MobileViewHomeOverride.mxml

- Le conteneur View qui définit la page de recherche remplace la zone de titre et de la zone d'action de la commande ActionBar, comme suit :

SearchViewOverride.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SearchViewOverride.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark">
<s:layout>
    <s:VerticalLayout paddingTop="10"
    paddingLeft="10" paddingRight="10"/>
</s:layout>
<fx:Script>
<![CDATA[
protected function
    button1_clickHandler(event:MouseEvent):void {
// Perform a search.
}
]]>
</fx:Script>
```

```
<!-- Override the title to insert a TextInput control. -->
<s:titleContent>
    <s:TextInput text="Enter search text ..." textAlpha="0.5"
    width="250"/>
</s:titleContent>
<!-- Override the action area to insert a Search button. -->
<s:actionContent>
<s:Button label="Search" click="button1_clickHandler(event);"/>
</s:actionContent>
<s:Label text="Search View"/>
<s:TextArea text="Search results appear here ..."
    height="75%"/>
</s:View>
```

Flex 4.5

Configuration de l’application en fonction de l’orientation

Configuration de l’orientation

- Principe : SearchViewStates
  - Un mobile définit l'orientation d'une application automatiquement lorsque le dispositif de change d'orientation
  - Pour configurer votre application pour les différentes orientations, Flex définit deux états d'affichage qui correspondent aux orientations portrait et paysage
  - L'exemple suivant utilise l'état d'affichage pour contrôler la propriété mise en page d'un conteneur de groupe basée sur l'orientation actuelle :

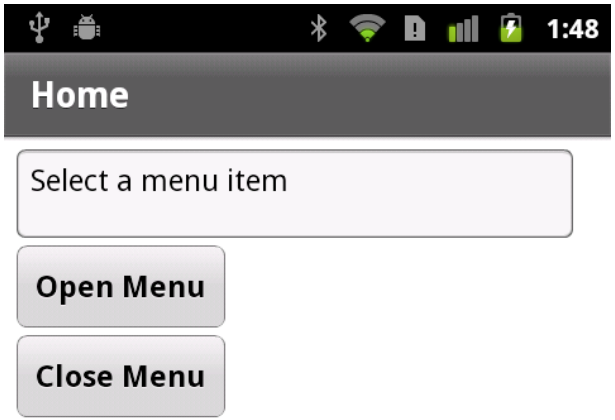
Exemple

```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"  title="Search">
    <s:layout>
      <s:VerticalLayout paddingTop="10"/>
    </s:layout>
    <s:states>
      <s:State name="portrait"/>
      <s:State name="landscape"/>
    </s:states>
    <s:Group>
      <s:layout> <s:VerticalLayout/></s:layout>
      <s:layout.landscape><s:HorizontalLayout/></s:layout.landscape>
      <s:TextInput text="Enter search text" textAlpha="0.5"/>
      <s:Button label="Search"/>
    </s:Group>
    <s:TextArea text="search results" textAlpha="0.5"/>
</s:View>
```

Interface utilisateur et agencement

Définir des menus

- Principe : ViewMenuSkin
  - Le conteneur ViewMenu définit un menu en bas du conteneur View
  - Chaque conteneur View définit son propre menu spécifique à ce point de vue



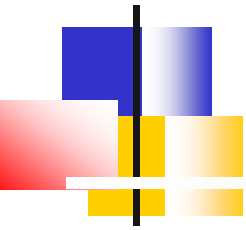
- Le conteneur ViewMenu définit un menu avec une seule hiérarchie des boutons de menu
- Autrement dit, vous ne pouvez pas créer des menus avec des sous-menus
- Les enfants du conteneur ViewMenu sont définis comme des commandes ViewMenuItem
- Chaque contrôle ViewMenuItem représente un seul bouton dans le menu

Créer un menu

- Utilisez la propriété View.viewMenuItems pour définir le menu pour une vue
- La propriété View.viewMenuItems prend un vecteur de contrôle ViewMenuItem, comme le montre l'exemple suivant : ViewMenuHome.mxml
- On a utilisé la propriété View.viewMenuItems pour ajouter cinq éléments de menu, où chacun est représenté par un contrôle ViewMenuItem
- Chaque contrôle ViewMenuItem utilise la propriété label pour spécifier le texte qui apparaît dans le menu pour cet élément

Principe

- Utilisez la propriété View.viewMenuItems pour définir le menu pour une vue
- La propriété View.viewMenuItems prend un vecteur de contrôle ViewMenuItem, comme le montre l'exemple suivant : ViewMenuHome.mxml
- On a utilisé la propriété View.viewMenuItems pour ajouter cinq éléments de menu, où chacun est représenté par un contrôle ViewMenuItem
- Chaque contrôle ViewMenuItem utilise la propriété label pour spécifier le texte qui apparaît dans le menu pour cet élément



TD

- Énoncé
  - Regardez l’exemple : FlexMobileBasicsSolution
  - Essayez de vous en inspirer pour faire une application personnelle