Développement d'applications mobiles avec ADOBE® FLEX® et ADOBE® FLASH® BUILDER™

Informations juridiques

Sommaire

Chapitre 1 : Prise en main

Premiers pas avec les applications mobiles

La version Adobe Flex 4.5 permet d'utiliser Flex et Adobe Flash Builder sur les smartphones et les tablettes. En tirant profit d'Adobe AIR, vous pouvez maintenant développer des applications mobiles dans Flex avec la même aisance et la même qualité que sur les plateformes d'ordinateurs de bureau.

De nombreux composants Flex existants ont été étendus pour fonctionner sur les périphériques mobiles, y compris l'ajout de la prise en charge du défilement tactile. Flex 4.5 contient également un ensemble de nouveaux composants conçus pour simplifier la création d'applications qui suivent les modèles de conception standard pour les téléphones et les tablettes.

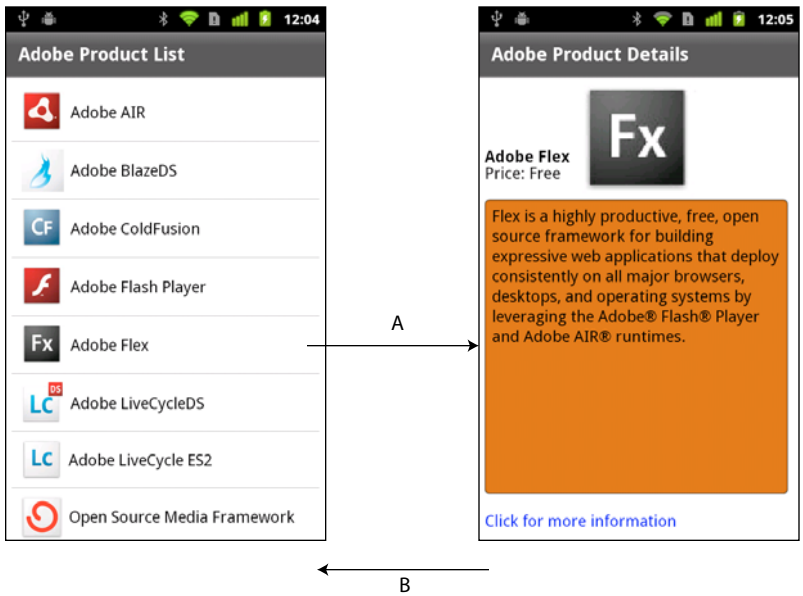
Flash Builder a également été mis à jour pour fournir de nouvelles fonctionnalités pour la prise en charge du développement d'applications pour les périphériques mobiles. Avec Flash Builder, vous pouvez développer, tester et déboguer des applications sur votre ordinateur de bureau ou directement sur votre périphérique mobile.

Conception d'une application mobile

En raison de la taille d'écran réduite des périphériques mobiles, les applications mobiles sont généralement conçues différemment des applications pour navigateur. Lors du développement d'applications mobiles, vous divisez en général le contenu en une série de vues à afficher sur un périphérique mobile.

Chaque vue contient les composants qui se rapportent à une tâche individuelle ou qui contiennent un ensemble unique d'informations. L'utilisateur passe généralement d'une vue à l'autre pour obtenir plus de détails en appuyant sur les composants d'une vue. L'utilisateur peut alors utiliser le bouton Retour du périphérique pour retourner à une vue précédente ou intégrer la navigation dans l'application.

Dans l'exemple suivant, la vue initiale de l'application montre une liste de produits :



A. Sélectionnez un élément de liste pour changer les vues dans l'application. B. Utilisez le bouton Retour du périphérique pour retourner à la vue précédente.

L'utilisateur sélectionne un produit dans la liste pour obtenir plus d'informations. La sélection change la vue pour afficher une description détaillée du produit.

Si vous concevez une application pour les plateformes mobiles, Web et d'ordinateurs de bureau, concevez des interfaces utilisateur distinctes pour chaque plateforme. Toutefois, les applications peuvent partager un code d'accès au modèle et aux données sous-jacent entre toutes les plateformes.

Création d'applications pour téléphones et tablettes

Pour une application destinée à des tablettes, les limites de taille d'écran sont moins contraignantes qu'avec les téléphones. Vous n'êtes pas tenu de structurer une application de tablette autour de petites vues. A la place, vous pouvez créer l'application en utilisant le conteneur Spark Application standard avec les composants et habillages mobiles pris en charge.

Remarque : *vous pouvez créer une application pour un téléphone mobile basée sur le conteneur Spark Application. Toutefois, vous utilisez généralement les conteneurs ViewNavigatorApplication et TabbedViewNavigatorApplication à la place.*

Créez un projet mobile dans Flash Builder pour les tablettes comme vous le feriez pour les téléphones. Les applications pour les tablettes et les téléphones requièrent le même thème mobile pour tirer profit des composants et habillages optimisés pour les applications mobiles.

Création d'applications mobiles dans Flash Builder

Flash Builder introduit un flux de travail productif de conception, création et débogage dans le développement mobile. L'objectif des fonctionnalités mobiles dans Flash Builder est de rendre le développement d'une application mobile basée sur ActionScript ou Flex aussi simple que le développement d'une application de bureau ou Web.

Flash Builder offre deux options pour les tests et le débogage. Vous pouvez lancer et déboguer l'application sur l'ordinateur de bureau en utilisant AIR Debug Launcher (ADL). Pour bénéficier d'un contrôle plus important, lancez et déboguez l'application directement sur un périphérique mobile. Dans les deux cas, vous pouvez utiliser les capacités de débogage de Flash Builder, y compris la définition de points d'arrêt et l'examen de l'état de l'application à l'aide des volets Variables et Expressions.

Lorsque votre application est prête à être déployée, utilisez le processus Exporter vers une version validée, tout comme vous le feriez pour préparer une application de bureau ou Web. La principale différence tient au fait que lorsque vous exportez une version validée d'un projet mobile, Flash Builder groupe la version en tant que programme d'installation natif et non pas en tant que fichier .air. Par exemple, sur Android, Flash Builder produit un fichier .apk qui ressemble à un package d'application Android natif. Le programme d'installation natif permet de distribuer les applications AIR de la même façon que les applications natives sur chaque plateforme.

Déploiement d'applications mobiles dans Flash Builder

Déployez des applications mobiles créées dans Flex en utilisant Adobe AIR pour les périphériques mobiles. Tout périphérique sur lequel vous souhaitez déployer une application mobile doit prendre en charge AIR.

Vos applications peuvent bénéficier pleinement de l'intégration d'AIR à la plateforme mobile. Par exemple, une application mobile peut gérer un bouton de retour et de menu matériel et accéder au stockage local. Vous pouvez également bénéficier de toutes les fonctions fournies par AIR pour les périphériques mobiles. Ces fonctions comprennent la géolocalisation, l'accéléromètre et l'intégration d'une caméra.

Sur un périphérique mobile, il n'est pas nécessaire d'installer AIR avant d'exécuter une application créée dans Flex. La première fois qu'un utilisateur exécute une application créée dans Flex, l'utilisateur est invité à télécharger AIR.

Pour vous familiariser avec AIR et pour plus d'informations sur les fonctionnalités de AIR, consultez les documents suivants :

- A propos d'Adobe AIR
- Appel et arrêt d'une application AIR
- Utilisation des informations d'exécution AIR et du système d'exploitation
- Utilisation des fenêtres natives AIR
- Utilisation des bases de données SQL locales dans AIR

Lors du développement d'applications mobiles, vous ne pouvez pas utiliser les composants Flex suivants pour AIR : WindowedApplication et Window. A la place, utilisez les conteneurs ViewNavigatorApplication et TabbedViewNavigatorApplication. Dans le cas du développement d'applications destinées à des tablettes, vous pouvez également utiliser le conteneur Spark Application.

Pour plus d'informations, voir Utilisation des composants Flex AIR et « Définition d'une application mobile et d'un écran de démarrage » à la page 27.

Utilisation du thème Mobile dans votre application

Un *thème* définit l'aspect et l'ergonomie des composants visuels d'une application. Un thème peut simplement définir la palette chromatique ou la police commune d'une application, ou peut redéfinir entièrement l'habillage de l'ensemble des composants utilisés par l'application.

Vous pouvez définir des styles CSS sur les composants Flex uniquement si le thème actuel comprend ces styles. Pour déterminer si le thème actuel prend en charge le style CSS, affichez l'entrée du style dans le Guide de référence ActionScript 3.0 pour la plate-forme Adobe Flash.

Flex prend en charge trois thèmes principaux : Mobile, Spark et Halo. Le thème Mobile définit l'aspect par défaut des composants Flex lorsque vous créez une application mobile. Afin de rendre certains composants Flex compatibles avec le thème Mobile, Adobe a créé de nouveaux habillages pour les composants. Par conséquent, certains composants ont les habillages spécifiques d'un thème.

Les applications créées avec Flex peuvent cibler différents périphériques mobiles, chacun présentant des tailles et des résolutions d'écran différentes. Flex simplifie le processus de production d'applications indépendantes de la résolution en proposant des habillages indépendants des PPP pour les composants mobiles. Pour plus d'informations sur les habillages mobiles, voir « Notions de base sur l'habillage mobile » à la page 96.

Pour plus d'informations sur les styles et les thèmes, voir Styles et thèmes et « Styles mobiles » à la page 96.

Ressources de la communauté

Découvrez les nouvelles fonctions intégrées dans Flex 4.5 et Flash Builder 4.5, dans :

- Introducing Adobe Flex 4.5 SDK de Deepa Subramaniam, chef de produit chez Adobe.
- Développement mobile à l'aide d'Adobe Flex 4.5 SDK et de Flash Builder 4.5 de Narciso Jaramillo, chef de produit chez Adobe.
- Nouveautés de Flash Builder 4.5 d'Andrew Shorten, chef de produit chez Adobe.

Le Pôle de développement Flex contient de nombreuses ressources conçues pour vous aider à commencer à créer des applications mobiles à l'aide de Flex 4.5 :

- Articles, liens et didacticiels de prise en main
- Exemples d'applications réelles créées dans Flex

- Livre de cuisine Flex Cookbook, qui contient les réponses aux problèmes de codage courants
- Liens vers la communauté Flex et vers d'autres sites dédiés à Flex

Adobe TV est une autre ressource, qui contient des vidéos élaborées par des ingénieurs, des experts produit et des clients Adobe sur le développement d'applications dans Flex. L'une des vidéos disponibles est Création de votre première application mobile dans Flash Builder 4.5.


Vous trouverez ici des informations sur Flash Builder 4.5 mobile compilées par Holly Schinsky.


Mark Doherty, évangéliste Adobe, a publié une vidéo sur la génération d'applications pour les ordinateurs de bureau, les téléphones portables et les tablettes.


James Ward, spécialiste d'Adobe, propose une vidéo concernant la génération d'applications mobiles avec Flex 4.5.


Le blogueur Joseph Labrecque a communiqué sur une démonstration de Flex 4.5 mobile.


Le blogueur Fabio Biondi a communiqué sur la création d'un lecteur YouTube utilisant AIR pour les dispositifs Android à l'aide de Flash Builder.

Différences entre le développement d'applications mobiles, de bureau et de navigateur

Utilisez Flex pour développer des applications pour les environnements de déploiement suivants :

Navigateur Déployez l'application en tant que fichier SWF à utiliser dans Flash Player s'exécutant dans un navigateur.

Bureau Déployez une application AIR autonome pour un ordinateur de bureau, tel qu'un ordinateur exécutant Windows ou un Macintosh.

Mobile Déployez une application AIR autonome pour un périphérique mobile, tel qu'un téléphone ou une tablette.

Les exécutions sur Flash Player et AIR sont similaires. Vous pouvez effectuer la plupart des mêmes opérations dans l'un ou l'autre des environnements d'exécution. AIR vous permet non seulement de déployer des applications autonomes en dehors d'un navigateur, mais vous fournit en outre une intégration étroite à la plateforme hôte. Cette intégration permet par exemple d'accéder au système de fichiers du périphérique, de créer et d'utiliser des bases de données SQL locales, etc.

Considérations liées à la conception et au développement d'applications mobiles

Les applications destinées aux périphériques mobiles à écran tactile diffèrent des applications de bureau et de navigateur de plusieurs manières :

- Pour permettre une manipulation simple par entrée tactile, les composants mobiles disposent généralement de zones actives plus étendues que dans les applications de bureau ou de navigateur.
- Les modèles d'interaction pour les actions telles que le défilement sont différents sur les périphériques à écran tactile.
- En raison de la surface limitée de l'écran, les applications mobiles sont conçues en général de façon à présenter à l'écran seulement une faible partie de l'interface utilisateur à la fois.
- La conception de l'interface utilisateur doit prendre en compte les différences de résolution d'écran entre les périphériques.

- Les performances d'UC et d'unité GPU sont plus limitées sur les téléphones et les tablettes que sur les périphériques de bureau.
- En raison de la mémoire limitée disponible sur les périphériques mobiles, les applications doivent être soucieuses d'économiser la mémoire.
- Les applications mobiles peuvent être fermées et redémarrées à tout moment, par exemple, lorsque le périphérique reçoit un appel ou un texto.

Par conséquent, la création d'une application pour un périphérique mobile ne se résume pas au redimensionnement d'une application de bureau pour un écran de plus petite taille. Flex vous permet de créer différentes interfaces utilisateur appropriées pour chaque facteur de forme, tout en partageant le code d'accès au modèle et aux données sous-jacent entre les projets mobiles, de navigateur et de bureau.

Restrictions liées à l'utilisation de Spark et des composants MX dans une application mobile

Utilisez le composant Spark défini lors de la création des applications mobiles dans Flex. Les composants Spark sont définis dans les composants Spark.* packages. Toutefois, pour des raisons de performance ou parce que tous les composants Spark n'ont pas d'habillage pour le thème Mobile, les applications mobiles ne prennent pas en charge la totalité des composants Spark.

A l'exception des contrôles graphiques MX et du contrôle MX Spacer, les applications mobiles ne prennent pas en charge l'ensemble de composants MX défini dans les packagesmx.*.

Le tableau ci-dessous répertorie les composants que vous pouvez utiliser, que vous ne pouvez pas utiliser ou qui requièrent un soin particulier pour être utilisés dans une application mobile :

Composant	**Composant**	**Utilisation dans une application mobile ?**	**Remarques**
Spark ActionBar Spark BusyIndicator Spark TabbedViewNavigator Spark TabbedViewNavigatorApplication	Spark View Spark ViewMenu Spark ViewNavigator Spark ViewNavigatorApplication	Oui	Ces nouveaux composants prennent en charge les applications mobiles.
Spark Button Spark CheckBox Spark DataGroup Spark Group/HGroup/VGroup/TileGroup Spark Image/BitmapImage Spark Label	Spark List Spark RadioButton/RadioButtonGroup Spark SkinnableContainer Spark Scroller Spark TextArea Spark TextInput	Oui	La plupart de ces composants disposent d'habillages pour le thème Mobile. Label, Image et BitmapImage peuvent être utilisés, même s'ils ne disposent pas d'un habillage mobile. Certains conteneurs de présentation Spark, comme Group et ses sous-classes, n'ont pas d'habillage. Par conséquent, vous pouvez les utiliser dans une application mobile.
Autres composants Spark habillables		Déconseillé	Les composants Spark habillables autres que ceux figurant dans la liste ci-dessus sont déconseillés car ils n'ont pas d'habillage pour le thème Mobile. Si le composant n'a pas d'habillage pour le thème Mobile, vous pouvez en créer un pour votre application.

Composant	Composant	Utilisation dans une application mobile ?	Remarques
Spark DataGrid	Spark RichEditableText Spark RichText	Déconseillé	Ces composants sont déconseillés pour des raisons de performance. Même si vous pouvez les utiliser dans une application mobile, cela risque d'affecter les performances. En ce qui concerne le contrôle DataGrid, les performances sont basées sur la quantité de données dont vous effectuez le rendu. Quant aux contrôles RichEditableText et RichText, leur performance est basée sur la quantité de texte et sur le nombre de contrôles présents dans l'application.
Composants MX autres que Spacer et les graphiques		Non	Les applications mobiles ne prennent pas en charge les composants MX tels que MX Button, CheckBox, List ou DataGrid. Ces composants correspondent aux composants Flex 3 dans les contrôles MX* et les conteneurs MX*.
MX Spacer		Oui	Spacer n'utilise pas d'habillage, de sorte qu'il peut être utilisé dans une application mobile.
Composants graphiques MX		Oui, mais avec des implications pour les performances.	Vous pouvez utiliser les contrôles graphiques MX, tels que AreaChart et BarChart, dans une application mobile. Les contrôles graphiques MX sont dans les graphiques MX.mx.*. Toutefois, leurs performances sur un périphérique mobile peuvent être amoindries selon la taille et le type des données graphiques. Par défaut, Flash Builder n'inclut pas les composants MX dans le chemin d'accès aux bibliothèques des projets mobiles. Pour utiliser les composants graphiques MX dans une application, ajoutez les fichiers mx.swc et charts.swc à votre chemin d'accès à la bibliothèque.

Les fonctionnalités Flex ci-dessous ne sont pas prises en charge dans les applications mobiles :

- Pas de prise en charge des opérations de glisser-déposer.
- Pas de prise en charge du contrôle ToolTip.
- Pas de prise de charge des bibliothèques RSL.

Considérations liées aux performances avec les applications mobiles

En raison des contraintes de performances qui pèsent sur les périphériques mobiles, certains aspects du développement d'applications mobiles diffèrent de ceux du développement d'applications de navigateur et de bureau. Les considérations liées aux performances comprennent :

- **Rédaction des rendus d'élément dans ActionScript**

  Pour les applications mobiles, le défilement dans les listes doit être aussi performant que possible. Rédigez des rendus d'élément dans ActionScript pour bénéficier de performances optimales. Il est possible de rédiger les rendus d'élément dans MXML, mais les performances de vos applications peuvent en souffrir.

  Flex fournit deux rendus d'élément qui sont optimisés à utiliser dans une application mobile : spark.components.LabelItemRenderer et spark.components.IconItemRenderer. Pour plus d'informations à propos de ces rendus d'élément, voir Using a mobile item renderer with a Spark list-based control.

  Pour plus d'informations sur la création de rendus d'élément personnalisés dans ActionScript, voir Custom Spark item renderers. Pour plus d'informations sur les différences entre les rendus d'élément mobiles et de bureau, voir Differences between mobile and desktop item renderers.

- **Utilisation d'ActionScript et de graphiques FXG compilés ou de bitmaps pour développer des habillages personnalisés**

  Les habillages mobiles livrés avec Flex sont rédigés dans ActionScript avec des graphiques FXG compilés afin d'offrir des performances optimales. Vous pouvez rédiger les habillages dans MXML, mais les performances de votre application peuvent en souffrir, selon le nombre de composants qui utilisent les habillages MXML. Pour des performances optimales, rédigez les habillages dans ActionScript et utilisez des graphiques FXG compilés. Pour plus d'informations, voir Habillage Spark et Graphiques FXG et MXML.

- **Utilisation de composants de texte indépendants de l'infrastructure Text Layout Framework (TLF)**

  Bon nombre des contrôles de texte Spark dépendent de TLF. L'utilisation de contrôles TLF dans une application mobile peut affecter les performances. Pour plus d'informations à propos de TLF, voir About the Spark text controls.

  Le contrôle Spark Label ne dépend pas de TLF. Les contrôles Spark TextInput et TextArea disposent d'habillages pour le thème mobile qui ne dépendent pas de TLF. Pour des résultats optimaux, utilisez les contrôles Label, TextInput et TextArea dans votre application, sauf lorsque vous rédigez des rendus d'élément personnalisés. Dans les rendus d'élément personnalisés, utilisez le contrôle StyleableTextField. Pour plus d'informations, voir Custom Spark item renderers.

  Les contrôles Spark RichText et RichEditableText dépendent de TLF. Vous pouvez utiliser ces contrôles pour afficher un contenu riche, mais leur utilisation risque de nuire aux performances.

- **Soyez prudent lorsque vous utilisez des composants graphiques MX dans une application mobile.**

  Vous pouvez utiliser les contrôles graphiques MX, tels que les contrôles AreaChart et BarChart, dans une application mobile. Toutefois, ils peuvent nuire aux performances selon la taille et le type des données graphiques.


Le blogueur Nahuel Foronda a créé une série d'articles sur Mobile ItemRenderer dans ActionScript.


Le blogueur Rich Tretola a créé une entrée de cookbook sur la création d'une liste avec un élément ItemRenderer pour une application mobile.

Chapitre 2 : Environnement de développement

Création d'une application Android dans Flash Builder

Voici un flux de travail général permettant de créer une application mobile Flex pour la plateforme Google Android. Ce flux de travail suppose que vous avez déjà conçu votre application mobile. Pour plus d'informations, voir « Conception d'une application mobile » à la page 1 .

Exigences liées à AIR

Les projets mobiles Flex et les projets mobiles ActionScript requièrent AIR 2.6. Vous pouvez exécuter des projets mobiles sur des périphériques physiques qui prennent en charge AIR 2.6. Vous pouvez installer AIR 2.6 sur des périphériques Android qui exécutent Android 2.2 ou version ultérieure.

Remarque : *si vous ne disposez pas d'un périphérique prenant en charge AIR 2.6, vous pouvez utiliser Flash Builder pour lancer et déboguer des applications mobiles sur le bureau.*

Chaque version du SDK Flex inclut la version requise d'Adobe AIR. Si vous avez installé des applications mobiles sur un périphérique exécutant une version antérieure du SDK Flex, désinstallez AIR du périphérique. Flash Builder installe la version correcte d'AIR lorsque vous exécutez ou déboguez une application mobile sur un périphérique.

Création d'une application

1 Dans Flash Builder, sélectionnez Fichier > Nouveau > Projet Flex Mobile.

Un projet Flex Mobile est un type particulier de projet AIR. Suivez les invites de l'assistant de nouveau projet comme vous le feriez pour tout autre projet AIR dans Flash Builder. Pour plus d'informations, voir Création de projets mobiles Flex.

Pour définir des préférences mobiles spécifiques à Android, voir « Définition des préférences de projet mobile » à la page 12.

Lorsque vous créez un projet Flex Mobile, Flash Builder génère les fichiers suivants pour le projet :

- *`ProjectName`*`.mxml`

  Le fichier d'application par défaut pour le projet.

  Par défaut, Flash Builder attribue à ce fichier le même nom qu'au projet. Si le nom du projet contient des caractères non autorisés par ActionScript, Flash Builder nomme ce fichier Main.mxml. Le fichier MXML contient la balise Spark Application de base du projet. La balise Spark Application peut être ViewNavigatorApplication ou TabbedViewNavigatorApplication.

  Généralement, vous n'ajoutez pas directement de contenu au fichier d'application par défaut, à l'exception du contenu ActionBar affiché dans toutes les vues. Pour ajouter du contenu au contrôle ActionBar, définissez les propriétés `navigatorContent`, `titleContent` ou `actionContent`.

- *`ProjectName`*`HomeView.mxml`

Le fichier représentant la vue initiale du projet. Flash Builder place le fichier dans un package de vues. L'attribut `firstView` de la balise ViewNavigatorApplication dans *`ProjectName`*`.mxml` spécifie ce fichier comme vue d'ouverture par défaut de l'application.

Pour plus d'informations sur la définition des vues, voir « Définition de vues dans une application mobile » à la page 30.

Vous pouvez aussi créer un projet mobile ActionScript seulement. Voir « Création d'un projet mobile ActionScript » à la page 11.

2 (Facultatif) Ajoutez du contenu au contrôle ActionBar du fichier de l'application principale.

Le contrôle ActionBar affiche le contenu et les fonctionnalités qui s'appliquent à l'application ou à la vue actuelle de l'application. Ajoutez ici le contenu que vous souhaitez afficher dans toutes les vues de l'application. Voir « Définition de contrôles de navigation, de titre et d'action dans une application mobile » à la page 45.

3 Disposez le contenu de la vue initiale de votre application.

Utilisez Flash Builder en mode Création ou en mode Source pour ajouter des composants à une vue.

Utilisez uniquement des composants pris en charge par Flash pour le développement mobile. En mode Création comme en mode Source, Flash Builder vous guide dans l'utilisation des composants pris en charge. Voir « Interface utilisateur et présentation » à la page 19

Dans la vue, ajoutez du contenu au contrôle ActionBar qui n'est visible que dans cette vue.

4 (Facultatif) Ajoutez toutes les vues que vous souhaitez inclure dans votre application.

Dans l'Explorateur de packages Flash Builder, sous le menu contextuelle correspondant au package de vues de votre projet, sélectionnez Nouveau composant MXML. L'assistant Nouveau composant MXML vous guide lors de la création de la vue.

Pour plus d'informations sur les vues, voir « Définition de vues dans une application mobile » à la page 30.

5 (Facultatif) Ajoutez des rendus d'élément optimisés pour les applications mobiles pour les composants List.

Adobe fournit IconItemRenderer, un rendu d'élément basé sur ActionScript à utiliser avec les applications mobiles. Voir Using a mobile item renderer with a Spark list-based control.

6 Configurez les configurations de lancement pour exécuter et déboguer l'application.

Vous pouvez exécuter ou déboguer l'application sur le bureau ou sur un périphérique.

Une configuration de lancement est requise pour exécuter ou déboguer une application à partir de Flash Builder. La première fois que vous exécutez ou déboguez une application mobile, Flash Builder vous invite à configurer une configuration de lancement.

Lors de l'exécution ou du débogage d'une application mobile sur un périphérique, Flash Builder installe l'application sur le périphérique.

Voir « Exécution et débogage des applications mobiles » à la page 111.

7 Exportez l'application en tant que package d'installation.

Utilisez Exporter vers une version validée pour créer des packages qui pourront être installés sur des périphériques mobiles. Flash Builder crée des packages pour la plateforme sélectionnée pour l'exportation. Voir « Exportation de packages Android APK pour publication » à la page 117.

Brent Arnold, expert Flex certifié par Adobe, a conçu les didacticiels vidéos suivants pour vous permettre de :

•Créer une application Flex mobile simple pour la plateforme Android

- Créer une application Flex mobile avec plusieurs vues
- Créer une application Flex mobile à l'aide d'un contrôle de listes utilisant Spark

Création d'une application iOS dans Flash Builder

Voici un flux de travail général permettant de créer une application mobile ActionScript pour la plateforme Apple iOS.

1 Avant de commencer à créer l'application mobile ActionScript, suivez la procédure répertoriée dans la section « Connexion de périphériques Apple iOS » à la page 18.

2 Dans Flash Builder, sélectionnez Fichier > Nouveau > Projet ActionScript Mobile.

Sélectionnez la plateforme cible Apple iOS et définissez les préférences de projet mobile. Pour plus d'informations sur la définition des préférences de projet mobile, voir « Définition des préférences de projet mobile » à la page 12.

Suivez les invites de l'assistant de nouveau projet comme vous le feriez dans tout autre assistant de création de projets dans Flash Builder. Pour plus d'informations, voir Création de projets mobiles ActionScript.

3 Rédigez le code de l'application dans le fichier d'application ActionScript principal.

4 Configurez les configurations de lancement pour exécuter et déboguer l'application. Vous pouvez exécuter ou déboguer l'application sur le bureau ou sur un périphérique connecté.

Pour plus d'informations, voir « Débogage d'une application sur un périphérique Apple iOS » à la page 115.

5 Exportez ou déployez l'application en tant qu'application de package iOS (IPA, iOS Package Application).

Pour plus d'informations, voir « Exportation de packages Apple iOS pour publication » à la page 118 et « Déploiement d'une application sur un périphérique Apple iOS » à la page 119.

Voir aussi

Création d'une application mobile (vidéo)

Création d'une application BlackBerry Tablet OS dans Flash Builder

Flash Builder 4.5.1 comprend un plug-in de Research In Motion (RIM) qui permet de créer et grouper des applications Flex et ActionScript pour BlackBerry® Tablet OS.

Création d'une application

Voici un flux de travail général permettant de créer des applications pour BlackBerry Tablet OS.

1 Avant de commencer à créer une application mobile, installez le SDK BlackBerry Tablet OS pour AIR depuis le site de développement d'applications BlackBerry Tablet OS.

Le SDK BlackBerry Tablet OS pour AIR fournit des API qui permettent de créer des applications Flex et ActionScript basées sur AIR.

Pour plus d'informations sur l'installation du SDK BlackBerry Tablet OS, voir le guide de prise en main BlackBerry Tablet OS.

2 Pour créer une application AIR basée sur Flex, dans Flash Builder, sélectionnez Fichier > Nouveau > Projet Flex Mobile.

   Suivez les invites de l'assistant de nouveau projet comme vous le feriez pour tout autre projet AIR dans Flash Builder. Veillez à sélectionner BlackBerry Tablet OS comme plateforme cible.

   Pour plus d'informations, voir Création de projets mobiles Flex.

3 Pour créer une application AIR basée sur ActionScript, dans Flash Builder, sélectionnez Fichier > Nouveau > Projet ActionScript Mobile.

   Suivez les invites de l'assistant de nouveau projet comme vous le feriez pour tout autre projet AIR dans Flash Builder. Veillez à sélectionner BlackBerry Tablet OS comme plateforme cible.

   Pour plus d'informations, voir Création de projets mobiles ActionScript.

Signature, groupement et déploiement d'une application

Pour plus d'informations sur la signature, le groupement et le déploiement de l'application, voir le guide de développement du SDK BlackBerry Tablet OS pour Adobe AIR par RIM.

Lisez également l'article relatif à l'utilisation de Flash Builder afin de grouper des applications pour les périphériques BlackBerry Tablet OS par Andrew Shorten, chef de produit Adobe.

Vous trouverez des ressources supplémentaires sur le développement BlackBerry Tablet OS provenant d'Adobe et de RIM sur la page Adobe Developer Connection.

Création d'un projet mobile ActionScript

Utilisez Flash Builder pour créer une application mobile ActionScript. L'application que vous créez est basée sur l'API d'Adobe AIR.

1 Sélectionnez Fichier > Nouveau > Projet mobile ActionScript.

2 Entrez un nom de projet et un emplacement. L'emplacement par défaut est l'espace de travail actuel.

3 Utilisez le SDK Flex 4.5 par défaut qui prend en charge le développement d'applications mobiles.

   Cliquez sur Suivant.

4 Sélectionnez les plateformes cibles pour votre application et spécifiez les paramètres de projet mobile pour chaque plateforme.

   Pour plus d'informations sur les paramètres de projet mobile, voir « Définition des préférences de projet mobile » à la page 12.

5 Cliquez sur Terminer ou sur Suivant pour indiquer d'autres options de configuration et chemins de génération.

   Pour plus d'informations sur les options de configuration de projet et les chemins de génération, voir Chemins de génération et autres options de configuration du projet.

Définition des préférences de projet mobile

Définition des configurations de périphériques

Flash Builder utilise des configurations de périphériques pour afficher des aperçus des tailles d'écran des périphériques dans la vue Création, ou pour lancer des applications sur le bureau à l'aide d'AIR Debug Launcher (ADL). Voir « Gestion des configurations de lancement » à la page 111.

Pour définir des configurations de périphériques, ouvrez la fenêtre Préférences et sélectionnez Flash Builder > Configuration des périphériques.

Flash Builder fournit plusieurs configurations de périphérique par défaut. Vous pouvez ajouter, modifier ou supprimer des configurations de périphérique supplémentaires. Vous ne pouvez pas modifier les configurations par défaut fournies par Flash Builder.

Cliquez sur le bouton Restaurer les valeurs par défaut pour rétablir les configurations de périphériques par défaut, sans supprimer les configurations que vous avez ajoutées. De plus, si vous avez ajouté une configuration de périphérique avec un nom correspondant à celui d'une configuration par défaut, Flash Builder remplace la configuration ajoutée par les paramètres par défaut.

Les configurations de périphérique contiennent les propriétés suivantes :

Propriété	Description
Nom du périphérique	Nom unique du périphérique.
Plateforme	Plateforme du périphérique. Sélectionnez une plateforme dans la liste des plateformes prises en charge.
Taille plein écran	Largeur et hauteur de l'écran du périphérique.
Taille d'écran utilisable	Taille standard d'une application sur le périphérique. Il s'agit de la taille prévue d'une application lancée en mode non-plein écran, en tenant compte du dispositif chrome du système, comme la barre d'état.
Pixels par pouce	Pixels par pouce sur l'écran du périphérique.

Choix des plateformes cibles

Flash Builder prend en charge les plateformes cibles en fonction du type d'application.

Pour sélectionner une plateforme, ouvrez la fenêtre Préférences et sélectionnez Flash Builder > Plateformes cibles.

Pour tous les plug-ins tiers, reportez-vous à la documentation correspondante.

Choix d'un modèle d'application

Lorsque vous créez une application mobile, vous pouvez faire un choix parmi les modèles d'application suivants :

Vide Utilise la balise Spark Application en tant qu'élément d'application de base.

Utilisez cette option pour créer une application personnalisée sans utiliser la navigation standard dans les vues.

Application basée sur une vue Utilise la balise Spark ViewNavigatorApplication en tant qu'élément d'application de base pour créer une application dotée d'une vue unique.

Vous pouvez spécifier le nom de la vue initiale.

Application à onglets Utilise la balise Spark TabbedViewNavigatorApplication comme élément d'application de base pour créer une application basée sur des onglets.

Pour ajouter un onglet, saisissez le nom de l'onglet et cliquez sur Ajouter. Vous pouvez modifier l'ordre des onglets en cliquant sur Haut et Bas. Pour supprimer un onglet d'une application, sélectionnez un onglet et cliquez sur Supprimer.

Le nom de la vue correspond au nom de l'onglet auquel est ajouté le mot « View ». Par exemple, si vous nommez un onglet FirstTab, Flash Builder génère une vue nommée FirstTabView.

Pour chaque onglet que vous créez, un nouveau fichier MXML est généré dans le package « Views ».

Remarque : *le nom du package n'est pas configurable à l'aide de l'assistant Projet Flex Mobile.*

Les fichiers MXML sont générés conformément aux règles suivantes :

- Si le nom de l'onglet est un nom de classe ActionScript valide, Flash Builder génère le fichier MXML en ajoutant le suffixe « View » au nom de l'onglet.
- Si le nom de l'onglet n'est pas un nom de classe valide, Flash Builder modifie le nom de l'onglet en supprimant les caractères non valides et en insérant des caractères initiaux valides. Si le nom modifié n'est pas acceptable, Flash Builder modifie le nom de fichier MXML en le remplaçant par « ViewN », où N correspond à la position de la vue, en commençant par N=1.


Brent Arnold, expert Flex certifié par Adobe, a créé un didacticiel vidéo sur l'utilisation du modèle d'application à onglets.

Choix des autorisations d'une application mobile

Lorsque vous créez une application mobile, vous pouvez spécifier ou modifier les droits par défaut d'une plateforme cible. Les autorisations sont spécifiées au moment de la compilation et ne peuvent pas être modifiées dans l'environnement d'exécution.

Commencez par sélectionner la plateforme cible, puis définissez les autorisations pour chaque plateforme, le cas échéant. Vous pourrez modifier les autorisations par la suite dans le fichier XML descripteur de l'application.

Des plug-ins tiers fournissent une prise en charge de plateformes supplémentaires pour les projets Flex et ActionScript. Pour les autorisations spécifiques aux plateformes, reportez-vous à la documentation du périphérique.

Autorisations pour la plateforme Google Android

Pour la plateforme Google Android, vous pouvez définir les autorisations suivantes :

INTERNET Autorise les demandes réseau et le débogage à distance.

L'autorisation INTERNET est sélectionnée par défaut. Si vous désélectionnez ce droit, vous ne pouvez pas déboguer votre application sur un périphérique.

WRITE_EXTERNAL_STORAGE Autorise l'écriture sur un périphérique externe.

Sélectionnez ce droit pour permettre à l'application d'écrire sur une carte mémoire externe du périphérique.

READ_PHONE_STATE Coupe le son au cours d'un appel entrant.

Sélectionnez ce droit pour permettre à l'application de couper le son au cours des appels téléphoniques. Par exemple, vous pouvez sélectionner ce droit si votre application lit les sons en arrière-plan.

ACCESS_FINE_LOCATION Autorise l'accès à un emplacement GPS.

Sélectionnez ce droit pour permettre à l'application d'accéder aux données GPS à l'aide de la classe Geolocation.

DISABLE_KEYGUARD and WAKE_LOCK Interdit la mise en veille du périphérique.

Sélectionnez ce droit pour empêcher le périphérique de se mettre en veille à l'aide des paramètres de la classe SystemIdleMode.

CAMERA Permet d'accéder à une caméra.

Sélectionnez ce droit pour permettre à l'application d'accéder à une caméra.

RECORD_AUDIO Permet d'accéder à un microphone.

Sélectionnez ce droit pour permettre à l'application d'accéder à un microphone.

ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE Autorise l'accès aux informations concernant les interfaces réseau associées au périphérique.

Sélectionnez ce droit pour permettre à l'application d'accéder aux informations réseau à l'aide de la classe NetworkInfo.

Pour plus d'informations sur la définition des propriétés des applications mobiles, voir la Documentation Adobe AIR.

Autorisations pour la plateforme Apple iOS

La plateforme Apple iOS utilise la validation dans l'environnement d'exécution pour les autorisations au lieu des autorisations prédéfinies. En d'autres termes, si une application souhaite accéder à une fonction spécifique de la plateforme Apple iOS qui requiert des droits d'utilisateur, une fenêtre contextuelle apparaît, demandant l'autorisation.

Choix des paramètres de plateforme

Les paramètres de plateforme vous permettent de sélectionner une gamme de périphériques cibles. Selon la plateforme sélectionnée, vous pouvez choisir le périphérique cible ou une gamme de périphériques cibles. Vous pouvez sélectionner un périphérique spécifique ou tous les périphériques pris en charge par la plateforme.

Des plug-ins tiers fournissent une prise en charge de plateformes supplémentaires pour les projets Flex et ActionScript. Pour les autorisations spécifiques aux plateformes, reportez-vous à la documentation du périphérique.

Paramètres de plateforme pour la plateforme Google Android

il n'existe pas de paramètres spécifiques à la plateforme Google Android.

Paramètres de plateforme pour la plateforme Apple iOS

Pour un projet mobile Flex ou un projet mobile ActionScript, vous pouvez spécifier les périphériques cibles suivants pour la plateforme Apple iOS :

iPhone/iPod Touch Les applications utilisant cette gamme cible sont répertoriées comme compatibles uniquement avec les périphériques iPhone et iPod Touch dans le magasin Apple App.

iPad Les applications utilisant cette gamme cible sont répertoriées comme compatibles uniquement avec les périphériques iPad dans le magasin Apple App.

Tous Les applications utilisant cette gamme cible sont répertoriées comme compatibles avec les périphériques iPhone/iPod Touch et iPad dans l'App Store d'Apple. Il s'agit de l'option par défaut.

Choix des paramètres d'application

Réorientation automatique Fait pivoter l'application lorsque l'utilisateur tourne le périphérique. Lorsque ce paramètre n'est pas activé, votre application apparaît toujours dans la même orientation.

Plein écran Affiche l'application en mode plein écran sur le périphérique. Lorsque ce paramètre est activé, la barre d'état du périphérique n'apparaît pas au-dessus de l'application. Votre application remplit tout l'écran.

Si vous souhaitez cibler votre application vers plusieurs types de périphériques aux densités d'écran variables, sélectionnez Redimensionner automatiquement l'application pour différentes densités d'écran. La sélection de cette option redimensionne automatiquement l'application et gère les changements de densité des périphériques, le cas échéant. Voir « Définition du redimensionnement d'application » à la page 15.

Définition du redimensionnement d'application

Vous utilisez le redimensionnement d'application mobile pour créer une application mobile unique compatible avec des périphériques de taille d'écran et de densité différentes.

Les écrans des périphériques mobiles possèdent des densités d'écran, ou valeurs PPP(points par pouce), variables. Vous pouvez spécifier une valeur de PPP de 160, 240 ou 320, selon la densité d'écran du périphérique cible. Lorsque vous activez le redimensionnement automatique, Flex optimise la façon dont il affiche l'application pour la densité d'écran de chaque périphérique.

Par exemple, supposez que vous spécifiez une valeur PPP cible de 160 et activez le redimensionnement automatique. Lorsque vous exécutez l'application sur un périphérique doté d'une valeur PPP de 320, Flex redimensionne automatiquement l'application par un facteur 2. En d'autres termes, Flex agrandit tout de 200 %.

Pour spécifier la valeur PPP cible, définissez-la comme propriété `applicationDPI` de la balise `<s:ViewNavigatorApplication>` ou `<s:TabbedViewNavigatorApplication>` dans le fichier de l'application principale :

```
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.HomeView"
    applicationDPI="160">
```

Si vous choisissez de ne pas utiliser le redimensionnement automatique de votre application, vous devez gérer manuellement les changements de densité pour votre présentation, selon les besoins. Toutefois, Flex adapte les habillages à la densité de chaque périphérique.

Pour plus d'informations à propos de la création d'applications mobiles indépendamment de la densité, voir « Prise en charge de plusieurs tailles d'écran et valeurs PPP dans une application mobile » à la page 71.

Connexion des périphériques Google Android

Vous pouvez connecter un périphérique Google Android à votre ordinateur de développement pour visualiser ou déboguer l'application sur le périphérique Android.

Périphériques Android pris en charge

Les projets mobiles Flex et les projets mobiles ActionScript requièrent AIR 2.6. Vous pouvez exécuter ou déboguer des projets mobiles seulement sur des périphériques physiques qui prennent en charge AIR 2.6. Vous pouvez installer AIR 2.6 sur des périphériques Android qui exécutent Android 2.2 ou version ultérieure.

Configuration de périphériques Android

Pour exécuter et déboguer les applications Flex mobiles à partir d'un périphérique Android, activez le débogage USB comme indiqué ci-dessous :

1 Sur le périphérique, procédez comme suit pour vous assurer que le débogage USB est activé :

   a Appuyez sur le bouton Accueil pour afficher l'écran d'accueil.

   b Accédez à Paramètres, puis sélectionnez Applications > Développement.

   c Activez le débogage USB.

2 Connectez le périphérique à votre ordinateur à l'aide d'un câble USB.

3 Etendez vers le bas la zone de notification située en haut de l'écran. Vous devez voir une indication de type USB connecté ou Connexion USB.

   a Appuyez sur USB connecté ou sur Connexion USB.

   b Si un ensemble d'options apparaît, parmi lesquelles le mode Charge seulement, sélectionnez le mode Charge seulement et appuyez sur OK.

   c Si vous voyez un bouton permettant de désactiver le mode stockage de masse, cliquez dessus pour désactiver le stockage de masse.

4 (Windows uniquement) Installez le pilote USB approprié à votre périphérique. Voir « Installation de pilotes de périphérique USB pour les périphériques Android (Windows) » à la page 16.

5 Etirez vers le bas la zone de notification situe en haut de l'écran.

   Si le débogage USB n'apparaît pas parmi les entrées, vérifiez le mode USB comme décrit à l'étape 3 ci-dessus. Assurez-vous que le mode USB n'est pas défini sur Mode PC.

Remarque : *une configuration supplémentaire est nécessaire pour le débogage. Voir « Exécution et débogage d'une application mobile sur un périphérique » à la page 112.*

Installation de pilotes de périphérique USB pour les périphériques Android (Windows)

Pilotes de périphérique et configurations

Les plateformes Windows requièrent l'installation d'un pilote USB pour connecter un périphérique Android à votre ordinateur de développement. Flash Builder fournit un pilote de périphérique et une configuration pour plusieurs périphériques Android.

Ces configurations de pilotes de périphérique sont répertoriées dans le fichier `android_winusb.inf`. Windows Device Manager accède à ce fichier au cours de l'installation du pilote de périphérique. Flash Builder installe `android_winusb.inf` à l'emplacement suivant :

```
<Adobe Flash Builder 4.5 Home>\utilities\drivers\android\android_winusb.inf
```

Pour obtenir la liste complète des périphériques pris en charge, voir Périphériques certifiés. Dans le cas de périphériques Android ne figurant pas dans la liste, vous pouvez mettre à jour le fichier `android_winusb.inf` afin d'y ajouter les pilotes USB. Voir « Ajout de configurations de pilotes de périphérique USB Android » à la page 17.

Installation d'un pilote de périphérique USB

1 Connectez votre périphérique Android au port USB de votre ordinateur.

2 Accédez à l'emplacement suivant :

```
<Flash Builder>/utilities/drivers/android/
```

Installez le pilote USB à l'aide de l'Assistant Ajout de nouveau matériel détecté de Windows ou du Gestionnaire de périphériques Windows.

Ajout de configurations de pilotes de périphérique USB Android

Si vous avez un périphérique Android pris en charge qui ne figure pas dans la section « Installation de pilotes de périphérique USB pour les périphériques Android (Windows) » à la page 16, mettez à jour le fichier `android_winusb.inf` pour inclure le périphérique.

1. Branchez le périphérique à un port USB de votre ordinateur. Windows vous indique que le pilote est introuvable.
2. A l'aide du Gestionnaire de périphériques Windows, ouvrez l'onglet Détails des propriétés du périphérique.
3. Sélectionnez la propriété ID de matériel pour afficher l'ID du matériel.
4. Ouvrez le fichier `android_winusb.inf` dans un éditeur de texte. Recherchez `android_winusb.inf` à l'emplacement suivant :

   ```
   <Adobe Flash Builder 4.5 Home>\utilities\drivers\android\android_winusb.inf
   ```

5. Notez les éléments qui apparaissent dans le fichier relatif à votre architecture : `[Google.NTx86]` ou `[Google.NTamd64]`. Les listes contiennent un commentaire descriptif et une ou plusieurs lignes dotées de l'ID de matériel, comme cela est illustré ici :

   ```
   . . .
   [Google.NTx86]
   ; HTC Dream
   %CompositeAdbInterface% = USB_Install, USB\VID_0BB4&PID_0C02&MI_01
   . . .
   ```

6. Copiez et collez un commentaire et une description de matériel. Pour le pilote de périphérique que vous souhaitez ajouter, modifiez la liste comme suit :

   a. Pour le commentaire, indiquez le nom du périphérique.

   b. Remplacez l'ID de matériel par celui qui a été identifié à l'étape 3 ci-dessus.

   Par exemple :

   ```
   . . .
   [Google.NTx86]
   ; NEW ANDROID DEVICE
   %CompositeAdbInterface%     = USB_Install, NEW HARDWARE ID
   . . .
   ```

7. Utilisez le gestionnaire de périphériques Windows pour installer le périphérique, comme décrit dans « Installation de pilotes de périphérique USB pour les périphériques Android (Windows) » à la page 16 ci-dessus.

   Au cours de l'installation, Windows affiche un avertissement indiquant que le pilote provient d'un éditeur inconnu. Toutefois, le pilote permet à Flash Builder d'accéder à votre périphérique.

   Important : *si Windows ne reconnaît toujours pas le périphérique, vous devez installer le pilote USB approprié disponible auprès du fabricant du périphérique. Voir la page relative aux fabricants de pilotes USB pour obtenir des liens vers les sites Web de plusieurs fabricants de périphériques depuis lesquels vous pouvez télécharger le pilote USB approprié pour votre périphérique.*

Brent Arnold, expert Flex certifié par Adobe, partage ses astuces qui permettent à votre périphérique Android d'être reconnu par Windows.

Connexion de périphériques Apple iOS

Vous pouvez connecter un périphérique Apple iOS à votre ordinateur de développement pour déboguer ou déployer l'application sur le périphérique iOS.

Périphériques iOS pris en charge

Flash Builder fournit des configurations de périphérique pour les périphériques Apple iPhone, iTouch et iPad.

Ouvrez la fenêtre Préférences et sélectionnez Flash Builder > Configuration des périphériques pour afficher les configurations de périphérique pour tous les périphériques Apple pris en charge.

Préparation à l'installation et au déploiement d'une application sur un périphérique iOS

Connectez-vous au site du Pôle de développement iOS à l'aide de votre ID de compte de développeur iPhone, puis procédez comme suit :

1. Créez un fichier de demande de signature de certificat. Ce fichier permet d'obtenir un certificat de développement iPhone.

   Pour plus d'informations, voir Génération d'une demande de signature de certificat.

2. Générez un certificat de développement Apple (extension de nom de fichier .cer) et un fichier de profil d'approvisionnement (extension de fichier .mobileprovision). Suivez les instructions fournies dans le Pôle de développement iOS.

   Lors de la création du profil d'approvisionnement, répertoriez les ID des périphériques sur lesquels vous souhaitez installer l'application.

   Pour plus d'informations, voir la documentation Apple destinée aux développeurs.

3. Convertissez le certificat de développement Apple au format de certificat P12.

   Pour plus d'informations sur la conversion du certificat de développement Apple au format de certificat P12, voir Conversion d'un certificat de développement en fichier P12.

Chapitre 3 : Interface utilisateur et présentation

Présentation d'une application mobile

Utilisation des vues et des sections pour présenter une application mobile

Une application mobile comporte un ou plusieurs écrans, ou *vues*. Par exemple, une application mobile peut posséder trois vues :

1 Une vue d'accueil qui vous permet d'ajouter des coordonnées

2 Une vue de contacts contenant la liste des contacts existants

3 Une vue de recherche permettant d'effectuer des recherches dans votre liste de contacts

Application mobile simple

L'image suivante présente l'écran principal d'une application mobile simple créée dans Flex :

***A.** Contrôle ActionBar **B.** Zone de contenu*

Cette figure présente les zones principales d'une application mobile :

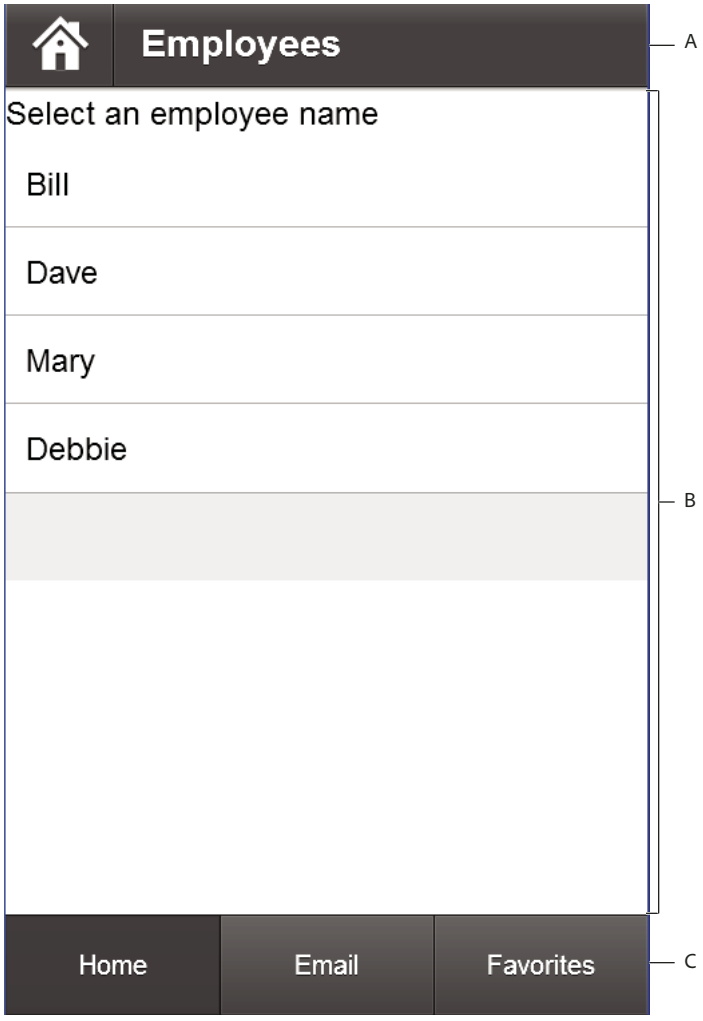
Contrôle ActionBar Le contrôle ActionBar vous permet d'afficher des informations contextuelles à propos de l'état actuel de l'application. Ces informations comprennent une zone de titre, une zone réservée aux contrôles permettant de naviguer dans l'application et une zone réservée aux contrôles permettant de réaliser une action. Vous pouvez ajouter un contenu commun dans le contrôle ActionBar qui s'applique à l'ensemble de l'application, et vous pouvez ajouter des éléments spécifiques à une vue individuelle.

Zone de contenu La zone de contenu affiche les écrans individuels, ou *vues*, qui constituent l'application. Les utilisateurs naviguent dans les vues de l'application en utilisant les composants intégrés à l'application et les contrôles d'entrée du périphérique mobile.

Une application mobile comprenant des sections

Une application plus complexe pourrait définir plusieurs zones, ou *sections*, de l'application. Par exemple, l'application pourrait comprendre une section contacts, une section e-mail, une section favoris et d'autres sections. Chaque section de l'application contient une ou plusieurs vues. Les vues individuelles peuvent être partagées entre les sections, ce qui vous évite d'avoir à définir la même vue à plusieurs reprises.

La figure ci-dessous présente une application mobile qui inclut une barre d'onglets au bas de la fenêtre de l'application :



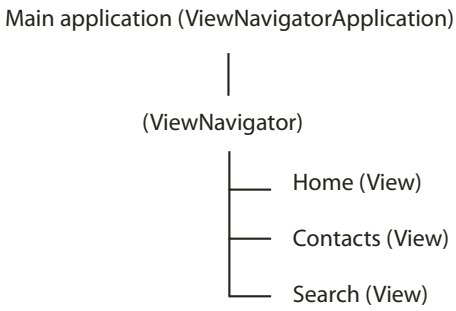
***A.** Contrôle ActionBar**B.** Zone de contenu**C.** Barre d'onglets*

Flex utilise le contrôle ButtonBarBase pour implémenter la barre d'onglets. Chaque bouton de la barre d'onglets correspond à une section différente. Sélectionnez un bouton dans la barre d'onglets pour modifier la section actuelle.

Chaque section de l'application définit son propre contrôle ActionBar. Par conséquent, la barre d'onglets est commune à l'ensemble de l'application, alors que le contrôle ActionBar est spécifique à chaque section.

Présentation d'une application mobile simple

La figure suivante présente l'architecture d'une application mobile simple :



La figure présente une application composée de quatre fichiers. Une application mobile contient un fichier de l'application principale et un fichier pour chaque vue. Il n'existe pas de fichier distinct pour le conteneur ViewNavigator. Le conteneur ViewNavigatorApplication le crée.

Remarque : *ce schéma présente l'architecture de l'application, mais ne représente pas l'application en cours d'exécution. Lors de l'exécution, une seule vue est active et résidente dans la mémoire. Pour plus d'informations, voir «* *Navigation parmi les vues d'une application mobile » à la page 23.*

Classes utilisées dans une application mobile

Utilisez les classes suivantes pour définir une application mobile :

Classe	Description
ViewNavigatorApplication	Définit le fichier de l'application principale. Le conteneur ViewNavigatorApplication n'accepte pas d'enfants.
ViewNavigator	Contrôle la navigation parmi les vues d'une application. Le conteneur ViewNavigator crée également le contrôle ActionBar. Le conteneur ViewNavigatorApplication crée automatiquement un seul conteneur ViewNavigator pour l'ensemble de l'application. Utilisez les méthodes du conteneur ViewNavigator pour basculer entre les différentes vues.
Vue	Définit les vues de l'application, chaque vue étant définie dans un fichier MXML ou ActionScript distinct. Une instance du conteneur View représente chaque vue de l'application. Définissez chaque vue dans un fichier MXML ou ActionScript distinct.

Utilisez le conteneur ViewNavigatorApplication pour définir le fichier d'application principal, comme l'illustre l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSectionSimple.mxml -->
<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 seul objet ViewNavigator qui définit le contrôle ActionBar. Vous utilisez le conteneur ViewNavigator pour naviguer parmi les vues de l'application.

Ajout d'un conteneur View à une application mobile

Chaque application mobile comporte au moins une vue. Même si le fichier de l'application principale crée le conteneur ViewNavigator, il ne définit aucune des vues utilisées dans l'application.

Chaque vue d'une application correspond à un conteneur View défini dans un fichier ActionScript ou MXML. Chaque vue contient une propriété `data` qui spécifie les données associées à cette vue. Les vues peuvent utiliser la propriété `data` pour échanger des informations entre elles tandis que l'utilisateur navigue dans l'application.

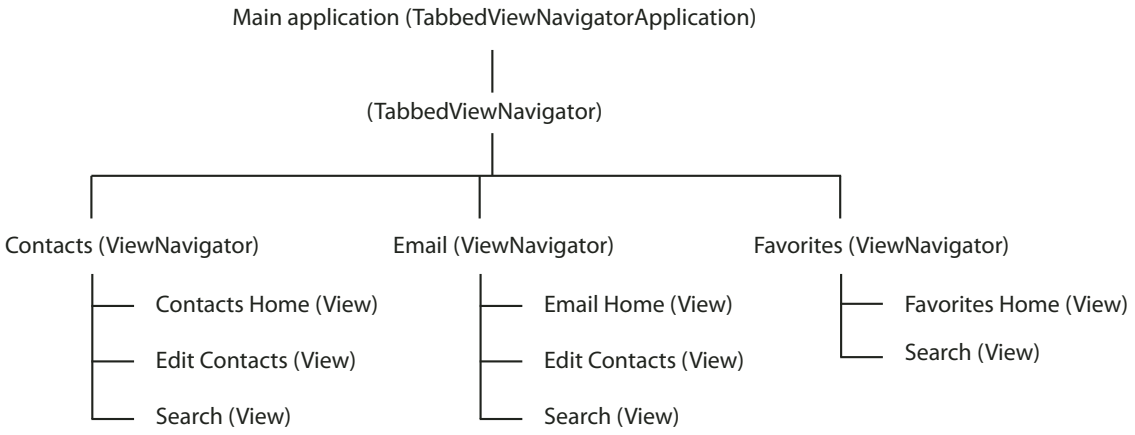
Utilisez la propriété `ViewNavigatorApplication.firstView` pour spécifier le fichier qui définit la première vue de l'application. Dans l'application précédente, la propriété `firstView` spécifie `views.HomeView`. L'exemple suivant présente le fichier HomeView.mxml qui définit cette vue :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\HomeView.mxml -->
<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>
```

Le blogueur David Hassoun a communiqué sur les bases de ViewNavigator.

Présentation d'une application mobile comprenant plusieurs sections.

Une application mobile peut collecter des vues associées dans différentes sections de l'application. Par exemple, la figure suivante présente l'organisation d'une application mobile comprenant trois sections.



N'importe quelle section peut utiliser n'importe quelle vue. En d'autres termes, une vue n'appartient pas à une section particulière. La section définit simplement une manière d'organiser un ensemble de vues et de naviguer parmi elles. Dans la figure, la vue Search fait partie de chaque section de l'application.

Au moment de l'exécution, une seule vue est active et résidente dans la mémoire. Pour plus d'informations, voir « Navigation parmi les vues d'une application mobile » à la page 23.

Classes utilisées dans une application mobile comprenant plusieurs sections

Le tableau suivant répertorie les classes que vous utilisez pour créer une application mobile comprenant plusieurs sections :

Classe	Description
TabbedViewNavigatorApplication	Définit le fichier d'application principal. Le seul enfant autorisé du conteneur TabbedViewNavigatorApplication est ViewNavigator. Définissez un conteneur ViewNavigator pour chaque section de l'application.
TabbedViewNavigator	Contrôle la navigation parmi les sections qui constituent l'application. Le conteneur TabbedViewNavigatorApplication crée automatiquement un seul conteneur TabbedViewNavigator pour l'ensemble de l'application. Le conteneur TabbedViewNavigator crée la barre d'onglets que vous utilisez pour naviguer parmi les sections.
ViewNavigator	Définissez un conteneur ViewNavigator pour chaque section. Le conteneur ViewNavigator contrôle la navigation parmi les vues qui constituent la section. Il crée également le contrôle Actionbar pour la section.
Vue	Définit les vues de l'application. Une instance du conteneur View représente chaque vue de l'application. Définissez chaque vue dans un fichier MXML ou ActionScript distinct.

Une application mobile divisée en plusieurs sections contient un fichier de l'application principale et un fichier qui définit chacune des vues. Utilisez le conteneur TabbedViewNavigatorApplication pour définir le fichier d'application principal, comme l'illustre l'exemple suivant :

```
<?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>
```

Utilisation de ViewNavigator dans une application à plusieurs sections

Le seul composant enfant accepté pour le conteneur TabbedViewNavigatorApplication est ViewNavigator. Chaque section de l'application correspond à un conteneur ViewNavigator différent.

Utilisez le conteneur ViewNavigator pour naviguer parmi 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 de la section.

Utilisation de TabbedViewNavigator dans une application à plusieurs sections

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'êtes pas tenu d'ajouter une logique à l'application pour naviguer parmi les sections.

Navigation parmi les vues d'une application mobile

Une pile d'objets View contrôle la navigation dans une application mobile. L'objet View supérieur de la pile définit la vue actuellement visible.

Le conteneur ViewNavigator gère la pile. Pour changer de vue, poussez un nouvel objet View sur la pile ou retirez l'objet View actuel de la pile. Le fait de retirer de la pile l'objet View actuellement visible détruit l'objet View et ramène l'utilisateur à la vue précédente sur la pile.

Dans une application dotée de sections, utilisez la barre d'onglets pour naviguer parmi les sections. Comme un conteneur ViewNavigator différent définit chaque section, changer de section revient à changer le conteneur ViewNavigator et la pile actuels. L'objet View situé au sommet de la pile du nouveau conteneur ViewNavigator devient la vue actuelle.

Pour économiser la mémoire, par défaut, le conteneur ViewNavigator s'assure qu'une seule vue est en mémoire à la fois. Toutefois, il conserve les données des vues précédentes de la pile. Par conséquent, lorsque l'utilisateur revient à la vue précédente, celle-ci peut être réinstanciée avec les données appropriées.

Remarque : *le conteneur View définit la propriété* `destructionPolicy`*. S'il est défini sur* `auto`*, la valeur par défaut, le conteneur ViewNavigator détruit la vue lorsqu'elle n'est pas active. S'il est défini sur* `none`*, la vue est mise en mémoire cache.*

Le blogueur Mark Lochrie a communiqué sur Flash Builder 4.5 ViewNavigator.

Méthodes de navigation ViewNavigator

Utilisez les méthodes suivantes de la classe ViewNavigator pour contrôler la navigation :

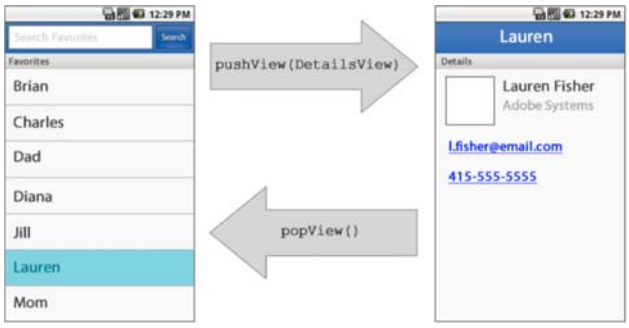
pushView() Poussez un objet View sur la pile. L'objet View communiqué en tant qu'argument à la méthode `pushView()` devient la vue actuelle.

popView() Retirez l'objet View actuel de la pile de navigation et détruisez l'objet View. L'objet View précédent sur la pile devient la vue actuelle.

popToFirstView() Retirez tous les objets View de la pile et détruisez-les, à l'exception du premier objet View de la pile. Le premier objet View de la pile devient la vue actuelle.

popAll() Videz la pile du conteneur ViewNavigator et détruisez tous les objets View. Votre application affiche une vue vide.

La figure suivante présente deux vues. Pour changer la vue actuelle, utilisez la méthode `ViewNavigator.pushView()` pour pousser un objet View qui représente la nouvelle vue sur la pile. La méthode `pushView()` invite le conteneur ViewNavigator à basculer l'affichage vers le nouvel objet View.



Installation et élimination d'objets View pour changer de vue.

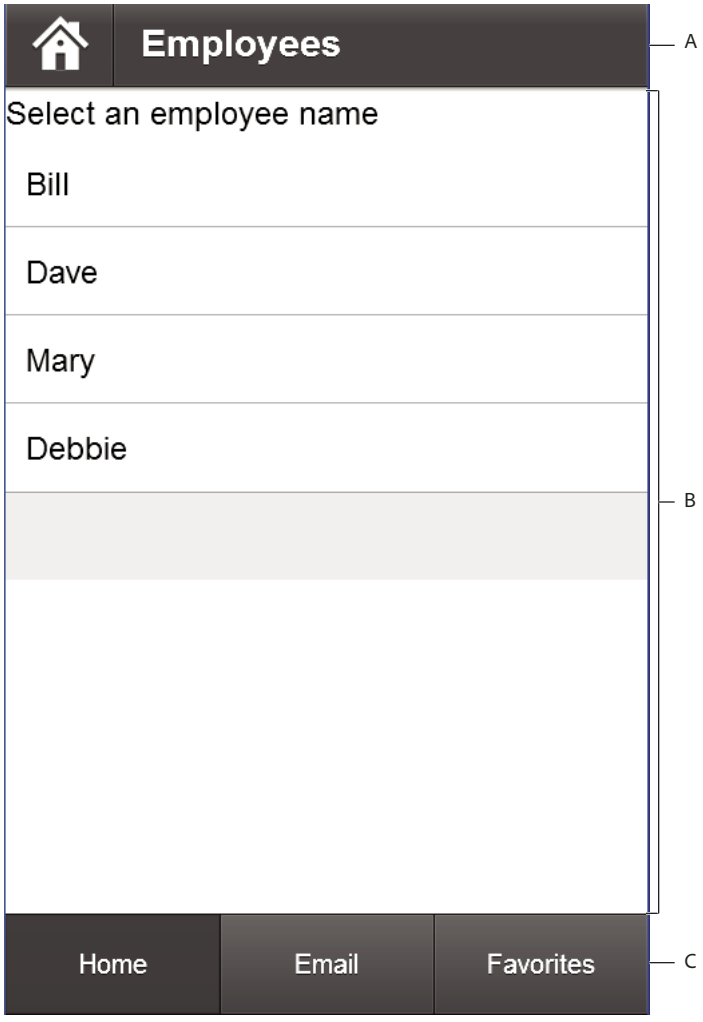
Utilisez la méthode `ViewNavigator.popView()` pour éliminer l'objet View actuel de la pile. Le conteneur ViewNavigator ramène l'affichage à l'objet View précédent sur la pile.

Remarque : *le périphérique mobile lui-même contrôle en grande partie la navigation dans une application mobile. Par exemple, les applications mobiles intégrées à Flex gèrent automatiquement le bouton retour sur les périphériques mobiles. Par conséquent, vous n'avez pas à ajouter la prise en charge du bouton retour dans l'application. Lorsque l'utilisateur appuie sur le bouton retour du périphérique mobile, Flex appelle automatiquement la méthode* `popView()` *pour restaurer la vue précédente.*

Le blogueur David Hassoun a communiqué surla gestion des données dans une vue.

Définition de la navigation pour une application à plusieurs sections

Dans la figure suivante, les vues sont organisées en plusieurs sections. Un conteneur ViewNavigator différent définit chaque section. Chaque section contient une ou plusieurs vues :



***A.** ActionBar **B.** Zone de contenu **C.** Barre d'onglets*

Pour modifier la vue dans la section en cours, qui correspond au conteneur ViewNavigator actuel, utilisez les méthodes `pushView()` et `popView()`.

Pour modifier la section actuelle, utilisez la barre d'onglets. Lorsque vous changez de section, vous basculez vers le conteneur ViewNavigator de la nouvelle section. L'affichage change et présente l'objet View qui se trouve actuellement au-dessus de la pile du nouveau ViewNavigator.

Vous pouvez aussi programmer le changement de section à l'aide de la propriété `TabbedViewNavigator.selectedIndex`. Cette propriété contient l'index de base 0 du navigateur de vues sélectionné.

Gestion des saisies de l'utilisateur dans une application mobile

La saisie utilisateur requiert une gestion différente dans une application mobile par rapport à une application de bureau ou de navigateur. Dans une application de bureau créée pour AIR ou dans une application de navigateur conçue pour Flash Player, les principaux dispositifs de saisie sont une souris et un clavier. Dans le cas de périphériques mobiles, le dispositif de série principal est un écran tactile. Un périphérique mobile possède souvent un type de clavier et certains périphériques disposent également d'une méthode de saisie à cinq directions (gauche, droite, haut, bas et sélection).

La classe mx.core.UIComponent définit la propriété de style `interactionMode` que vous utilisez pour configurer des composants pour le type de saisie utilisé par l'application. Pour les thèmes Halo et Spark, la valeur par défaut est `mouse`, ce qui indique que la souris est le dispositif de saisie principal. Pour le thème Mobile, la valeur par défaut est `touch`, ce qui indique que l'écran tactile constitue le dispositif de saisie principal.

Prise en charge des touches du matériel

Les applications définies par les conteneurs ViewNavigatorApplication ou TabbedViewNavigatorApplication répondent aux touches matérielles Retour et Menu d'un périphérique. Lorsque l'utilisateur appuie sur la touche Retour, l'application revient à la vue précédente. S'il n'y a pas de vue précédente, l'application se ferme et l'écran d'accueil du périphérique apparaît.

Lorsque l'utilisateur appuie sur le bouton Retour, la vue active de l'application reçoit un événement `backKeyPressed`. Vous pouvez annuler l'action de la touche Retour en appelant `preventDefault()` dans le gestionnaire d'événement pour l'événement `backKeyPressed`.

Lorsque l'utilisateur appuie sur le bouton Menu, le conteneur ViewMenu de la vue actuelle apparaît, s'il est défini. Le conteneur ViewMenu définit un menu situé au bas du conteneur View. Chaque conteneur View définit son propre menu, spécifique à cette vue.

Le contrôle View actif envoie un événement `menuKeyPressed` lorsque l'utilisateur appuie sur la touche Menu. Pour annuler l'action du bouton Menu et empêcher l'affichage de ViewMenu, appelez la méthode `preventDefault()` dans le gestionnaire d'événement pour l'événement `menuKeyPressed`.

Pour plus d'informations, voir « Définition de menus dans une application mobile » à la page 53.

Gestion des événements de souris et tactiles dans une application mobile

AIR génère différents événements pour indiquer différents types de saisie. Ces événements comprennent :

Evénements de souris Evénements générés par une intervention de l'utilisateur effectuée au moyen d'une souris ou d'un écran tactile. Les événements de souris comprennent `mouseOver`, `mouseDown` et `mouseUp`.

Evénements tactiles Evénements générés sur des périphériques qui détectent les contacts de l'utilisateur avec le périphérique, par exemple le contact d'un doigt sur un écran tactile. Les événements tactiles comprennent `touchTap`, `touchOver` et `touchMove`. Lorsqu'un utilisateur utilise un périphérique équipé d'un écran tactile, il touche généralement l'écran du doigt ou à l'aide d'un dispositif de pointage.

Evénements de mouvement Evénements générés par des interactions tactiles multipoint, telles que la pression de deux doigts simultanément sur un écran tactile. Les événements de mouvement comprennent `gesturePan`, `gestureRotate` et `gestureZoom`. Par exemple, sur certains périphériques, il est possible d'effectuer un geste de pincement pour produire un zoom arrière dans une image.

Prise en charge intégrée des événements de souris

La structure Flex et le jeu de composants Flex intègrent la prise en charge des événements de souris, mais pas des événements tactiles ni de mouvement. Par exemple, l'utilisateur interagit avec les composants Flex dans une application mobile en utilisant l'écran tactile. Les composants répondent aux événements de souris, tels que `mouseDown` et `mouseOver`, mais pas aux événements tactiles ni de mouvement.

Par exemple, l'utilisateur appuie sur l'écran tactile pour sélectionner le contrôle Flex Button. Le contrôle Button utilise les événements `mouseUp` et `mouseDown` pour signaler que l'utilisateur est intervenu sur le contrôle. Le contrôle Scroller utilise les événements `mouseMove` et `mouseUp` pour indiquer que l'utilisateur défile dans l'écran.

Paul Trani, fameux développeur Adobe, explique la gestion des événements tactiles et gestuels dans le document Touch Events and Gesture on Mobile.

Contrôle des événements générés par AIR

La propriété `flash.ui.Multitouch.inputMode` contrôle les événements générés par AIR et Flash Player. La propriété `flash.ui.Multitouch.inputMode` peut posséder l'une des valeurs suivantes :

- `MultitouchInputMode.NONE` AIR traite les événements de souris, mais pas les événements tactiles ni les gestes.
- `MultitouchInputMode.TOUCH_POINT` AIR traite les événements de souris et tactiles, mais pas les gestes. Dans ce mode, la structure Flex reçoit les mêmes événements de souris que pour `MultitouchInputMode.NONE`.
- `MultitouchInputMode.GESTURE` AIR traite les événements de souris et les gestes, mais pas les événements tactiles. Dans ce mode, la structure Flex reçoit les mêmes événements de souris que pour `MultitouchInputMode.NONE`.

Comme le montre la liste, quelle que soit la valeur paramétrée de la propriété `flash.ui.Multitouch.inputMode`, AIR envoie toujours les événements de souris. Par conséquent, les composants Flex peuvent toujours répondre aux interactions de l'utilisateurs réalisées à l'aide d'un écran tactile.

Flex vous permet d'utiliser n'importe quelle valeur pour la propriété `flash.ui.Multitouch.inputMode` dans votre application. Par conséquent, même si les composants Flex ne répondent pas aux événements tactiles ni de mouvement, vous pouvez ajouter une fonctionnalité à votre application pour répondre à tout événement. Par exemple, vous pouvez ajouter un gestionnaire d'événement au contrôle Button afin de gérer les événements tactiles, tels que `touchTap`, `touchOver` et `touchMove`.

Le Guide du développeur ActionScript 3.0 présente une vue d'ensemble de la gestion de la saisie utilisateur sur différents périphériques et de l'utilisation de la saisie tactile, tactile multipoint et basée sur le mouvement. Pour plus de détails, voir:

- Notions de base sur l'interaction utilisateur
- Saisie tactile, multitouch et gestes

Définition d'une application mobile et d'un écran de démarrage

Création d'un conteneur d'application mobile

La première balise d'une application mobile est en général l'une des suivantes :

- La balise `<s:ViewNavigatorApplication>` définit une application mobile comprenant une seule section.
- La balise `<s:TabbedViewNavigatorApplication>` définit une application mobile comprenant plusieurs sections.

Lorsque vous développez des applications destinées à des tablettes, les limites de taille d'écran sont moins contraignantes qu'avec les téléphones. Par conséquent, pour une tablette, vous n'avez pas à structurer votre application autour de petites vues. A la place, vous pouvez créer l'application en utilisant le conteneur Spark Application standard avec les composants et habillages mobiles pris en charge.

Remarque : *lors du développement d'une application mobile quelconque, vous pouvez utiliser le conteneur Spark Application, même pour les téléphones. Toutefois, le conteneur Spark Application n'inclut pas de prise en charge pour la navigation entre vues, la persistance des données ni les boutons Retour et Menu du périphérique. Pour plus d'informations, voir « Différences entre les conteneurs d'application mobile et le conteneur d'application Spark » à la page 28.*

Les conteneurs d'application mobile présentent les caractéristiques suivantes par défaut :

Caractéristiques	Conteneurs ViewNavigatorApplication et TabbedViewNavigatorApplication Spark
Taille par défaut	100 % de hauteur et 100 % de largeur pour occuper tout l'espace de l'écran.
Présentation des enfants	Définie par les conteneurs View individuels qui constituent les vues de l'application.
Marges intérieures par défaut	0 pixel.
Barres de défilement	Aucune. Si vous ajoutez des barres de défilement à l'habillage du conteneur de l'application, les utilisateurs peuvent faire défiler l'ensemble de l'application. Cela comprend la zone du contrôle ActionBar et de la barre d'onglets de l'application. En général, vous ne voulez pas faire défiler ces zones de la vue. Par conséquent, ajoutez des barres de défilement aux conteneurs View individuels de l'application, plutôt qu'à l'habillage du conteneur de l'application.

Différences entre les conteneurs d'application mobile et le conteneur d'application Spark

Les conteneurs d'application mobile Spark présentent la plupart des mêmes fonctionnalités que le conteneur de l'application Spark. Par exemple, vous appliquez des styles aux conteneurs d'application mobile de la même manière que vous les appliquez au conteneur Spark Application.

Les conteneurs d'application mobile Spark présentent plusieurs caractéristiques qui diffèrent de celles du conteneur Spark Application :

- **Prise en charge de la persistance**

  Prise en charge du stockage des données sur un disque et de leur téléchargement à partir d'un disque. La persistance permet aux utilisateurs d'interrompre une application mobile, par exemple pour répondre à un appel téléphonique, puis de restaurer l'état de l'application à la fin de l'appel.

- **Prise en charge de la navigation dans les vues**

  Le conteneur ViewNavigatorApplication crée automatiquement un seul conteneur ViewNavigator afin de contrôler la navigation entre les vues.

  Le conteneur TabbedViewNavigatorApplication crée automatiquement un conteneur TabbedViewNavigator individuel pour contrôler la navigation entre les sections.

- **Prise en charge des boutons Retour et Menu du périphérique**

  Lorsque l'utilisateur appuie sur le bouton Retour, l'application revient à la vue précédente de la pile. Lorsque l'utilisateur appuie sur le bouton Menu, le conteneur ViewMenu de la vue actuelle apparaît, s'il est défini.

Pour plus d'informations sur le conteneur Spark Application, voir A propos du conteneur Application.

Gestion d'événements de niveau application

La classe NativeApplication représente une application AIR. Elle fournit des informations sur l'application, les fonctions de l'ensemble de l'application et envoie des événements au niveau de l'application. Vous pouvez accéder à l'instance de la classe NativeApplication qui correspond à votre application mobile en utilisant la propriété statique `NativeApplication.nativeApplication`.

Par exemple, la classe NativeApplication définit les événements `invoke` et `exiting` que vous pouvez gérer dans votre application mobile. L'exemple suivant fait référence à la classe NativeApplication pour définir un gestionnaire d'événement pour l'événement `exiting` :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkNativeApplicationEvent.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            protected function creationCompleteHandler(event:FlexEvent):void {
                // Reference NativeApplication to assign the event handler.
              NativeApplication.nativeApplication.addEventListener(Event.EXITING, myExiting);
            }

            protected function myExiting(event:Event):void {
                // Handle exiting event.
            }
        ]]>
    </fx:Script>

</s:ViewNavigatorApplication>
```

Notez que vous accédez au ViewNavigator à l'aide de la propriété `ViewNavigatorApplication.navigator`.

Ajout d'un écran de démarrage à une application

Le conteneur Spark Application constitue une classe de base pour les conteneurs ViewNavigatorApplication et TabbedViewNavigatorApplication. S'il est utilisé avec le thème Spark, le conteneur Spark Application prend en charge un preloader d'application pour indiquer l'avancement du téléchargement et de l'initialisation du fichier SWF d'une application. S'il est utilisé avec le thème Mobile, vous pouvez afficher un écran de démarrage à la place.

L'écran de démarrage apparaît au cours du démarrage de l'application. Pour configurer l'écran de démarrage, vous utilisez les propriétés `splashScreenImage`, `splashScreenScaleMode` et `splashScreenMinimumDisplayTime` de la classe d'application.

L'exemple utilise un écran de démarrage au format Letterbox dans une application mobile :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMobileSplashScreen.mxml -->
<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>
```

Remarque : *pour utiliser l'écran de démarrage dans une application de bureau, définissez la propriété* `Application.preloader` *sur spark.preloaders.SplashScreen. Ajoutez également le fichier frameworks\libs\mobile\mobilecomponents.swc au chemin de bibliothèque de l'application.*

Le blogueur Joseph Labrecque a communiqué sur AIR pour Android Splash Screen avec Flex 4.5.

Le blogueur Brent Arnold a créé une vidéo sur l'ajout d'un écran de démarrage pour une application Android.

Définition de vues dans une application mobile

Une application mobile définit en général plusieurs écrans, ou vues. Lorsque les utilisateurs naviguent dans l'application, ils passent d'une vue à l'autre.

Rendez la navigation intuitive pour l'utilisateur de l'application. Ainsi, lorsque l'utilisateur passe d'une vue à une autre, il s'attend à pouvoir revenir à la vue précédente. L'application peut définir un bouton Accueil ou d'autres aides à la navigation de niveau supérieur qui permettent à l'utilisateur d'accéder à différents emplacements de l'application depuis tout autre emplacement.

Pour définir les vues d'une application mobile, utilisez le conteneur View. Pour contrôler la navigation entre les vues d'une application mobile, utilisez le conteneur ViewNavigator.

Regardez cette vidéo de Peter Elst pour savoir comment créer une application gérée par les données avec plusieurs vues.

Utilisation de pushView() pour changer de vue

Utilisez la méthode `ViewNavigator.pushView()` pour placer une nouvelle vue sur la pile. Accédez au ViewNavigator à l'aide de la propriété `ViewNavigatorApplication.navigator`. La mise en place d'une vue modifie l'affichage de l'application en passant à la nouvelle vue.

La méthode `pushView()` présente la syntaxe suivante :

```
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 n'importe quelles données communiquées à la vue. Cet objet est écrit dans la propriété `View.data` de la nouvelle vue.

- `context` spécifie un objet arbitraire écrit dans la propriété `ViewNavigator.context`. Lorsque la nouvelle vue est créée, elle peut faire référence à cette propriété et exécuter une action en fonction de cette valeur. Par exemple, la vue peut afficher des données de différentes manières selon la valeur de `context`.
- `transition` spécifie la transition à utiliser lors du changement de vue. Pour plus d'informations sur les transitions liées aux vues, voir « Définition de transitions dans une application mobile » à la page 60

Utilisation de l'argument data pour faire passer un seul objet

Utilisez l'argument `data` pour faire passer un seul objet contenant les éventuelles données requises par la nouvelle vue. La vue peut alors accéder à l'objet en utilisant la propriété `View.data`, comme le montre l'exemple suivant :

```
<?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>
```

Dans cet exemple, la vue EmployeeView est définie dans le fichier EmployeeView.mxml. Cette vue utilise la propriété `data` pour accéder au prénom et au nom d'un employé et à l'ID de l'employé à partir de l'objet qui lui a été transmis.

La propriété `View.data` est garantie valide au moment de l'événement `ajout` pour le nouvel objet. Pour plus d'informations sur le cycle de vie d'un conteneur View, voir « Cycle de vie des conteneurs Spark ViewNavigator et View » à la page 40.

Communication de données à la première vue d'une application

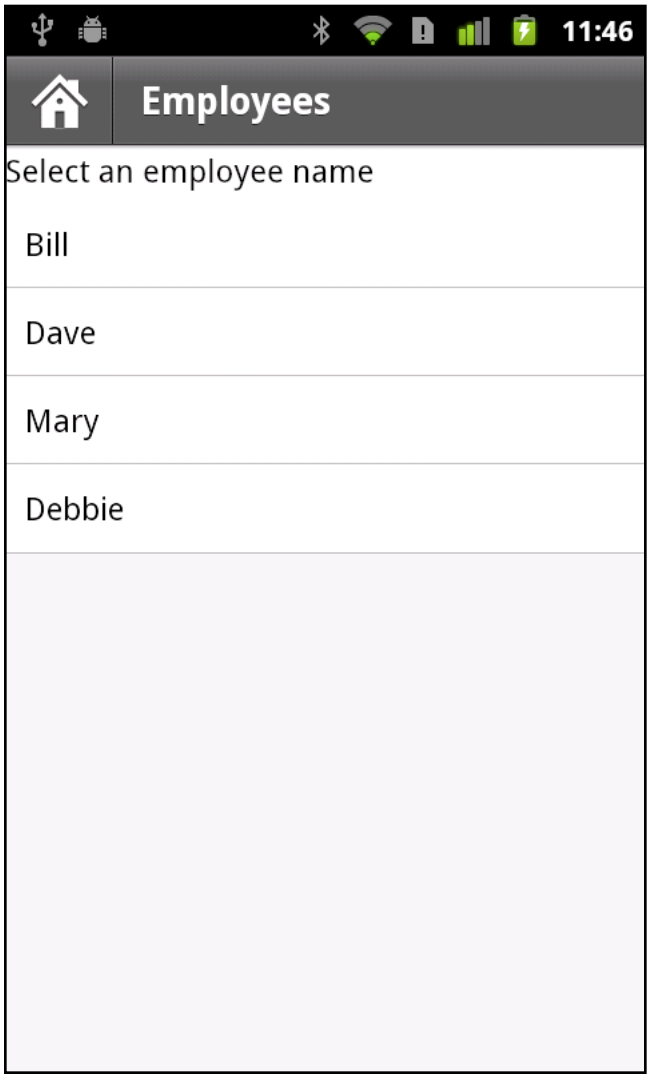
La propriété `ViewNavigatorApplication.firstView` et la propriété `ViewNavigator.firstView` définissent la première vue dans une application. Pour communiquer des données à la première vue, utilisez la propriété `ViewNavigatorApplication.firstViewData` ou la propriété `ViewNavigator.firstViewData`.

Communication de données à une vue

Dans l'exemple suivant, vous définissez 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 parmi les vues définies par l'application.

```
<?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>
```

Cet exemple définit un bouton Accueil dans la zone de navigation du contrôle ActionBar. La sélection du bouton Accueil élimine toutes les vues de la pile et revient à la première vue. La figure suivante illustre cette application :



Le fichier EmployeeMainView.mxml définit la première vue de l'application, comme le montre l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeMainView.mxml -->
<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>
```

Cette vue définit un contrôle List qui permet à l'utilisateur de sélectionner un nom d'employé. La sélection d'un nom incite le gestionnaire d'événement correspondant à l'événement `change` à placer sur la pile une instance d'une vue différente, nommée EmployeeView. Le fait de pousser une instance de EmployeeView provoque le changement par l'application de la vue EmployeeView.

La méthode `pushView()` dans cet exemple utilise deux arguments : la nouvelle vue et un Objet qui définit les données à communiquer à la nouvelle vue. Dans cet exemple, vous communiquez l'objet de données correspondant à l'élément actuellement sélectionnée dans le contrôle List.

L'exemple suivant illustre la définition de la vue EmployeeView :

```
<?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>
```

La vue EmployeeView affiche les trois champs du fournisseur de données du contrôle List. EmployeeView accède aux données qui lui sont communiquées à l'aide de la propriété `View.data`.


Le blogueur Steve Mathews a créé une entrée de cookbook sur la transmission des données entre les vues.


La blogueuse Holly Schinsky a communiqué sur la persistance et la gestion des données dans la gestion des données mobiles de Flex 4.5.

Retour de données d'une vue

La méthode `ViewNavigator.popView()` retourne le contrôle de la vue actuelle à la vue précédente de la pile. Lorsque la méthode `popView()` s'exécute, la vue actuelle est détruite et la vue précédente sur la pile est rétablie. La restauration du conteneur View précédent implique la réinitialisation de sa propriété `data` dans la pile,

Pour obtenir une description complète du cycle de vie d'une vue, y compris des événements envoyés au cours de sa création, voir « Cycle de vie des conteneurs Spark ViewNavigator et View » à la page 40.

La nouvelle vue est restaurée avec l'objet `data` original correspondant au moment où elle a été désactivée. Par conséquent, vous n'utilisez pas généralement l'objet `data` original pour transmettre en retour les données de l'ancienne vue à la nouvelle vue. A la place, remplacez la méthode `createReturnObject()` de l'ancienne vue. La méthode `createReturnObject()` retourne un seul objet.

Retour du type d'objet

L'objet retourné par la méthode `createReturnObject()` est écrit dans la propriété `ViewNavigator.poppedViewReturnedObject`. Le type de données de la propriété `poppedViewReturnedObject` est ViewReturnObject.

ViewReturnObject définit deux propriétés, `context` et `object`. La propriété `object` contient l'objet retourné par la méthode `createReturnObject()`. La propriété `context` contient la valeur de l'argument `context` qui a été transmis à la vue lorsque la vue a été placée sur la pile de navigation à l'aide de `pushView()`.

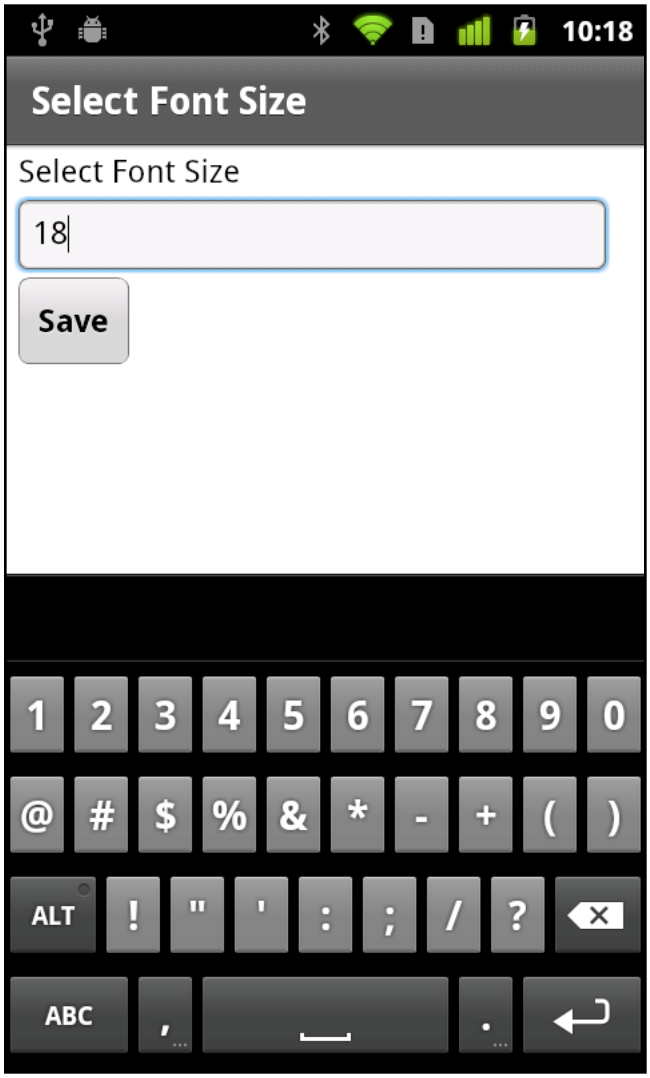
La propriété `poppedViewReturnedObject` est garantie comme étant définie dans la nouvelle vue avant que la vue ne reçoive l'événement `ajout`. Si la propriété `poppedViewReturnedObject.object` est nulle, aucune donnée n'a été retournée.

Exemple : Communication de données à une vue

L'exemple suivant, SelectFont.mxml, présente une vue qui vous permet de définir une taille de police. Le remplacement de la méthode `createReturnObject()` retourne la valeur sous forme de numéro. Le champ `fontSize` de la propriété `data` transmise à partir de la vue précédente définit la valeur initiale du contrôle TextInput :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SelectFont.mxml -->
<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>
```

La figure suivante présente la vue définie par SelectFont.mxml :



La vue de l'exemple suivant, MainFontView.mxml, utilise la vue définie dans SetFont.mxml. La vue MainFontView.mxml définit les éléments suivants :

- Un contrôle Button dans le conteneur ActionBar pour passer à la vue définie par SetFont.mxml.
- Un gestionnaire d'événement pour l'événement `ajout` qui détermine pour la première fois si la propriété `View.data` est nulle. Si la valeur est null, le gestionnaire d'événement ajoute le champ `data.fontSize` à la propriété `View.data`.

  Si la propriété `data` n'a pas la valeur null, le gestionnaire d'événement définit la taille de la police sur la valeur figurant dans le champ `data.fontSize`.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\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;

            // Change to the SelectFont view, and pass the current data property.
            // The data property contains the fontSize field with the current font size.
            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 the data property is null,
                // initialize it and create the data.fontSize field.
                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>
```

Configuration d'une application pour l'orientation Portrait ou Paysage

Un périphérique mobile définit l'orientation d'une application automatiquement lorsque l'orientation du périphérique change. Pour configurer votre application pour différentes orientations, Flex définit deux états d'affichage qui correspondent aux orientations Portrait et Paysage :`portrait` et `landscape`. Utilisez ces états de vue pour définir les caractéristiques de votre application en fonction de l'orientation.

L'exemple suivant utilise l'état d'affichage pour contrôler la propriété `layout` d'un conteneur Group en fonction de l'orientation actuelle :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SearchViewStates.mxml -->
<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>
```

Cet exemple définit une vue de recherche. Le conteneur Group contrôle la présentation du texte de recherche saisi et du bouton de recherche. En mode portrait, le conteneur Group utilise la présentation verticale. Le passage de la présentation au mode paysage incite le conteneur Group à utiliser une présentation horizontale.

Définition d'un habillage personnalisé pour prendre en charge les modes de présentation

Vous pouvez définir une classe d'habillages personnalisée pour une application mobile. Si l'habillage prend en charge la présentation portrait et paysage, votre habillage doit gérer les états des vues `portrait` et `paysage`.

Vous pouvez configurer une application de telle sorte qu'elle ne modifie pas l'orientation de la présentation lorsque l'utilisateur fait pivoter le périphérique. Pour ce faire, modifiez le fichier XML de l'application, celui qui se termine par -app.xml, afin de définir les propriétés suivantes :

- Pour empêcher l'application de modifier l'orientation de la présentation, définissez la propriété `<autoOrients>` sur `false`.
- Pour définir l'orientation, définissez la propriété `<aspectRatio>` sur `portrait` ou `paysage`.

Définission du mode d'incrustation d'un conteneur Spark ViewNavigator

Par défaut, la barre d'onglets et le contrôle ActionBar d'une application mobile définissent une zone qui ne peut pas être utilisée par les vues de l'application. Cela signifie que votre contenu ne peut pas utiliser toute la taille de l'écran du périphérique mobile.

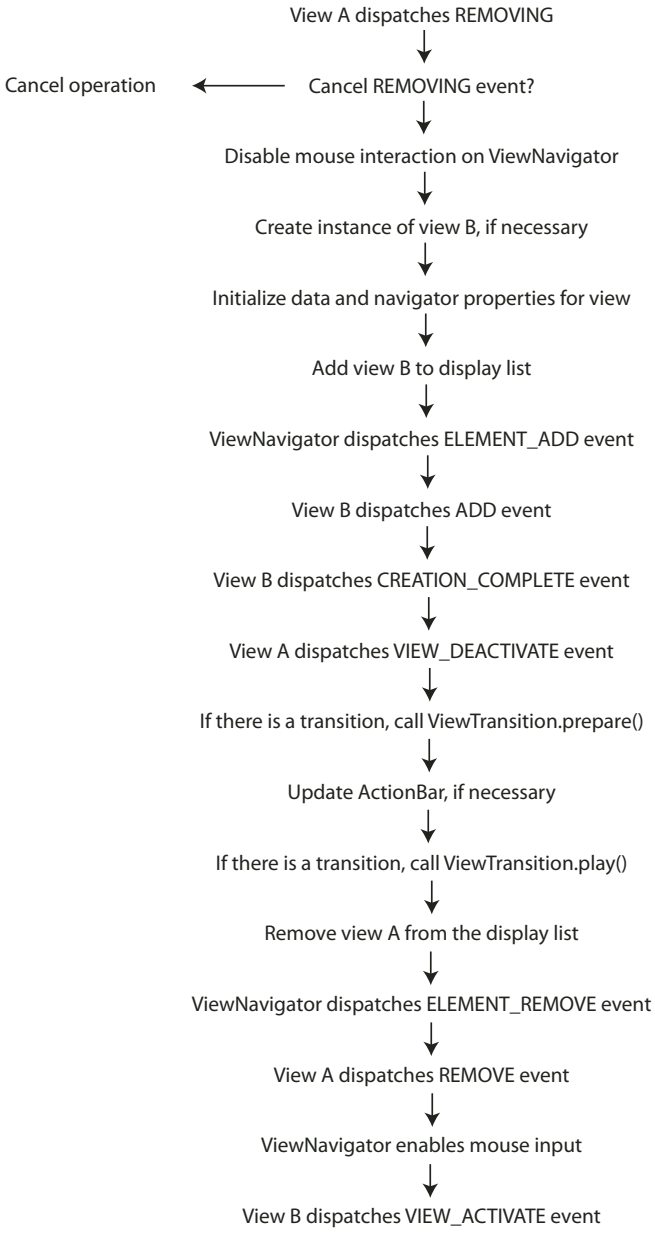
Vous pouvez toutefois utiliser la propriété `ViewNavigator.overlayControls` pour modifier la présentation par défaut de ces composants. Lorsque vous définissez la propriété `overlayControls` sur `true`, la zone de contenu de l'application couvre toute la largeur et toute la hauteur de l'écran. Le contrôle ActionBar et la barre d'onglets chevauchent la zone de contenu avec une valeur alpha qui les rend partiellement transparents.

La classe d'habillage pour le conteneur ViewNavigator, spark.skins.mobile.ViewNavigatorSkin, définit les états des vues afin de gérer les différentes valeurs de la propriété `overlayControls`. Lorsque la propriété `overlayControls` est `true`, « AndOverlay » est annexé au nom de l'état actuel. Par exemple, l'habillage de ViewNavigator est l'état « portrait » par défaut. Lorsque la propriété `overlayControls` est `true`, l'état de l'habillage du navigateur passe à « portraitAndOverlay ».

Cycle de vie des conteneurs Spark ViewNavigator et View

Flex procède à une série d'opérations lorsque vous passez d'une vue à une autre dans une application mobile. A différents stades du processus de changement de vue, Flex envoie des événements. Vous pouvez surveiller ces événements pour effectuer des actions au cours du processus. Vous pouvez par exemple utiliser l'événement `suppression` pour annuler le passage d'une vue à une autre.

Le tableau suivant décrit le processus de passage de la vue actuelle, la Vue A, à une autre vue, la Vue B :



Définition d'onglets dans une application mobile

Définition des sections d'une application

Utilisez le conteneur TabbedViewNavigatorApplication pour définir une application mobile comprenant plusieurs sections. Le conteneur TabbedViewNavigatorApplication crée automatiquement un conteneur TabbedMobileNavigator. Le conteneur TabbedViewNavigator crée une barre d'onglets pour prendre en charge la navigation entre les sections de l'application.

Chaque conteneur ViewNavigator définit une section différente de l'application. Utilisez la propriété `navigators` du conteneur TabbedViewNavigatorApplication pour spécifier les conteneurs ViewNavigator.

Dans l'exemple ci-dessous, vous définissez trois sections correspondant aux trois balises ViewNavigator. Chaque ViewNavigator définit la première vue qui apparaît lorsque vous accédez à la section:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMultipleSections.mxml -->
<s:TabbedViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">

    <s:navigators>
        <s:ViewNavigator label="Employees" firstView="views.EmployeeMainView"/>
        <s:ViewNavigator label="Contacts" firstView="views.ContactsMainView"/>
        <s:ViewNavigator label="Search" firstView="views.SearchView"/>
    </s:navigators>

</s:TabbedViewNavigatorApplication>
```

Remarque : *vous n'avez pas à spécifier la balise enfant `navigators` dans MXML car il s'agit de la propriété par défaut de TabbedViewNavigator.*

Chaque ViewNavigator gère une pile de navigation distincte. Par conséquent, les méthodes ViewNavigator, telles que `pushView()` et `popView()` concernent la section actuellement active. Le bouton retour sur le périphérique mobile retourne le contrôle vers la vue précédente sur la pile du conteneur ViewNavigator actuel. Le changement de vue n'altère pas la section actuelle.

Vous n'avez pas à ajouter une logique particulière à l'application pour la navigation dans les sections. Le conteneur TabbedViewNavigator crée automatiquement une barre d'onglets au bas de l'application afin de contrôler la navigation entre les sections.

Même si cela n'est pas obligatoire, vous pouvez ajouter un contrôle programmatique de la section actuelle. Pour modifier les sections par programmation, définissez la propriété `TabbedViewNavigator.selectedIndex` sur l'index de la section désirée. Les index des sections sont de base 0, c'est-à-dire que la première section de l'application est à l'index 0, la deuxième à l'index 1, etc.

Brent Arnold, expert Flex certifié par Adobe, a conçu une vidéo concernant l'utilisation d'une pile de navigation ViewNavigator.

Gestion des événements de changement de section

Lorsque la section change, le conteneur TabbedViewNavigator envoie les événements suivants :

- L'événement `changing` est envoyé juste avant le passage d'une section à une autre. Pour empêcher le changement de section, appelez la méthode `preventDefault()` dans le gestionnaire d'événement pour l'événement `changing`.
- L'événement `change` est envoyé juste après le changement de section.

Configuration de l'ActionBar avec des sections multiples

Un contrôle ActionBar est associé à un conteneur ViewNavigator. Par conséquent, vous pouvez configurer le contrôle ActionBar pour chaque section lorsque vous définissez le conteneur ViewNavigator de la section. Dans l'exemple suivant, vous configurez le contrôle ActionBar séparément pour chaque conteneur ViewNavigator qui définit les trois sections différentes de l'application :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMultipleSectionsAB.mxml -->
<s:TabbedViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first section in the application.
                tabbedNavigator.selectedIndex = 0;
                // Switch to the first view in the section.
                ViewNavigator(tabbedNavigator.selectedNavigator).popToFirstView();
            }
        ]]>
    </fx:Script>

    <s:navigators>
        <s:ViewNavigator label="Employees" firstView="views.EmployeeMainView">
            <s:navigationContent>
                <s:Button icon="@Embed(source='assets/Home.png')"
                    click="button1_clickHandler(event)"/>
            </s:navigationContent>
        </s:ViewNavigator>
        <s:ViewNavigator label="Contacts" firstView="views.ContactsMainView">
            <s:navigationContent>
                <s:Button icon="@Embed(source='assets/Home.png')"
                    click="button1_clickHandler(event)"/>
            </s:navigationContent>
        </s:ViewNavigator>
        <s:ViewNavigator label="Search" firstView="views.SearchView">
            <s:navigationContent>
                <s:Button icon="@Embed(source='assets/Home.png')"
                    click="button1_clickHandler(event)"/>
            </s:navigationContent>
        </s:ViewNavigator>
    </s:navigators>

</s:TabbedViewNavigatorApplication>
```

La figure suivante montre cette application avec l'onglet Contacts sélectionné dans la barre d'onglets :

Il est également possible de définir l'ActionBar dans chaque vue de l'application. De cette manière, chaque vue utilise le même contenu ActionBar, où que vous l'utilisiez dans l'application.

Contrôle de la barre d'onglets

Masquage du contrôle de la barre d'onglets dans une vue

N'importe quelle vue peut masquer la barre d'onglets si vous définissez la propriété `View.tabBarVisible` sur `false`. Par défaut, la propriété `tabBarVisible` a la valeur `true` pour afficher la barre d'onglets.

Vous pouvez également faire appel aux méthodes `TabbedViewNavigator.hideTabBar()` et `TabbedViewNavigator.showTabBar()` pour contrôler la visibilité.

Brent Arnold, expert Flex certifié par Adobe, a créé une vidéo sur le masquage de la barre des onglets.

Application d'un effet à la barre d'onglets du conteneur TabbedViewNavigator

Par défaut, la barre d'onglets utilise un effet de glissement pour ses effets d'affichage et de masquage. La barre d'onglets n'utilise aucun effet lorsque vous changez l'onglet actuellement sélectionné.

Vous pouvez modifier l'effet par défaut de la barre d'onglets pour un effet d'affichage ou de masquage en remplaçant les méthodes `TabbedViewNavigator.createTabBarHideEffect()` et `TabbedViewNavigator.createTabBarShowEffect()`. Après avoir masqué la barre d'onglets, pensez à définir les propriétés `visible` et `includeInLayout` de la barre d'onglets sur `false`.

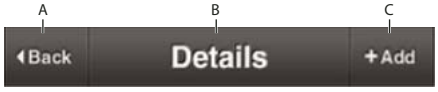
Définition de contrôles de navigation, de titre et d'action dans une application mobile

Configuration du 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 auxquels les utilisateurs peuvent accéder en tout point de l'application ou dans une vue spécifique. Vous pouvez par exemple utiliser le contrôle ActionBar pour ajouter un bouton d'accueil, un bouton de recherche ou d'autres options.

Pour une application mobile disposant d'une seule section, ce qui signifie un seul conteneur ViewNavigator, toutes les vues partagent la même barre d'action. Pour une application mobile comprenant plusieurs sections, et par conséquent, plusieurs conteneurs ViewNavigator, chaque section définit sa propre barre d'action.

Utilisez le contrôle ActionBar pour définir la zone de la barre d'action. Le contrôle ActionBar définit trois zones distinctes, comme le montre la figure suivante :



***A**. Zone de navigation **B**. Zone de titre **C**. Zone d'action*

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 d'accueil dans la zone de navigation.

  Utilisez la propriété `navigationContent` pour définir les composants qui apparaissent dans la zone de navigation. Utilisez la propriété `navigationLayout` pour définir la présentation de la zone de navigation.

- **Zone de titre**

  Contient soit une chaîne contenant le texte du titre, soit des composants. Si vous spécifiez des composants, vous ne pouvez pas spécifier une chaîne de titre.

  Utilisez la propriété `title` pour spécifier la chaîne qui doit apparaître dans la zone de titre. Utilisez la propriété `titleContent` pour définir les composants qui figurent dans la zone de titre. Utilisez la propriété `titleLayout` pour définir la présentation de la zone de titre. Si vous spécifiez une valeur pour la propriété `titleContent`, l'habillage ActionBar ignore la propriété `title`.

- **Zone d'action**

Contient des composants qui définissent des actions que l'utilisateur peut effectuer dans une vue. Par exemple, vous pouvez définir un bouton de recherche ou d'actualisation dans la zone d'action.

Utilisez la propriété `actionContent` pour définir les composants qui apparaissent dans la zone d'action. Utilisez la propriété `actionLayout` pour définir la présentation de la zone d'action.

Même si Adobe recommande d'utiliser les zones de navigation, de titre et d'action décrites, il n'existe aucune restriction sur le type de composant que vous placez dans ces zones.

Définition de propriétés ActionBar dans le conteneur ViewNavigatorApplication, ViewNavigator ou View

Vous pouvez définir les propriétés qui définissent le contenu du contrôle ActionBar dans le conteneur ViewNavigatorApplication, dans le conteneur ViewNavigator ou dans des conteneurs View individuels. Le conteneur View possède la priorité la plus élevée, suivi du conteneur ViewNavigator, puis du conteneur ViewNavigatorApplication. Par conséquent, les propriétés que vous définissez dans le conteneur ViewNavigatorApplication s'appliquent à l'application entière, mais vous pouvez les remplacer dans le conteneur ViewNavigator ou View.

Remarque : *un contrôle ActionBar est associé à un conteneur ViewNavigator, de sorte qu'il est spécifique à une seule section d'une application mobile. Vous ne pouvez donc pas configurer un contrôle ActionBar à partir des conteneurs TabbedViewNavigatorApplication et TabbedViewNavigator.*

Le blogueur Brent Arnold a créé un didacticiel vidéo concernant l'utilisation d'ActionBar.

Exemple : personnalisation d'un contrôle ActionBar Spark au niveau de l'application

L'exemple suivant présente le fichier de l'application principale d'une application mobile :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkActionBarSimple.mxml -->
<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>
```

Cet exemple définit un bouton Accueil dans la zone de contenu de navigation du contrôle ActionBar et un bouton Actualiser dans la zone de contenu.

L'exemple suivant définit le conteneur View MobileViewHome qui détermine la première vue de l'application. Le conteneur View définit une chaîne de titre, « Vue Accueil », mais ne remplace ni le contenu de la navigation, ni les zones de contenu d'action du contrôle ActionBar :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\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>
```

Exemple : personnalisation d'un contrôle ActionBar dans un conteneur View

Cet exemple utilise un fichier d'application principal avec une section individuelle qui définit un bouton Accueil dans la zone de navigation du conteneur ViewNavigatorApplication. Il définit également un bouton Recherche dans la zone d'action :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkActionBarOverride.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    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>
```

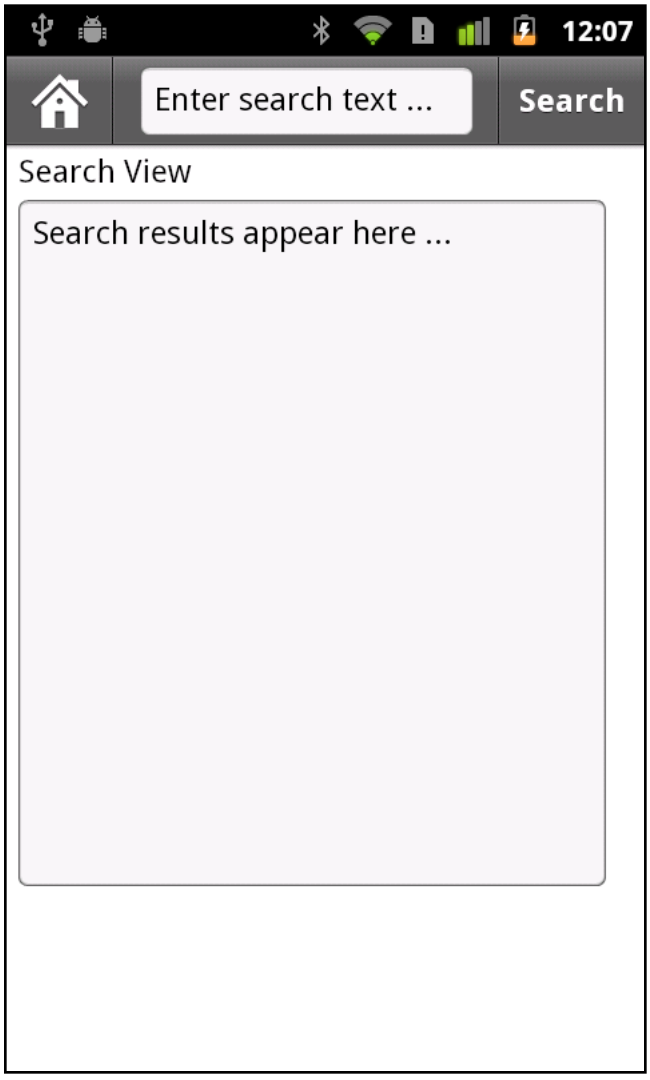
La première vue de cette application est la vue MobileViewHomeOverride. La vue MobileViewHomeOverride définit un contrôle Button afin de naviguer jusqu'à un deuxième conteneur View qui définit une page Recherche :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\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>
```

Le conteneur View qui définit la page Recherche remplace la zone de titre et la zone d'action du contrôle ActionBar, comme indiqué ci-dessous :

```
<?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>
```

La figure suivante présente le contrôle ActionBar pour cette vue :



Dans la mesure où la vue Recherche ne remplace pas la zone de navigation du contrôle ActionBar, la zone de navigation affiche toujours le bouton Accueil.

Masquage du contrôle ActionBar

Vous pouvez masquer le contrôle ActionBar dans une vue quelconque en définissant la propriété `View.actionBarVisible` sur `false`. Par défaut, la propriété `actionBarVisible` est `true` pour afficher le contrôle ActionBar.

Utilisez la méthode `ViewNavigator.hideActionbar()` pour masquer le contrôle ActionBar pour toutes les vues contrôlées par le conteneur ViewNavigator, comme l'indique l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSectionNoAB.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.HomeView"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            protected function creationCompleteHandler(event:FlexEvent):void {
            // Access the ViewNavigator using the ViewNavigatorApplication.navigator property.
                navigator.hideActionBar();
            }
        ]]>
    </fx:Script>

</s:ViewNavigatorApplication>
```

Vous pouvez définir un effet personnalisé pour l'ActionBar lorsque celui-ci est masqué, ou lorsqu'il est rendu visible. Par défaut, le contrôle ActionBar fait appel à l'effet Animer comme effet d'affichage et de masquage. Vous pouvez modifier l'effet par défaut en substituant les méthodes `ViewNavigator.createActionBarHideEffect()` et `ViewNavigator.createActionBarShowEffect()`. Après avoir affiché un effet qui masque le contrôle ActionBar, définissez ses propriétés `visible` et `includeInLayout` sur `false` afin qu'il ne soit plus inclus dans la présentation de la vue.

Utilisation des barres de défilement dans une application mobile

Considérations liées à l'utilisation des barres de défilement dans une application mobile

Généralement, si le contenu occupe plus que la zone visible à l'écran, l'application affiche des barres de défilement. Utilisez le contrôle Scroller pour ajouter des barres de défilement dans votre application. Pour plus d'informations, voir Scrolling Spark containers.

La zone active d'une barre de défilement est la zone de l'écran dans laquelle vous placez le curseur de la souris pour effectuer un défilement. Dans une application de bureau ou de navigateur, la zone active correspond à la zone visible de la barre de défilement. Dans une application mobile, les barres de défilement sont masquées même lorsque le contenu est plus important que la zone visible de l'écran. Le masquage des barres de défilement permet à l'application d'utiliser toute la largeur et toute la hauteur de l'écran.

Une application mobile doit faire une distinction entre le moment où l'utilisateur agit sur un contrôle, par exemple en sélectionnant un contrôle Button, et le moment où l'utilisateur souhaite faire défiler l'affichage. Pour ce qui est des barres de défilement dans une application mobile, il faut garder à l'esprit que les composants Flex changent souvent d'apparence en réaction à une intervention de l'utilisateur.

Ainsi, lorsque l'utilisateur appuie sur un contrôle Button, le bouton change d'aspect pour indiquer qu'il est sélectionné. Lorsque l'utilisateur relâche le bouton, celui-ci reprend l'aspect correspondant à l'état désélectionné. Toutefois, lors du défilement, si l'utilisateur touche l'écran au niveau du bouton, vous ne souhaitez pas que le bouton change d'aspect.

Steven Shongrunden, ingénieur Adobe, présente un exemple d'utilisation des barres de défilement dans son article sur l'enregistrement de la position de défilement entre des vues dans une application mobile Flex.

Evénements et barres de défilement

Les composants Flex s'appuient sur les événements pour indiquer qu'une intervention de l'utilisateur s'est produite. En réponse à l'intervention de l'utilisateur, le composant peut changer d'apparence ou effectuer une autre action.

Les développeurs d'application utilisent les événements pour gérer les interventions de l'utilisateur. Par exemple, vous utilisez généralement l'événement `click` du contrôle Button pour exécuter un gestionnaire d'événement en réponse à la sélection du bouton par l'utilisateur. Utilisez l'événement `change` du contrôle List pour exécuter un gestionnaire d'événement lorsque l'utilisateur sélectionne un élément dans la liste.

Le mécanisme de défilement Flex repose sur l'événement `mouseDown`. Cela signifie que le mécanisme de défilement écoute les événements `mouseDown` pour déterminer si une opération de défilement doit être lancée.

Interprétation d'un geste utilisateur en tant qu'opération de défilement

Par exemple, une application est composée de plusieurs contrôles Button dans un conteneur prenant en charge le défilement :

1. Utilisez votre doigt pour appuyer sur un contrôle Button. Le bouton envoie un événement `mouseDown`.
2. Flex diffère la réponse à l'intervention de l'utilisateur pendant une durée prédéfinie. Le délai permet de garantir que l'utilisateur sélectionne le bouton et ne tente pas d'effectuer un défilement dans l'écran.

   Si, au cours de cette période, vous déplacez votre doigt sur une distance supérieure à celle qui est prédéfinie, Flex interprète ce geste comme une action de défilement. La distance sur laquelle vous devez déplacer votre doigt pour que le geste soit interprété comme un défilement est d'environ 0,08 pouce. Cette distance correspond à environ 20 pixels sur un périphérique de 252 PPP.

   Comme vous avez déplacé votre doigt avant l'expiration du délai, le contrôle Button ne reconnaît jamais l'intervention. Le bouton n'envoie pas d'événement et ne change jamais d'aspect.
3. Une fois le délai écoulé, le contrôle Button reconnaît l'intervention de l'utilisateur. Le bouton change d'aspect pour indiquer qu'il a été sélectionné.

   Utilisez la propriété `touchDelay` du contrôle pour configurer la durée de ce délai. La valeur par défaut est de 100 ms. Si vous définissez la propriété `touchDelay` sur 0, il n'y a pas de délai et le défilement est immédiatement déclenché.
4. Après l'expiration du délai et lorsque Flex a envoyé les événements de souris, vous déplacez votre doigt sur une distance supérieure à 20 pixels. Le contrôle Button revient à son état normal et l'action de défilement débute.

   Dans ce cas, le bouton a changé d'aspect car le délai a expiré. Toutefois, lorsque vous déplacez votre doigt sur une distance supérieure à 20 pixels, même au-delà du délai, Flex interprète ce geste comme un mouvement de défilement.

Remarque : *les composants Flex prennent en charge de nombreux types d'événement en plus des événements de souris. Lorsque vous travaillez avec des composants, vous décidez de la manière dont votre application réagit à ces événements. Au moment de l'événement* `mouseDown`*, l'intention exacte de l'utilisateur est ambiguë. L'utilisateur pouvait avoir l'intention d'interagir avec le composant ou d'effectuer un défilement. En raison de cette ambiguité, Adobe recommande d'écouter les événements* `click` *ou* `mouseUp` *au lieu de l'événement* `mouseDown`*.*

Gestion des événements de défilement

Pour signaler le début d'une opération de défilement, le composant qui envoie l'événement `mouseDown` , lequel envoie un événement `touchInteractionStarting` de propagation. Si cet événement n'est pas annulé, le composant envoie un événement `touchInteractionStart` de propagation.

Lorsqu'un composant détecte un événement `touchInteractionStart`, il ne doit pas tenter de répondre au geste utilisateur. Par exemple, lorsqu'un contrôle Button détecte un événement `touchInteractionStart`, il désactive tous les indicateurs visuels qu'il a définis en fonction de l'événement `mouseDown` initial.

Si un composant ne souhaite pas autoriser le défilement à commencer, le composant peut appeler la méthode `preventDefault()` dans le gestionnaire d'événement pour l'événement `touchInteractionStarting`.

Pour signaler le début d'une opération de défilement, le composant qui envoie l'événement `mouseDown`, lequel envoie un événement `touchInteractionStarting` de propagation.

Comportement de défilement en fonction du point de contact initial

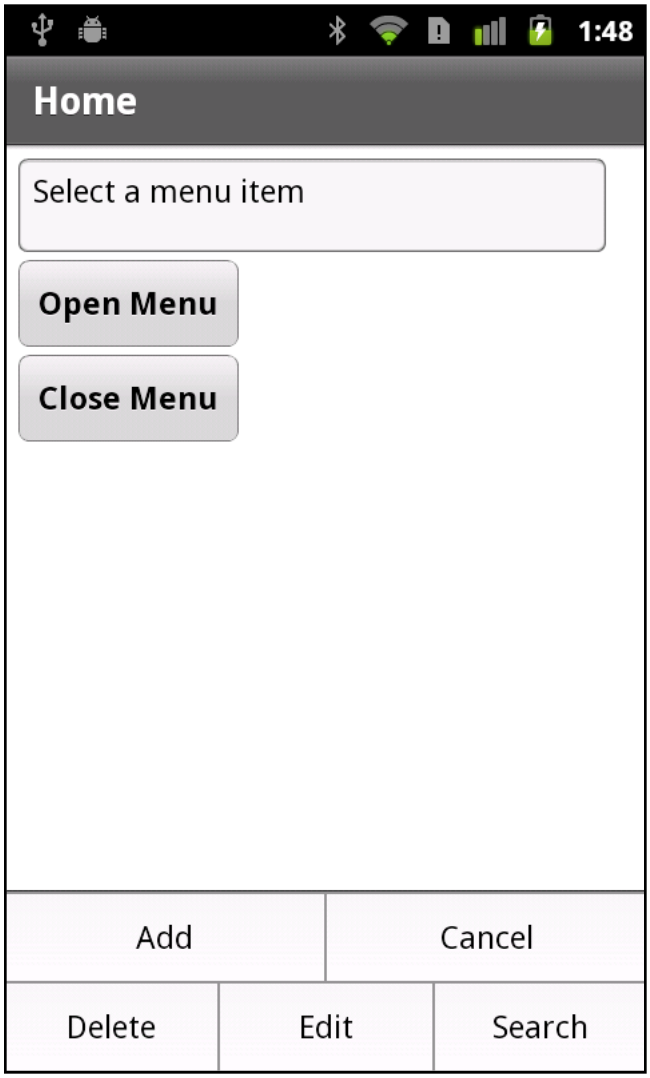
Le tableau suivant décrit la manière dont le défilement est géré en fonction de l'emplacement du point de contact initial :

Elément sélectionné	**Comportement**
Espace vide, texte non modifiable, texte non sélectionnable	Aucun composant ne reconnaît le mouvement. Le composant Scroller attend que l'utilisateur déplace le point de contact de plus de 20 pixels avant de déclencher le défilement.
Elément dans un contrôle List	Au-delà du délai défini, le rendu d'élément pour l'élément sélectionné modifie l'affichage pour passer à l'état sélectionné. Toutefois, si à tout moment, l'utilisateur déplace son doigt sur une distance supérieure à 20 pixels, l'élément change d'aspect pour revenir à l'état normal et le défilement est déclenché.
Button, CheckBox, RadioButton, DropDownList	Une fois le délai écoulé, l'état `mouseDown` est affiché. Toutefois, si l'utilisateur déplace le point de contact de plus de 20 pixels, le contrôle change d'aspect et présente son état normal et le défilement est déclenché.
Composant Button dans un rendu d'élément List	Le rendu d'élément n'est pas mis en surbrillance. Le composant Button ou Scroller gère le mouvement, de la même manière que dans le cas du composant Button normal.

Définition de menus dans une application mobile

Le conteneur ViewMenu définit un menu au bas d'un conteneur View dans une application mobile. Chaque conteneur View définit son propre menu spécifique à cette vue.

La figure suivante présente le conteneur ViewMenu dans une application :



Le conteneur ViewMenu définit un menu avec une seule hiérarchie de boutons de menu. En d'autres termes, il ne permet pas de créer des menus comprenant des sous-menus.

Les enfants du conteneur ViewMenu sont définis comme des contrôles ViewMenuItem. Chaque contrôle ViewMenuItem représente un seul bouton dans le menu.

Holly Schinsky, experte Flex certifiée par Adobe, a publié un article sur son blog concernant l'utilisation des menus dans une application mobile Flex 4.5.

Brent Arnold, expert Flex certifié par Adobe, a créé une vidéo sur la création d'un menu pour une application mobile Flex.

Interaction de l'utilisateur avec le conteneur ViewMenu

Ouvrez le menu en utilisant la clé de menu matérielle sur le périphérique mobile. Vous pouvez également l'ouvrir par programmation.

La sélection d'un bouton de menu ferme tout le menu. Le contrôle ViewMenuItem envoie un événement `click` lorsque l'utilisateur sélectionne un bouton de menu.

Pendant que le menu est ouvert, appuyez sur le bouton retour ou de menu du périphérique pour fermer le menu. Le menu se ferme également si vous appuyez sur l'écran en dehors du menu.

Le caret est le bouton de menu qui est actuellement actif. Utilisez le contrôle à cinq directions du périphérique ou les touches directionnelles pour modifier le caret. Appuyez sur la touche Entrée du périphérique ou sur le contrôle à cinq directions pour sélectionner l'élément de caret et fermer le menu.

Création d'un menu dans une application mobile

Utilisez la propriété `View.viewMenuItems` pour définir le menu pour une vue. La propriété `View.viewMenuItems` utilise un vecteur de contrôles ViewMenuItem, comme le montre l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\ViewMenuHome.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Home">

    <fx:Script>
        <![CDATA[
            // The event listener for the click event.
            private function itemClickInfo(event:MouseEvent):void {
                switch (event.currentTarget.label) {
                    case "Add" :
                        myTA.text = "Add selected";
                        break;
                    case "Cancel" :
                        myTA.text = "Cancel selected";
                        break;
                    case "Delete" :
                        myTA.text = "Delete selected";
                        break;
                    case "Edit" :
                        myTA.text = "Edit selected";
                        break;
                    case "Search" :
                        myTA.text = "Search selected";
                        break;
                    default :
                        myTA.text = "Error";
                }
```

```
            }
        ]]>
    </fx:Script>

    <s:viewMenuItems>
        <s:ViewMenuItem label="Add" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Cancel" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Delete" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Edit" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Search" click="itemClickInfo(event);"/>
    </s:viewMenuItems>

    <s:VGroup paddingTop="10" paddingLeft="10">
        <s:TextArea id="myTA" text="Select a menu item"/>
        <s:Button label="Open Menu"
            click="mx.core.FlexGlobals.topLevelApplication.viewMenuOpen=true;"/>
        <s:Button label="Close Menu"
            click="mx.core.FlexGlobals.topLevelApplication.viewMenuOpen=false;"/>
    </s:VGroup>
</s:View>
```

Dans cet exemple, vous utilisez la propriété `View.viewMenuItems` pour ajouter cinq éléments de menu, où chaque élément de menu est représenté par un contrôle ViewMenuItem. Chaque contrôle ViewMenuItem utilise la propriété `label` pour spécifier le texte qui apparaît dans le menu correspondant à cet élément.

Remarquez que vous ne définissez pas explicitement le conteneur ViewMenu. Le conteneur View crée automatiquement une instance du conteneur ViewMenu pour contenir les contrôles ViewMenuItem.

Utilisez le style `icône` ducontrôle de ViewMenuItem.

Le contrôle ViewMenuItem définit la propriété de style `icône` que vous pouvez utiliser pour inclure une image. Vous pouvez utiliser le style `icône` avec ou sans la propriété `label`.

Gestion de l'événement `clic` du contrôle ViewMenuItem

Chaque contrôle de ViewMenuItem définit également un gestionnaire d'événement pour l'événement `clic`. Le contrôle ViewMenuItem envoie l'événement `clic` lorsque l'utilisateur sélectionne l'événement. Dans cet exemple, tous les éléments de menu utilisent le même gestionnaire d'événement. Vous pouvez toutefois choisir de définir un gestionnaire d'événement distinct pour chaque événement `clic`.

Ouverture du contrôle ViewMenuItem par programmation

Vous ouvrez le menu en utilisant la touche de menu matérielle de votre périphérique. Cette application définit également deux contrôles Button pour ouvrir et fermer le menu par programmation.

Pour ouvrir le menu par programmation, définissez la propriété `viewMenuOpen` du conteneur d'application sur `true`. Pour fermer le menu, définissez la propriété sur `false`. La propriété `viewMenuOpen` est définie dans la classe ViewNavigatorApplicationBase, la classe de base des conteneurs ViewNavigatorApplication et TabbedViewNavigatorApplication.

Application d'un habillage aux composants ViewMenu et ViewMenuItem

Utilisez les habillages pour contrôler l'aspect des composants ViewMenu et ViewMenuItem. La classe d'habillages par défaut de ViewMenu est spark.skins.mobile.ViewMenuSkin. La classe d'habillages par défaut de ViewMenuItem est spark.skins.mobile.ViewMenuItemSkin.

Le blogueur Daniel Demmel explique comment appliquer un habillage Gingerbread black à la commande ViewMenu.

Les classes d'habillages utilisent les états d'habillage, tels que normal, fermé et désactivé, pour contrôler l'aspect de l'habillage. Les habillages définissent également les transitions permettant de contrôler l'aspect du menu lorsqu'il change d'état d'affichage.

Pour plus d'informations, voir « Notions de base sur l'habillage mobile » à la page 96.

Définition de la présentation d'un conteneur ViewMenu

La classe ViewMenuLayout définit la présentation du menu des vues. Le menu peut contenir plusieurs lignes selon le nombre d'éléments de menu.

Règles de présentation de ViewMenuItem

La propriété `requestedMaxColumnCount` de la classe ViewMenuLayout définit le nombre maximum d'éléments de menu sur une ligne. Par défaut, la propriété `requestedMaxColumnCount` est définie sur trois.

Les règles suivantes définissent la manière dont la classe ViewMenuLayout réalise la présentation :

- Si vous définissez trois éléments de menu ou moins, où la propriété `requestedMaxColumnCount` contient la valeur par défaut de trois, les éléments de menu s'affichent sur une seule ligne. Tous les éléments de menu ont la même taille.

  Si vous définissez quatre éléments de menu ou plus, soit davantage d'éléments de menu que ceux spécifiés par la propriété `requestedMaxColumnCount`, le conteneur ViewMenu crée plusieurs lignes.

- Si le nombre d'éléments de menu est divisible par la propriété `requestedMaxColumnCount`, chaque ligne contient le même nombre d'éléments de menu. Tous les éléments de menu ont la même taille.

  Par exemple, la propriété `requestedMaxColumnCount` est définie sur la valeur par défaut de trois et vous définissez six éléments de menu. Le menu affiche deux lignes, chacune contenant trois éléments de menu.

- Si le nombre d'éléments de menu n'est pas divisible par la propriété `requestedMaxColumnCount`, les lignes peuvent contenir un nombre différent d'éléments de menu. La taille des éléments de menu dépend du nombre d'éléments de menu par ligne.

  Par exemple, la propriété `requestedMaxColumnCount` est définie sur la valeur par défaut de trois et vous définissez huit éléments de menu. Le menu affiche trois lignes. La première ligne contient deux éléments de menu. Les deuxième et troisième lignes contiennent chacune trois éléments.

Création d'une présentation ViewMenuItem personnalisée

La classe ViewMenuLayout contient des propriétés permettant de modifier les espaces entre les éléments de menu et le nombre par défaut d'éléments de menu figurant sur chaque ligne. Vous pouvez aussi créer votre propre présentation personnalisée pour le menu en créant votre classe de présentations.

Par défaut, la classe spark.skins.mobile.ViewMenuSkin définit l'habillage du conteneur ViewMenu. Pour appliquer une classe ViewMenuLayout personnalisée au conteneur ViewMenu, définissez une nouvelle classe d'habillages pour le conteneur ViewMenu.

La classe ViewMenuSkin par défaut inclut une définition pour un conteneur Group nommé `contentGroup`, comme le montre l'exemple suivant :

```
...
    <s:Group id="contentGroup" left="0" right="0" top="3" bottom="2"
        minWidth="0" minHeight="0">
        <s:layout>
            <s:ViewMenuLayout horizontalGap="2" verticalGap="2" id="contentGroupLayout"
                requestedMaxColumnCount="3"
                requestedMaxColumnCount.landscapeGroup="6"/>
        </s:layout>
    </s:Group>
...
```

Votre classe d'habillages doit également définir un conteneur nommé `contentGroup`. Ce conteneur utilise la propriété `layout` pour spécifier votre classe de présentation personnalisée.

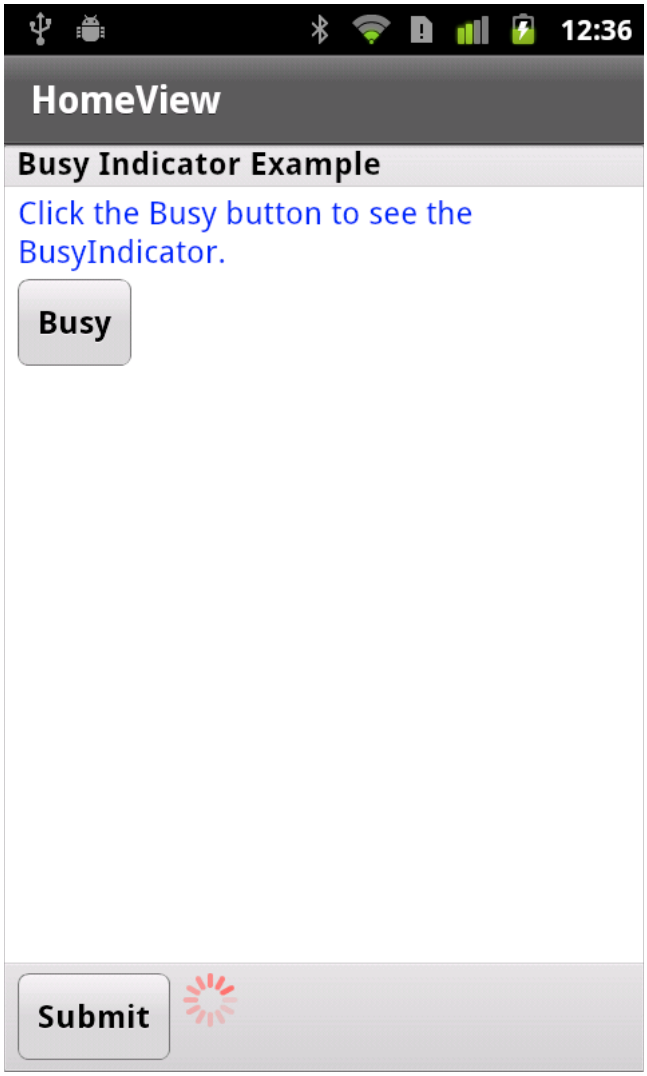
Vous pouvez alors appliquer votre classe d'habillages personnalisés dans l'application, comme le montre l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\ViewMenuSkin.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.ViewMenuHome">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|ViewMenu {
            skinClass: ClassReference("skins.MyVMSkin");
        }
    </fx:Style>
</s:ViewNavigatorApplication>
```

Affichage d'une indication visuelle pour une opération longue durée dans une application mobile

Le contrôle Spark BusyIndicator affiche un bouton fléché à 12 rayons. Vous utilisez le contrôle BusyIndicator pour fournir une indication visuelle signalant qu'une opération de longue durée est en cours.

La figure suivante présente le contrôle BusyIndicator dans la zone de barre de contrôle d'un conteneur Spark Panel, à côté du bouton Envoyer :



Rendez visible le contrôle BusyIndicator pendant le déroulement d'une opération de longue durée. Une fois l'opération terminée, masquez le contrôle.

Par exemple, vous pouvez créer une instance du contrôle BusyIndicator dans un gestionnaire d'événement, éventuellement le gestionnaire d'événement qui démarre le processus de longue durée. Dans le gestionnaire d'événement, appelez la méthode `addElement()` pour ajouter le contrôle à un conteneur. Une fois le processus terminé, appelez `removeElement()` pour supprimer le contrôle BusyIndicator du conteneur.

Une autre option consiste à utiliser la propriété `visible` du contrôle pour l'afficher et le masquer. Dans l'exemple suivant, vous ajoutez le contrôle BusyIndicator à la zone de barre de contrôle d'un conteneur Spark Panel dans un conteneur View :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\SimpleBusyIndicatorHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">

    <s:Panel id="panel" title="Busy Indicator Example"
        width="100%" height="100%">
        <s:controlBarContent>
            <s:Button label="Submit" />
            <s:BusyIndicator id="bi"
                visible="false"
                symbolColor="red"/>
        </s:controlBarContent>

        <s:VGroup left="10" right="10" top="10" bottom="10">
            <s:Label width="100%" color="blue"
                text="Click the Busy button to see the BusyIndicator."/>
            <s:Button label="Busy"
                click="{bi.visible = !bi.visible}" />
        </s:VGroup>
    </s:Panel>
</s:View>
```

Dans cet exemple, la propriété `visible` du contrôle BusyIndicator est initialement définie sur `false` pour être masquée. Cliquez sur le bouton Occupé pour définir la propriété `visible` sur `true` afin d'afficher le contrôle.

Le contrôle BusyIndicator ne tourne que lorsqu'il est visible. Par conséquent, lorsque vous définissez la propriété `visible` sur `false`, le contrôle ne requiert aucun cycle de traitement.

Remarque : *la définition de la propriété `visible` sur `false` masque le contrôle, mais celui-ci est encore inclus dans la présentation de son conteneur parent. Pour exclure le contrôle de la présentation, définissez les propriétés `visible` et `includeInLayout` sur `false`.*

Le contrôle Spark BusyIndicator ne prend pas en charge les habillages. Vous pouvez cependant utiliser les styles pour définir la couleur et l'intervalle de rotation du bouton fléché. Dans l'exemple précédent, vous définissez la couleur de l'indicateur à l'aide de la propriété `symbolColor`.

Définition de transitions dans une application mobile

Les transitions entre vues Spark définissent la manière dont le passage d'un conteneur View à un autre apparaît à l'écran. Les transitions fonctionnent en appliquant une animation au cours du changement de vue. Utilisez les transitions pour créer des interfaces attrayantes pour vos applications mobiles.

Par défaut, le conteneur View existant glisse hors de l'écran tandis que la nouvelle vue glisse sur l'écran. Vous pouvez aussi personnaliser ces transitions. Par exemple, votre application définit un formulaire dans le conteneur View qui présente seulement quelques champs, mais un conteneur View suivant présente des champs supplémentaires. Au lieu de glisser d'une vue à l'autre, vous pouvez utiliser une transition par retournement ou fondu-enchaîné.

Flex fournit les classes suivantes de transition entre vues que vous pouvez utiliser lorsque vous modifiez les conteneurs View :

Transition	Description
CrossFadeViewTransition	Effectue une transition par fondu-enchaîné entre la vue existante et la nouvelle vue. La vue existante s'efface progressivement pour laisser place à la nouvelle vue.
FlipViewTransition	Effectue une transition par retournement entre la vue existante et la nouvelle vue. Vous pouvez définir la direction et le type de retournement.
SlideViewTransition	Effectue une transition par glissement entre la vue existante et la nouvelle vue. La vue existante glisse hors de l'écran, tandis que la nouvelle vue glisse dans l'écran. Vous pouvez contrôler la direction et le type du glissement. Cette transition correspond à la transition entre vues par défaut utilisée par Flex.
ZoomViewTransition	Effectue une transition par zoom entre la vue existante et la nouvelle vue. Vous pouvez effectuer un zoom arrière sur la vue existante ou un zoom avant sur la nouvelle vue.

Remarque : *les transitions entre vues dans les applications mobiles ne sont pas associées aux transitions Flex standard. Les transitions Flex standard définissent les effets affichés au cours d'un changement d'état. Les opérations de navigation du conteneur ViewNavigator déclenchent les transitions entre vues. Les transitions entre vues ne peuvent pas être définies dans MXML et elles n'interagissent pas avec les états.*

Holly Schinsky, experte Flex certifiée par Adobe, a publié un article sur son blog concernant l'utilisation des transitions entre les vues mobiles dans Flex 4.5.

Brent Arnold, expert Flex certifié par Adobe, a conçu un didacticiel vidéo sur l'utilisation des transitions entre les vues mobiles.

Application d'une transition

Vous appliquez une transition lorsque vous modifiez le conteneur View actif. Dans la mesure où les transitions entre vues interviennent lorsque vous changez de conteneur View, vous les contrôlez par le biais du conteneur ViewNavigator.

Par exemple, vous pouvez utiliser les méthodes suivantes de la classe ViewNavigator pour changer la vue actuelle :

- `pushView()`
- `popView()`
- `popToFirstView()`
- `popAll()`
- `replaceView()`

Ces méthodes utilisent toutes un argument facultatif qui définit la transition à utiliser lors du changement de vue.

Vous pouvez également changer la vue actuelle en utilisant les touches de navigation de votre périphérique, telles que le bouton retour. Lorsque vous changez la vue à l'aide d'une touche matérielle, ViewNavigator utilise les transitions par défaut définies par les propriétés `ViewNavigator.defaultPopTransition` et `ViewNavigator.defaultPushTransition`. Par défaut, ces propriétés spécifient l'utilisation de la classe SlideViewTransition.

L'exemple suivant montre le fichier d'application principal qui initialise les propriétés `defaultPopTransition` et `defaultPushTransition` de façon à utiliser une transition FlipViewTransition :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkViewTrans.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainViewTrans"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.transitions.FlipViewTransition;

            // Define a flip transition.
            public var flipTrans:FlipViewTransition = new FlipViewTransition();

            // Set the default push and pop transitions of the navigator
            // to use the flip transition.
            protected function creationCompleteHandler(event:FlexEvent):void {
                navigator.defaultPopTransition = flipTrans;
                navigator.defaultPushTransition = flipTrans;
            }

            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                // Use the default pop view transition defined by
                // the ViewNavigator.defaultPopTransition property.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

    <s:navigationContent>
        <s:Button icon="@Embed(source='assets/Home.png')"
            click="button1_clickHandler(event)"/>
    </s:navigationContent>
</s:ViewNavigatorApplication>
```

La première vue, EmployeeMainViewTrans.mxml, définit une transition CrossFadeViewTransition. Elle transmet ensuite la transition CrossFadeViewTransition en tant qu'argument à la méthode `pushView()` lors d'un passage à la vue EmployeeView :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeMainViewTrans.mxml -->
<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;
            import spark.transitions.CrossFadeViewTransition;

            // Define two transitions: a cross fade and a flip.
            public var xFadeTrans:CrossFadeViewTransition = new CrossFadeViewTransition();

            // Use the cross fade transition on a push(),
            // with a duration of 100 ms.
            protected function myList_changeHandler(event:IndexChangeEvent):void {
                xFadeTrans.duration = 1000;
               navigator.pushView(views.EmployeeView, myList.selectedItem, null, xFadeTrans);
            }
        ]]>
    </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>
```

La vue EmployeeView est définie dans le fichier EmployeeView.mxml, comme le montre l'exemple suivant :

```
<?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>
```

Application d'une transition au contrôle ActionBar

Par défaut, le contrôle ActionBar n'est pas inclus dans la transition d'une vue à l'autre. A la place, le contrôle ActionBar utilise une transition par glissement lors du changement de vue, quelle que soit la transition spécifiée. Pour inclure le contrôle ActionBar dans la transition lors du changement de vue, définissez la propriété `transitionControlsWithContent` de la classe de transition sur `true`.

Utilisation d'une classe d'amortissement

Une transition se déroule en deux phases : une *phase d'accélération* suivie d'une *phase de décélération*. Vous pouvez modifier les propriétés d'accélération et de décélération d'une transition en utilisant une classe d'amortissement. L'amortissement permet de créer un rythme plus réaliste d'accélération et de décélération. Vous pouvez aussi utiliser une classe d'amortissement pour créer un effet de rebond ou contrôler d'autres types de mouvement.

Flex fournit les classes d'amortissement Spark dans le package spark.effects.easing. Ce package comprend des classes pour les types d'amortissement les plus courants, parmi lesquels Bounce, Linear et Sine (Rebond, Linéaire et Sinusoïdal). Pour plus d'informations sur l'utilisation de ces classes, voir Utilisation de classes d'amortissement Spark.

L'exemple suivant présente une modification de l'application définie dans la section précédente. Cette version ajoute une classe d'amortissement Bounce à la transition FlipView :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkViewTransEasier.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainViewTransEaser"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.transitions.FlipViewTransition;

            // Define a flip transition.
            public var flipTrans:FlipViewTransition = new FlipViewTransition();

            // Set the default push and pop transitions of the navigator
            // to use the flip transition.
            // Specify the Bounce class as the easer for the flip.
            protected function creationCompleteHandler(event:FlexEvent):void {
                flipTrans.easer = bounceEasing;
                flipTrans.duration = 1000;
                navigator.defaultPopTransition = flipTrans;
                navigator.defaultPushTransition = flipTrans;
```

```
            }

            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                // Use the default pop view transition defined by
                // the ViewNavigator.defaultPopTransition property.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

    <fx:Declarations>
        <s:Bounce id="bounceEasing"/>
    </fx:Declarations>

    <s:navigationContent>
        <s:Button icon="@Embed(source='assets/Home.png')"
                  click="button1_clickHandler(event)"/>
    </s:navigationContent>
</s:ViewNavigatorApplication>
```

Pour visualiser le rebond, veillez à utiliser le bouton Retour sur le périphérique.

Cycle de vie de la transition entre vues

Une transition entre vues s'effectue en deux phases principales : la phase de *préparation* et la phase d'*exécution*.

Trois méthodes de la classe de transition définissent la phase de préparation. Ces méthodes sont appelées dans l'ordre suivant :

1 `captureStartValues()`

   Lorsque cette méthode est appelée, le conteneur ViewNavigator a créé la nouvelle vue mais n'a pas validé la nouvelle vue ou mis à jour le contenu du contrôle ActionBar et de la barre d'onglets. Utilisez cette méthode pour capturer les valeurs initiales des composants qui jouent un rôle dans la transition.

2 `captureEndValues()`

   Lorsque cette méthode est appelée, la nouvelle vue a été entièrement validée et le contrôle ActionBar et la barre d'onglets reflètent l'état de la nouvelle vue. La transition peut utiliser cette méthode pour capturer les éventuelles valeurs dont elle a besoin à partir de la nouvelle vue.

3 `prepareForPlay()`

   Cette méthode permet à la transition d'initialiser l'instance d'effet utilisée pour animer les composants de la transition.

La phase d'exécution débute lorsque le conteneur ViewNavigator appelle la méthode `play()` de la transition. A ce stade, la nouvelle vue a été créée et validée, et le contrôle ActionBar et la barre d'onglets ont été initialisés. La transition envoie un événement `start` et les éventuelles instances d'effet créées au cours de la phase de préparation sont maintenant invoquées en appelant la méthode `play()` de l'effet.

Une fois la transition entre vues terminée, la transition envoie un événement `end`. La classe de base des transitions ViewTransitionBase définit la méthode `transitionComplete()` que vous pouvez appeler pour envoyer l'événement `end`. Il est important que la transition nettoie les éventuels objets temporaires et supprime les écouteurs qu'elle a créés avant d'envoyer l'événement de fin.

Après l'appel de la méthode `transitionComplete()`, le conteneur ViewNavigator finalise le processus de changement de vue et réinitialise la transition à son état non initialisé.

Chapitre 4 : Conception d'applications et flux de travail

Activation de la persistance dans une application mobile

Une application destinée à un périphérique mobile est souvent interrompue par d'autres actions, telles qu'un texto, un appel téléphonique ou d'autres applications mobiles. Généralement, lorsqu'une application interrompue est relancée, l'utilisateur s'attend à ce que l'état précédent de l'application soit restauré. Le mécanisme de persistance permet au périphérique de restaurer l'application dans son état antérieur.

La structure Flex fournit deux types de persistance pour les applications mobiles. La *persistance en mémoire* conserve les données de la vue lorsque l'utilisateur navigue dans l'application. La *persistance de session* restaure les données si l'utilisateur quitte l'application, puis la redémarre. La persistance de session est importante dans les applications mobiles car un système d'exploitation mobile peut fermer des applications à tout moment (par exemple, lorsque la mémoire est insuffisante).

Le blogueur Steve Mathews a créé une entrée de cookbook sur la persistance des données simples dans une application mobile Flex 4.5.

La blogueuse Holly Schinsky a communiqué sur la persistance et la gestion des données dans la gestion des données mobiles de Flex 4.5.

Persistance en mémoire

Les conteneurs de type Vue prennent en charge la persistance en mémoire à travers la propriété `View.data`. La propriété `data` d'une vue existante est automatiquement enregistrée lorsque la section sélectionnée change ou qu'une nouvelle vue est placée sur la pile ViewNavigator, provoquant l'élimination de la vue existante. La propriété `data` de la vue est restaurée lorsque le contrôle revient à la vue et que celle-ci est réinstanciée et activée. Par conséquent, la persistance en mémoire permet de conserver les informations d'état d'une vue au moment de l'exécution.

Persistance de session

La persistance de session conserve des informations d'état relatives à l'application entre deux exécutions de l'application. Les conteneurs ViewNavigatorApplication et TabbedViewNavigatorApplication définissent la propriété `persistNavigatorState` pour l'implémentation de la persistance de session. Définissez `persistNavigatorState` sur `true` pour activer la persistance de session. Par défaut, `persistNavigatorState` est défini sur `false`.

Si elle est activée, la persistance de session enregistre l'état de l'application sur disque en utilisant un objet partagé local, nommé `FxAppCache`. L'application peut également faire appel à des méthodes du spark.managers.PersistenceManager pour enregistrer des informations supplémentaires dans l'objet partagé local.

Persistance de session ViewNavigator

Le conteneur ViewNavigator prend en charge la persistance de session en enregistrant l'état de sa pile de vues sur disque lors de l'arrêt de l'application. Cet enregistrement inclut la propriété `data` de la vue actuelle.

Lors du redémarrage de l'application, la pile de ViewNavigator est réinitialisée et l'utilisateur obtient la même vue et le même contenu que ceux visibles lors de l'arrêt de l'application. Dans la mesure où la pile contient une copie de la propriété `data` pour chaque vue, les vues qui appartenaient auparavant à la pile peuvent être recréées lorsqu'elles sont activées.

Persistance de session TabbedViewNavigator

Pour le conteneur TabbedViewNavigator, la persistance de session enregistre l'onglet actuellement sélectionné dans la barre d'onglets à la fermeture de l'application. L'onglet correspond au conteneur ViewNavigator et à la pile de vues qui compose l'onglet. L'enregistrement comprend la propriété `data` de la vue actuelle. Ainsi, lors du redémarrage de l'application, l'onglet actif et le conteneur ViewNavigator associé sont restaurés à l'état auquel ils se trouvaient lors de l'arrêt de l'application.

Remarque : *dans le cas d'une application définie par le conteneur TabbedViewNavigatorApplication, seule la pile du conteneur ViewNavigator actif est enregistrée. Ainsi, lors du redémarrage de l'application, seul l'état du conteneur ViewNavigator actif est restauré.*

Représentation des données de persistance de session

Le mécanisme de persistance utilisé par Flex n'est ni chiffré ni protégé. Par conséquent, les données conservées sont stockées dans un format qui peut facilement être interprété par un autre programme ou un autre utilisateur. N'utilisez pas ce mécanisme de persistance pour conserver des informations sensibles telles que les identifiants de connexion. Vous avez la possibilité de rédiger votre propre gestionnaire de persistance qui vous offre une meilleure protection. Pour plus d'informations, voir « Personnalisation du mécanisme de persistance » à la page 70.

Utilisation de la persistance de session

L'exemple suivant définit la propriété `persistNavigatorState` sur `true` d'une application, ce qui permet d'activer la persistance de session :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSectionPersist.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView"
    persistNavigatorState="true">

    <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>
```

Cette application utilise le fichier EmployeeMainView.mxml en tant que première vue. Celui-ci définit un contrôle Liste qui vous permet de sélectionner un nom d'utilisateur :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeMainView.mxml -->
<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>
```

Pour afficher la persistance de session, ouvrez l'application et sélectionnez « Dave » dans le contrôle Liste pour accéder à la vue 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>
```

La vue EmployeeView.mxml affiche les données relatives à « Dave ». Ensuite, quittez l'application. Au redémarrage de l'application, la vue EmployeeView.mxml est à nouveau disponible et affiche les même données que celles visibles lors de l'arrêt de l'application.

Accès aux données dans un objet partagé local

Les informations dans un objet partagé local sont enregistrées sous la forme d'une paire clé:valeur. Les méthodes de PersistenceManager, par exemple `setProperty()` et `getProperty()`, reposent sur la clé permettant d'accéder à la valeur associée dans l'objet partagé local.

Vous pouvez utiliser la méthode `setProperty()` permettant d'enregistrer vos propres paires clé:valeur dans l'objet partagé local. La méthode `setProperty()` présente la signature suivante :

```
setProperty(key:String, value:Object):void
```

Utilisez la méthode `getProperty()` pour accéder à la valeur pour une clé spécifique. La méthode `getProperty()` présente la signature suivante :

```
getProperty(key:String):Object
```

Lorsque la propriété `persistNavigatorState` est définie sur `true`, le gestionnaire de persistance enregistre automatiquement deux paires clé:valeur dans l'objet partagé local lors de l'arrêt de l'application :

- `applicationVersion`

  Version de l'application telle que décrite par le fichier application.xml.

- `navigatorState`

  Etat de vue du navigateur, correspondant à la pile du conteneur ViewNavigator actif.

Réalisation de la persistance manuelle

Si la propriété `persistNavigatorState` est définie sur `true`, Flex réalise automatiquement la persistance de session. Il vous est tout de même possible de conserver des données d'application lorsque la propriété `persistNavigatorState` est définie sur `false`. Dans ce cas, implémentez votre propre mécanisme de persistance en faisant appel aux méthodes du PersistenceManager.

Utilisez les méthodes `setProperty()` et `getProperty()` pour enregistrer et lire les informations de l'objet partagé local. Appelez la méthode `load()` pour initialiser le PersistenceManager. Appelez les méthodes `save()` pour enregistrer des données sur disque.

Remarque : *si la propriété `persistNavigatorState` est définie sur `false`, Flex n'enregistre pas automatiquement la pile de vues du conteneur ViewNavigator actif lors de l'arrêt de l'application, ni ne la restaure lors du démarrage de l'application.*

Gestion des événements de persistance

Vous pouvez utiliser les événements suivants des conteneurs d'application mobile pour développer un mécanisme de persistance personnalisé :

- `navigatorStateSaving`
- `navigatorStateLoading`

Vous pouvez annuler l'enregistrement de l'état d'une application en appelant la méthode `preventDefault()` dans le gestionnaire de l'événement `navigatorStateSaving`. Annulez le chargement de l'application au redémarrage en appelant la méthode `preventDefault()` dans le gestionnaire de l'événement `navigatorStateLoading`.

Personnalisation du mécanisme de persistance

Lorsque la persistance de session est activée, l'application s'ouvre sur la vue qui était affichée au moment du dernier arrêt de l'application. Vous devez stocker suffisamment d'informations dans la propriété `data` de la vue (ou ailleurs, par exemple en tant qu'objet partagé) afin de pouvoir restaurer complètement l'état de l'application.

Par exemple, la vue restaurée doit effectuer des calculs basés sur la propriété `data` de la vue. L'application doit alors reconnaître le moment où elle est redémarrée et réaliser les calculs nécessaires. Il est également possible de substituer les méthodes `serializeData()` et `deserializePersistedData()` de la vue de sorte à réaliser vos propres actions à l'arrêt ou au redémarrage de l'application.

Types de données intégrées pris en charge pour la persistance de session

Le mécanisme de persistance prend automatiquement en charge tous les types de données intégrées, y compris : Number, String, Array, Vector, Object, uint, int et Boolean. Ces types de données sont enregistrées automatiquement par le mécanisme de persistance.

Classes personnalisées prises en charge pour la persistance de session

De nombreuses applications font appel à des classes personnalisées pour définir les données. Si une classe personnalisée contient des propriétés définies par des types de données intégrées, le mécanisme de persistance peut automatiquement enregistrer et charger la classe. Toutefois, vous devez enregistrer la classe à l'aide du mécanisme de persistance en appelant la méthode flash.net.registerClassAlias(). Généralement, vous appelez cette méthode dans l'événement `preinitialize` de l'application, avant que le stockage de persistance ne soit initialisé ou que des données y soient enregistrées.

Si vous définissez une classe complexe, une qui utilise des types de données autres que ceux des données intégrées, vous devez convertir ce type de données en type pris en charge, par exemple, String. De même, si la classe définit des variables privées, celles-ci ne sont pas automatiquement conservées. Pour prendre en charge la classe complexe dans le mécanisme de persistance, la classe doit implémenter l'interface flash.utils.IExternalizable. Cette interface nécessite que la classe implémente les méthodes `writeExternal()` et `readExternal()` permettant d'enregistrer et de restaurer une instance de la classe.

Prise en charge de plusieurs tailles d'écran et valeurs PPP dans une application mobile

Consignes pour la prise en charge de plusieurs tailles d'écran et valeurs PPP

Pour développer une application indépendante de la plateforme, soyez conscient des différents périphériques de sortie. Les périphériques peuvent avoir une taille, ou résolution, d'écran différente et une valeur PPP, ou densité, différente.

L'ingénieur Flex Jason SJ présente deux approches différentes en termes de création d'applications mobiles indépendantes de la résolution sur son blog.

Terminologie

La *résolution* correspond au nombre de pixels en hauteur fois le nombre de pixels en largeur : à savoir le nombre total de pixels qu'un périphérique prend en charge.

La valeur *PPP* est le nombre de points par pouce carré : à savoir la densité de pixels sur l'écran d'un périphérique. Le terme PPP peut également être interprété comme le nombre de pixels par pouce.

Prise en charge des valeurs PPP par Flex

Les fonctionnalités Flex ci-dessous simplifient le processus de création d'applications indépendantes de la résolution et de la valeur PPP :

Habillages Habillages prenant en charge les valeurs PPP pour les composants mobiles. Les habillages mobiles par défaut n'ont pas besoin de codage supplémentaire pour être adaptés à la résolution de la plupart des écrans.

applicationDPI Une propriété qui définit la taille pour laquelle vos habillages personnalisés sont créés. Supposez que vous définissez cette propriété sur une valeur PPP quelconque et qu'un utilisateur exécute l'application sur un périphérique doté d'une valeur PPP différente. Flex redimensionne tous les élément d'une application sur la valeur PPP du périphérique utilisé.

Les habillages mobiles par défaut sont dépendants des PPP, avec ou sans redimensionnement des PPP. En conséquence, si vous n'utilisez pas de composants présentant des tailles statiques ou des habillages personnalisés, vous n'avez généralement pas besoin de définir la propriété `applicationDPI`.

Présentations dynamiques

Les présentations dynamiques vous permettent de surmonter les différences de résolution. Par exemple, la définition de la largeur d'un contrôle sur 100 % remplit toujours toute la largeur de l'écran, que sa résolution soit de 480x854 ou de 480x800.

Définition de la propriété applicationDPI

Lorsque vous créez des applications indépendantes de la densité, vous pouvez définir la valeur PPP cible sur la balise d'application racine. (Pour les applications mobiles, la balise racine est `<s:ViewNavigatorApplication>`, `<s:TabbedViewNavigatorApplication>` ou `<s:Application>`.)

Vous définissez la valeur de la propriété `applicationDPI` sur 160, 240 ou 320, selon la résolution approximative de votre périphérique cible. Par exemple :

```
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.DensityView1"
    applicationDPI="320">
```

Lorsque vous définissez la propriété `applicationDPI`, vous définissez effectivement une échelle pour l'application lorsqu'elle est comparée à la résolution réelle du périphérique cible (`runtimeDPI`) au moment de l'exécution. Par exemple, si vous définissez la propriété `applicationDPI` sur 160 et que le périphérique possède une valeur `runtimeDPI` de 160, le facteur d'échelle est de 1 (pas de redimensionnement). Si vous définissez la propriété `applicationDPI` sur 240, le facteur d'échelle est de 1,5 (Flex agrandit tout de 150 %). Avec une valeur de 320, le facteur d'échelle est de 2 et Flex agrandit tout de 200 %.

Dans certains cas, un redimensionnement par un facteur non entier peut générer des artefacts indésirables en raison de l'interpolation, comme par exemple des lignes floues.

Désactivation du redimensionnement des PPP

Pour désactiver le redimensionnement des PPP pour l'application, ne définissez pas la valeur de la propriété `applicationDPI`.

Compréhension des propriétés applicationDPI et runtimeDPI

Le tableau suivant décrit deux propriétés de la classe Application qui interviennent lors de l'utilisation d'applications à des résolutions différentes :

Propriété	Description
`applicationDPI`	La densité cible ou PPP de l'application. Lorsque vous spécifiez une valeur pour cette propriété, Flex applique un facteur d'échelle à l'application racine. Par conséquent, une application conçue pour une valeur PPP est redimensionnée et présente un aspect correct sur un autre périphérique doté d'une valeur PPP différente. Le facteur d'échelle est calculé en comparant la valeur de cette propriété à la propriété `runtimeDPI`. Ce facteur d'échelle est appliqué à toute l'application, y compris les éléments de préchargement, les pop-ups et tous les composants de l'image. Lorsqu'elle n'est pas spécifiée, cette propriété retourne la même valeur que la propriété `runtimeDPI`. Cette propriété ne peut pas être définie dans ActionScript ; elle ne peut être définie que dans MXML. Vous ne pouvez pas modifier la valeur de cette propriété au moment de l'exécution.
`runtimeDPI`	La densité, ou valeur PPP, du périphérique sur lequel l'application est en cours d'exécution. Retourne la valeur de la propriété `Capabilities.screenDPI`, arrondie à l'une des constantes définies par la classe DPIClassification. Cette propriété est en lecture seule.

Création d'applications indépendantes de la résolution et des PPP

Les applications indépendantes de la résolution et de la valeur PPP présentent les caractéristiques suivantes :

Images Les images vectorielles sont redimensionnées en toute transparence pour correspondre à la résolution réelle du périphérique cible. En revanche, les images bitmap ne sont pas toujours redimensionnées aussi efficacement. Dans ce cas, vous pouvez charger les images bitmap à des résolutions différentes, en fonction de la résolution du périphérique, en utilisant la classe MultiDPIBitmapSource.

Texte La taille de la police de texte (pas le texte lui-même) est redimensionnée pour correspondre à la résolution.

Structures Utilisez des présentations dynamiques pour vous assurer que l'application présente un aspect correct une fois redimensionnée. En général, évitez d'utiliser des présentations basées sur les contraintes lorsque vous spécifiez les bordures des pixels par des valeurs absolues. Si vous utilisez des contraintes, choisissez la propriété de la valeur `applicationDPI` en tenant compte du redimensionnement.

Redimensionnement N'utilisez pas les propriétés `scaleX` et `scaleY` sur l'objet Application. Lorsque vous définissez la propriété `applicationDPI`, Flex se charge du redimensionnement.

Styles Vous pouvez utiliser des feuilles de style pour personnaliser les propriétés de style pour le système d'exploitation du périphérique cible et les paramètres PPP de l'application.

Habillages Les habillages Flex du thème mobile utilisent la valeur PPP de l'application pour déterminer les éléments à utiliser au moment de l'exécution. Tous les éléments d'habillage visuels définis par des fichiers FXG sont adaptés au périphérique cible.

Taille de l'application Ne définissez pas explicitement la hauteur et la largeur de l'application. De même, lors du calcul de la taille des composants personnalisés ou des popups, n'utilisez pas les propriétés `stageWidth` et `stageHeight`. Utilisez de préférence la propriété `SystemManager.screen`.

Détermination de la valeur PPP de l'environnement d'exécution

Au démarrage de votre application, celle-ci obtient la valeur de la propriété `runtimeDPI` auprès de la propriété Flash Player `Capabilities.screenDPI`. Cette propriété est mappée sur l'une des constantes définies par la classe DPIClassification. Par exemple, un Droid s'exécutant à 232 PPP est mappé sur la valeur PPP 240 de l'environnement d'exécution. Les valeurs PPP des périphériques ne correspondent pas toujours exactement aux constantes DPIClassification (160, 240 ou 320). Ils sont plutôt mappés sur ces classifications, en fonction d'un certain nombre de valeurs cibles.

Les mappages sont les suivants :

Constante DPIClassification	160 PPP	240 PPP	320 PPP
PPP du périphérique réel	<200	>=200 et <280	>=280

Vous pouvez personnaliser ces mappages pour remplacer le comportement par défaut ou pour régler les périphériques qui signalent de manière incorrecte leur propre valeur PPP. Pour plus d'informations, voir « Remplacement de la valeur PPP par défaut » à la page 83.

Choix de la mise à l'échelle automatique ou non

Choisir d'utiliser la mise à l'échelle automatique (en définissant la valeur de la propriété `applicationDPI`) revient à trouver un compromis entre la commodité et le niveau de précision en termes de pixels. Si vous définissez la propriété `applicationDPI` de sorte que l'application soit automatiquement mise à l'échelle, Flex utilise les habillages destinés à la propriété `applicationDPI`. Flex adapte ces habillages à la densité réelle du périphérique. D'autres éléments de l'application et les positions de la présentation sont également adaptés.

Si vous souhaitez utiliser la mise à l'échelle automatique et que vous créez vos propres habillages ou éléments destinés à une valeur PPP unique, généralement vous effectuez les opérations suivantes :

- créer un jeu unique d'habillages et de présentations vue/composant qui sont destinés à la propriété `applicationDPI` que vous spécifiez ;
- créer plusieurs versions des éléments de bitmap utilisés dans vos habillages ou ailleurs dans votre application et spécifiez-les à l'aide de la classe MultiDPIBitmapSource. Les éléments vectoriels et le texte contenu dans vos habillages n'ont pas besoin de dépendre de la densité si vous utilisez la mise à l'échelle automatique.
- N'utilisez pas la règle `@media` dans vos feuilles de style, car l'application prend en compte une seule valeur PPP cible.
- testez l'application sur des périphériques de différentes densités pour vérifier si l'apparence de l'application mise à l'échelle est acceptable sur chaque périphérique ; En particulier, vérifiez les périphériques qui entraînent un redimensionnement par un facteur non entier. Par exemple, si `applicationDPI` a la valeur 160, testez l'application sur des périphériques dotés d'une valeur PPP de 240.

Si vous choisissez de ne pas utiliser la mise à l'échelle automatique (en ne définissant pas la propriété `applicationDPI`), obtenez la valeur `applicationDPI`. Utilisez cette propriété pour déterminer la classification PPP réelle du périphérique et adaptez l'application au moment de l'exécution en procédant comme suit :

- créez plusieurs jeux d'habillages et de présentations destinés à chaque classification PPP d'environnement d'exécution, ou créez un seul jeu d'habillages et de présentations qui s'adapte de façon dynamique aux différentes densités (les habillages Flex intégrés adoptent la deuxième approche : chaque classe d'habillage contrôle la propriété `applicationDPI` et se définit automatiquement en conséquence) ;
- utilisez des règles `@media` dans vos feuilles de style pour filtrer les règles CSS en fonction de la classification PPP du périphérique (généralement, les polices et les valeurs de marge sont personnalisées pour chaque valeur PPP) ;
- testez l'application sur des périphériques de différentes densités pour garantir l'adaptation appropriée des habillages et des présentations.

Sélection de styles en fonction des PPP

Flex inclut la prise en charge de l'application des styles en fonction du système d'exploitation cible et de la valeur PPP de l'application dans CSS. Vous appliquez les styles en utilisant la règle `@media` qui figure dans votre feuille de style. La règle `@media` fait partie de la spécification CSS ; Flex développe cette règle pour inclure des propriétés supplémentaires : `application-dpi` et `os-platform`. Vous utilisez ces propriétés pour appliquer de manière sélective les styles en fonction de la valeur PPP de l'application et de la plateforme sur laquelle l'application est exécutée.

L'exemple suivant définit la propriété de style par défaut du contrôle Spark Button, `fontSize`, sur 12. Si le périphérique utilise 240 PPP et s'exécute sur le système d'exploitation Android, la propriété `fontSize` est 10.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/MediaQueryValuesMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="320">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        @namespace mx "library://ns.adobe.com/flex/mx";
        s|Button {
            fontSize: 12;
        }
        @media (os-platform: "Android") and (application-dpi: 240) {
            s|Button {
                fontSize: 10;
            }
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

Valeurs de la propriété application-dpi

La propriété CSS `application-dpi` est comparée à la valeur de la propriété de style `applicationDPI` qui est définie sur l'application racine. Les valeurs valides pour la propriété CSS `application-dpi` sont les suivantes :

- `160`
- `240`
- `320`

Chacune des valeurs prises en charge pour `application-dpi` possède une constante correspondante dans la classe DPIClassification.

Valeurs de la propriété os-platform

La propriété CSS `os-platform` est mise en correspondance avec la valeur de la propriété `flash.system.Capabilities.version` de Flash Player. Les valeurs valides pour la propriété CSS `os-platform` sont les suivantes :

- `Android`
- `iOS`
- `Macintosh`
- `Linux`
- `QNX`

- `Windows`

La correspondance ne respecte pas la casse.

Si aucune des entrées ne correspond, Flex recherche une correspondance secondaire en comparant les trois premiers caractères avec la liste des plateformes prises en charge.

Valeurs par défaut des propriétés application-dpi et os-platform

Si vous ne définissez pas explicitement une expression contenant les propriétés `application-dpi` ou `os-platform`, toutes les expressions sont supposées correspondre.

Opérateurs dans la règle @media

La règle `@media` prend en charge les opérateurs courants "and" et "not". Elle prend également en charge les listes séparées par des virgules. La séparation des expressions par une virgule implique une condition "or".

Lorsque vous utilisez l'opérateur "not", le mot "not" doit être le premier mot-clé dans l'expression. Cet opérateur nie l'expression entière et pas seulement la propriété qui suit le mot "not". En raison du bogue SDK-29191, l'opérateur "not" doit être suivi d'un type de support, tel que "all", avant une ou plusieurs expressions.

L'exemple suivant présente l'utilisation de ces opérateurs courants :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/MediaQueryValuesMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="320">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        @namespace mx "library://ns.adobe.com/flex/mx";
        /* Every os-platform @ 160dpi */
        @media (application-dpi: 160) {
            s|Button {
                fontSize: 10;
            }
        }
        /* IOS only @ 240dpi */
        @media (application-dpi: 240) and (os-platform: "IOS") {
            s|Button {
                fontSize: 11;
            }
        }
        /* IOS at 160dpi or Android @ 160dpi */
        @media (os-platform: "IOS") and (application-dpi:160), (os-platform: "ANDROID") and
```

```
(application-dpi: 160) {
            s|Button {
                fontSize: 13;
            }
        }
        /* Every os-platform except Android @ 240dpi */
        @media not all and (application-dpi: 240) and (os-platform: "Android") {
            s|Button {
                fontSize: 12;
            }
        }
        /* Every os-platform except IOS @ any DPI */
        @media not all and (os-platform: "IOS") {
            s|Button {
                fontSize: 14;
            }
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

Sélection d'éléments bitmap en fonction des PPP

Les éléments d'image bitmap s'affichent généralement de façon optimale seulement à la résolution pour laquelle ils sont conçus. Cette limitation peut poser des problèmes lorsque vous concevez des applications pour différentes résolutions. La solution consiste à créer plusieurs bitmaps, chacun ayant sa propre résolution, puis à charger le fichier approprié en fonction de la valeur de la propriété `runtimeDPI` de l'application.

Les composants Spark BitmapImage et Image ont une propriété `source` de type Objet. En raison de cette propriété, vous pouvez passer une classe qui définit les éléments à utiliser. Dans ce cas, vous passez la classe MultiDPIBitmapSource pour mapper différentes sources, selon la valeur de la propriété `runtimeDPI`.

L'exemple suivant charge une image différente, selon les PPP :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/MultiSourceView3.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Image with MultiDPIBitmapSource">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;

            private function doSomething():void {
                /* The MultiDPIBitmapSource's source data. */
                myTA.text =
myImage.source.getSource(FlexGlobals.topLevelApplication.applicationDPI).toString();
            }


        ]]>
    </fx:Script>
    <s:Image id="myImage">
        <s:source>
            <s:MultiDPIBitmapSource
                    source160dpi="assets/low-res/bulldog.jpg"
                    source240dpi="assets/med-res/bulldog.jpg"
                    source320dpi="assets/high-res/bulldog.jpg"/>
        </s:source>
    </s:Image>
    <s:Button id="myButton" label="Click Me" click="doSomething()"/>
    <s:TextArea id="myTA" width="100%"/>
</s:View>
```

Lorsque vous utilisez les classes BitmapImage et Image avec MultiDPIBitmapSource dans une application de *bureau*, la propriété `source160dpi` est utilisée pour la source.

La propriété `icon` du contrôle Button utilise également une classe en tant qu'argument. En conséquence, vous pouvez aussi utiliser un objet MultiDPIBitmapSource en tant que source pour l'icône de Button. Vous pouvez définir la source de l'icône en ligne, comme le présente l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/MultiSourceView2.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Icons Inline">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;

            private function doSomething():void {
                /* The MultiDPIBitmapSource's source data. */
                myTA.text =
dogButton.getStyle("icon").getSource(FlexGlobals.topLevelApplication.applicationDPI).toStrin
g();
            }
        ]]>
    </fx:Script>
    <s:Button id="dogButton" click="doSomething()">
        <s:icon>
            <s:MultiDPIBitmapSource id="dogIcons"
                    source160dpi="@Embed('../../assets/low-res/bulldog.jpg')"
                    source240dpi="@Embed('../../assets/med-res/bulldog.jpg')"
                    source320dpi="@Embed('../../assets/high-res/bulldog.jpg')"/>
        </s:icon>
    </s:Button>
    <s:TextArea id="myTA" width="100%"/>
</s:View>
```

Vous pouvez également définir des icônes en les déclarant dans un bloc `<fx:Declarations>` et en attribuant la source par liaison des données, comme le montre l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/MultiSourceView1.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:mx="library://ns.adobe.com/flex/mx"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Icons in Declarations">
    <fx:Declarations>
        <s:MultiDPIBitmapSource id="dogIcons"
                source160dpi="@Embed('../../assets/low-res/bulldog.jpg')"
                source240dpi="@Embed('../../assets/med-res/bulldog.jpg')"
                source320dpi="@Embed('../../assets/high-res/bulldog.jpg')"/>
    </fx:Declarations>
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;

            private function doSomething():void {
                /* The MultiDPIBitmapSource's source data. */
                myTA.text =
dogIcons.getSource(FlexGlobals.topLevelApplication.applicationDPI).toString();
            }
        ]]>
    </fx:Script>
    <s:Button id="dogButton" icon="{dogIcons}" click="doSomething()"/>
    <s:TextArea id="myTA" width="100%"/>
</s:View>
```

Si la propriété `runtimeDPI` est mappe sur une propriété `source`*`XXX`*`dpi` qui est `null` ou sur une chaîne vide (""), Flash Player utilise la densité la plus élevée suivante comme source. Si cette valeur est également `null` ou vide, la densité la plus basse suivante est utilisée. Si cette valeur est *également*`null` ou vide, Flex attribue `null` comme source et aucune image n'est affichée. En d'autres termes, vous ne pouvez pas spécifier explicitement qu'aucune image ne doit être affichée pour une valeur PPP particulière.

Sélection des éléments d'habillage en fonction des PPP

La logique dans les constructeurs des habillages mobiles par défaut choisit les éléments en fonction de la valeur de la propriété `applicationDPI`. Ces classes sélectionnent les éléments qui correspondent le mieux à la valeur PPP cible. Lorsque vous créez des habillages personnalisés qui fonctionnent avec et sans redimensionnement des PPP, utilisez la propriété `applicationDPI` et non pas la propriété `runtimeDPI`.

Par exemple, la classe spark.skins.mobile.ButtonSkin utilise une instruction switch/case qui sélectionne les éléments FXG conçus pour des valeurs PPP particulières, comme ci-après :

```
switch (applicationDPI) {
    case DPIClassification.DPI_320: {
        upBorderSkin = spark.skins.mobile320.assets.Button_up;
        downBorderSkin = spark.skins.mobile320.assets.Button_down;
        ...
        break;
    }
    case DPIClassification.DPI_240: {
        upBorderSkin = spark.skins.mobile240.assets.Button_up;
        downBorderSkin = spark.skins.mobile240.assets.Button_down;
        ...
        break;
    }
}
```

Outre la sélection conditionnelle des éléments FXG, les classes d'habillages mobiles définissent également les valeurs des autres propriétés de style comme les écarts de présentation et les marges intérieures. Ces paramètres sont basés sur la valeur PPP du périphérique cible.

Non-définition de la propriété applicationDPI

Si vous ne définissez pas la propriété `applicationDPI`, les habillages utilisent par défaut la propriété `runtimeDPI`. Ce mécanisme garantit qu'un habillage dont les valeurs sont basées sur la propriété `applicationDPI`, et non pas sur la propriété `runtimeDPI`, utilise la ressource appropriée avec et sans redimensionnement des PPP.

Lors de la création d'habillages personnalisés, vous pouvez choisir d'ignorer le paramètre `applicationDPI`. Cela produit un habillage toujours redimensionné pour correspondre à la valeur PPP du périphérique cible, mais son aspect risque de ne pas être parfait si ses éléments ne sont pas spécifiquement conçus pour cette valeur PPP.

Utilisation de la propriété applicationDPI dans CSS

Utilisez la valeur de la propriété `applicationDPI` dans le sélecteur CSS `@media` pour personnaliser les styles utilisés par votre application pour périphérique mobile ou tablette sans créer d'habillages personnalisés. Pour plus d'informations, voir « Sélection de styles en fonction des PPP » à la page 75.

Détermination manuelle du facteur d'échelle et de la valeur PPP actuelle

Pour demander manuellement à une application de périphérique mobile ou de tablette de sélectionner des éléments en fonction de la valeur PPP du périphérique cible, vous pouvez calculer le facteur d'échelle au moment de l'exécution. Pour ce faire, vous divisez la valeur de la propriété `runtimeDPI` par celle de la propriété de style `applicationDPI` :

```
import mx.core.FlexGlobals;
var curDensity:Number = FlexGlobals.topLevelApplication.runtimeDPI;
var curAppDPI:Number = FlexGlobals.topLevelApplication.applicationDPI;
var currentScalingFactor:Number = curDensity / curAppDPI;
```

Vous pouvez utiliser le facteur de mise à l'échelle calculé pour sélectionner manuellement des éléments. L'exemple suivant définit les emplacements personnalisés des éléments bitmap pour chaque valeur PPP. Il charge ensuite une image à partir de cet emplacement personnalisé :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/DensityMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark"
     firstView="views.DensityView1"
     applicationDPI="240" initialize="initApp()">

    <fx:Script>
        <![CDATA[
                [Bindable]
                public var densityDependentDir:String;
                [Bindable]
                public var curDensity:Number;
                [Bindable]
                public var appDPI:Number;
                [Bindable]
                public var curScaleFactor:Number;

                public function initApp():void {
                    curDensity = runtimeDPI;
                    appDPI = applicationDPI;
                    curScaleFactor  =  appDPI / curDensity;
                    switch (curScaleFactor) {
                        case 1: {
                            densityDependentDir = "../../assets/low-res/";
                            break;
                        }
                        case 1.5: {
                            densityDependentDir = "../../assets/med-res/";
                            break;
                        }
                        case 2: {
                            densityDependentDir = "../../assets/high-res/";
                            break;
                        }
                    }
                }
        ]]>
    </fx:Script>

</s:ViewNavigatorApplication>
```

La vue qui utilise le facteur d'échelle est comme suit :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/DensityView1.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Home"
        creationComplete="initView()">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;
            [Bindable]
            public var imagePath:String;
            private function initView():void {
                label0.text = "App DPI:" + FlexGlobals.topLevelApplication.appDPI;
                label1.text = "Cur Density:" + FlexGlobals.topLevelApplication.curDensity;
          label2.text = "Scale Factor:" + FlexGlobals.topLevelApplication.curScaleFactor;
              imagePath = FlexGlobals.topLevelApplication.densityDependentDir + "bulldog.jpg";

                ta1.text = myImage.source.toString();
            }
        ]]>
    </fx:Script>

    <s:Image id="myImage" source="{imagePath}"/>
    <s:Label id="label0"/>
    <s:Label id="label1"/>
    <s:Label id="label2"/>
    <s:TextArea id="ta1" width="100%"/>
</s:View>
```

Remplacement de la valeur PPP par défaut

Une fois la valeur PPP de l'application définie, votre application est redimensionnée en fonction de la valeur PPP signalée par le périphérique sur lequel elle s'exécute. Dans certains cas, les périphériques signalent des valeurs PPP incorrectes ou vous souhaitez remplacer la méthode de sélection de la valeur PPP par défaut par une méthode de redimensionnement personnalisée.

Vous pouvez remplacer le comportement de redimensionnement par défaut d'une application en remplaçant les mappages PPP par défaut. Par exemple, si un périphérique signale à tort être doté de 240 PPP au lieu de 160 PPP, vous pouvez créer un mappage personnalisé qui recherche ce périphérique et le classe comme doté de 160 PPP.

Pour remplacer la valeur PPP d'un périphérique particulier, pointez la propriété `runtimeDPIProvider` de la classe Application sur une sous-classe de la classe RuntimeDPIProvider. Dans votre sous-classe, remplacez la méthode Get de `runtimeDPI` et ajoutez une logique fournissant un mappage de valeur PPP personnalisé. N'ajoutez pas de dépendances aux autres classes dans la structure, telles que UIComponent. Cette sous-classe ne peut appeler que les API Player.

L'exemple suivant définit un mappage de valeur PPP personnalisé pour un périphérique dont la propriété `Capabilities.os` correspond à "Mac 10.6.5" :

```
package {
import flash.system.Capabilities;
import mx.core.DPIClassification;
import mx.core.RuntimeDPIProvider;
public class DPITestClass extends RuntimeDPIProvider {
    public function DPITestClass() {
    }

    override public function get runtimeDPI():Number {
        // Arbitrary mapping for Mac OS.
        if (Capabilities.os == "Mac OS 10.6.5")
            return DPIClassification.DPI_320;

        if (Capabilities.screenDPI < 200)
            return DPIClassification.DPI_160;

        if (Capabilities.screenDPI <= 280)
            return DPIClassification.DPI_240;

        return DPIClassification.DPI_320;
    }
}
}
```

L'application ci-dessous utilise DPITestClass pour déterminer une valeur PPP de l'environnement d'exécution à utiliser pour le redimensionnement. Elle pointe sur la propriété de la classe ViewNavigatorApplication`runtimeDPIProvider` :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/DPIMappingOverrideMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark"
     firstView="views.DPIMappingView"
     applicationDPI="160"
     runtimeDPIProvider="DPITestClass">

</s:ViewNavigatorApplication>
```

L'exemple suivant illustre également une sous-classe de la classe RuntimeDPIProvider. Dans ce cas, la classe personnalisée vérifie la résolution x et y du périphérique pour déterminer si ce dernier signale de manière incorrecte sa valeur PPP :

```
package
{
import flash.system.Capabilities;
import mx.core.DPIClassification;
import mx.core.RuntimeDPIProvider;
public class SpecialCaseMapping extends RuntimeDPIProvider {
    public function SpecialCaseMapping() {
    }

    override public function get runtimeDPI():Number {
        /* A tablet reporting an incorrect DPI of 240. We could use
            Capabilities.manufacturer to check the tablet's OS as well. */
        if (Capabilities.screenDPI == 240 &&
            Capabilities.screenResolutionY == 1024 &&
            Capabilities.screenResolutionX == 600) {
            return DPIClassification.DPI_160;
        }

        if (Capabilities.screenDPI < 200)
            return DPIClassification.DPI_160;

        if (Capabilities.screenDPI <= 280)
            return DPIClassification.DPI_240;

        return DPIClassification.DPI_320;
    }
}
}
```

Chapitre 5 : Texte

Utilisation de texte dans une application mobile

Consignes liées au texte dans une application mobile

Certains contrôles de texte Spark ont été optimisés pour être utilisés dans les applications mobiles. Dans la mesure du possible, utilisez les contrôles de texte suivants :

- Spark TextArea
- Spark TextInput
- Spark Label (sauf si vous devez utiliser une police intégrée ; dans ce cas, utilisez les contrôles Spark TextArea ou TextInput)

Certaines considérations doivent être prises en compte lorsque vous utilisez des contrôles de texte dans une application iOS. L'ingénieur Flex Jason SJ les décrit sur son blog.

TLF dans une application mobile

D'une manière générale, évitez les contrôles de texte qui utilisent la structure TLF (Text Layout Framework) dans les applications mobiles. Les habillages mobiles des contrôles TextArea et TextInput sont optimisés pour les applications mobiles et, à la différence de leurs équivalents de type bureau, ils n'utilisent pas TLF. TLF est utilisé dans les applications de bureau pour fournir un ensemble riche de contrôles sur le rendu de texte.

Evitez d'utiliser les contrôles de texte suivants dans une application mobile, car ils utilisent TLF et leurs habillages ne sont pas optimisés pour les applications mobiles :

- Spark RichText
- Spark RichEditableText

Habillages pour les contrôles de texte mobiles

Lorsque vous créez une application mobile, Flex applique automatiquement le thème mobile. En conséquence, les contrôles basés sur le texte utilisent les habillages mobiles. Ces habillages sont optimisés pour les applications mobiles, mais ils ne prennent pas en charge les fonctions suivantes des habillages Spark standard :

- TLF
- Bidirectionnalité et mise en miroir
- Polices CFF (Compact Font Format) pour l'intégration
- RichEditableText pour le rendu de texte (à la place, les habillages mobiles utilisent StyleableTextField)

Saisie à l'aide d'un clavier logiciel

Lorsqu'un utilisateur met l'accent sur un contrôle de texte qui accepte la saisie, les périphériques mobiles sans clavier affichent un clavier logiciel à l'écran. Les développeurs ne contrôlent pas actuellement la configuration de ce clavier logiciel.

Utilisation du contrôle Spark Label dans une application mobile

Le contrôle Spark Label est idéalement adapté aux lignes uniques de texte non modifiable et non sélectionnable.

Le contrôle Label utilise FTE, lequel n'est pas aussi performant que les contrôles de texte qui ont été optimisés pour les applications mobiles, tels que TextInput et TextArea. Toutefois, le contrôle Label n'utilise pas TLF, de sorte qu'il est généralement plus performant que les contrôles tels que RichText et RichEditableText, lesquels implémentent TLF.

D'une manière générale, utilisez de manière limitée les contrôles Spark Label dans les applications mobiles. N'utilisez pas le contrôle Spark Label dans des habillages ou des rendus d'élément.

N'utilisez pas le contrôle Label lorsque vous intégrez des polices dans une application mobile, car le contrôle Label utilise CFF. Utilisez le contrôle TextArea à la place. Pour plus d'informations, voir « Intégration de polices dans une application mobile » à la page 93.

Utilisation du contrôle Spark TextArea dans une application mobile

Le contrôle Spark TextArea est un contrôle de saisie de texte qui permet aux utilisateurs de saisir et de modifier plusieurs lignes de texte. Le contrôle Spark TextArea a été optimisé pour les applications mobiles.

Dans une application mobile, le contrôle TextArea utilise la classe spark.skins.mobile.TextAreaSkin pour son habillage. Cet habillage utilise la classe StyleableTextField plutôt que la classe RichEditableText pour le rendu de texte. En conséquence, le contrôle TextArea ne prend pas en charge TLF. Il prend en charge uniquement un sous-ensemble de styles disponibles sur le contrôle TextArea avec l'habillage Spark de bureau.

Dans la mesure où le contrôle TextArea ne prend pas en charge TLF, vous ne pouvez pas utiliser le contenu `textFlow`, `content`, ni les propriétés `selectionHighlighting`. En outre, vous ne pouvez pas utiliser les méthodes suivantes :

- `getFormatOfRange()`
- `setFormatOfRange()`

Utilisation du contrôle Spark TextInput dans une application mobile

Le contrôle Spark TextInput est un contrôle de saisie de texte qui permet aux utilisateurs de saisir et de modifier une seule ligne de texte. Il a été optimisé pour les applications mobiles.

Dans une application mobile, le contrôle TextInput utilise la classe spark.skins.mobile.TextInputSkin pour son habillage. Cet habillage utilise la classe StyleableTextField plutôt que la classe RichEditableText pour le rendu de texte. En conséquence, le contrôle TextInput ne prend pas en charge TLF. Il prend en charge uniquement un sous-ensemble de styles disponibles sur le contrôle TextInput avec l'habillage Spark de bureau.

Dans certains cas (par exemple, lorsque vous souhaitez utiliser une police intégrée), remplacez un contrôle Label par un contrôle TextIntput. Pour que le contrôle TextInput se comporte davantage comme un contrôle Label, définissez les propriétés `editable` et `selectable` sur `false`. Vous pouvez également supprimer la bordure autour d'un contrôle TextInput en créant un habillage personnalisé. Pour plus d'informations, voir « Notions de base sur l'habillage mobile » à la page 96.

Utilisation des contrôles RichText et RichEditableText dans une application mobile

Essayez d'éviter d'utiliser les contrôles RichText et RichEditableText dans les applications mobiles. Ces contrôles ne possèdent pas d'habillage mobile et ne sont pas optimisés pour les applications mobiles. Si vous utilisez ces contrôles, vous utilisez TLF, ce qui exige une importante puissance de traitement.

Contrôles de texte MX

Vous ne pouvez pas utiliser des contrôles de texte MX tels que MX Text et MX Label dans les applications mobiles. Utilisez les équivalents Spark à la place.

Styles de texte dans une application mobile

Les styles pris en charge par la classe StyleableTextField déterminent les styles qui sont pris en charge par les contrôles de texte dans le thème mobile.

Les styles suivants sont les seuls pris en charge par les classes TextInput et TextArea dans une application mobile :

- `textAlign`
- `fontFamily`
- `fontWeight`
- `fontStyle`
- `color`
- `fontSize`
- `textDecoration`
- `textIndent`
- `leading`
- `letterSpacing`

Lorsque vous utilisez le thème mobile, le contrôle Label prend en charge le jeu de styles standard associé.

Contrôles de texte dans les habillages et les rendus d'élément dans une application mobile

Les contrôles de texte dans une application mobile utilisent le thème mobile. Dans le thème mobile, les habillages utilisent la classe StyleableTextField pour rendre le texte. Cette classe se trouve dans le groupement spark.components.supportClasses.*.

Par exemple, la classe TextAreaSkin mobile définit la propriété `textDisplay` comme suit :

```
textDisplay = StyleableTextField(createInFontContext(StyleableTextField));
```

Lorsque vous rendez du texte dans un habillage mobile personnalisé ou créez un rendu d'élément ActionScript à utiliser dans une application mobile, utilisez la classe StyleableTextField. Elle est optimisée pour les applications mobiles.

La classe StyleableTextField est une sous-classe légère de la classe Flash TextField. Elle implémente l'interface IEditableText (laquelle constitue elle-même une extension de IDisplayText).

La classe StyleableTextField n'implémente pas les interfaces IUIComponent ou ILayoutElement, de sorte qu'elle n'est pas utilisable directement dans MXML. Elle est conçue pour être utilisée dans les habillages ActionScript et les rendus d'élément ActionScript.

Pour plus d'informations sur l'habillage des composants mobiles, voir « Notions de base sur l'habillage mobile » à la page 96. Pour plus d'informations à propos des rendus d'élément ActionScript, voir Création d'un rendu d'élément Spark dans ActionScript.

Interventions de l'utilisateur liées à du texte dans une application mobile

Vous pouvez utiliser des gestes tels que le glissement avec les contrôles de texte. L'exemple suivant écoute un événement de glissement et vous indique la direction dans laquelle le glissement a été effectué :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/TextAreaEventsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="TextArea swipe event"
        viewActivate="view1_viewActivateHandler(event)">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            import flash.events.TransformGestureEvent;
            import mx.events.FlexEvent;

            protected function swipeHandler(event:TransformGestureEvent):void {
                // event.offsetX shows the horizontal direction of the swipe (1 is right, -1
is left)
                swipeEvent.text = event.type + " " + event.offsetX;
                if (swipeText.text.length == 0) {
                    swipeText.text = "Swipe again to make text go away."
                }
                else {
                    swipeText.text = "";
                }
            }

            protected function view1_viewActivateHandler(event:FlexEvent):void {
                swipeText.addEventListener(TransformGestureEvent.GESTURE_SWIPE,swipeHandler);
            }

        ]]>
    </fx:Script>
    <s:VGroup>
        <s:TextArea id="swipeText" height="379"
            editable="false" selectable="false"
            text="Swipe to make text go away."/>
        <s:TextInput id="swipeEvent" />
    </s:VGroup>

</s:View>
```

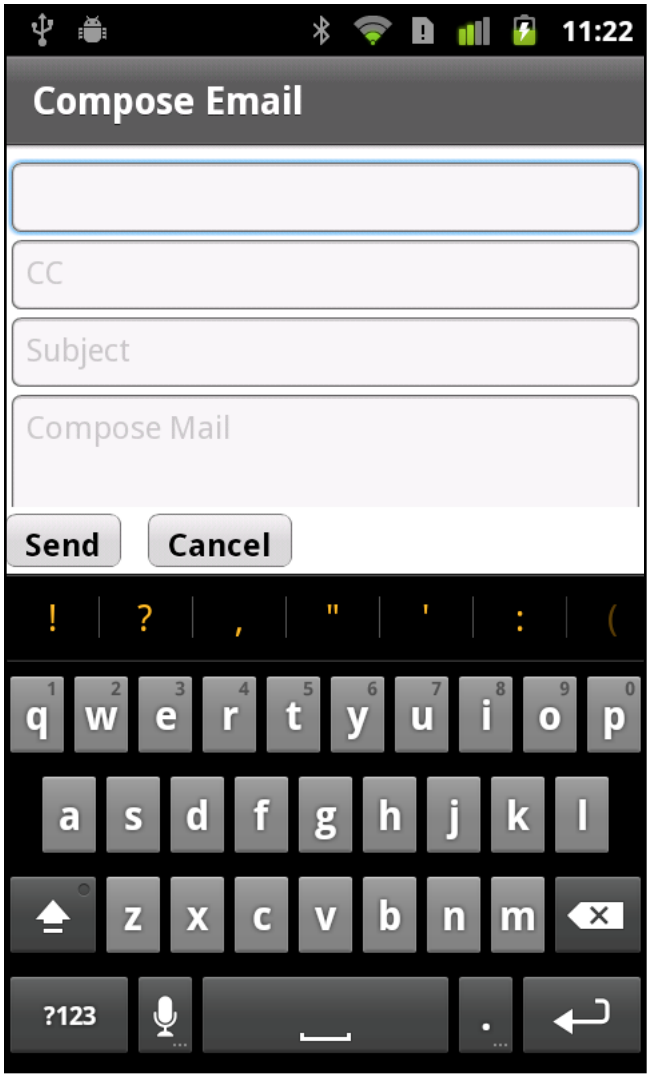
Les gestes de toucher-glisser sélectionnent toujours du texte, mais seulement si le contrôle de texte est sélectionnable ou modifiable. Dans certains cas, vous ne souhaiterez peut-être pas sélectionner du texte en effectuant un geste de toucher+glisser ou de glissement seul sur un contrôle de texte. Dans ce cas, définissez les propriétés `selectable` et `editable` sur `false` ou réinitialisez la sélection en appelant la méthode `selectRange(0,0)` dans le gestionnaire d'événement de glissement.

Si le texte se trouve dans un composant Scroller, ce dernier n'effectue un défilement que si le geste est effectué en dehors du composant de texte.

Prise en charge du clavier à l'écran dans une application mobile

De nombreux périphériques ne sont pas équipés d'un clavier physique. Ils utilisent un clavier qui apparaît à l'écran lorsque cela est nécessaire. Le clavier à l'écran, également appelé clavier logiciel ou virtuel, se ferme une fois que l'utilisateur a saisi les informations ou lorsqu'il annule l'opération.

La figure suivante montre une application qui utilise le clavier à l'écran :



Dans la mesure où le clavier occupe une partie de l'écran, Flex doit s'assurer qu'une application fonctionne toujours dans la zone réduite de l'écran. Par exemple, l'utilisateur sélectionne un contrôle TextInput, provoquant l'ouverture du clavier à l'écran. Une fois le clavier ouvert, Flex redimensionne automatiquement l'application en fonction de la zone d'écran disponible. Flex défile ensuite dans l'application afin que le contrôle TextInput soit visible au-dessus du clavier.

Le blogueur Peter Elst a communiqué sur le contrôle du clavier programmable dans les applications Flex Mobile.

Interaction de l'utilisateur avec le clavier à l'écran

Le clavier à l'écran s'ouvre automatiquement lorsqu'un contrôle de saisie de texte est utilisé. les contrôles de saisie de texte comprennent TextInput et TextArea.

Vous pouvez configurer d'autres types de contrôles pour ouvrir le clavier, tels que les contrôles Button ou ButtonBar. Pour ouvrir le clavier lorsqu'un contrôle autre qu'un contrôle de saisie de texte est activé, définissez la propriété `needsSoftKeyboard` du contrôle sur `true`. Tous les composants Flex héritent cette propriété de la classe InteractiveObject.

Remarque : *les contrôles de saisie de texte ouvrent toujours le clavier lorsqu'ils sont activés. Ils ignorent la propriété* `needsSoftKeyboard` *et sa définition est sans effet sur les contrôles de saisie de texte.*

Le clavier reste ouvert jusqu'à ce que les actions suivantes soient réalisées :

- L'utilisateur active un contrôle qui ne reçoit pas de saisie de texte.

  Si l'utilisateur passe à un autre contrôle de saisie de texte ou à un contrôle dont la propriété `needsSoftKeyboard` est définie sur `true`, le clavier reste ouvert.

- L'utilisateur annule la saisie en appuyant sur le bouton retour du périphérique.

Configuration de l'application pour le clavier à l'écran

Pour prendre en charge le clavier à l'écran, l'application peut effectuer les actions suivantes à l'ouverture du clavier :

- Redimensionner l'application en fonction de l'espace disponible à l'écran afin que le clavier ne soit pas superposé à l'application.
- Faite défiler le conteneur parent du contrôle de saisie de texte activé afin de s'assurer que le contrôle est visible.

Configuration du système pour le clavier à l'écran

Le clavier à l'écran n'est pas pris en charge dans les applications qui s'exécutent en mode plein écran. Par conséquent, dans le fichier app.xml, assurez-vous que l'attribut `<fullScreen>` est défini sur `false`, la valeur par défaut.

Assurez-vous que le mode de rendu de l'application est défini sur le mode du processeur. Le mode de rendu est contrôlé dans le fichier descripteur app.xml de l'application par l'attribut `<renderMode>`. Assurez-vous que l'attribut `<renderMode>` est défini sur `cpu`, la valeur par défaut, et non sur `gpu`.

Remarque : *l'attribut* `<renderMode>` *n'est pas inclus par défaut dans le fichier app.xml. Pour modifier son paramètre, ajoutez-le en tant qu'entrée dans l'attribut* `<initialWindow>`*. S'il n'est pas inclus dans le fichier app.xml, il a la valeur par défaut de* `cpu`*.*

Redimensionnement de l'application à l'ouverture du clavier à l'écran

La propriété `resizeForSoftKeyboard` du conteneur Application détermine le comportement de redimensionnement de l'application. Si elle est définie sur `true`, l'application se redimensionne pour s'adapter à la zone disponible de l'écran lorsque le clavier s'ouvre. L'application reprend sa taille normale à la fermeture du clavier.

L'exemple ci-dessous montre le fichier d'application principal pour une application qui prend en charge le redimensionnement de l'application en définissant la propriété `resizeForSoftKeyboard` sur `true` :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMobileKeyboard.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.SparkMobileKeyboardHomeView"
    resizeForSoftKeyboard="true">

</s:ViewNavigatorApplication>
```

Pour activer le redimensionnement de l'application, assurez-vous que l'attribut `<softKeyboardBehavior>` qui figure dans le fichier descripteur app.xml de l'application est défini sur `none`. La valeur par défaut de l'attribut `<softKeyboardBehavior>` est `none`. Cette valeur par défaut configure AIR pour un déplacement de l'ensemble de l'espace disponible afin que le composant texte activé soit visible.

Défilement dans le conteneur parent à l'ouverture du clavier à l'écran

Pour prendre en charge le défilement, groupez le conteneur parent des éventuels contrôles de saisie de texte dans un composant Scroller. Lorsqu'un composant qui ouvre le clavier est activé, le composant Scroller fait automatiquement défiler le composant dans la vue. Le composant peut aussi être l'enfant de plusieurs conteneurs imbriqués du composant Scroller.

Le conteneur parent doit être un conteneur GroupBase ou SkinnableContainer ou une sous-classe de GroupBase ou de SkinnableContainer. Le composant activé doit implémenter l'interface IVisualElement et doit être activable.

En enveloppant le conteneur parent dans un composant Scroller, vous pouvez faire défiler le conteneur pendant que le clavier est ouvert. Par exemple, un conteneur détient plusieurs contrôles de saisie de texte. Vous défilez alors d'un contrôle de saisie de texte à un autre pour entrer des données. Le clavier reste ouvert tant que vous sélectionnez un contrôle de saisie de texte ou jusqu'à ce que vous sélectionniez un composant dont la propriété `needsSoftKeyboard` est définie sur `true`.

Lorsque le clavier se ferme, le conteneur parent peut être plus petit que l'espace disponible à l'écran. Si le conteneur est plus petit que l'espace disponible à l'écran, le composant Scroller rétablit les positions à 0, c'est-à-dire en haut du conteneur.

L'exemple suivant présente un conteneur View contenant plusieurs contrôles TextInput et un composant Scroller :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SparkMobileKeyboardHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Compose Email">

    <s:Scroller width="100%" top="10" bottom="50">
        <s:VGroup paddingTop="3" paddingLeft="5" paddingRight="5" paddingBottom="3">
            <s:TextInput prompt="To" width="100%"/>
            <s:TextInput prompt="CC" width="100%"/>
            <s:TextInput prompt="Subject" width="100%"/>
            <s:TextArea height="400" width="100%" prompt="Compose Mail"/>
        </s:VGroup>
    </s:Scroller>

    <s:HGroup width="100%" gap="20"
        bottom="5" horizontalAlign="left">
        <s:Button label="Send" height="40"/>
        <s:Button label="Cancel" height="40"/>
    </s:HGroup>
</s:View>
```

Le conteneur VGroup est le conteneur parent des contrôles TextInput. Le composant Scroller enveloppe le conteneur VGroup, de sorte que chaque contrôle TextInput apparaît au-dessus du clavier lorsqu'il est activé.

Pour plus d'informations à propos du composant Scroller, voir Scrolling Spark containers.

Gestion des événements de clavier à l'écran

Le tableau ci-dessous décrit les événements associés au clavier :

Evénement	Moment où il est distribué
`softKeyboardActivating`	Juste avant l'ouverture du clavier
`softKeyboardActivate`	Juste après l'ouverture du clavier
`softKeyboardDeactivate`	Après la fermeture du clavier

Tous les composants Flex héritent ces événements de la classe flash.display.InteractiveObject.

Dans un gestionnaire d'événement, utilisez la propriété`softKeyboardRect` de la classe flash.display.Stage pour déterminer la taille et l'emplacement du clavier à l'écran.

Sur Android, le clavier à l'écran envoie les événements `KEY_UP` et `KEY_DOWN` lorsqu'il est utilisé par l'utilisateur. Sur iOS, les événements `KEY_UP` et `KEY_DOWN` ne sont pas envoyés. Vous pouvez à la place écouter l'événement `CHANGE` du contrôle de texte associé pour répondre à l'entrée clavier.

Intégration de polices dans une application mobile

Lorsque vous compilez une application mobile avec des polices intégrées, Flex utilise par défaut des polices non CFF. Les polices CFF utilisent FTE. En général, évitez d'utiliser FTE dans une application mobile.

Dans la mesure où le contrôle Label utilise FTE (et par conséquent, des polices CFF), utilisez les contrôles TextArea ou TextInput lors de l'intégration de polices dans une application mobile.

Dans votre CSS, définissez `embedAsCFF` sur `false`, comme le montre l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/Main.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                      xmlns:s="library://ns.adobe.com/flex/spark"
                      firstView="views.EmbeddingFontsView">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        @font-face {
            src: url("../assets/MyriadWebPro.ttf");
            fontFamily: myFontFamily;
            embedAsCFF: false;
        }
        .customStyle {
            fontFamily: myFontFamily;
            fontSize: 24;
        }
    </fx:Style>
</s:ViewNavigatorApplication>
```

Le contrôle TextArea de la vue EmbeddingFontView applique le sélecteur de type :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/EmbeddingFontsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Embedded Fonts">
    <s:TextArea id="ta1"
        width="100%"
        styleName="customStyle"
        text="This is a TextArea control that uses an embedded font."/>

</s:View>
```

Si vous utilisez des sélecteurs de classe (tels que `s|TextArea`) pour appliquer des styles (ou intégrer des polices), définissez le sélecteur de classe dans le fichier de l'application principale. Vous ne pouvez pas définir des sélecteurs de classe dans une vue d'une application mobile.

Pour plus d'informations, voir Utilisation de polices incrustées.

Utilisation de texte HTML dans des contrôles mobiles

Pour utiliser la propriété `htmlText` dans des contrôles de texte mobiles, accédez à la propriété sur le sous-contrôle StyleableTextField. L'exemple suivant crée un contrôle TextInput et TextArea et ajoute un marquage HTML au contenu :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/HTMLTextView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HTMLText View"
        creationComplete="initView()">

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.supportClasses.StyleableTextField;

            private function initView():void {
                StyleableTextField(ta1.textDisplay).htmlText = "TextArea <b>bold</b>
<i>italic</i>.";
                StyleableTextField(ti1.textDisplay).htmlText = "TextInput <b>bold</b>
<i>italic</i>.";
            }
        ]]>
    </fx:Script>

        <s:TextArea id="ta1" width="100%"/>
        <s:TextInput id="ti1" width="100%"/>
</s:View>
```

La définition de style par défaut n'est pas incluse, de sorte que si vous ajoutez un hyperlien, le contrôle de texte n'affiche pas les couleurs de soulignement et de lien. Vous pouvez ajouter des styles à l'aide d'un objet StyleSheet, comme le montre l'exemple suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/HTMLLinkView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HTMLText with Link"
        creationComplete="initView()">

    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <fx:Script>
        <![CDATA[
            import spark.components.supportClasses.StyleableTextField;

            private function initView():void {
                var styles:String = "a { color: #33CCFF; } a:hover { color: #3366CC; text-
decoration: underline; }";
                var myStyleSheet:StyleSheet = new StyleSheet();
                myStyleSheet.parseCSS(styles);

                StyleableTextField(ta1.textDisplay).styleSheet = myStyleSheet;
                StyleableTextField(ta1.textDisplay).htmlText = "Click <a
href='http://www.adobe.com'>here</a>.";
            }
        ]]>
    </fx:Script>

    <s:TextArea id="ta1" width="100%"/>

</s:View>
```

Chapitre 6 : Habillage

Notions de base sur l'habillage mobile

Comparaison entre les habillages de bureau et mobiles

Les habillages mobiles sont plus légers que leurs homologues de bureau. En conséquence, ils présentent de nombreuses différences, par exemple :

- Les habillages mobiles sont rédigés dans ActionScript. Les habillages ActionScript exclusifs fournissent les meilleures performances sur les périphériques mobiles.
- Les habillages mobiles étendent la classe spark.skins.mobile.supportClasses.MobileSkin. Cette classe étend UIComponent, comparée à la classe SparkSkin qui étend la classe Skin.
- Les habillages mobiles utilisent le FXG compilé ou le dessin ActionScript simple pour que leurs éléments graphiques améliorent les performances. En revanche, les habillages destinés aux applications de bureau utilisent généralement des graphiques MXML pour une grande partie de leurs dessins.
- Les habillages mobiles n'ont pas besoin de déclarer les états d'habillage. Comme les habillages sont rédigés dans ActionScript, les états doivent être implémentés dans le cadre de procédures.
- Les habillages mobiles ne prennent pas en charge les transitions entre états.
- Les habillages mobiles sont disposés manuellement. Comme les habillages mobiles n'étendent pas Group, ils ne prennent pas en charge les présentations Spark. Par conséquent, leurs enfants sont positionnés manuellement dans ActionScript.
- Les habillages mobiles ne prennent pas en charge tous les styles. Le thème mobile omet certains styles en fonction des performances ou d'autres différences dans les habillages mobiles.

Outre les différences liées aux performances, Flash Builder utilise également les fichiers d'habillage mobile de manière différente. Cela se vérifie tout spécialement pour les thèmes mobiles utilisés pour les projets de la bibliothèque. Le blogueur Jeffry Houser explique comment régler ce problème.

Composant d'hôte mobile

Les habillages mobiles déclarent généralement une propriété publique `hostComponent`. Cette propriété n'est pas obligatoire, mais elle est recommandée. La propriété `hostComponent` doit être du même type que le composant qui utilise l'habillage. Par exemple, l'habillage ActionBarSkin déclare `hostComponent` comme étant du type ActionBar :

```
public var hostComponent:ActionBar;
```

Flex définit la valeur de la propriété `hostComponent` lorsque le composant charge l'habillage pour la première fois.

Comme avec les habillages de bureau, vous pouvez utiliser le composant hôte pour accéder aux propriétés et aux méthodes du composant auxquelles l'habillage est attaché. Vous pouvez par exemple accéder aux propriétés publiques du composant hôte ou ajouter un écouteur d'événement au composant hôte depuis l'intérieur de la classe d'habillages.

Styles mobiles

Les habillages mobiles prennent en charge un sous-ensemble des propriétés de style que leurs homologues de bureau prennent en charge. Le thème mobile définit cet ensemble de styles.

Le tableau suivant définit les propriétés de style disponibles pour les composants lors de l'utilisation du thème mobile :

Propriété Style	Prise en charge par	Héritant/N'héritant pas
`accentColor`	Button, ActionBar, ButtonBar	Héritant
`backgroundAlpha`	ActionBar	N'héritant pas
`backgroundColor`	Application	N'héritant pas
`borderAlpha`	List	N'héritant pas
`borderColor`	List	N'héritant pas
`borderVisible`	List	N'héritant pas
`chromeColor`	ActionBar, Button, ButtonBar, CheckBox, HSlider, RadioButton	Héritant
`color`	Tous les composants avec du texte	Héritant
`contentBackgroundAlpha`	TextArea, TextInput	Héritant
`contentBackgroundColor`	TextArea, TextInput	Héritant
`focusAlpha`	Tous les composants détaillables	N'héritant pas
`focusBlendMode`	Tous les composants détaillables	N'héritant pas
`focusColor`	Tous les composants détaillables	Héritant
`focusThickness`	Tous les composants détaillables	N'héritant pas
`paddingBottom`	TextArea, TextInput	N'héritant pas
`paddingLeft`	TextArea, TextInput	N'héritant pas
`paddingRight`	TextArea, TextInput	N'héritant pas
`paddingTop`	TextArea, TextInput	N'héritant pas
`selectionColor`	ViewMenuItem	Héritant

Tous les composants comprenant du texte prennent également en charge les styles de texte standard tels que `fontFamily`, `fontSize`, `fontWeight` et `textDecoration`.

Pour voir si le thème mobile prend en charge une propriété de style, ouvrez la description du composant dans le Guide de référence du langage ActionScript. Un grand nombre de ces limitations de style proviennent du fait que les composants mobiles de type texte n'utilisent pas la structure TLF (Text Layout Framework). En effet, les habillages mobiles remplacent les contrôles de texte basés sur TLF par des composants plus légers. Pour plus d'informations, voir « Utilisation de texte dans une application mobile » à la page 86.

Le thème mobile ne prend pas en charge les propriétés de style `rollOverColor`, `cornerRadius` ni `dropShadowVisible`.

L'ingénieur Flex Jason SJ présente les styles des habillages mobiles sur son blog.

Parties de l'habillage mobile

En ce qui concerne les parties de l'habillage, les habillages mobiles doivent respecter les mêmes normes que les habillages de bureau. Si un composant comporte une partie d'habillage obligatoire, l'habillage mobile doit déclarer une propriété publique du type approprié.

Exceptions

Toutes les parties d'habillage ne sont pas obligatoires. Par exemple, le composant Spark Button comporte les parties d'habillage facultatives `iconDisplay` et `labelDisplay`. En conséquence, la classe mobile ButtonSkin peut déclarer une propriété `iconDisplay` du type BitmapImage. Elle peut aussi déclarer une propriété `labelDisplay` du type StyleableTextField.

La partie `labelDisplay` ne définit pas de propriété `id` car les styles qu'elle utilise sont tous des styles de texte héritant. En outre, le type StyleableTextField n'est pas un composant UIComponent et ne possède donc pas de propriété `id`. La partie `iconDisplay` ne prend pas en charge les styles, de sorte qu'elle ne définit pas de propriété `id`.

Définition de styles à l'aide des fonctionnalités CSS avancées

Si vous souhaitez définir des styles sur la partie habillage avec le sélecteur avancé CSS `id`, l'habillage doit également définir la propriété `id` de la partie habillage. Par exemple, la partie d'habillage d'ActionBar`titleDisplay` définit une propriété `id`, de sorte qu'elle peut être stylée à l'aide des fonctionnalités CSS avancées ; par exemple :

```
@namespace s "library://ns.adobe.com/flex/spark";
s|ActionBar #titleDisplay {
    color:red;
}
```

Thème mobile

Le thème mobile détermine les styles qu'une application mobile prend en charge. Le nombre de styles disponibles avec le thème mobile est un sous-ensemble du thème Spark (avec quelques ajouts mineurs). Vous pouvez voir une liste complète des styles pris en charge par le thème mobile dans « Styles mobiles » à la page 96.

Thème par défaut des applications mobiles

Le thème des applications mobiles est défini dans le fichier themes/Mobile/mobile.swc. Ce fichier définit les styles globaux pour les applications mobiles, ainsi que les paramètres par défaut de chacun des composants mobiles. Les habillages mobiles de ce fichier de thème sont définis dans les groupements spark.skins.mobile.flash.sampler.*. Ce package contient la classe de base MobileSkin.

Le fichier de thème mobile.swc est inclus par défaut dans les projets mobiles Flash Builder, mais le fichier SWC ne figure pas dans l'Explorateur de packages.

Lorsque vous créez un nouveau projet mobile dans Flash Builder, ce thème est appliqué par défaut.

Modification du thème

Pour modifier le thème, utilisez l'argument de compilateur `theme` pour spécifier le nouveau thème ; par exemple :

```
-theme+=myThemes/NewMobileTheme.swc
```

Pour plus d'informations sur les thèmes, voir A propos des thèmes.

L'ingénieur Flex Jason SJ présente de quelle façon créer et incruster un thème dans une application sur son blog.

Etats des habillages mobiles

La classe MobileSkin remplace le mécanisme d'états de la classe UIComponent et n'utilise pas l'implémentation des états d'affichage des applications de bureau. En conséquence, les habillages mobiles déclarent seulement les états d'habillage du composant hôte qui sont implémentés par l'habillage. Ils modifient l'état dans le cadre de procédures, en fonction seulement du nom de l'état. A l'opposé, les habillages de bureau doivent déclarer tous les états, qu'ils soient utilisés ou non. Les habillages de bureau utilisent également les classes dans le package mx.states.* pour modifier les états.

La plupart des habillages mobiles implémentent moins d'états que leurs homologues de bureau. Par exemple, la classe spark.skins.mobile.ButtonSkin implémente les états `up`, `down` et `disabled`. L'habillage spark.skins.spark.ButtonSkin implémente tous ces états ainsi que l'état `over`. L'habillage mobile ne définit pas le comportement de l'état `over` car cet état ne serait pas couramment utilisé sur un périphérique tactile.

Méthode commitCurrentState()

Les classes d'habillages mobiles définissent le comportement de leurs états dans la méthode `commitCurrentState()`. Vous pouvez ajouter un comportement à un habillage mobile pour prendre en charge des états supplémentaires en modifiant la méthode `commitCurrentState()` dans votre classe d'habillages personnalisés.

Propriété currentState

L'aspect d'un habillage dépend de la valeur de la propriété `currentState`. Par exemple, dans la classe mobile ButtonSkin, la valeur de la propriété `currentState` détermine la classe FXG qui est utilisée comme classe limitrophe :

```
if (currentState == "down")
    return downBorderSkin;
else
    return upBorderSkin;
```

Pour plus d'informations sur la propriété `currentState`, voir Create and apply view states.

Graphiques mobiles

Les habillages mobiles utilisent généralement le FXG compilé pour leurs éléments graphiques. Les habillages destinés aux applications de bureau, en revanche, utilisent généralement des graphiques MXML pour une grande partie de leurs dessins.

Graphiques bitmap intégrés

Vous pouvez utiliser des graphiques bitmap intégrés dans vos classes, car ils se comportent correctement en général. Toutefois, le redimensionnement des images bitmap pour plusieurs densités d'écran pose parfois problème. Le redimensionnement peut être plus efficace si vous créez plusieurs éléments différents, un pour chaque densité d'écran.

Graphiques dans le thème mobile par défaut

Les habillages mobiles dans le thème mobile par défaut utilisent les graphiques FXG qui sont optimisés pour la valeur PPP du périphérique cible. Les habillages chargent les graphiques en fonction de la valeur de la propriété `applicationDPI` de l'application racine. Par exemple, lorsqu'un contrôle CheckBox est chargé sur un périphérique doté d'une valeur PPP de 320, la classe CheckBoxSkin utilise le graphique spark.skins.mobile320.assets.CheckBox_up.fxg pour la propriété `upIconClass`. Pour une valeur PPP de 160, elle utilise le graphique spark.skins.mobile160.assets.CheckBox_up.fxg.

L'exemple *de bureau* suivant monte les différents graphiques utilisés par l'habillage CheckBox pour différentes valeurs PPP :

```
<?xml version="1.0"?>
<!-- mobile_skins/ShowCheckBoxSkins.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:skins160="spark.skins.mobile160.assets.*"
    xmlns:skins240="spark.skins.mobile240.assets.*"
    xmlns:skins320="spark.skins.mobile320.assets.*">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <!--
    NOTE: You must add the mobile theme directory to source path
    to compile this example.

    For example:
    mxmlc -source-path+=\frameworks\projects\mobiletheme\src\ ShowCheckBoxSkins.mxml
    -->
    <s:Label text="160 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins160:CheckBox_down/>
        <skins160:CheckBox_downSymbol/>
        <skins160:CheckBox_downSymbolSelected/>
        <skins160:CheckBox_up/>
        <skins160:CheckBox_upSymbol/>
        <skins160:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="240 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins240:CheckBox_down/>
        <skins240:CheckBox_downSymbol/>
        <skins240:CheckBox_downSymbolSelected/>
        <skins240:CheckBox_up/>
        <skins240:CheckBox_upSymbol/>
        <skins240:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="320 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins320:CheckBox_down/>
        <skins320:CheckBox_downSymbol/>
        <skins320:CheckBox_downSymbolSelected/>
        <skins320:CheckBox_up/>
        <skins320:CheckBox_upSymbol/>
        <skins320:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <s:Label text="down, downSymbol, downSymbolSelected, up, upSymbol, upSymbolSelected"/>
</s:Application>
```

Pour plus d'informations sur les résolutions et les valeurs PPP dans les applications mobiles, voir « Prise en charge de plusieurs tailles d'écran et valeurs PPP dans une application mobile » à la page 71.

Dans les habillages ActionScript, vous pouvez aussi utiliser des vecteurs mis en mémoire cache en tant que bitmaps. Le seul inconvénient tient au fait que vous ne pouvez pas utiliser de transitions qui nécessitent la recréation des pixels, telles que les transitions alpha. Pour plus d'informations, voir www.adobe.com/devnet/air/flex/articles/writing_multiscreen_air_apps.html.

Création d'habillages pour une application mobile

Lors de la personnalisation d'habillages mobiles, vous créez une classe d'habillages mobile personnalisée. Dans certains cas, vous modifiez également les éléments utilisés par une classe d'habillages mobiles.

Lorsque vous modifiez une classe d'habillages mobiles, vous pouvez modifier les interactions basées sur l'état, implémenter la prise en charge de nouveaux styles ou ajouter ou supprimer des composants enfants de l'habillage. En général, vous commencez par le code source d'un habillage existant et l'enregistrez en tant que nouvelle classe.

Vous pouvez également modifier les éléments utilisés par les habillages mobiles pour modifier les propriétés visuelles de l'habillage, telles que sa taille, sa couleur ou ses dégradés et arrière-plans. Dans ce cas, vous modifiez également les éléments FXG utilisés par les habillages. Les fichiers *.fxg utilisés par les habillages mobiles sont stockés dans le répertoire spark/skins/mobile/assets.

Toutes les propriétés visuelles des habillages mobiles ne sont pas définies dans des fichiers *.fxg. Par exemple, la couleur d'arrière-plan de l'habillage Button est définie par la propriété de style `chromeColor` dans la classe ButtonSkin. Elle n'est pas définie dans un élément FXG. Dans ce cas, vous modifieriez la classe d'habillage pour changer la couleur d'arrière-plan.

L'ingénieur Flex Jason SJ présente de quelle façon créer des habillages pour les applications mobiles sur son blog.

Création d'une classe d'habillages mobiles

Lors de la création d'une classe d'habillages mobiles personnalisée, la solution la plus simple consiste à utiliser comme base une classe d'habillages mobiles existante. Modifiez ensuite cette classe et utilisez-la comme un habillage personnalisé.

Pour créer une classe d'habillages personnalisés :

1. Créez un répertoire dans votre projet (par exemple, customSkins). Ce répertoire correspond au nom du package de vos habillages personnalisés. La création d'un package n'est pas requise, mais il est judicieux d'organiser les habillages personnalisés dans un package individuel.
2. Créez une classe d'habillages personnalisés dans le nouveau répertoire. Donnez à cette nouvelle classe le nom de votre choix, par exemple CustomButtonSkin.as.
3. Copiez le contenu de la classe d'habillages que vous utilisez comme base pour la nouvelle classe. Par exemple, si vous utilisez ButtonSkin comme classe de base, copiez le contenu du fichier spark.skins.mobile.ButtonSkin dans la nouvelle classe d'habillages personnalisés.
4. Modifiez la nouvelle classe. Par exemple, apportez les modifications minimales suivantes à la classe CustomButtonSkin :
   - Modifiez l'emplacement du groupement :

     ```
     package customSkins
     //was: package spark.skins.mobile
     ```

   - Modifiez le nom de la classe dans la déclaration de classe. Développez également la classe sur laquelle votre nouvel habillage est basé, pas la classe d'habillages de base :

```
public class CustomButtonSkin extends ButtonSkin
// was: public class ButtonSkin extends ButtonSkinBase
```

- Modifiez le nom de la classe dans le constructeur :

```
public function CustomButtonSkin()
//was: public function ButtonSkin()
```

5 Modifiez la classe d'habillages personnalisés. Par exemple, ajoutez la prise en charge d'états supplémentaires ou de nouveaux composants enfants. De même, certains composants graphiques sont définis dans la classe d'habillages elle-même, de sorte que vous pouvez modifier certains éléments.

Pour faciliter la lecture de votre classe d'habillages, vous pouvez en général supprimer des méthodes quelconques de l'habillage personnalisé que vous ne remplacez pas.

La classe d'habillages personnalisés ci-dessous étend ButtonSkin et remplace la méthode `drawBackground()` par une logique personnalisée. Elle remplace le dégradé linéaire par un dégradé radial pour le remplissage de l'arrière-plan.

```
package customSkins {
    import mx.utils.ColorUtil;
    import spark.skins.mobile.ButtonSkin;
    import flash.display.GradientType;
    import spark.skins.mobile.supportClasses.MobileSkin;
    import flash.geom.Matrix;
    public class CustomButtonSkin extends ButtonSkin {

        public function CustomButtonSkin() {
            super();
        }
        private static var colorMatrix:Matrix = new Matrix();
        private static const CHROME_COLOR_ALPHAS:Array = [1, 1];
        private static const CHROME_COLOR_RATIOS:Array = [0, 127.5];

        override protected function drawBackground(unscaledWidth:Number,
unscaledHeight:Number):void {
            super.drawBackground(unscaledWidth, unscaledHeight);

            var chromeColor:uint = getStyle("chromeColor");
            /*
```

```
            if (currentState == "down") {
                graphics.beginFill(chromeColor);
            } else {
            */
            var colors:Array = [];
           colorMatrix.createGradientBox(unscaledWidth, unscaledHeight, Math.PI / 2, 0, 0);
            colors[0] = ColorUtil.adjustBrightness2(chromeColor, 70);
            colors[1] = chromeColor;
            graphics.beginGradientFill(GradientType.RADIAL, colors, CHROME_COLOR_ALPHAS,
CHROME_COLOR_RATIOS, colorMatrix);
            // }
            graphics.drawRoundRect(layoutBorderSize, layoutBorderSize,
                unscaledWidth - (layoutBorderSize * 2),
                unscaledHeight - (layoutBorderSize * 2),
                layoutCornerEllipseSize, layoutCornerEllipseSize);
            graphics.endFill();
        }

    }
}
```

6 Dans votre application, appliquez l'habillage personnalisé en utilisant l'une des méthodes décrites dans « Application d'un habillage mobile personnalisé » à la page 109. L'exemple suivant utilise la propriété `skinClass` de la balise du composant pour appliquer l'habillage `customSkins.CustomButtonSkin` :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/CustomButtonSkinView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark" title="Home">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Button label="Click Me" skinClass="customSkins.CustomButtonSkin"/>

</s:View>
```

Méthodes de cycle de vie des habillages mobiles

Lors de la création de classes d'habillages personnalisés, familiarisez-vous avec les méthodes UIComponent suivantes. Ces méthodes protégées héritées définissent les enfants et les membres d'un habillage et l'aident à interagir avec d'autres composants de la liste d'affichage.

- `createChildren()` — Permet de créer tous les graphiques ou les objets de texte enfants requis par l'habillage.
- `commitProperties()` — Permet de copier les données du composant dans l'habillage, le cas échéant.
- `measure()` — Permet de mesurer l'habillage aussi efficacement que possible et de stocker les résultats dans les propriétés `measuredWidth` et `measuredHeight` de l'habillage.
- `updateDisplayList()` — Permet de définir la position et la taille des graphiques et du texte. Réalisez tout schéma ActionScript requis. Cette méthode appelle les méthodes `drawBackground()` et `layoutContents()` dans l'habillage.

Pour plus d'informations à propos de l'utilisation de ces méthodes, voir Implémentation du composant.

Méthodes courantes de personnalisation des habillages mobiles

De nombreux habillages mobiles implémentent les méthodes suivantes :

- `layoutContents()` — Positionne les enfants pour l'habillage, tels que les éléments dropshadow et label. Les classes d'habillages mobiles ne prennent pas en charge les présentations Spark telles que HorizontalLayout et VerticalLayout. Présentez manuellement les enfants de l'habillage dans une méthode telle que `layoutContents()`.
- `drawBackground()` — Restitue un arrière-plan pour l'habillage. Les cas d'utilisation standard incluent le dessin des styles `chromeColor`, `backgroundColor` ou `contentBackgroundColor` en fonction de la forme de l'habillage. Cette méthode peut également être utilisée pour teinter, comme par exemple avec la méthode `applyColorTransform()`.
- `commitCurrentState()` — Définit les comportements d'état pour les habillages mobiles. Vous pouvez ajouter ou supprimer des états pris en charge, ou changer le comportement d'états existants en modifiant cette méthode. Cette méthode est appelée lorsque l'état a été modifié. La plupart des classes d'habillages remplacent cette méthode. Pour plus d'informations, voir « Etats des habillages mobiles » à la page 98.

Création d'éléments FXG personnalisés

La plupart des éléments visuels des habillages mobiles sont définis à l'aide de FXG. FXG est une syntaxe déclarative permettant de définir les graphiques statiques. Vous pouvez utiliser un outil graphique tel qu'Adobe Fireworks, Adobe Illustrator ou Adobe Catalyst pour exporter un document FXG. Vous pouvez alors utiliser le document FXG dans votre habillage mobile. Vous pouvez aussi créer des documents FXG dans un éditeur de texte, même si les graphiques complexes peuvent être difficiles à écrire entièrement.

Les habillages mobiles utilisent généralement des fichiers FXG pour définir les états d'un habillage. Par exemple, la classe CheckBoxSkin utilise les fichiers FXG suivants pour définir l'aspect de sa case et de son symbole de coche :

- CheckBox_down.fxg
- CheckBox_downSymbol.fxg
- CheckBox_downSymbolSelected.fxg
- CheckBox_up.fxg
- CheckBox_upSymbol.fxg
- CheckBox_upSymbolSelected.fxg

Si vous ouvrez ces fichiers dans un éditeur de texte, ils apparaissent comme suit :

Etats des cases (down, downSymbol, downSymbolSelected, up, upSymbol et upSymbolSelected)

Fichiers FXG pour différentes résolutions

La plupart des habillages mobiles possèdent trois jeux de fichiers graphiques FXG, un pour chaque résolution cible par défaut. Par exemple, des versions différentes des six classes CheckBoxSkin figurent dans les répertoires spark/skins/mobile160, spark/skins/mobile240 et spark/skins/mobile320.

Lorsque vous créez un habillage personnalisé, vous pouvez procéder d'une des façons suivantes :

- Utilisez l'un des habillages par défaut comme base (habituellement 160 PPP). Ajoutez une logique qui redimensionne l'habillage personnalisé pour l'adapter au périphérique sur lequel l'application s'exécute en définissant la propriété `applicationDPI` sur l'objet Application.
- Créez les trois versions de l'habillage personnalisé (160, 240 et 320 PPP) pour un affichage optimal.

Certains habillages mobiles utilisent un jeu individuel de fichiers FXG pour leurs éléments graphiques et ne possèdent pas de graphiques pour des valeurs PPP spécifiques. Ces éléments sont stockés dans le répertoire spark/skins/mobile/assets. Par exemple, les habillages ViewMenuItem et les habillages de barre de boutons TabbedViewNavigator ne possèdent pas de versions pour des valeurs PPP spécifiques, si bien que tous leurs éléments FXG sont stockés dans ce répertoire.

Personnalisation d'un fichier FXG

Vous pouvez ouvrir un fichier FXG existant et le personnaliser, ou en créer un et l'exporter à partir d'un éditeur graphique tel qu'Adobe Illustrator. Après avoir modifié le fichier FXG, appliquez-le à votre classe d'habillages.

Pour créer un habillage personnalisé en modifiant un fichier FXG :

1. Créez une classe d'habillages personnalisés et placez-la dans le répertoire customSkins, comme cela est décrit dans « Création d'une classe d'habillages mobiles » à la page 101.
2. Créez un sous-répertoire sous le répertoire customSkins, par exemple assets. La création d'un sous-répertoire est facultative, mais permet d'organiser vos fichiers FXG et vos classes d'habillages.
3. Créez un fichier dans le répertoire assets et copiez à l'intérieur le contenu d'un fichier FXG existant. Par exemple, créez un fichier nommé CustomCheckBox_upSymbol.fxg. Copiez le contenu du fichier spark/skins/mobile160/assets/CheckBox_upSymbol.fxg dans le nouveau fichier CustomCheckBox_upSymbol.fxg.
4. Modifiez le nouveau fichier FXG. Par exemple, remplacez la logique permettant de tracer un symbole de coche par un « X » rempli avec des entrées de dégradé :

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- mobile_skins/customSkins/assets/CustomCheckBox_upSymbol.fxg -->
<Graphic xmlns="http://ns.adobe.com/fxg/2008" version="2.0"
    viewWidth="32" viewHeight="32">
    <!-- Main Outer Border -->
    <Rect x="1" y="1" height="30" width="30" radiusX="2" radiusY="2">
        <stroke>
            <SolidColorStroke weight="1" color="#282828"/>
        </stroke>
    </Rect>
    <!-- Replace check mark with an "x" -->
    <Group x="2" y="2">
        <Line xFrom="3" yFrom="3" xTo="25" yTo="25">
            <stroke>
               <LinearGradientStroke caps="none" weight="8" joints="miter" miterLimit="4">
                    <GradientEntry color="#FF0033"/>
                    <GradientEntry color="#0066FF"/>
                </LinearGradientStroke>
            </stroke>
        </Line>
        <Line xFrom="25" yFrom="3" xTo="3" yTo="25">
            <stroke>
            <stroke>
               <LinearGradientStroke caps="none" weight="8" joints="miter" miterLimit="4">
                    <GradientEntry color="#FF0033"/>
                    <GradientEntry color="#0066FF"/>
                </LinearGradientStroke>
            </stroke>
            </stroke>
        </Line>
    </Group>
</Graphic>
```

5 Dans la classe d'habillages personnalisés, importez la nouvelle classe FXG et appliquez-la à une propriété. Par exemple, dans la classe CustomCheckBox :

   1 Importez le nouveau fichier FXG.

   ```
   //import spark.skins.mobile.assets.CheckBox_upSymbol;
   import customSkins.assets.CustomCheckBox_upSymbol;
   ```

   2 Ajoutez le nouvel élément à la classe d'habillages personnalisés. Par exemple, modifiez la valeur de la propriété `upSymbolIconClass` pour qu'elle pointe sur votre nouvel élément FXG :

   ```
   upSymbolIconClass = CustomCheckBox_upSymbol;
   ```

La classe complète d'habillages personnalisés se présente comme suit :

```
// mobile_skins/customSkins/CustomCheckBoxSkin.as
package customSkins {
    import spark.skins.mobile.CheckBoxSkin;
    import customSkins.assets.CustomCheckBox_upSymbol;

    public class CustomCheckBoxSkin extends CheckBoxSkin {
        public function CustomCheckBoxSkin() {
            super();
            upSymbolIconClass = CustomCheckBox_upSymbol; // was CheckBox_upSymbol
        }
    }
}
```

Pour plus d'informations sur l'utilisation et l'optimisation des éléments FXG pour les habillages, voir Optimisation des FXG.

Affichage des fichiers FXG dans des applications

Dans la mesure où les fichiers FXG sont écrits en langage XML, il peut être difficile de visualiser l'aspect final du produit. Vous pouvez écrire une application Flex qui importe et rend les fichiers FXG en les ajoutant en tant que composants, puis en les enveloppant dans un conteneur Spark.

Pour ajouter des fichiers FXG en tant que composants à votre application, ajoutez l'emplacement des fichiers source au chemin source de votre application. Par exemple, pour monter des éléments FXG mobiles dans une application Web, ajoutez le thème mobile à votre chemin source. Le compilateur peut alors trouver les fichiers FXG.

L'exemple *de bureau* ci-dessous restitue les différents éléments FXG du composant CheckBox lorsque vous l'utilisez dans une application mobile. Ajoutez le répertoire frameworks\projects\mobiletheme\src\ à l'argument `source-path` du compilateur lorsque vous compilez cet exemple.

```
<?xml version="1.0"?>
<!-- mobile_skins/ShowCheckBoxSkins.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:skins160="spark.skins.mobile160.assets.*"
    xmlns:skins240="spark.skins.mobile240.assets.*"
    xmlns:skins320="spark.skins.mobile320.assets.*">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <!--
    NOTE: You must add the mobile theme directory to source path
    to compile this example.

    For example:
    mxmlc -source-path+=\frameworks\projects\mobiletheme\src\ ShowCheckBoxSkins.mxml
    -->
    <s:Label text="160 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins160:CheckBox_down/>
        <skins160:CheckBox_downSymbol/>
        <skins160:CheckBox_downSymbolSelected/>
        <skins160:CheckBox_up/>
        <skins160:CheckBox_upSymbol/>
        <skins160:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="240 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins240:CheckBox_down/>
        <skins240:CheckBox_downSymbol/>
        <skins240:CheckBox_downSymbolSelected/>
        <skins240:CheckBox_up/>
        <skins240:CheckBox_upSymbol/>
        <skins240:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="320 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins320:CheckBox_down/>
        <skins320:CheckBox_downSymbol/>
        <skins320:CheckBox_downSymbolSelected/>
        <skins320:CheckBox_up/>
        <skins320:CheckBox_upSymbol/>
        <skins320:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <s:Label text="down, downSymbol, downSymbolSelected, up, upSymbol, upSymbolSelected"/>
</s:Application>
```

Utilisation de texte dans les habillages mobiles personnalisés

Pour rendre le texte dans les habillages mobiles, vous utilisez la classe StyleableTextField. Cette classe de texte est optimisée pour les applications mobiles. Elle développe la classe TextField et implémente les interfaces ISimpleStyleClient et IEditableText.

Les habillages mobiles pour plusieurs composants utilisent la classe StyleableTextField, y compris :

- ActionBar
- Bouton
- TextArea
- TextInput

Pour plus d'informations sur l'utilisation des contrôles de texte dans les applications mobiles, voir « Contrôles de texte MX » à la page 88.

TLF dans les habillages mobiles

Pour des raisons de performances, essayez d'éviter les classes qui utilisent TLF dans les habillages mobiles. Dans certains cas, comme par exemple avec le composant Spark Label, vous pouvez utiliser des classes qui utilisent FTE.

Utilisation de htmlText dans les habillages mobiles

Vous pouvez définir la propriété `htmlText` directement sur une instance de la classe StyleableTextField. Pour plus d'informations, voir « Utilisation de texte HTML dans des contrôles mobiles » à la page 94.

Définition de StyleTextField

Vous définissez généralement l'objet StyleableTextField dans la méthode `createChildren()` de l'habillage mobile. Pour instancier un objet StyleableTextField dans un habillage mobile, utilisez la méthode `UIComponent.createInFontContext()`, comme le montre l'exemple suivant :

```
import spark.components.supportClasses.StyleableTextField;
textDisplay = StyleableTextField(createInFontContext(StyleableTextField));
```

Application de styles à StyleableTextField

Dans la méthode `createChildren()`, vous appelez généralement la méthode `getStyle()` sur les propriétés de style que vous voulez que votre objet StyleableTextField prenne en charge dans l'habillage. Vous définissez également des propriétés sur l'habillage que vous voulez que l'objet StyleableTextField utilise ; par exemple :

```
textDisplay.multiline = true;
textDisplay.editable = true;
textDisplay.lineBreak = getStyle("lineBreak");
```

Appelez la méthode `commitStyles()` pour valider les informations de style dans le champ de texte après avoir appelé `setStyle()`. En général, appelez cette méthode dans les méthodes `measure()` et `updateDisplayList()` de l'habillage.

Ajout de StyleableTextField à la liste d'affichage

Après avoir défini l'objet StyleableTextField, vous l'ajoutez à la liste d'affichage à l'aide de la méthode `addElement()` :

```
addElement(textDisplay);
```

Vous utilisez uniquement la méthode `addElement()` pour ajouter les enfants des habillages mobiles dans les groupes et conteneurs Spark. Dans le cas contraire, vous utilisez la méthode `addChild()`.

Gestes avec du texte

Les gestes de toucher-glisser sélectionnent toujours du texte (lorsque le texte est sélectionnable ou modifiable). Si le texte se trouve dans un composant Scroller, ce dernier n'effectue un défilement que si le geste est effectué en dehors du composant de texte. Ces gestes fonctionnent uniquement lorsque le texte est modifiable et sélectionnable.

Modification du texte en texte modifiable et sélectionnable

Pour rendre le texte modifiable et sélectionnable, définissez les propriétés `editable` et `selectable` sur `true`:

```
textDisplay.editable = true;
textDisplay.selectable = true;
```

Bidirectionnalité avec StyleableTextField

La bidirectionnalité n'est pas prise en charge pour le texte dans la classe StyleableTextField.

Application d'un habillage mobile personnalisé

Vous pouvez appliquer un habillage personnalisé à votre composant mobile de la même manière que vous appliquez un habillage personnalisé à un composant dans une application de bureau.

Application d'un habillage dans ActionScript

```
// Call the setStyle() method:
myButton.setStyle("skinClass", "MyButtonSkin");
```

Application d'un habillage dans MXML

```
<!-- Set the skinClass property: -->
<s:Button skinClass="MyButtonSkin"/>
```

Application d'un habillage dans CSS

```
// Use type selectors for mobile skins, but only in the root document:
s|Button {
    skinClass: ClassReference("MyButtonSkin");
}
```

ou

```
// Use class selectors for mobile skins in any document:
.myStyleClass {
    skinClass: ClassReference("MyButtonSkin");
}
```

Exemple d'application d'un habillage mobile personnalisé

L'exemple suivant montre les trois méthodes qui permettent d'appliquer un habillage mobile personnalisé à des composants mobiles :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/ApplyingMobileSkinsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Home">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import customSkins.CustomButtonSkin;
            private function changeSkin():void {
                b3.setStyle("skinClass", customSkins.CustomButtonSkin);
            }
        ]]>
    </fx:Script>

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        .customButtonStyle {
            skinClass: ClassReference("customSkins.CustomButtonSkin");
        }
    </fx:Style>

    <s:Button id="b1" label="Click Me" skinClass="customSkins.CustomButtonSkin"/>
    <s:Button id="b2" label="Click Me" styleName="customButtonStyle"/>
    <s:Button id="b3" label="Click Me" click="changeSkin()"/>

</s:View>
```

Lorsque vous utilisez un sélecteur CSS *type* pour appliquer un habillage personnalisé, définissez-le dans le fichier d'application mobile racine. Vous ne pouvez pas définir des sélecteurs de type dans une vue mobile, qui est identique à un composant personnalisé. Vous pouvez néanmoins définir des styles dans ActionScript, MXML ou CSS avec un sélecteur de classe dans toute vue ou document de votre application mobile.

Chapitre 7 : Exécution et débogage des applications mobiles

Gestion des configurations de lancement

Flash Builder utilise des configurations de lancement lorsque vous exécutez ou déboguez des applications mobiles. Vous pouvez spécifier s'il faut lancer l'application sur le bureau ou sur un périphérique connecté à votre ordinateur.

Pour créer une configuration de lancement, procédez comme suit :

1 Sélectionnez Exécuter > Exécuter les configurations pour ouvrir la boîte de dialogue d'exécution des configurations.

Pour ouvrir la boîte de dialogue Configurations de débogage, sélectionner Exécuter > Configurations de débogage. Voir « Exécution et débogage d'une application mobile sur un périphérique » à la page 112.

Vous pouvez également accéder à la commande Exécuter ou Déboguer les configurations dans la liste déroulante du bouton Exécuter ou Déboguer de la barre d'outils Flash Builder.

2 Développez le nœud des applications mobiles. Cliquez sur le bouton Crée une configuration de lancement dans la barre d'outils de la boîte de dialogue.

3 Spécifiez une plateforme cible dans la liste déroulante.

4 Spécifiez une méthode de lancement :

- **Sur le bureau**

  Exécute l'application sur votre bureau à l'aide de AIR Debug Launcher (ADL), selon une configuration de périphérique que vous avez spécifiée. Cette méthode de lancement n'est pas une vraie émulation d'exécution de l'application sur un périphérique. Elle vous permet toutefois de voir la mise en forme de l'application et d'interagir avec l'application. Voir « Prévisualisation des applications avec ADL » à la page 112.

  Cliquez sur Configurer pour modifier les configurations du périphérique. Voir « Configuration des informations du périphérique pour un aperçu sur le bureau » à la page 112.

- **Sur le périphérique**

  Installez et exécutez l'application sur le périphérique.

  Pour la plateforme Google Android, Flash Builder installe l'application sur votre périphérique et lance l'application. Flash Builder accède au périphérique connecté au port USB de l'ordinateur. Pour plus d'informations, voir « Exécution et débogage d'une application mobile sur un périphérique » à la page 112.

  Windows nécessite un pilote USB pour connecter un périphérique Android à l'ordinateur. Pour plus d'informations, voir « Installation de pilotes de périphérique USB pour les périphériques Android (Windows) » à la page 16.

5 Indiquez si vous souhaitez effacer les données de l'application à chaque lancement, le cas échéant.

Exécution et débogage d'une application mobile sur le bureau

Pour des tâches initiales de test ou de débogage, ou si vous ne possédez pas de périphérique mobile, Flash Builder vous permet d'exécuter et de déboguer les applications sur le bureau en utilisant AIR Debug Launcher (ADL).

Avant d'exécuter ou de déboguer une application mobile pour la première fois, vous devez définir une configuration de lancement. Spécifiez la plateforme cible et Sur le bureau comme méthode de lancement. Voir « Gestion des configurations de lancement » à la page 111.

Configuration des informations du périphérique pour un aperçu sur le bureau

Les propriétés d'une configuration de périphérique déterminent la manière dont l'application apparaît dans ADL et dans le mode Création de Flash Builder.

La fenêtre « Définition des configurations de périphériques » à la page 12 répertorie les configurations prises en charge. Les configurations des périphériques n'affectent pas l'aspect de l'application sur le périphérique.

Densité d'écran

Vous pouvez afficher votre application sur le bureau de développement ou afficher la présentation de l'application dans le mode Création de Flash Builder. Flash Builder utilise une densité d'écran de 240 PPP. L'aspect d'une application lors de sa prévisualisation diffère parfois de son aspect sur un périphérique qui prend en charge une densité de pixels différente.

Prévisualisation des applications avec ADL

Lors de la prévisualisation d'une application sur le bureau, Flash Builder lance l'application à l'aide d'ADL. ADL fournit un menu Périphérique avec les raccourcis correspondants pour émuler les boutons du périphérique.

Par exemple, pour émuler le bouton retour d'un périphérique, sélectionnez Périphérique > Retour. Sélectionnez Périphérique > Rotation gauche ou Périphérique > Rotation droite pour émuler la rotation du périphérique. Les options de rotation sont désactivées si vous n'avez pas sélectionné l'orientation automatique.

Faites glisser le curseur dans une liste pour émuler le défilement de la liste sur un périphérique.

Brent Arnold, expert Flex certifié par Adobe, a créé un didacticiel vidéo sur l'utilisation d'ADL pour obtenir un aperçu d'une application mobile sur le bureau.

Exécution et débogage d'une application mobile sur un périphérique

Vous pouvez utiliser Flash Builder pour exécuter ou déboguer une application mobile à partir de votre bureau de développement ou d'un périphérique.

Vous exécutez et déboguez les applications en fonction d'une configuration de lancement que vous définissez. Flash Builder partage la configuration de lancement entre l'exécution et le débogage de l'application. Lorsque vous utilisez Flash Builder pour déboguer une application sur un périphérique, Flash Builder installe une version de débogage de l'application sur le périphérique.

Remarque : *si vous exportez une version validée vers un périphérique, vous installez une version non déboguée de l'application. La version non déboguée ne convient pas au débogage.*

Pour plus d'informations, voir Gestion des configurations de lancement.

Débogage d'une application sur un périphérique Google Android

Sur un périphérique Android, le débogage requiert Android 2.2 ou version ultérieure.

Vous pouvez effectuer le débogage dans l'un des scénarios suivants :

Débogage via USB Pour déboguer une application via une connexion USB, connectez le périphérique à l'ordinateur hôte par le biais d'un port USB. Lors d'un débogage via USB, Flash Builder groupe toujours l'application, puis l'installe et la lance sur le périphérique avant le démarrage du débogage. Assurez-vous que le périphérique est connecté au port USB de l'ordinateur hôte au cours de la session de débogage complète.

Débogage via un réseau Lorsque vous déboguez une application via le réseau, le périphérique et l'ordinateur hôte doivent utiliser le même réseau. L'ordinateur hôte et le périphérique peuvent être connectés au réseau via Wi-Fi, Ethernet ou Bluetooth.

Lors d'un débogage via un réseau, Flash Builder vous permet de déboguer une application qui est déjà installée sur un périphérique connecté, sans réinstaller l'application. Connectez le périphérique à l'ordinateur hôte via un port USB seulement au cours du groupement et au cours de l'installation de l'application sur le périphérique. Vous pouvez déconnecter le périphérique du port USB au cours du débogage. Toutefois, assurez-vous qu'il existe une connexion réseau entre le périphérique et l'ordinateur hôte tout au long de la session de débogage.

Préparation au débogage de l'application

Avant de commencer le débogage via USB ou via un réseau, procédez comme suit :

1 (Windows) Vérifiez que le pilote USB approprié est installé.

Dans Windows, installez le pilote Android USB. Reportez-vous à la documentation accompagnant la version du SDK Android pour plus d'informations. Pour plus d'informations, voir « Installation de pilotes de périphérique USB pour les périphériques Android (Windows) » à la page 16.

2 Veillez à ce que le débogage USB soit activé sur le périphérique.

Dans les paramètres du périphérique, accédez à Applications > Développement, et activez le débogage USB.

Recherche de périphériques connectés

Lorsque vous exécutez ou déboguez une application mobile sur un périphérique, Flash Builder vérifie les périphériques connectés. Si Flash Builder détecte un seul périphérique connecté en ligne, il déploie et lance l'application. Dans le cas contraire, Flash Builder lance la boîte de dialogue Choisir un périphérique pour les scénarios suivants :

- Aucun périphérique n'est trouvé
- Un seul périphérique déconnecté est trouvé ou sa version du système d'exploitation n'est pas prise en charge
- Plusieurs périphériques connectés trouvés

Si plusieurs périphériques sont détectés, la boîte de dialogue Choisir un périphérique répertorie les périphériques et leur état (en ligne ou hors connexion). Sélectionnez le périphérique à lancer.

La boîte de dialogue Choisir un périphérique répertorie la version du système d'exploitation et la version d'AIR. Si Adobe AIR n'est pas installé sur le périphérique, Flash Builder l'installe automatiquement.

Configuration du débogage réseau

Effectuez cette procédure seulement si vous déboguez une application via un réseau.

Préparation au débogage par le biais du réseau

Avant de déboguer une application via le réseau, procédez comme suit :

1 Dans Windows, ouvrez le port 7935 (port du débogueur Flash Player) et le port 7 (port echo/ping).

Pour obtenir des instructions détaillées, reportez-vous à cet article Microsoft TechNet.

Dans Windows Vista, désélectionnez la case Connexion réseau sans fil dans Pare-feu Windows > Modifier les paramètres > Avancés.

2 Sur le périphérique, configurez les paramètres sans fil dans Paramètres > Sans fil et Réseau.

Sélection d'une interface réseau principale

Votre ordinateur hôte peut être connecté à plusieurs interfaces réseau simultanément. Toutefois, vous pouvez sélectionner une interface réseau principale à utiliser pour le débogage. Vous sélectionnez cette interface en ajoutant une adresse d'hôte dans le fichier de package Android APK.

1 Dans Flash Builder, ouvrez la fenêtre Préférences.

2 Sélectionnez Flash Builder > Plateformes cibles.

La boîte de dialogue répertorie toutes les interfaces réseau disponibles sur l'ordinateur hôte.

3 Sélectionnez l'interface réseau que vous souhaitez incorporer dans le package Android APK.

Assurez-vous que l'interface réseau sélectionnée est accessible à partir du périphérique. Si le périphérique ne peut pas accéder à l'interface réseau sélectionnée pendant qu'il établit une connexion, Flash Builder affiche une boîte de dialogue vous invitant à fournir l'adresse IP de l'ordinateur hôte.

Débogage de l'application

1 Connectez le périphérique par le biais d'un port USB ou via une connexion réseau.

2 Sélectionnez Exécuter > Déboguer les configurations pour mettre au point une configuration de lancement destinée au débogage.

- Pour la Méthode de lancement, choisissez Sur le périphérique.
- Sélectionnez Déboguer via USB ou Déboguer par le biais du réseau.

  Lors du premier débogage de l'application via un réseau, vous pouvez installer l'application sur le périphérique via USB. Pour déboguer une application via connexion USB, connectez le périphérique à l'ordinateur hôte par le biais d'un port USB.

  Une fois l'application installé, si vous ne voulez pas vous connecter via USB pour des sessions de débogage suivantes, désélectionnez l'option Installer l'application sur le périphérique via USB.

- (Facultatif) Effacez les données de l'application à chaque lancement.

  Sélectionnez cette option pour conserver l'état de l'application pour chaque session de débogage. Cette option s'applique uniquement si l'occurrence sessionCachingEnabled est définie sur True dans votre application.

3 Sélectionnez Déboguer pour lancer une session de débogage.

Le débogueur se lance et attend le démarrage de l'application. La session de débogage commence lorsque le débogueur établit une connexion avec le périphérique.

Lorsque vous essayez de déboguer sur un périphérique via un réseau, l'application affiche parfois une boîte de dialogue qui vous invite à saisir une adresse IP. Cette boîte de dialogue indique que le débogueur n'a pas pu se connecter. Assurez-vous que le périphérique est correctement connecté au réseau sans fil et que l'ordinateur qui exécute Flash Builder est accessible depuis ce réseau.

Remarque : *sur un réseau d'entreprise, le réseau d'un hôtel ou un autre réseau invité, parfois le périphérique ne peut pas se connecter à l'ordinateur, même si les deux sont sur le même réseau.*

Si vous effectuez un débogage via un réseau et que l'application a été installée auparavant sur le périphérique, démarrez le débogage en tapant l'adresse IP de l'ordinateur hôte.

Brent Arnold, expert Flex certifié par Adobe, a conçu un didacticiel vidéo traitant du débogage d'une application sur USB pour un périphérique Android.

Voir aussi

Débogage et applications groupées pour les périphériques (vidéo)

Débogage d'une application sur un périphérique Apple iOS

Pour déboguer une application sur un périphérique Apple iOS, déployez et installez manuellement le package iOS de débogage (fichier IPA) sur le périphérique iOS. Le déploiement automatique n'est pas pris en charge pour la plateforme Apple iOS.

1. Connectez le périphérique Apple iOS à votre ordinateur de développement.
2. Lancez iTunes sur le périphérique iOS.

   Remarque : *vous devez disposer d'iTunes pour installer l'application sur le périphérique iOS et obtenir l'ID du périphérique iOS.*
3. Dans Flash Builder, sélectionnez Exécuter > Configurations de débogage.
4. Dans la boîte de dialogue Configurations de débogage, procédez comme suit :
   - **a** Sélectionnez l'application que vous souhaitez déboguer.
   - **b** Sélectionnez la plateforme cible Apple iOS.
   - **c** Sélectionnez la méthode de lancement Sur le périphérique.
   - **d** Sélectionnez l'une des méthodes de groupement suivantes :

     Standard Utilisez cette méthode pour grouper une version de l'application de qualité analogue à une version validée qui peut s'exécuter sur des périphériques Apple iOS. L'application groupée de cette façon offre des performances semblables à celles du package final de la version validée et peut être soumise à l'App Store d'Apple.

     Toutefois, notez que cette méthode de création d'un fichier iOS de débogage (IPA) prend plusieurs minutes.

     Rapide Utilisez cette méthode pour créer rapidement un fichier IPA, puis exécutez et déboguez le fichier sur le périphérique. Cette méthode est appropriée à des fins de test d'application. L'application traitée de cette façon présente des performances inférieures à celles d'une version validée et ne peut donc pas être soumise à l'App Store d'Apple.
   - **e** Cliquez sur Configurer pour sélectionner le certificat de signature de code, le fichier de configuration et le contenu du groupement appropriés.
   - **f** Cliquez sur Configuration du débogage réseau pour sélectionner l'interface réseau que vous souhaitez incorporer dans le package iOS de débogage.

Remarque : *votre ordinateur hôte peut être connecté à plusieurs interfaces réseau simultanément. Toutefois, vous pouvez sélectionner une interface réseau principale à utiliser pour le débogage.*

g Cliquez sur Déboguer. Flash Builder affiche une boîte de dialogue vous invitant à fournir un mot de passe. Entrez votre mot de passe de certificat P12.

Flash Builder génère le fichier IPA de débogage et le place dans le dossier bin-debug.

5 Sur le périphérique iOS, procédez comme suit :

1 (Facultatif) Dans iTunes, sélectionnez Fichier > Ajouter à la bibliothèque et recherchez dans l'arborescence le fichier de profil d'approvisionnement mobile (extension de nom de fichier .mobileprovision) que vous avez obtenu d'Apple.

2 Dans iTunes, sélectionnez Fichier > Ajouter à la bibliothèque et recherchez dans l'arborescence le fichier IPA de débogage que vous avez généré à l'étape 4.

3 Synchronisez le périphérique iOS avec iTunes en sélectionnant Fichier > Synchroniser.

4 Flash Builder essaie de se connecter à l'adresse d'hôte spécifiée dans le fichier IPA de débogage. Si l'application ne peut pas se connecter à l'adresse d'hôte, Flash Builder affiche une boîte de dialogue vous invitant à fournir l'adresse IP de l'ordinateur hôte.

Remarque : *si vous n'avez pas modifié votre code ni vos ressources depuis la création du dernier package IPA de débogage, Flash Builder ignore le groupement et effectue le débogage de l'application. Cela signifie que vous pouvez lancer l'application installée sur le périphérique et cliquer sur Déboguer pour vous connecter au débogueur Flash Builder. De cette manière, vous pouvez effectuer plusieurs débogages successifs sans chaque fois grouper l'application.*

Chapitre 8 : Groupement et exportation d'une application mobile

Utilisez la fonction Flash Builder Exporter vers une version validée pour grouper et exporter la version validée d'une application mobile. Une version validée est généralement la version finale de l'application que vous souhaitez télécharger ou essayer sur un périphérique.

Vous pouvez exporter un package d'application spécifique à une plateforme pour l'installer ultérieurement sur un périphérique. Le package ainsi obtenu peut être déployé et installé de la même manière qu'une application native.

Exportation de packages Android APK pour publication

Avant d'exporter une application mobile, vous pouvez personnaliser les autorisations Android. Personnalisez manuellement les paramètres dans le fichier descripteur de l'application. Ces paramètres figurent dans le bloc `<android>` du fichier bin-debug/*app_name*-app.xml. Pour plus d'informations, voir Définition des propriétés d'une application AIR.

Si vous exportez l'application pour l'installer ultérieurement sur un périphérique, installez le package d'application à l'aide des outils fournis par le fournisseur du système d'exploitation du périphérique.

1 Dans Flash Builder, sélectionnez Projet > Exporter vers une version validée.

2 Sélectionnez le projet et l'application que vous souhaitez exporter.

3 Sélectionnez les plateformes cibles et l'emplacement où exporter le projet.

4 Exportez et signez un package d'application spécifique à la plateforme.

   Vous pouvez grouper votre application avec une signature numérique pour chaque plateforme cible ou en tant qu'application AIR signée numériquement pour le bureau.

   Vous pouvez également exporter l'application en tant que fichier AIRI intermédiaire qui peut être signé ultérieurement. Si vous sélectionnez cette option, utilisez ultérieurement l'outil de ligne de commande AIR adt pour grouper le fichier AIRI en tant que fichier APK. Installez ensuite le fichier APK sur le périphérique à l'aide d'outils spécifiques à la plateforme (par exemple, avec le SDK Android, utilisez adb). Pour plus d'informations sur l'utilisation des outils de ligne de commande pour grouper votre application, voir « Développement et déploiement d'une application mobile sur la ligne de commande » à la page 120.

5 À partir de la page Paramètres de groupement, vous pouvez spécifier le certificat numérique, le contenu du groupement ainsi que les paramètres de boutique d'applications.

   Signature numérique Cliquez sur l'onglet Signature numérique pour créer ou localiser un certificat numérique qui représente l'identité de l'éditeur de l'application. Vous pouvez aussi spécifier un mot de passe pour le certificat sélectionné.

   Si vous créez un certificat, il est *auto-signé*. Vous pouvez obtenir un certificat commercial signé auprès d'un fournisseur de certificats. Voir Signature numérique de vos applications AIR.

   Contenu du groupement (Facultatif) Cliquez sur l'onglet Contenu du groupement pour spécifier les fichiers à inclure dans le groupement.

Déploiement Si vous voulez également installer l'application sur un périphérique, cliquez sur la page Déploiement et sélectionnez Installer et lancer l'application sur tout périphérique connecté. Assurez-vous que vous avez connecté un ou plusieurs périphériques aux ports USB de votre ordinateur.

Si AIR n'est pas déjà installé sur le périphérique d'un utilisateur, vous pouvez sélectionner ou spécifier une URL de façon à télécharger Adobe AIR pour le package d'application. L'URL par défaut renvoie vers Android Market. Vous pouvez toutefois remplacer l'URL par défaut en saisissant votre propre URL ou en sélectionnant l'URL qui pointe vers un emplacement de l'App Store Amazon.

6 Cliquez sur Terminer.

Flash Builder crée `ApplicationName.apk` dans le répertoire spécifié dans le premier volet (par défaut, au niveau supérieur de votre projet). Si le périphérique a été connecté à votre ordinateur au cours de l'exportation, Flash Builder installe l'application sur le périphérique.

Brent Arnold, expert Flex certifié par Adobe, a créé un didacticiel vidéo sur l'exportation d'une application mobile pour la plateforme Android.

Exportation de packages Apple iOS pour publication

Vous pouvez créer et exporter un package iOS pour une distribution ponctuelle ou pour une soumission à l'App Store d'Apple.

1 Dans Flash Builder, sélectionnez Projet > Exporter vers une version validée.

2 Sélectionnez Apple iOS en tant que plateforme cible pour exporter et signer un package IPA.

Cliquez sur Suivant.

3 Avant de sélectionner le type de package, assurez-vous d'avoir obtenu le certificat de signature de code, le mot de passe et le profil d'approvisionnement appropriés auprès d'Apple. Pour soumettre l'application à l'App Store d'Apple, vous avez besoin d'un profil de configuration de distribution. Pour plus d'informations, voir Obtention de fichiers de développement auprès d'Apple.

Vous pouvez sélectionner l'un des types de packages suivants :

Distribution ponctuelle pour une distribution limitée Pour une distribution limitée de l'application

Package final de la version validée pour l'App Store d'Apple Pour soumettre l'application à l'App Store d'Apple.

4 (Facultatif) Cliquez sur l'onglet Contenu du groupement pour spécifier les fichiers à inclure dans le package

5 Cliquez sur Terminer.

Flash Builder valide la configuration des paramètres du package, puis compile l'application. Une fois le groupement terminé, vous pouvez installer le fichier IPA sur un périphérique Apple iOS connecté.

Serge Jespers, évangéliste Adobe, explique dans cette vidéo Adobe TV comment générer et exporter des applications iOS à l'aide de Flash Builder.

Pour grouper le fichier IPA à l'aide de l'outil ADT (AIR Developer Tool), voir Packages iOS dans *Création d'applications AIR*.

Chapitre 9 : Déploiement

Déploiement d'une application sur un périphérique mobile

Déploiement d'une application sur un périphérique Google Android

Vous pouvez utiliser Flash Builder pour déployer et installer une application directement sur un périphérique Android. Lorsque vous installez un package sur un périphérique sur lequel Adobe AIR n'est pas installé, Flash Builder installe automatiquement AIR.

1 Connectez le périphérique Google Android à votre ordinateur de développement.

   Flash Builder accède au périphérique connecté au port USB de l'ordinateur. Assurez-vous d'avoir configuré les pilotes de périphérique USB requis. Voir « Connexion des périphériques Google Android » à la page 15.

2 Dans Flash Builder, sélectionnez Exécuter > Exécuter les configurations. Dans la boîte de dialogue Exécuter les configurations, sélectionnez l'application mobile que vous voulez déployer.

3 Sélectionnez la méthode de configuration de lancement Sur le périphérique.

4 (Facultatif) Indiquez si vous souhaitez effacer les données de l'application à chaque lancement.

5 Cliquez sur Appliquer.

Flash Builder déploie et lance l'application sur le périphérique Android.

Brent Arnold, expert Flex certifié par Adobe, a conçu un didacticiel vidéo traitant de la configuration et de l'exécution de votre application sur un périphérique Android.

Déploiement d'une application sur un périphérique Apple iOS

Sur un périphérique iOS, vous devez déployer et installer manuellement une application (fichier IPA), car la plateforme Apple iOS ne prend pas en charge le déploiement automatique.

Avant de déployer une application sur un périphérique iOS, vous devez obtenir le certificat de signature de code, le mot de passe et le profil d'approvisionnement appropriés auprès d'Apple. Pour plus d'informations, voir « Connexion de périphériques Apple iOS » à la page 18.

1 Connectez le périphérique Apple iOS à votre ordinateur de développement.

2 Lancez iTunes sur votre ordinateur de développement.

   Remarque : *vous devez disposer d'iTunes pour installer l'application sur le périphérique iOS et obtenir l'UDID (Unique Device Identifier) du périphérique iOS.*

3 Dans Flash Builder, sélectionnez Exécuter > Exécuter les configurations.

4 Dans la boîte de dialogue Exécuter les configurations, procédez comme suit :

   a Sélectionnez l'application que vous souhaitez déployer.

   b Sélectionnez la plateforme cible Apple iOS.

   c Sélectionnez la méthode de lancement Sur le périphérique.

d Sélectionnez l'une des méthodes de groupement suivantes :

Standard Utilisez cette méthode pour grouper une version de l'application de qualité analogue à une version validée qui peut s'exécuter sur des périphériques Apple iOS.

La méthode standard de création de packages traduit le code en octets du fichier SWF de l'application en instructions ARM avant de créer le package. En raison de cette étape de traduction supplémentaire, cette méthode de création d'un fichier d'application (IPA) prend plusieurs minutes. La méthode standard prend plus de temps que la méthode rapide. Toutefois, l'application traitée de cette façon présente des performances dignes d'une version validée et peut donc être soumise à l'App Store d'Apple.

Rapide Utilisez cette méthode pour créer rapidement un fichier IPA.

La méthode rapide de création de packages fait l'impasse sur la traduction du code en octets et regroupe simplement le fichier SWF et les ressources de l'application avec l'environnement d'exécution AIR précompilé. La méthode rapide de création de packages est plus rapide que la méthode standard. Toutefois, l'application traitée de cette façon présente des performances inférieures à celle d'une version validée et ne peut donc pas être soumise à l'App Store d'Apple.

Remarque : *entre les méthodes de création de packages standard et rapide, la différence réside dans l'environnement d'exécution ou le fonctionnement.*

e Cliquez sur Configurer pour sélectionner le certificat de signature de code, le fichier de configuration et le contenu du groupement appropriés.

f Cliquez sur Exécuter. Flash Builder affiche une boîte de dialogue vous invitant à fournir un mot de passe. Entrez votre mot de passe de certificat P12.

Flash Builder génère le fichier IPA et le place dans le dossier bin-debug.

5 Suivez les étapes ci-dessous sur votre ordinateur de développement :

1. Dans iTunes, sélectionnez Fichier > Ajouter à la bibliothèque et recherchez dans l'arborescence le fichier de profil d'approvisionnement mobile (extension de nom de fichier .mobileprovision) que vous avez obtenu d'Apple.

   Vous pouvez également ajouter le fichier de profil d'approvisionnement mobile par glisser-déposer dans iTunes.

2. Dans iTunes, sélectionnez Fichier > Ajouter à la bibliothèque et recherchez dans l'arborescence le fichier IPA que vous avez généré à l'étape 4.

   Vous pouvez également ajouter le fichier IPA par glisser-déposer dans iTunes.

3. Synchronisez le périphérique iOS avec iTunes en sélectionnant Fichier > Synchroniser.

L'application est déployée sur le périphérique iOS et vous pouvez la lancer.


Holly Schinsky, experte Flex certifiée par Adobe, explique la procédure de développement iOS, y compris les étapes permettant d'obtenir des ID de périphérique, un certificat de signature de code, un mot de passe et un profil de configuration auprès d'Apple.

Développement et déploiement d'une application mobile sur la ligne de commande

Vous pouvez créer une application mobile sans Flash Builder. Vous pouvez utiliser à la place les outils de ligne de commande mxmlc, adl et adt.

Le processus général de développement et de déploiement d'une application mobile sur un périphérique à l'aide des outils de ligne de commande est détaillé ci-dessous. Chacune de ces étapes sera décrite ultérieurement de façon plus détaillée :

1 Compilez l'application à l'aide de l'outil mxmlc.

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

Cette étape requiert que l'ensemble de paramètres `configname` soit défini sur « airmobile ».

2 Testez l'application dans AIR Debug Launcher (ADL) à l'aide de l'outil adl.

```
adl MyMobileApp-app.xml -profile mobileDevice
```

Cette étape nécessite la création d'un fichier descripteur de l'application et son utilisation en tant qu'argument dans l'outil adl. Vous devez également spécifier le profil `mobileDevice`.

3 Groupez l'application à l'aide de l'outil adt.

```
adt -package -target apk SIGN_OPTIONS MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

Cette étape requiert la création préalable d'un certificat.

4 Déployez l'application sur votre périphérique mobile. Pour déployer l'application sur un périphérique Android, utilisez l'outil adb.

```
adb install -r MyMobileApp.apk
```

Cette étape requiert la connexion préalable de votre périphérique mobile à votre ordinateur via USB.

Compilation d'une application mobile à l'aide de mxmlc

Vous pouvez compiler des applications mobiles avec le compilateur de ligne de commande mxmlc. Pour utiliser mxmlc, affectez au paramètre `configname` la valeur `airmobile` ; par exemple :

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

En affectant `+configname=airmobile`, vous invitez le compilateur à utiliser le fichier airmobile-config.xml. Ce fichier se trouve dans le répertoire sdk/frameworks. Ce fichier effectue les tâches suivantes :

- S'applique au thème mobile.swc.
- Effectuez les modifications suivantes dans le chemin d'accès à la bibliothèque :
  - Supprime libs/air du chemin d'accès à la bibliothèque. Les applications mobiles ne prennent pas en charge les classes Window et WindowedApplication.
  - Supprime libs/mx du chemin d'accès à la bibliothèque. Les applications mobiles ne prennent pas en charge les composants MX (autres que les graphiques).
  - Ajoute libs/mobile au chemin d'accès à la bibliothèque.
- Supprime les espaces de nom ns.adobe.com/flex/mx et www.adobe.com/2006/mxml. Les applications mobiles ne prennent pas en charge les composants MX (autres que les graphiques).
- Désactive l'accessibilité.
- Supprime les entrées RSL ; les applications mobiles ne prennent pas en charge les RSL.

Le compilateur mxmlc génère un fichier SWF.

Test d'une application mobile à l'aide d'adl

Vous utilisez AIR Debug Launcher (ADL) pour tester une application mobile. Vous utilisez ADL pour exécuter et tester une application sans être tenu de la grouper ni de l'installer auparavant sur un périphérique.

Débogage à l'aide de l'outil adl

ADL imprime les instructions trace et les erreurs d'exécution au format de sortie standard, mais ne prend pas en charge les points d'arrêt ni d'autres fonctions de débogage. Vous pouvez utiliser un environnement de développement intégré tel que Flash Builder pour les problèmes complexes de débogage.

Lancement de l'outil adl

Pour lancer l'outil adl à partir de la ligne de commande, passez votre fichier descripteur d'application de l'application mobile et définissez le paramètre `profile` sur `mobileDevice`, comme le montre l'exemple suivant :

```
adl MyMobileApp-app.xml -profile mobileDevice
```

Le profil `mobileDevice` définit un ensemble de fonctionnalités pour les applications qui sont installées sur des périphériques mobiles. Pour obtenir des informations précises à propos du profil `mobileDevice`, voir Capacités des différents profils.

Création d'un descripteur d'application

Si vous n'avez pas utilisé Flash Builder pour compiler votre application, vous devez créer manuellement le fichier descripteur de l'application. Vous pouvez utiliser le fichier /sdk/samples/descriptor-sample.xml en tant que base. En général, au minimum, effectuez les modifications suivantes :

- Pointez l'élément `<initialWindow><content>` sur le nom du fichier SWF de votre application mobile :

```
<initialWindow>
    <content>MyMobileApp.swf</content>
    ...
</initialWindow>
```

- Modifiez le titre de l'application, car c'est ainsi qu'il apparaît sous l'icône de l'application sur votre périphérique mobile. Pour modifier le titre, éditez l'élément `<name><text>` :

```
<name>
    <text xml:lang="en">MyMobileApp by Nick Danger</text>
</name>
```

- Ajoutez un bloc `<android>` pour définir les autorisations spécifiques à Android pour l'application. Selon les services utilisés par votre périphérique, vous pouvez souvent utiliser l'autorisation suivante :

```
<application>
    ...
    <android>
        <manifestAdditions>
            <![CDATA[<manifest>
                <uses-permission android:name="android.permission.INTERNET"/>
            </manifest>]]>
        </manifestAdditions>
    </android>
</application>
```

Vous pouvez aussi utiliser le fichier descripteur pour définir la hauteur et la largeur de l'application, l'emplacement des fichiers d'icône, les informations relatives à la version et d'autres détails concernant l'emplacement d'installation.

Pour plus d'informations à propos de la création et de la modification des fichiers descripteurs de l'application, voir fichiers descripteurs d'application AIR.

Groupement d'une application mobile à l'aide d'adt

Vous utilisez AIR Developer Tool (ADT) pour grouper les applications mobiles sur la ligne de commande. L'outil adt peut créer un fichier APK que vous pouvez déployer sur un périphérique mobile Android.

Création d'un certificat

Avant de créer un fichier APK, créez un certificat. A des fins de développement, vous pouvez utiliser un certificat auto-signé. Vous pouvez créer un certificat auto-signé avec l'outil adt, comme le montre l'exemple suivant :

```
adt -certificate -cn SelfSign -ou QE -o "Example" -c US 2048-RSA newcert.p12 password
```

L'outil adt crée le fichier newcert.p12 dans le répertoire en cours. Vous passez ce certificat à adt lorsque vous groupez votre application. N'utilisez pas de certificats auto-signés pour les applications de production. Ils fournissent uniquement une assurance restreinte aux utilisateurs. Pour plus d'informations sur la signature des fichiers d'installation AIR à l'aide d'un certificat émis par une autorité de certification reconnue, voir Signature des applications AIR.

Création du fichier de package

Pour créer le fichier APK pour Android, passez les détails sur l'application à l'outil adt, y compris le certificat, comme l'illustre l'exemple suivant :

```
adt -package -target apk -storetype pkcs12 -keystore newcert.p12 -keypass password
MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

La sortie de l'outil adt est un fichier *appname*.apk.

Groupement pour iOS

Pour grouper des applications mobiles pour iOS, vous devez obtenir un certificat de développement d'Apple, ainsi qu'un fichier de configuration. Cela exige que vous rejoigniez le programme des développeurs Apple. Pour plus d'informations, voir « Préparation à l'installation et au déploiement d'une application sur un périphérique iOS » à la page 18.


Piotr Walczyszyn, expert en programmation Flex, explique comment grouper l'application avec ADT en utilisant Ant pour les périphériques iOS.


Dans son blog, Valentin Simonov fournit des informations supplémentaires sur la façon de publier votre application sur iOS.

Déploiement d'une application mobile sur un périphérique à l'aide d'adb

Vous utilisez Android Debug Bridge (adb) pour déployer le fichier APK sur un périphérique mobile qui exécute Android. L'outil adb fait partie du SDK Android.

Connexion du périphérique à un ordinateur

Avant d'exécuter adb pour déployer le fichier APK sur votre périphérique mobile, connectez le périphérique à votre ordinateur. Sur les systèmes Windows et Linux, la connexion d'un périphérique requiert les pilotes USB.

Pour plus d'informations sur l'installation des pilotes USB pour votre périphérique, voir Utilisation des périphériques matériels.

Déploiement de l'application sur un périphérique local

Après avoir connecté le périphérique à l'ordinateur, vous pouvez déployer l'application sur le périphérique. Pour déployer l'application à l'aide de l'outil adb, utilisez l'option `install` et affectez le nom de votre fichier APK, comme le montre l'exemple suivant :

```
adb install -r MyMobileApp.apk
```

Utilisez l'option `-r` pour remplacer l'application si vous l'avez installée au préalable. Dans le cas contraire, vous devez désinstaller l'application déployée chaque fois que vous souhaitez déployer une version plus récente sur le périphérique mobile.

Déploiement de l'application dans des magasins en ligne


Lee Brimlow montre comment déployer une nouvelle application AIR pour Android dans Android Market.


Christian Cantrell explique comment déployer l'application sur Amazon Appstore pour Android.

Voir aussi

Android Debug Bridge