ADOBE® FLEX® 4.6 および ADOBE® FLASH® BUILDER™ 4.6 を使用したモバイルアプリケーションの開発

法律上の注意

最終更新日 2011/12/12

目次

第 1 章：はじめに

モバイルアプリケーションの使用について

Adobe Flex では、Flex と Adobe Flash Builder をスマートフォンとタブレット向けの開発に使用できます。Adobe AIR を使用すると、デスクトッププラットフォームと同じ簡単さと品質で、Flex でモバイルアプリケーションを開発できるようになりました。

既存の Flex コンポーネントの多くが、タッチベースのスクロールの追加サポートを含めて、モバイルデバイスで動作するように拡張されました。Flex には、電話機とタブレットの標準デザインパターンに従ったアプリケーションを容易に作成できるように設計された新しい一連のコンポーネントも含まれています。

Flash Builder にも、モバイルデバイス用のアプリケーション開発をサポートする新機能が追加されています。Flash Builder では、アプリケーションの開発、テストおよびデバッグをデスクトップ上で行うことも、モバイルデバイス上で直接行うこともできます。


アドビのエバンジェリストの Mark Doherty が、デスクトップ、モバイルフォンおよびタブレットのアプリケーション作成に関するビデオを公開しています。


アドビのエバンジェリストの James Ward が、Flex を使用したモバイルアプリケーションの作成 に関するビデオを公開しています。


Adobe Community Professional の Joseph Labrecque 氏が、Mobile Flex Demonstration というブログを公開しています。

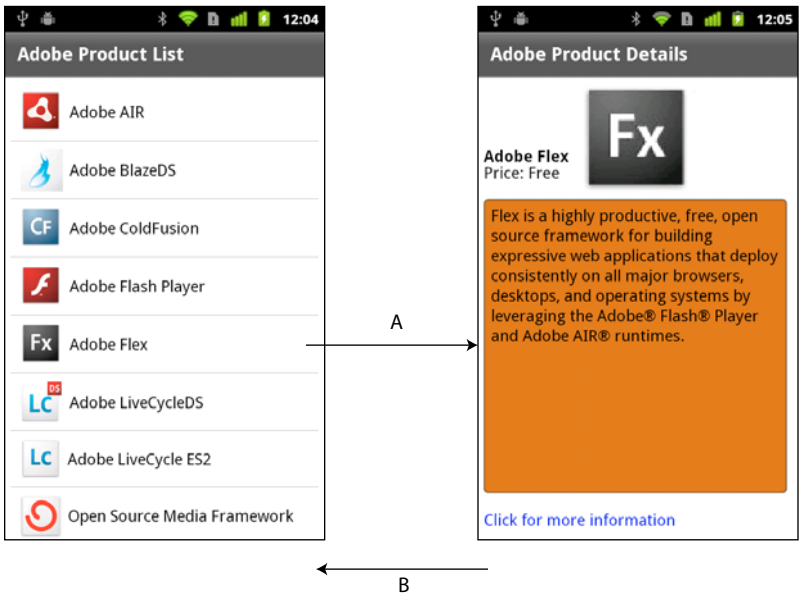

Flash 開発者の Fabio Biondi は、Flash Builder を使用して Android デバイス用の AIR ベースの YouTube Player を作成しました。

モバイルアプリケーションのデザイン

モバイルデバイスで使用できるスクリーンサイズはブラウザーと比較して小さいため、モバイルアプリケーションは通常、ブラウザーベースのアプリケーションとは異なるデザインパターンに従います。モバイルアプリケーション用に開発する際には、通常はモバイルデバイスで表示するための一連のビューにコンテンツを分割します。

各ビューには、1 つのタスクに特化したコンポーネントまたは単一セットの情報を含むコンポーネントが含まれています。通常は、ビューのコンポーネントをタップすると、1 つのビューから別のビューに「ドリルダウン」つまり変更されます。ユーザーはデバイスの戻るボタンを使用して前のビューに戻ることができます。または、アプリケーションにナビゲーションを作成することもできます。

次の例では、アプリケーションの基本ビューに製品リストが表示されています。



A. リストアイテムを選択するとアプリケーションのビューが変更されます。B. デバイスの戻るボタンを使用して前のビューに戻ります。

ユーザーはリストの製品を選択して、詳しい情報を入手します。選択すると、ビューが変更されて、製品の詳しい説明が表示されます。

モバイル、Web およびデスクトッププラットフォームで使用するアプリケーションをデザインしている場合、通常はプラットフォームごとに別々のユーザーインターフェイスをデザインします。ただし、基礎となるモデルおよびデータアクセスコードはすべてのプラットフォームのアプリケーションで共有できます。

電話機およびタブレット用のアプリケーションの作成

タブレット用のアプリケーションの場合は、電話機の場合ほどスクリーンサイズの制限に注意する必要はありません。タブレット用のアプリケーションを小さなビューで構成する必要はありません。代わりに、標準の Spark の Application コンテナを使用し、サポートされているモバイルコンポーネントとスキンを使用して、アプリケーションを作成できます。

注意：Spark の Application コンテナに基づいてモバイルフォン用のアプリケーションを作成できます。ただし、通常は ViewNavigatorApplication コンテナと TabbedViewNavigatorApplication コンテナを代わりに使用します。

電話機の場合と同様にタブレットの場合も、Flash Builder でモバイルプロジェクトを作成してください。モバイルアプリケーション用に最適化されたコンポーネントとスキンを利用するためには、タブレットと電話機のアプリケーションで同じモバイルテーマを使用する必要があります。

Flash Builder でのモバイルアプリケーションの作成

Flash Builder には、モバイル開発用の生産性の高いデザイン、ビルドおよびデバッグのワークフローが用意されています。Flash Builder のモバイル機能の目標は、デスクトップや Web のアプリケーション開発と同じように簡単に、ActionScript ベースまたは Flex ベースのモバイルアプリケーションを開発できるようにすることです。

Flash Builder には、テストとデバッグ用の 2 つのオプションが用意されています。AIR Debug Launcher（ADL）を使用して、デスクトップでアプリケーションを起動してデバッグできます。さらに細かく制御する場合は、モバイルデバイスでアプリケーションを直接起動してデバッグします。いずれの場合も、ブレークポイントの設定、変数パネルと式パネルを使用したアプリケーションの状態の調査などの Flash Builder のデバッグ機能を使用できます。

アプリケーションをデプロイする準備が整ったら、デスクトップや Web のアプリケーションを準備する場合と同様に、リリースビルドの書き出し処理を使用します。主な違いは、モバイルプロジェクトのリリースビルドを書き出す際には、.air ファイルではなくネイティブインストーラーとしてビルドがパッケージされることです。例えば、Android の場合は、ネイティブの Android アプリケーションパッケージと同じような .apk ファイルが Flash Builder で作成されます。ネイティブインストーラーでは、各プラットフォームのネイティブアプリケーションと同じ方法で AIR ベースのアプリケーションを配布できます。

AIR でのモバイルアプリケーションのデプロイ

Flex で構築したモバイルアプリケーションは、モバイルデバイス対応の Adobe AIR を使用してデプロイします。モバイルアプリケーションのデプロイ先のデバイスでは、AIR がサポートされている必要があります。

作成したアプリケーションでは、AIR とモバイルプラットフォームとの統合を十分に活用できます。例えば、モバイルアプリケーションでは、ハードウェアの戻るボタンやメニューのボタンを処理したり、ローカル記憶域にアクセスしたりできます。モバイルデバイス用に提供されている AIR のすべての機能も利用できます。これらの機能には、地理位置情報、加速度計およびカメラの統合が含まれます。

Flex で構築したアプリケーションを実行する前に、モバイルデバイスに AIR をインストールする必要はありません。Flex で構築したアプリケーションを最初に実行すると、AIR をダウンロードするように求めるメッセージが表示されます。

AIR と AIR の機能について詳しく理解するには、次のトピックを参照してください。

- Adobe AIR について
- AIR アプリケーションの呼び出しと終了
- AIR ランタイムとオペレーティングシステムに関する情報の操作
- AIR ネイティブウィンドウの操作
- AIR を使用したローカル SQL データベースの操作

モバイルアプリケーションを開発する際には、WindowedApplication と Window という AIR 用の Flex コンポーネントは使用できません。代わりに、ViewNavigatorApplication コンテナと TabbedViewNavigatorApplication コンテナを使用してください。タブレット用のモバイルアプリケーションを開発する際には、Spark の Application コンテナも使用できます。

詳しくは、Using the Flex AIR components および 33 ページの「モバイルアプリケーションとスプラッシュ画面の定義」を参照してください。

アプリケーションでの Mobile テーマの使用

テーマは、アプリケーションのビジュアルコンポーネントのルック＆フィールを定義します。テーマを使って、アプリケーションのカラースキームや一般的フォントなど簡単なものを定義したり、アプリケーションで使用するすべてのコンポーネントの外観を変更したりすることができます。

Flex コンポーネントに対して CSS スタイルを設定できるのは、現在のテーマにそのスタイルが含まれている場合のみです。現在のテーマで CSS スタイルがサポートされているかどうかを判断するには、Adobe Flash Platform 用 ActionScript 3.0 リファレンスガイドのスタイルのエントリを参照してください。

Flex では、Mobile、Spark および Halo の 3 つの主要なテーマがサポートされています。モバイルアプリケーションの作成時には、Mobile テーマによって Flex コンポーネントのデフォルトの外観が定義されます。Flex コンポーネントと Mobile テーマとの互換性を維持するために、Adobe では、一部のコンポーネント用に新しいスキンを作成しました。そのため、一部のコンテンツにはテーマに固有のスキンがあります。

Flex を使用すると、スクリーンサイズや解像度がそれぞれ異なる様々なモバイルデバイスを対象としたアプリケーションを構築できます。Flex では、DPI に依存しないスキンをモバイルコンポーネントに提供することで、解像度に依存しないアプリケーションの作成プロセスを単純化しています。モバイルスキンについて詳しくは、154 ページの「モバイルのスキン適用の基本」を参照してください。

スタイルおよびテーマについて詳しくは、Styles and themes および 154 ページの「モバイルのスタイル」を参照してください。

コミュニティリソース

Flex と Flash Builder の新機能については、次を参照してください。

- Flex 4.5 SDK のご紹介（アドビのプロダクトマネージャー Deepa Subramaniam による記事）
- Flex SDK と Flash Builder を使用したモバイルアプリケーション開発（アドビのプロダクトデザイナー Narciso Jaramillo による記事）
- What's new in Flex 4.6 SDK（アドビのプロダクトマネージャー Jacob Surber による記事）および What's New in Flash Builder 4.6（アドビのプロダクトマネージャー Adam Lehman による記事）

Flex デベロッパーセンターには、Flex を使用してモバイルアプリケーションの構築を開始するのに役立つ多くのリソースがあります。

- 「はじめに」の記事、リンクおよびチュートリアル
- Flex で実際に構築したアプリケーションサンプル
- 一般的なコーディングの問題に対する答えが掲載された Flex Cookbook
- Flex のコミュニティおよび Flex 専用の他のサイトへのリンク

別のリソースとして、Flex で開発したアプリケーションに関して、アドビの技術者、プロダクトエバンジェリストおよびお客様が作成したビデオが掲載される Adobe TV があります。その中には、Flash Builder 4.5 での初めてのモバイルアプリケーションのビルドというビデオもあります。

モバイル、デスクトップおよびブラウザーのアプリケーション開発の相違

次のデプロイメント環境に対しては、Flex を使用してアプリケーションを開発します。

ブラウザー ブラウザーで動作する Flash Player で使用される SWF ファイルとしてアプリケーションをデプロイします。

デスクトップ Windows コンピューターや Macintosh など、デスクトップコンピューター用のスタンドアロン AIR アプリケーションをデプロイします。

モバイル 電話機やタブレットなど、モバイルデバイス用のスタンドアロン AIR アプリケーションをデプロイします。

Flash Player ランタイムと AIR ランタイムは似ています。どちらのランタイムでもほとんど同じ操作を実行できます。AIR を使用すると、スタンドアロンアプリケーションをブラウザー外の環境にデプロイできることに加えて、ホストプラットフォームと緊密に統合できます。この統合によって、デバイスのファイルシステムにアクセスする機能、ローカル SQL データベースを作成して操作する機能などを使用できるようになります。

モバイルアプリケーションのデザインおよびデプロイに関する考慮事項

モバイルタッチスクリーンデバイス用のアプリケーションは、デスクトップおよびブラウザーのアプリケーションといくつかの点で異なります。

- タッチ入力による操作を容易にするために、一般にモバイルコンポーネントのヒット領域は、デスクトップやブラウザーのアプリケーションより大きくなっています。
- タッチスクリーンデバイスでは、スクロールなどのアクションのインタラクションパターンが異なります。
- スクリーン領域が限られているため、モバイルアプリケーションでは、一般にスクリーンに一度に表示されるユーザーインターフェイスの量が少なくなるようにデザインされています。
- ユーザーインターフェイスのデザインでは、デバイス間のスクリーン解像度の相違を考慮に入れる必要があります。
- 電話機やタブレットでは、デスクトップデバイスよりも CPU と GPU のパフォーマンスが制限されています。
- モバイルデバイスでは使用できるメモリーが限られているため、アプリケーションでメモリーを注意深く節約する必要があります。
- モバイルアプリケーションは、デバイスが通話やテキストメッセージを受信したときなどに、いつでも終了して再起動される場合があります。

そのため、モバイルデバイス用のアプリケーションの作成には、デスクトップアプリケーションを異なるスクリーンサイズにスケールダウンする以上のことが求められます。Flex では、基礎となるモデルおよびデータアクセスコードをモバイル、ブラウザーおよびデスクトップのプロジェクトで共有しながら、各フォームファクターに適した個別のユーザーインターフェイスを作成できます。

モバイルアプリケーションで Spark および MX コンポーネントを使用する際の制限事項

Flex でモバイルアプリケーションを作成するときには、Spark コンポーネントセットを使用します。Spark コンポーネントは、spark.components.* パッケージで定義されています。ただし、パフォーマンス上の理由により、また、すべての Spark コンポーネントに Mobile テーマのスキンがあるわけではないため、モバイルアプリケーションでは Spark コンポーネントセット全体はサポートされていません。

モバイルアプリケーションでは、mx.* パッケージで定義されている MX コンポーネントセットはサポートされていません。ただし、MX のチャートコントロールと Spacer コントロールはサポートされています。

次の表に、モバイルアプリケーションで、使用できる、使用できないまたは使用に注意を要するコンポーネントを示します。

コンポーネント	コンポーネント	モバイルで使用できるかどうか	メモ
Spark ActionBar Spark BusyIndicator Spark Callout Spark CalloutButton Spark DateSpinner Spark SpinnerList Spark SpinnerListContainer	Spark TabbedViewNavigator Spark TabbedViewNavigatorApplication Spark ToggleSwitch Spark View Spark ViewMenu Spark ViewNavigator Spark ViewNavigatorApplication	はい	これらの新しいコンポーネントは、モバイルアプリケーションをサポートしています。
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	はい	これらのコンポーネントのほとんどには、Mobile テーマのスキンがあります。Label、Image および BitmapImage は、モバイルスキンがない場合でも使用できます。 Group とそのサブクラスなど、一部の Spark レイアウトコンテナにはスキンがありません。そのため、これらのコンポーネントはモバイルアプリケーションで使用できます。
その他の Spark スキナブルコンポーネント		非推奨	前述のコンポーネント以外のスキナブル Spark コンポーネントは、スキンが Mobile テーマ用ではないためお勧めしません。コンポーネントに Mobile テーマのスキンがない場合は、作成するアプリケーション用のスキンを作成できます。
Spark DataGrid	Spark RichEditableText Spark RichText	非推奨	これらのコンポーネントは、パフォーマンス上の理由からお勧めしません。これらのコンポーネントはモバイルアプリケーションで使用できますが、使用すると、パフォーマンスに影響を与える場合があります。 DataGrid コントロールの場合、パフォーマンスはレンダリングするデータの量に基づきます。RichEditableText コントロールと RichText コントロールの場合、パフォーマンスはアプリケーション内のテキストの量とコントロールの数に基づきます。

コンポーネント	コンポーネント	モバイルで使用できるかどうか	メモ
Spacer およびチャート以外の MX コンポーネント		いいえ	モバイルアプリケーションでは、MX の Button、CheckBox、List、DataGrid などの MX コンポーネントはサポートされていません。これらのコンポーネントは、mx.controls.* パッケージと mx.containers.* パッケージの Flex 3 コンポーネントに対応します。
MX の Spacer		はい	Spacer ではスキンを使用しないため、モバイルアプリケーションで使用できます。
MX のチャートコンポーネント		はい（パフォーマンスに影響あり）	AreaChart や BarChart などの MX チャートコントロールは、モバイルアプリケーションで使用できます。MX チャートコントロールは mx.charts.* パッケージ内にあります。 ただし、チャートデータのサイズおよびタイプによっては、モバイルデバイスでのパフォーマンスが最適でなくなる場合があります。 Flash Builder では、デフォルトでモバイルプロジェクトのライブラリパスに MX コンポーネントが含まれていません。アプリケーションで MX チャートコンポーネントを使用するには、mx.swc と charts.swc をライブラリパスに追加します。

モバイルアプリケーションでは、次の Flex 機能がサポートされていません。

- ドラッグ& ドロップ操作がサポートされていません。
- ToolTip コントロールがサポートされていません。
- RSL がサポートされていません。

モバイルアプリケーションのパフォーマンスに関する考慮事項

モバイルデバイスにはパフォーマンスの制約があるため、モバイルアプリケーションの開発には、ブラウザーアプリケーションやデスクトップアプリケーションの開発とは異なる面があります。次に、パフォーマンスに関する考慮事項をいくつか示します。

- **ActionScript でアイテムレンダラーを記述する**

  モバイルアプリケーションの場合は、リストのスクロールのパフォーマンスを最大限まで高める必要があります。最高のパフォーマンスを実現するには、ActionScript でアイテムレンダラーを記述します。MXML でアイテムレンダラーを記述することもできますが、アプリケーションのパフォーマンスが低下する場合があります。

  Flex には、モバイルアプリケーションでの使用のために最適化された spark.components.LabelItemRenderer と spark.components.IconItemRenderer という 2 つのアイテムレンダラーが用意されています。これらのアイテムレンダラーについて詳しくは、Using a mobile item renderer with a Spark list-based control を参照してください。

  ActionScript でのカスタムアイテムレンダラーの作成について詳しくは、Custom Spark item renderers を参照してください。モバイルとデスクトップのアイテムレンダラーの相違について詳しくは、Differences between mobile and desktop item renderers を参照してください。

- **ActionScript とコンパイル済みの FXG グラフィックまたはビットマップを使用してカスタムスキンを開発する**

Flex に付属しているモバイルスキンは、最高のパフォーマンスを実現するために、ActionScript とコンパイル済みの FXG グラフィックで記述されています。MXML でスキンを記述することもできますが、MXML スキンを使用するコンポーネントの数によっては、アプリケーションのパフォーマンスが低下する場合があります。最高のパフォーマンスを得るには、ActionScript でスキンを記述し、コンパイル済みの FXG グラフィックを使用します。詳しくは、Spark Skinning および FXG and MXML graphics を参照してください。

- **StageText を使用するテキスト入力コンポーネントの使用**

  TextInput や TextArea などのテキスト入力コンポーネントを追加する際には、デフォルトを使用してください。これらのコントロールは、テキスト入力の基盤メカニズムとして、ネイティブテキスト入力クラスにフックされている StageText を使用しています。このようにすると、自動訂正、自動キャピタライズ、テキスト制限、カスタムソフトキーボードなどのネイティブ機能に対するパフォーマンスとアクセスがよくなります。

  StageText を使用すると、このコントロールが入っているビューをスクロールできないなど、ある程度の欠点もあります。さらに、StageText ベースのコントロールに対しては、埋め込みフォントやカスタムサイズ設定も使用できません。これらが必要な場合は、TextField クラスに基づいたテキスト入力コントロールを使用できます。

  詳しくは、131 ページの「モバイルアプリケーションでのテキストの使用」を参照してください。

- **モバイルアプリケーションで MX チャートコンポーネントを使用する際には注意する**

  AreaChart や BarChart コントロールなどの MX チャートコントロールは、モバイルアプリケーションで使用できます。ただし、チャートデータのサイズおよびタイプによっては、パフォーマンスに影響する場合があります。


ブロガーの Nahuel Foronda 氏が、ActionScript の Mobile ItemRenderer に関する一連のアーティクルを公開しています。


ブロガーの Rich Tretola 氏が、Creating a List with an ItemRenderer にモバイルアプリケーション用のクックブックエントリを公開しています。

第 2 章：開発環境

Flash Builder での Android アプリケーションの作成

次に、Google Android プラットフォーム用の Flex モバイルアプリケーションを作成する場合の一般的なワークフローを示します。このワークフローでは、モバイルアプリケーションのデザインが既に終了していることを前提としています。詳しくは、1 ページの「モバイルアプリケーションのデザイン」を参照してください。

アドビのエバンジェリストの Mike Jones が、マルチプラットフォームゲーム Mode の開発時に習得した知識を 10 tips when developing for multiple devices で公開しています。

AIR の要件

Flex のモバイルプロジェクトおよび ActionScript のモバイルプロジェクトには、AIR 2.6 以降のバージョンが必要です。モバイルプロジェクトの実行は、AIR 2.6 以降のバージョンをサポートする物理デバイス上で行えます。

AIR 2.6 以降のバージョンは、Android 2.2 以降のバージョンが動作するサポート対象の Android デバイスにのみインストールできます。サポートされている Android デバイスの完全なリストは、Certified Devices を参照してください。また、Android デバイスで Adobe AIR を実行するための最低限の必要システム構成については、モバイルの必要システム構成を参照してください。

注意： AIR 2.6 以降のバージョンをサポートしているデバイスがない場合は、Flash Builder を使用して、デスクトップ上でモバイルアプリケーションを起動およびデバッグできます。

Flex の各バージョンには、必要な Adobe AIR SDK バージョンが含まれています。以前のバージョンの Flex からモバイルアプリケーションをデバイスにインストールした場合は、デバイスから AIR SDK をアンインストールしてください。デバイス上でモバイルアプリケーションを実行またはデバッグするときに、Flash Builder によって正しいバージョンの AIR がインストールされます。

アプリケーションの作成

1 Flash Builder で、ファイル／新規／ Flex モバイルプロジェクトを選択します。

Flex モバイルプロジェクトは、特殊なタイプの AIR プロジェクトです。Flash Builder で他の AIR プロジェクトを作成する場合と同様に、新規プロジェクトウィザードのプロンプトに従ってください。詳しくは、Flex mobile projects を参照してください。

Android 固有のモバイル環境設定をするには、15 ページの「モバイルプロジェクト環境設定の設定」を参照してください。

Flex モバイルプロジェクトを作成すると、プロジェクト用に次のファイルが生成されます。

- ***ProjectName***.mxml

  プロジェクトのデフォルトのアプリケーションファイルです。

  デフォルトでは、このファイルの名前はプロジェクト名と同じ名前になります。プロジェクトに不正な ActionScript 文字が含まれている場合は、Main.mxml という名前になります。この MXML ファイルには、プロジェクトのベース Spark アプリケーションタグが含まれています。ベース Spark アプリケーションタグは、ViewNavigatorApplication または TabbedViewNavigatorApplication の場合があります。

  通常、すべてのビューに表示される ActionBar コンテンツ以外は、デフォルトのアプリケーションファイルにコンテンツを直接追加することはありません。ActionBar にコンテンツを追加するには、navigatorContent、titleContent または actionContent の各プロパティを設定します。

- ***ProjectName***HomeView.mxml

  プロジェクトの基本ビューを表すファイルです。このファイルは、ビューパッケージに格納されます。***ProjectName***.mxml の ViewNavigatorApplication タグの firstView 属性で、このファイルがアプリケーションのデフォルトの開始ビューとして指定されます。

ビューの定義について詳しくは、38 ページの「モバイルアプリケーションでのビューの定義」を参照してください。

ActionScript のみのモバイルプロジェクトを作成することもできます。12 ページの「ActionScript モバイルプロジェクトの作成」を参照してください。

2 （オプション）メインアプリケーションファイルの ActionBar にコンテンツを追加します。

ActionBar には、アプリケーション、またはアプリケーションの現在のビューに適用されるコンテンツと機能が表示されます。この場所に、アプリケーションのすべてのビューに表示するコンテンツを追加します。58 ページの「モバイルアプリケーションでのナビゲーションコントロール、タイトルコントロールおよびアクションコントロールの定義」を参照してください。

3 アプリケーションの基本ビューのコンテンツをレイアウトします。

Flash Builder をデザインモードまたはソースモードで使用して、コンポーネントをビューに追加します。

モバイルアプリケーション開発用に Flex でサポートされているコンポーネントのみを使用してください。デザインモードとソースモードのいずれの場合も、Flash Builder の指示に従うことで、サポートされているコンポーネントを使用できます。25 ページの「ユーザーインターフェイスとレイアウト」を参照してください。

ビュー内で、そのビューにのみ表示するコンテンツを ActionBar に追加します。

4 （オプション）アプリケーションに含める他のビューを追加します。

Flash Builder パッケージエクスプローラーで、プロジェクトのビューパッケージのコンテキストメニューから、新規／MXML コンポーネントを選択します。新規 MXML コンポーネントウィザードの指示に従って、ビューを作成します。

ビューについて詳しくは、38 ページの「モバイルアプリケーションでのビューの定義」を参照してください。

5 （オプション）モバイル用に最適化された、List コンポーネントのアイテムレンダラーを追加します。

モバイルアプリケーションで使用するための ActionScript ベースのアイテムレンダラーである IconItemRenderer が用意されています。Using a mobile item renderer with a Spark list-based control を参照してください。

6 起動設定を設定して、アプリケーションを実行およびデバッグします。

デスクトップ上またはデバイス上でアプリケーションを実行およびデバッグできます。

Flash Builder からアプリケーションを実行またはデバッグするには、起動設定が必要です。モバイルアプリケーションを初めて実行またはデバッグするときに、Flash Builder によって起動設定を設定するように要求されます。

デバイス上でモバイルアプリケーションを実行またはデバッグするときに、そのデバイスにアプリケーションがインストールされます。

168 ページの「テストとデバッグ」を参照してください。

7 アプリケーションをインストーラーパッケージとして書き出します。

リリースビルドの書き出しを使用して、モバイルデバイスにインストールできるパッケージを作成します。書き出しのために選択したプラットフォーム用のパッケージが作成されます。176 ページの「Android APK のリリース用パッケージの書き出し」を参照してください。

Flex のアドビ認定エキスパートの Brent Arnold 氏が、役に立つ次のビデオチュートリアルを公開しています。

•Create a Flex mobile application with multiple views

- Create a Flex mobile application using a Spark-based List control

Flash Builder での iOS アプリケーションの作成

次に、Apple iOS プラットフォーム用のモバイルアプリケーションを作成する場合の一般的なワークフローを示します。

1 アプリケーションの作成を始める前に、20 ページの「Flash Builder を使用した Apple iOS の開発プロセス」の手順に必ず従ってください。

2 Flash Builder で、ファイル／新規／ Flex モバイルプロジェクトを選択します。

ターゲットプラットフォームとして Apple iOS を選択し、モバイルプロジェクト設定を設定します。

Flash Builder で他のプロジェクトビルドウィザードを使用する場合と同様に、新規プロジェクトウィザードのプロンプトに従ってください。詳しくは、10 ページの「アプリケーションの作成」を参照してください。

ActionScript のみのモバイルプロジェクトを作成することもできます。詳しくは、ActionScript モバイルプロジェクトの作成を参照してください。

3 起動設定を設定して、アプリケーションを実行およびデバッグします。デスクトップ上または接続したデバイス上でアプリケーションを実行およびデバッグできます。

詳しくは、172 ページの「Apple iOS デバイスでのアプリケーションのデバッグ」を参照してください。

4 Apple App Store にアプリケーションを書き出すか、iOS パッケージアプリケーション（IPA）をデバイスにデプロイします。

詳しくは、177 ページの「Apple iOS のリリース用パッケージの書き出し」および 174 ページの「Apple iOS デバイスへのアプリケーションのインストール」を参照してください。

関連項目

Beginning a Mobile Application（ビデオ）

Flash Builder での BlackBerry Tablet OS アプリケーションの作成

Flash Builder には、BlackBerry® Tablet OS 用に Flex と ActionScript の両方のアプリケーションを作成してパッケージ化できる Research In Motion（RIM）のプラグインが含まれています。

アプリケーションの作成

BlackBerry Tablet OS 用にアプリケーションを作成するための一般的なワークフローは次のとおりです。

1 モバイルアプリケーションの作成を始める前に、BlackBerry Tablet OS Application Development サイトから BlackBerry Tablet OS SDK for AIR をインストールします。

BlackBerry Tablet OS SDK for AIR では、AIR ベースの Flex と ActionScript のアプリケーションを作成できる API が提供されています。

BlackBerry Tablet OS SDK のインストールについて詳しくは、BlackBerry Tablet OS Getting Started Guide を参照してください。

2 Flex ベースの AIR アプリケーションを作成するには、Flash Builder で、ファイル／新規／ Flex モバイルプロジェクトを選択します。

Flash Builder で他の AIR プロジェクトを作成する場合と同様に、新規プロジェクトウィザードのプロンプトに従ってください。BlackBerry Tablet OS をターゲットプラットフォームとして選択したことを確認します。

詳しくは、Flex mobile projects を参照してください。

3 ActionScript ベースの AIR アプリケーションを作成するには、Flash Builder で、ファイル／新規／ ActionScript モバイルプロジェクトを選択します。

Flash Builder で他の AIR プロジェクトを作成する場合と同様に、新規プロジェクトウィザードのプロンプトに従ってください。BlackBerry Tablet OS をターゲットプラットフォームとして選択したことを確認します。

詳しくは、ActionScript モバイルプロジェクトの作成を参照してください。

アプリケーションの署名、パッケージ化およびデプロイ

アプリケーションの署名、パッケージ化およびデプロイについて詳しくは、RIM による BlackBerry Tablet OS SDK for Adobe AIR Development Guide を参照してください。

BlackBerry Tablet OS の開発に関する、Adobe および RIM からの追加のリソースが Adobe Developer Connection にあります。

ActionScript モバイルプロジェクトの作成

Flash Builder を使用して、ActionScript モバイルアプリケーションを作成します。作成するアプリケーションは Adobe AIR の API に基づいています。

1 ファイル／新規／ ActionScript モバイルプロジェクトを選択します。

2 プロジェクトの名前と場所を入力します。デフォルトの場所は、現在のワークスペースです。

3 モバイルアプリケーションの開発をサポートするデフォルトの Flex 4.6 SDK を使用します。

「次へ」をクリックします。

4 アプリケーションのターゲットプラットフォームを選択し、各プラットフォームのモバイルプロジェクトの設定を指定します。

モバイルプロジェクトの設定について詳しくは、15 ページの「モバイルプロジェクト環境設定の設定」を参照してください。

5 「終了」をクリックするか、または「次へ」をクリックしてさらに設定オプションとビルドパスを指定します。

プロジェクトの設定オプションとビルドパスについて詳しくは、Build paths, native extensions, and other project configuration options を参照してください。

ネイティブエクステンションの使用

ネイティブエクステンションを使用すると、ネイティブプラットフォームの機能をモバイルアプリケーションに組み込むことができます。

ネイティブエクステンションには、ActionScript クラスとネイティブコードが含まれます。ネイティブコードの実装により、純粋な ActionScript クラスではアクセスできない、デバイス固有の機能にアクセスできます。例えば、デバイスのバイブレーション機能にアクセスできます。

ネイティブコードの実装は、AIR ランタイムの外部で実行されるコードとして定義できます。拡張では、プラットフォーム固有の ActionScript クラスとネイティブコードの実装を定義します。ActionScript 拡張クラスでは、ActionScript クラスの ExtensionContext を使用して、ネイティブコードのデータにアクセスし、データ交換を行います。

拡張は、デバイスのハードウェアプラットフォームに固有のものです。プラットフォーム固有の複数の拡張を作成したり、複数のプラットフォームをターゲットにする単一の拡張を作成したりできます。例えば、Android と iOS の両方のプラットフォームをターゲットにするネイティブエクステンションを作成できます。ネイティブエクステンションは、次のモバイルデバイスでサポートされます。

- Android 2.2 以降のバージョンが動作する Android デバイス
- iOS 4.0 以降のバージョンが動作する iOS デバイス

クロスプラットフォームのネイティブエクステンションの作成について詳しくは、Adobe AIR 用ネイティブ拡張の開発を参照してください。

Adobe およびコミュニティによるネイティブエクステンションのサンプルのコレクションについては、Native extensions for Adobe AIR を参照してください。

ネイティブエクステンションのパッケージ化

ネイティブエクステンションをアプリケーション開発者に提供するには、次の手順に従って、必要なすべてのファイルを ActionScript ネイティブエクステンション（ANE）ファイルにパッケージ化します。

1 拡張の ActionScript ライブラリを SWC ファイルにビルドします。

2 拡張のネイティブライブラリをビルドします。拡張で複数のプラットフォームをサポートする必要がある場合は、ターゲットプラットフォームごとに 1 つのライブラリをビルドします。

3 拡張に使用する署名済み証明書を作成します。拡張に署名が行われていない場合、Flash Builder でプロジェクトに拡張を追加するときに警告が表示されます。

4 拡張記述ファイルを作成します。

5 拡張のすべての外部リソース（画像など）を含めます。

6 AIR 開発ツールを使用して、拡張パッケージを作成します。詳しくは、AIR のマニュアルを参照してください。

ActionScript エクステンションのパッケージ化について詳しくは、Adobe AIR 用ネイティブ拡張の開発を参照してください。

プロジェクトへのネイティブエクステンションの追加

SWC ファイルを追加するときと同じ方法で、ActionScript ネイティブエクステンション（ANE）ファイルをプロジェクトのビルドパスに追加します。

1 Flash Builder で Flex モバイルプロジェクトを作成するときに、「ビルドパス設定」ページの「ネイティブエクステンション」タブを選択します。

また、プロジェクトのプロパティダイアログボックスで「Flex ビルドパス」を選択して、拡張を追加することもできます。

2 ANE ファイル、または ANE ファイルを含むフォルダーを参照して、プロジェクトに追加します。デフォルトでは、ANE ファイルを追加すると、プロジェクトのアプリケーション記述ファイル（**project name**-app.xml）にエクステンション ID が追加されます。

次の場合は、追加した拡張にエラーシンボルが表示されます。

- 拡張の AIR ランタイムバージョンが、アプリケーションのランタイムバージョンよりも新しい
- アプリケーションのターゲットである選択したすべてのプラットフォームが拡張に含まれていない

注意：複数のプラットフォームをターゲットにする ActionScript ネイティブエクステンションを作成できます。AIR シミュレーターを使用して、このような ANE ファイルが含まれるアプリケーションを開発コンピューター上でテストする場合は、その ANE ファイルでコンピューターのプラットフォームがサポートされていることを確認してください。例えば、Windows で AIR シミュレーターを使用してアプリケーションをテストする場合、その ANE ファイルで Windows がサポートされていることを確認します。

アプリケーションパッケージへの ActionScript ネイティブエクステンションの追加

リリースビルドの書き出し機能を使用してモバイルアプリケーションを書き出すと、デフォルトでは、プロジェクトで使用されている拡張がアプリケーションパッケージ内に追加されます。

デフォルトの選択を変更するには、次の手順を行います。

1 リリースビルドの書き出しダイアログボックスで、パッケージ設定の下の「ネイティブエクステンション」タブを選択します。

2 プロジェクト内で参照されている ActionScript ネイティブエクステンション（ANE）ファイルが一覧表示され、ANE ファイルがプロジェクトで使用されているかどうかが表示されます。

プロジェクトで ANE ファイルが使用されている場合、そのファイルはアプリケーションパッケージでデフォルトで選択されます。

プロジェクトに ANE ファイルが含まれていても使用されていない場合は、コンパイラーで ANE ファイルが認識されません。その場合は、アプリケーションパッケージに追加されません。アプリケーションパッケージに ANE ファイルを追加するには、次の手順を行います。

a プロジェクトのプロパティダイアログボックスで、「Flex ビルドのパッケージ化」を選択し、必要なプラットフォームを選択します。

b アプリケーションパッケージに含める拡張を選択します。

iOS5 ネイティブエクステンションのサポート

iOS5 SDK の機能を使用するネイティブエクステンションをパッケージ化するには、AIR Developer Tool（ADT）に、iOS5 SDK の場所を設定する必要があります。

Mac OS では、Flash Builder でパッケージ設定ダイアログボックスを使用して iOS5 SDK の場所を選択できます。iOS SDK の場所を選択すると、選択した場所が -platformsdk ADT コマンド経由で渡されるようになります。

注意：この機能は、Windows では現在サポートされていません。

詳しくは、Adobe AIR 用ネイティブ拡張の開発を参照してください。

モバイルプロジェクト環境設定の設定

デバイス設定の設定

Flash Builder では、デバイス設定を使用して、デバイスのスクリーンサイズプレビューをデザインビューに表示したり、AIR Debug Launcher（ADL）を使用してデスクトップでアプリケーションを起動したりできます。169 ページの「デスクトッププレビュー用のデバイス情報の設定」を参照してください。

デバイス設定を設定するには、「環境設定」を開いて、Flash Builder ／デバイス設定を選択します。

Flash Builder には、デフォルトのデバイス設定がいくつか用意されています。また、それ以外のデバイス設定を追加、編集または削除できます。Flash Builder に用意されているデフォルト設定は変更できません。

「デフォルトに戻す」ボタンをクリックするとデフォルトのデバイス設定が復元されますが、追加した設定は削除されません。また、デフォルトのデバイス設定と同じ名前のデバイス設定を追加した場合は、追加した設定がデフォルト設定でオーバーライドされます。

デバイス設定には、次のプロパティがあります。

プロパティ	説明
デバイス名	デバイスの一意の名前です。
プラットフォーム	デバイスプラットフォームです。サポートされているプラットフォームのリストから、プラットフォームを選択します。
フルスクリーンサイズ	デバイスのスクリーンの幅と高さです。
使用可能なスクリーンサイズ	デバイスでのアプリケーションの標準サイズです。このサイズは、フルスクリーンモード以外で起動されたときに予期されるアプリケーションのサイズであり、ステータスバーなどのシステムクロームが考慮されたサイズです。
1 インチあたりのピクセル数 (ppi)	デバイスのスクリーンの 1 インチあたりのピクセル数です。

ターゲットプラットフォームの選択

Flash Builder では、アプリケーションタイプに基づいてターゲットプラットフォームがサポートされます。

プラットフォームを選択するには、「環境設定」を開いて、Flash Builder ／ターゲットプラットフォームを選択します。

すべてのサードパーティープラグインについては、関連するマニュアルを参照してください。

適切なテンプレートの選択

モバイルアプリケーションを作成するときは、次のアプリケーションテンプレートを選択できます。

空白 ベースアプリケーションエレメントとして Spark の Application タグを使用します。

標準のビューナビゲーションを使用せずに、カスタムアプリケーションを作成する場合は、このオプションを使用します。

ビューベースアプリケーション ベースアプリケーションエレメントとして Spark の ViewNavigatorApplication タグを使用し、単一のビューを持つアプリケーションを作成します。

基本ビューの名前を指定できます。

タブ付きアプリケーション ベースアプリケーションエレメントとして Spark の TabbedViewNavigatorApplication タグを使用し、タブ付きアプリケーションを作成します。

タブを追加するには、タブの名前を入力し、「追加」をクリックします。タブの順序は、「上へ」および「下へ」をクリックして変更できます。アプリケーションからタブを削除するには、タブを選択して「削除」をクリックします。

ビューの名前はタブ名の末尾に「View」が追加された名前になります。例えば、タブに FirstTab という名前を付けた場合は、FirstTabView という名前のビューが生成されます。

作成するタブごとに、views パッケージには新規の MXML ファイルが生成されます。

注意：Flex モバイルプロジェクトウィザードでは、パッケージ名を設定できません。

MXML ファイルは次の規則に従って生成されます。

- タブ名が有効な ActionScript クラス名である場合は、末尾に「View」が追加されたタブ名を使用して MXML ファイルが生成されます。
- タブ名が有効なクラス名でない場合は、無効な文字を削除して有効な開始文字を挿入することにより、タブ名が変更されます。変更された名前が使用不可の場合、MXML ファイル名は「ViewN」に変更されます。N はビューの位置を示し、1 から始まります。


Flex のアドビ認定エキスパートの Brent Arnold 氏が、タブ付きアプリケーションテンプレートの使用に関するビデオチュートリアルを公開しています。

モバイルアプリケーションの権限の選択

モバイルアプリケーションを作成するときに、ターゲットプラットフォームに対するデフォルトの権限を指定または変更できます。権限はコンパイル時に指定され、実行時には変更できません。

ターゲットプラットフォームをまず選択し、必要に応じて、プラットフォームごとに権限を設定します。権限は、後でアプリケーション記述 XML ファイルで編集できます。

サードパーティーのプラグインでは、Flex プロジェクトと ActionScript プロジェクトの両方について追加のプラットフォームがサポートされます。プラットフォーム固有の権限については、それぞれのデバイスの関連マニュアルを参照してください。

Google Android プラットフォームの権限

Google Android プラットフォームの場合は、次の権限を設定できます。

INTERNET ネットワークリクエストとリモートデバッグを許可します

INTERNET 権限はデフォルトで選択されます。この権限を選択解除すると、デバイス上でアプリケーションをデバッグできません。

WRITE_EXTERNAL_STORAGE 外部デバイスへの書き込みを許可します

この権限を選択すると、アプリケーションはデバイスの外部メモリカードに書き込むことができます。

READ_PHONE_STATE 電話の着信時に音声をミュートします

この権限を選択すると、アプリケーションは電話での通話中に音声をミュートできます。例えば、アプリケーションがバックグラウンドでオーディオを再生する場合に、この権限を選択できます。

ACCESS_FINE_LOCATION GPS 位置情報へのアクセスを許可します

この権限を選択すると、アプリケーションは Geolocation クラスを使用して GPS データにアクセスできます。

DISABLE_KEYGUARD および WAKE_LOCK デバイスのスリープモードを許可しません

この権限を選択すると、デバイスは SystemIdleMode クラスの設定を使用してスリープ状態になることができなくなります。

CAMERA カメラへのアクセスを許可します

この権限を選択すると、アプリケーションはカメラにアクセスできます。

RECORD_AUDIO マイクロホンへのアクセスを許可します

この権限を選択すると、アプリケーションはマイクロホンにアクセスできます。

ACCESS_NETWORK_STATE および ACCESS_WIFI_STATE デバイスと関連付けられているネットワークインターフェイスに関する情報へのアクセスを許可します

この権限を選択すると、アプリケーションは NetworkInfo クラスを使用してネットワーク情報にアクセスできます。

モバイルアプリケーションのプロパティ設定について詳しくは、Adobe AIR マニュアルを参照してください。

Apple iOS プラットフォームの権限

Apple iOS プラットフォームでは、定義済みの権限を使用せずに、実行時に権限が検証されます。つまり、アプリケーションで Apple iOS プラットフォームの特定の機能にアクセスするときに、ユーザー権限が必要な場合は、権限を確認するポップアップが表示されます。

プラットフォーム設定の選択

プラットフォーム設定により、ターゲットデバイスファミリを選択できます。選択したプラットフォームに応じて、ターゲットデバイスまたはターゲットデバイスファミリを選択できます。特定のデバイスを選択するか、プラットフォームがサポートしているすべてのデバイスを選択できます。

サードパーティーのプラグインでは、Flex プロジェクトと ActionScript プロジェクトの両方について追加のプラットフォームがサポートされます。プラットフォーム固有の設定については、それぞれのデバイスの関連マニュアルを参照してください。

Google Android のプラットフォーム設定

Google Android プラットフォームの場合は、プラットフォーム固有の設定はありません。

Apple iOS のプラットフォーム設定

Flex モバイルプロジェクトや ActionScript モバイルプロジェクトに対しては、Apple iOS プラットフォームの次のターゲットデバイスを指定できます。

iPhone/iPod Touch このターゲットファミリを使用しているアプリケーションは、Apple App Store で iPhone および iPod Touch デバイスのみと互換性があるアプリケーションとして表示されます。

iPad このターゲットファミリを使用しているアプリケーションは、Apple App Store で iPad デバイスのみと互換性があるアプリケーションとして表示されます。

すべて このターゲットファミリを使用しているアプリケーションは、Apple App Store で iPhone や iPod Touch および iPad デバイスの両方と互換性があるアプリケーションとして表示されます。このオプションがデフォルトです。

アプリケーション設定の選択

自動的に方向を変更 デバイスを回転すると、アプリケーションが回転します。この設定が有効でないと、アプリケーションが常に固定した方向に表示されます。

フルスクリーン アプリケーションをデバイスのフルスクリーモードで表示します。この設定が有効な場合、デバイスのステータスバーがアプリケーションの上に表示されません。アプリケーションがスクリーン全体に表示されます。

画面密度が異なる複数の種類のデバイスでアプリケーションをターゲットにする場合は、「画面密度に合わせてアプリケーションを自動サイズ変更」を選択します。このオプションを選択すると、デバイスに対してアプリケーションのサイズが自動的に変更され、必要に応じて、密度の変更が処理されます。18 ページの「アプリケーションのサイズ変更」を参照してください。

アプリケーションのサイズ変更

スクリーンサイズや密度が異なる複数のデバイスと互換性がある単一のモバイルアプリケーションを構築するには、モバイルアプリケーションのサイズ変更を使用します。

モバイルデバイスのスクリーンは、密度や DPI（dots per inch）が異なります。DPI 値には、ターゲットデバイスの画面密度に応じて、160、240 または 320 を指定できます。自動サイズ変更を有効にすると、各デバイスの画面密度に対してアプリケーションを表示する方法が最適化されます。

例えば、ターゲット DPI 値を 160 に指定して、自動サイズ変更を有効にしたとします。DPI 値が 320 のデバイスでアプリケーションを実行すると、2 の比率でアプリケーションが自動サイズ変更されます。つまり、すべてが 200 %拡大されます。

ターゲットの DPI 値を指定するには、メインアプリケーションファイルで、<s:ViewNavigatorApplication> タグまたは <s:TabbedViewNavigatorApplication> タグの applicationDPI プロパティに値を設定します。

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

アプリケーションの自動サイズ変更を無効にした場合は、必要に応じて、レイアウトの密度変更を手動で処理する必要があります。ただし、スキンは各デバイスの密度に適合して表示されます。

密度に依存しないモバイルアプリケーションの作成について詳しくは、119 ページの「モバイルアプリケーションでの複数のスクリーンサイズと DPI 値のサポート」を参照してください。

Google Android デバイスの接続

Google Android デバイスを開発用コンピューターに接続して、Android デバイス上でアプリケーションをプレビューしたり、デバッグしたりできます。

サポートされている Android デバイス

Flex のモバイルプロジェクトおよび ActionScript のモバイルプロジェクトには、AIR 2.6 以降のバージョンが必要です。モバイルプロジェクトの実行やデバッグは、AIR 2.6 以降のバージョンをサポートする物理デバイス上でのみ行えます。

AIR 2.6 は、Android 2.2 以降のバージョンが動作するサポート対象の Android デバイスにインストールできます。サポート対象のデバイスのリストについては、http://www.adobe.com/flashplatform/certified_devices/ を参照してください。また、Android デバイスで Adobe AIR を実行するための最低限の必要システム構成については、モバイルの必要システム構成を参照してください。

Android デバイスの設定

Flex モバイルアプリケーションを Android デバイスから実行およびデバッグするには、次の手順で USB デバッグを有効にします。

1 デバイスで、次の手順に従って、USB デバッグが有効になっていることを確認します。

   a ホームボタンをタップしてホームスクリーンを表示します。

   b 「Settings」に移動して、Applications ／ Development を選択します。

   c 「USB debugging」を有効にします。

2 デバイスを USB ケーブルでコンピューターに接続します。

3 スクリーン上部にある通知領域をプルダウンします。「USB Connected」または「USB Connection」のいずれかが表示されます。

   a 「USB Connected」または「USB Connection」をタップします。

   b 「Charge Only」モードを含む一連のオプションが表示された場合は、「Charge Only」を選択し、「OK」をタップします。

   c 大容量記憶装置モードをオフにするボタンが表示された場合は、ボタンをクリックして大容量記憶装置をオフにします。

4 （Windows のみ）デバイスに適切な USB ドライバーをインストールします。19 ページの「Android デバイス用の USB デバイスドライバーのインストール（Windows）」を参照してください。

5 スクリーン上部にある通知領域をプルダウンします。

   「USB debugging」がエントリとして表示されない場合は、前述の手順 3 の説明に従って、USB モードを選択します。USB モードが「PC Mode」に設定されていないことを確認します。

注意： デバッグ時には追加の設定が必要です。169 ページの「デバイスでのモバイルアプリケーションのテストとデバッグ」を参照してください。

Android デバイス用の USB デバイスドライバーのインストール（Windows）

デバイスドライバーと設定

Windows プラットフォームでは、Android デバイスを開発用コンピューターに接続するために USB ドライバーをインストールする必要があります。Flash Builder には、いくつかの Android デバイス用のデバイスドライバーと設定が用意されています。

これらのデバイスドライバーの設定は、android_winusb.inf に記述されています。Windows のデバイスマネージャーは、デバイスドライバーのインストール時にこのファイルにアクセスします。android_winusb.inf は次の場所にインストールされます。

```
<Adobe Flash Builder 4.6 Home>\utilities\drivers\android\android_winusb.inf
```

サポートされているすべてのデバイスの完全なリストは、Certified devices を参照してください。リストされていない Android デバイスの場合は、android_winusb.inf を USB ドライバーで更新できます。20 ページの「Android 用 USB デバイスドライバー設定の追加」を参照してください。

USB デバイスドライバーのインストール

1 Android デバイスをコンピューターの USB ポートに接続します。

2 次の場所に移動します。

   <Flash Builder>¥utilities¥drivers¥android¥

Windows の新しいハードウェアが見つかりましたウィザードまたは Windows のデバイスマネージャーを使用して、USB ドライバーをインストールします。

重要：Windows が引き続きデバイスを認識できない場合は、デバイス製造元の適切な USB ドライバーをインストールする必要があります。OEM USB drivers には、デバイス製造元の Web サイトへのリンクがあるので参照してください。ご使用のデバイスの適切な USB ドライバーをダウンロードできます。

Android 用 USB デバイスドライバー設定の追加

19 ページの「Android デバイス用の USB デバイスドライバーのインストール（Windows）」に記載されていないサポート対象 Android デバイスがある場合は、android_winusb.inf ファイルを更新してそのデバイスを記述します。

1. デバイスをコンピューターの USB ポートに接続します。Windows から、ドライバーを検出できないことが通知されます。
2. Windows のデバイスマネージャーを使用して、デバイスプロパティの「詳細」タブを開きます。
3. 「ハードウェア ID」プロパティを選択して、ハードウェア ID を表示します。
4. テキストエディターで android_winusb.inf を開きます。次の場所で android_winusb.inf を見つけます。

   ```
   <Adobe Flash Builder 4.6 Home>\utilities\drivers\android\android_winusb.inf
   ```

5. 現在のアーキテクチャに適用されるファイル内の記述（[Google.NTx86] または [Google.NTamd64]）を確認します。この記述には、次に示すようなわかりやすいコメントおよびハードウェア ID が指定された行があります。

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

6. コメントおよびハードウェアの記述をコピー＆ペーストします。追加するデバイスドライバーについて、次のように記述を編集します。

   a コメントには、デバイスの名前を指定します。

   b ハードウェア ID は、前述の手順 3 で確認したハードウェア ID で置き換えます。

   次に例を示します。

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

7. Windows のデバイスマネージャーを使用して、前述の 19 ページの「Android デバイス用の USB デバイスドライバーのインストール（Windows）」の説明に従ってデバイスをインストールします。

   インストール時に、Windows によって、不明な発行者からのドライバーであるという警告が表示されます。それでも、このドライバーを使用して Flash Builder でデバイスにアクセスできます。

Flash Builder を使用した Apple iOS の開発プロセス

Flash Builder を使用して iOS アプリケーションを開発する前に、iOS の開発プロセスと、必要な証明書を Apple から取得する方法を理解しておくことが重要です。

iOS の開発およびデプロイメントプロセスの概要

次の表に、iOS 開発プロセスの手順、必要な証明書の取得方法、および各手順の前提条件の簡単な一覧を示します。

これらの各手順について詳しくは、21 ページの「iOS アプリケーションの作成、デバッグまたはデプロイのための準備」を参照してください。

手順の番号	手順	場所	前提条件
1.	Apple Developer Program に参加します。	Apple Developer サイト	なし
2.	iOS デバイスの Unique Device Identifier（UDID）を登録します。	iOS Provisioning Portal	Apple Developer ID（手順 1）
3.	証明書署名要求（CSR）ファイル（*.certSigningRequest）を生成します。	• Mac OS の場合、キーチェーンアクセスプログラムを使用します。 • Windows の場合、OpenSSL を使用します。	なし
4.	iOS 開発用証明書または配布用証明書（*.cer）を生成します。	iOS Provisioning Portal	• Apple Developer ID（手順 1） • CSR ファイル（手順 3）
5.	iOS 開発用証明書または配布用証明書を P12 形式に変換します。	• Mac OS の場合、キーチェーンアクセスプログラムを使用します。 • Windows の場合、OpenSSL を使用します。	• Apple Developer ID（手順 1） • iOS 開発用証明書または配布用証明書（手順 4）
6.	アプリケーション ID を生成します。	iOS Provisioning Portal	Apple Developer ID（手順 1）
7.	プロビジョニングプロファイル（*.mobileprovision）を生成します。	iOS Provisioning Portal	• Apple Developer ID（手順 1） • iOS デバイスの UDID（手順 2） • アプリケーション ID（手順 6）
8.	アプリケーションを作成します。	Flash Builder	• Apple Developer ID（手順 1） • P12 の開発用証明書または配布用証明書（手順 5） • アプリケーション ID（手順 6）
9.	アプリケーションをデプロイします。	iTunes	• プロビジョニングプロファイル（手順 7） • アプリケーションパッケージ（手順 8）

iOS アプリケーションの作成、デバッグまたはデプロイのための準備

Flash Builder を使用して iOS アプリケーションを作成し、アプリケーションを iOS デバイスにデプロイしたり、Apple App Store に送信したりする前に、次の手順を行います。

1 Apple iOS Developer Program に参加します。

既存の Apple ID を使用してログインするか、または Apple ID を作成できます。Apple Developer Registration に従って、必要な手順を行います。

2 デバイスの Unique Device Identifier（UDID）を登録します。

この手順は、アプリケーションを Apple App Store ではなく、iOS デバイスにデプロイする場合にのみ該当します。アプリケーションを複数の iOS デバイスにデプロイする場合は、各デバイスの UDID を登録します。

iOS デバイスの UDID の取得

a 開発用コンピューターに iOS デバイスを接続して、iTunes を起動します。接続した iOS デバイスが、iTunes の「デバイス」セクションの下に表示されます。

b デバイス名をクリックして、iOS デバイスの概要を表示します。

c 「概要」タブで、「シリアル番号」をクリックして、iOS デバイスの 40 文字の UDID を表示します。


キーボードショートカットの Ctrl+C（Windows）または Command+C（Mac）を使用して、iTunes から UDID をコピーできます。

デバイスの UDID の登録

Apple ID を使用して iOS Provisioning Portal にログインし、デバイスの UDID を登録します。

3 証明書署名要求（CSR）ファイル（*.certSigningRequest）を生成します。

iOS 開発用証明書または配布用証明書を取得するための CSR を生成します。CSR は、キーチェーンアクセス（Mac の場合）または OpenSSL（Windows の場合）を使用して生成できます。生成する CSR でユーザー名と電子メールアドレスのみを提供する場合は、アプリケーションやデバイスに関する情報は入力しません。

CSR を生成すると、公開鍵と秘密鍵、および *.certSigningRequest ファイルが作成されます。公開鍵は CSR に組み込まれ、秘密鍵は要求に署名する場合に使用します。

CSR の生成について詳しくは、証明書署名要求の生成を参照してください。

4 必要に応じて、iOS 開発用証明書または iOS 配布用証明書（*.cer）を生成します。

注意：アプリケーションをデバイスにデプロイするには、開発用証明書が必要です。アプリケーションを Apple App Store にデプロイするには、配布用証明書が必要です。

iOS 開発用証明書の生成

a Apple ID を使用して iOS Provisioning Portal にログインし、「Development」タブを選択します。

b 「Request Certificate」をクリックして、生成してコンピューターに保存した CSR ファイル（手順 3）を参照します。

c CSR ファイルを選択し、「Submit」をクリックします。

d 「Certificates」ページで「Download」をクリックします。

e ダウンロードしたファイル（*.developer_identity.cer）を保存します。

iOS 配布用証明書の生成

f Apple ID を使用して iOS Provisioning Portal にログインし、「Distribution」タブを選択します。

g 「Request Certificate」をクリックして、生成してコンピューターに保存した CSR ファイル（手順 3）を参照します。

h CSR ファイルを選択し、「Submit」をクリックします。

i 「Certificates」ページで「Download」をクリックします。

j ダウンロードしたファイル（*.distribution_identity.cer）を保存します。

5 iOS 開発用証明書または iOS 配布用証明書を P12 ファイル形式（*.p12）に変換します。

iOS 開発用証明書または iOS 配布用証明書を P12 形式に変換すると、Flash Builder で iOS アプリケーションに電子署名できるようになります。P12 形式に変換すると、iOS 開発用証明書または配布用証明書と、関連付けられた秘密鍵が 1 つのファイルにまとめられます。

注意：AIR Debug Launcher（ADL）を使用して、デスクトップ上でアプリケーションのテストを行う場合は、iOS 開発用証明書または配布用証明書を P12 形式に変換する必要はありません。

キーチェーンアクセス（Mac の場合）または OpenSSL（Windows の場合）を使用して、Personal Information Exchange（*.p12）ファイルを生成します。詳しくは、P12 ファイルへの開発用証明書の変換を参照してください。

6 次の手順に従って、アプリケーション ID を生成します。

a Apple ID を使用して iOS Provisioning Portal にログインします。

b 「App IDs」ページに移動し、「New App ID」をクリックします。

c 「Manage」タブで、アプリケーションの説明を入力し、新しいバンドルシード ID を生成して、バンドル識別子を入力します。

アプリケーションは、それぞれ一意のアプリケーション ID を持ちます。この ID は、アプリケーション記述 XML ファイルで指定します。アプリケーション ID は、Apple から提供される 10 文字の「バンドルシード ID」と、それに続いてユーザーが指定する「バンドル識別子」で構成されます。指定するバンドル識別子は、アプリケーション記述ファイルのアプリケーション ID と一致する必要があります。例えば、アプリケーション ID が com.myDomain.* の場合、アプリケーション記述ファイルの ID は com.myDomain で始まる必要があります。

重要：ワイルドカードのバンドル識別子は、iOS アプリケーションの開発とテストには便利ですが、アプリケーションを Apple App Store にデプロイするときには使用できません。

7 開発プロビジョニングプロファイルファイル、または配布プロビジョニングプロファイルファイル（*.mobileprovision）を生成します。

注意：アプリケーションをデバイスにデプロイするには、開発プロビジョニングプロファイルが必要です。アプリケーションを Apple App Store にデプロイするには、配布プロビジョニングプロファイルが必要です。配布プロビジョニングプロファイルは、アプリケーションに署名するときに使用します。

開発プロビジョニングプロファイルの生成

a Apple ID を使用して iOS Provisioning Portal にログインします。

b Certificate ／ Provisioning の順に移動し、「New Profile」をクリックします。

c プロファイル名を入力し、iOS 開発用証明書、アプリケーション ID、およびアプリケーションのインストール先の UDID を選択します。

d 「Submit」をクリックします。

e 生成された開発プロビジョニングプロファイルファイル（*.mobileprovision）をダウンロードして、コンピューターに保存します。

配布プロビジョニングプロファイルの生成

f Apple ID を使用して iOS Provisioning Portal にログインします。

g Certificate ／ Provisioning の順に移動し、「New Profile」をクリックします。

h プロファイル名を入力し、iOS 配布用証明書とアプリケーション ID を選択します。デプロイする前にアプリケーションをテストする場合は、テストするデバイスの UDID を指定します。

i 「Submit」をクリックします。

j 生成されたプロビジョニングプロファイルファイル（*.mobileprovision）をダウンロードして、コンピューターに保存します。

関連項目

11 ページの「Flash Builder での iOS アプリケーションの作成」

iOS アプリケーションをテスト、デバッグまたはインストールする場合に選択するファイル

iOS デバイス上でテストするためにアプリケーションを実行、デバッグまたはインストールするには、実行／デバッグの構成ダイアログボックスで、次のファイルを選択します。

- P12 形式の iOS 開発用証明書（手順 5）
- アプリケーション ID（手順 6）が含まれるアプリケーション記述 XML ファイル
- 開発プロビジョニングプロファイル（手順 7）

詳しくは、172 ページの「Apple iOS デバイスでのアプリケーションのデバッグ」および 174 ページの「Apple iOS デバイスへのアプリケーションのインストール」を参照してください。

アプリケーションを Apple App Store にデプロイする場合に選択するファイル

アプリケーションを Apple App Store にデプロイするには、リリースビルドの書き出しダイアログボックスで、「パッケージタイプ」に「Apple App Store への最終リリースパッケージ」を選択し、次のファイルを選択します。

- P12 形式の iOS 配布用証明書（手順 5）
- アプリケーション ID（手順 6）が含まれるアプリケーション記述 XML ファイル

  注意： アプリケーションを Apple App Store に送信している間は、ワイルドカードのアプリケーション ID は使用できません。
- 配布プロビジョニングプロファイル（手順 7）

詳しくは、177 ページの「Apple iOS のリリース用パッケージの書き出し」を参照してください。

第 3 章：ユーザーインターフェイスとレイアウト

モバイルアプリケーションのレイアウト

ビューとセクションを使用したモバイルアプリケーションのレイアウト

モバイルアプリケーションは、1 つ以上のスクリーン（ビュー）で構成されます。例えば、モバイルアプリケーションに次の 3 つのビューがあるとします。

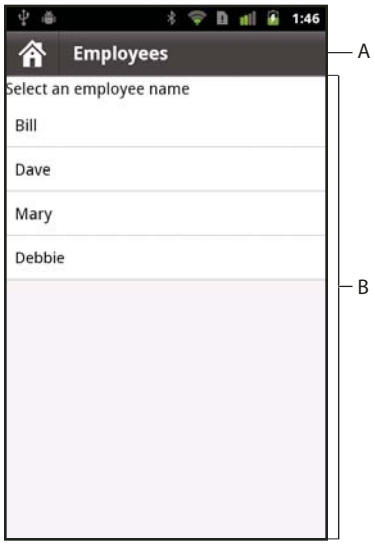
1 連絡先情報を追加できるホームビュー

2 既存の連絡先のリストを含む連絡先ビュー

3 連絡先のリストを検索する検索ビュー

簡単なモバイルアプリケーション

次の図は、Flex で構築された簡単なモバイルアプリケーションのメインスクリーンを示しています。



A. ActionBar コントロール **B.** コンテンツ領域

この図は、モバイルアプリケーションのメイン領域を示しています。

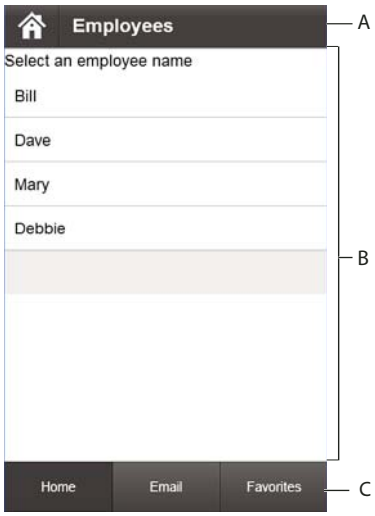
ActionBar コントロール ActionBar コントロールには、アプリケーションの現在の状態に関するコンテキスト情報を表示できます。この情報には、タイトル領域、アプリケーションをナビゲートするためのコントロール領域、およびアクションを実行するためのコントロール領域が含まれます。ActionBar コントロールには、アプリケーション全体に適用するグローバルコンテンツや、個々のビューに固有のアイテムを追加できます。

コンテンツ領域 コンテンツ領域には、アプリケーションを構成する個別のスクリーン（ビュー）が表示されます。ユーザーは、アプリケーションに組み込まれたコンポーネントおよびモバイルデバイスの入力コントロールを使用して、アプリケーションのビューをナビゲートします。

セクションがあるモバイルアプリケーション

より複雑なアプリケーションでは、複数の領域（セクション）があるアプリケーションを定義できます。例えば、連絡先、電子メール、お気に入りなどのセクションをアプリケーションに設定できます。アプリケーションの各セクションには、1 つ以上のビューが含まれます。各ビューはセクション間で共有できるため、同じビューを複数回定義する必要はありません。

次の図は、アプリケーションウィンドウの下部にタブバーが組み込まれたモバイルアプリケーションを示しています。



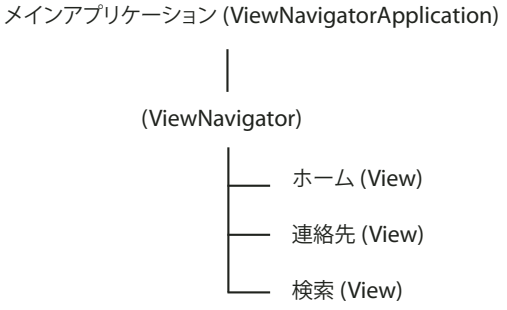
A. ActionBar コントロール **B.** コンテンツ領域 **C.** タブバー

Flex では、ButtonBarBase コントロールを使用してタブバーを実装します。タブバーの各ボタンは、それぞれのセクションに対応します。現在のセクションを変更するには、タブバーのボタンを選択します。

アプリケーションの各セクションでは、独自に ActionBar が定義されます。したがって、タブバーはアプリケーション全体で共通で、ActionBar は各セクションに固有です。

簡単なモバイルアプリケーションのレイアウト

次の図は、単純なモバイルアプリケーションのアーキテクチャを示しています。



この図は、4 つのファイルで構成されるアプリケーションを示しています。モバイルアプリケーションには、メインアプリケーションファイルと、各ビューにつき 1 つのファイルが含まれています。ViewNavigator 用の別のファイルはありません。ViewNavigatorApplication コンテナがファイルを作成します。

注意：この図はアプリケーションのアーキテクチャを示していますが、実行時のアプリケーションを表しているわけではありません。実行時にアクティブでメモリに存在するビューは 1 つのみです。詳しくは、29 ページの「モバイルアプリケーションのビューのナビゲート」を参照してください。

モバイルアプリケーションで使用するクラス

モバイルアプリケーションを定義するには、次のクラスを使用します。

クラス	説明
ViewNavigatorApplication	メインアプリケーションファイルを定義します。ViewNavigatorApplication コンテナには子を設定できません。
ViewNavigator	アプリケーションのビュー間のナビゲーションを制御します。ActionBar コントロールも作成します。 ViewNavigatorApplication コンテナは、アプリケーション全体に対応する単一の ViewNavigator コンテナを自動的に作成します。複数のビュー間を切り替えるには、ViewNavigator コンテナのメソッドを使用します。
View	アプリケーションのビューを定義します。各ビューは、別個の MXML ファイルまたは ActionScript ファイルに定義します。View コンテナのインスタンスは、アプリケーションの各ビューを表します。各ビューは、個別の MXML または ActionScript ファイルで定義します。

次の例に示すように、ViewNavigatorApplication コンテナを使用して、メインアプリケーションファイルを定義します。

```
<?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>
```

ViewNavigatorApplication コンテナは、ActionBar を定義する単一の ViewNavigator オブジェクトを自動的に作成します。ViewNavigator は、アプリケーションのビューをナビゲートするために使用します。

モバイルアプリケーションへの View コンテナの追加

すべてのモバイルアプリケーションには、少なくとも 1 つのビューを設定します。メインアプリケーションファイルによって ViewNavigator が作成されますが、アプリケーションで使用するビューは定義されません。

アプリケーションの各ビューは、ActionScript または MXML ファイルで定義した View コンテナに対応します。各 View には、そのビューに関連するデータを指定する data プロパティが含まれています。各 View では、ユーザーがアプリケーションをナビゲートしたときに、data プロパティを使用して相互に情報を渡すことができます。

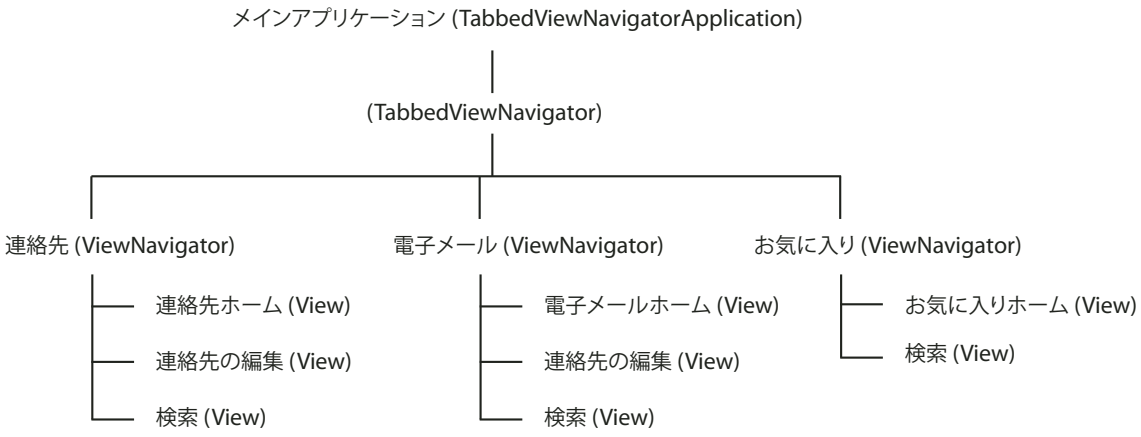
アプリケーションの最初のビューが定義されているファイルを指定するには、ViewNavigatorApplication.firstView プロパティを使用します。前のアプリケーションでは、firstView プロパティが views.HomeView を指定しています。次の例は、このビューを定義している HomeView.mxml ファイルを示しています。

```
<?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>
```


ブロガーの David Hassoun 氏が、ViewNavigator basics というブログを公開しています。

複数のセクションがあるモバイルアプリケーションのレイアウト

モバイルアプリケーションでは、アプリケーションの様々なセクションから関連するビューを収集できます。例えば、次の図は、3 つのセクションがあるモバイルアプリケーションの編成を示しています。



任意のセクションで任意の View を使用できます。つまり、ビューは特定のセクションに属するわけではありません。セクションでは、一連のビューを配置してナビゲートする方法を定義するだけです。この図では、アプリケーションのすべてのセクションに検索ビューが含まれています。

実行時にアクティブでメモリに存在するビューは 1 つのみです。詳しくは、29 ページの「モバイルアプリケーションのビューのナビゲート」を参照してください。

複数のセクションがあるモバイルアプリケーションで使用するクラス

次の表に、複数のセクションがあるモバイルアプリケーションの作成に使用するクラスを示します。

クラス	説明
TabbedViewNavigatorApplication	メインアプリケーションファイルを定義します。TabbedViewNavigatorApplication コンテナで設定可能な子は ViewNavigator のみです。ViewNavigator は、アプリケーションのセクションごとに 1 つ定義します。
TabbedViewNavigator	アプリケーションを構成するセクション間のナビゲーションを制御します。 TabbedViewNavigatorApplication コンテナは、アプリケーション全体に対応する単一の TabbedViewNavigator コンテナを自動的に作成します。TabbedViewNavigator コンテナは、セクション間のナビゲートに使用するタブバーを作成します。
ViewNavigator	ViewNavigator コンテナは、セクションごとに 1 つ定義します。ViewNavigator は、セクションを構成するビュー間のナビゲーションを制御します。セクションの ActionBar コントロールも作成します。
View	アプリケーションのビューを定義します。View コンテナのインスタンスは、アプリケーションの各ビューを表します。各ビューは、個別の MXML または ActionScript ファイルで定義します。

セクション化されたモバイルアプリケーションには、メインアプリケーションファイルと、各ビューを定義するファイルが含まれています。次の例に示すように、TabbedViewNavigatorApplication コンテナを使用して、メインアプリケーションファイルを定義します。

```
<?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>
```

複数のセクションがあるアプリケーションでの ViewNavigator の使用

TabbedViewNavigatorApplication コンテナで設定可能な子コンポーネントは ViewNavigator のみです。アプリケーションの各セクションは、別の ViewNavigator コンテナに対応します。

ViewNavigator コンテナは、各セクションのビューをナビゲートしたり、セクションの ActionBar コントロールを定義したりするために使用します。セクションの最初のビューが定義されているファイルを指定するには、ViewNavigator.firstView プロパティを使用します。

複数のセクションがあるアプリケーションでの TabbedViewNavigator の使用

TabbedViewNavigatorApplication コンテナは、TabbedViewNavigator タイプの単一コンテナを自動的に作成します。次に、TabbedViewNavigator コンテナが、アプリケーションの下部にタブバーを作成します。セクション間をナビゲートするためのロジックをアプリケーションに追加する必要はありません。

モバイルアプリケーションのビューのナビゲート

モバイルアプリケーションのナビゲーションは、View オブジェクトのスタックによって制御します。現在表示されているビューは、スタックの最上位にある View オブジェクトによって定義されています。

このスタックは、ViewNavigator コンテナによって維持されます。ビューを変更するには、新規の View オブジェクトをスタックにプッシュしたり、スタックから現在の View オブジェクトをポップしたりします。現在表示されている View オブジェクトをスタックからポップすると、その View オブジェクトが破棄され、スタックにある前のビューに戻ります。

セクションがあるアプリケーションでは、タブバーを使用してセクションをナビゲートします。各セクションは異なる ViewNavigator で定義されているので、セクションを変更するには、現在の ViewNavigator とスタックを変更します。新しい ViewNavigator のスタックの最上位にある View オブジェクトが現在のビューになります。

メモリを節約するために、ViewNavigator のデフォルトでは、メモリ内に存在するビューは一度に 1 つのみになっています。ただし、前のビューのデータはスタックで維持されます。したがって、ユーザーが前のビューに戻ると、適切なデータを使用してビューが再インスタンス化されます。

注意：View コンテナでは destructionPolicy プロパティが定義されます。デフォルトの auto に設定した場合、ビューがアクティブでなくなると、ViewNavigator によって破棄されます。none に設定した場合、ビューはメモリ内にキャッシュされます。

ブロガーの Mark Lochrie 氏が、ViewNavigator というブログを公開しています。

ViewNavigator のナビゲーションメソッド

ナビゲーションを制御するには、ViewNavigator クラスの次のメソッドを使用します。

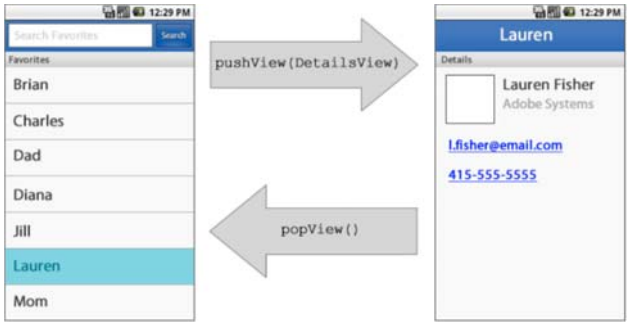
pushView() View オブジェクトをスタックにプッシュします。pushView() メソッドに引数として渡された View が、現在のビューになります。

popView() 現在の View オブジェクトをナビゲーションスタックからポップして、その View オブジェクトを破棄します。スタックにある前の View オブジェクトが現在のビューになります。

popToFirstView() スタックにある最初の View オブジェクトを除いて、スタックからすべての View オブジェクトをポップして破棄します。スタックにある最初の View オブジェクトが現在のビューになります。

popAll() ViewNavigator のスタックを空にしてすべての View オブジェクトを破棄します。アプリケーションには、空白のビューが表示されます。

次の図に 2 つのビューを示します。現在のビューを変更するには、ViewNavigator.pushView() メソッドを使用して、新しいビューを表す View オブジェクトをスタックにプッシュします。pushView() メソッドによって、ViewNavigator が表示を新しい View オブジェクトに切り替えます。



View オブジェクトのプッシュおよびポップによるビューの変更

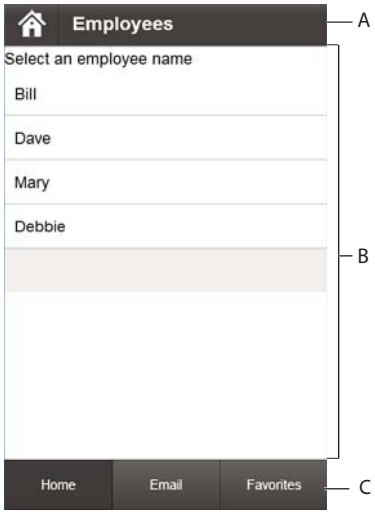
現在の View オブジェクトをスタックから削除するには、ViewNavigator.popView() メソッドを使用します。ViewNavigator によって、スタックにある前の View オブジェクトに表示が戻ります。

注意：モバイルアプリケーションでは、ナビゲーションの大部分をモバイルデバイス自身が制御します。例えば、Flex で構築されたモバイルアプリケーションは、モバイルデバイスの戻るボタンを自動的に処理します。したがって、戻るボタンのサポートをアプリケーションに追加する必要はありません。ユーザーがモバイルデバイスの戻るボタンを押すと、Flex によって popView() メソッドが自動的に呼び出され、前のビューが復元されます。

ブロガーの David Hassoun 氏が、ビューのデータ管理に関するブログを公開しています。

複数のセクションがあるアプリケーション用のナビゲーションの作成

次の図では、複数のセクションに View が配置されています。各セクションは、異なる ViewNavigator コンテナで定義します。各セクション内には 1 つ以上のビューがあります。



A. ActionBar **B.** コンテンツ領域 **C.** タブバー

現在の ViewNavigator に対応する現在のセクション内のビューを変更するには、pushView() および popView() メソッドを使用します。

現在のセクションを変更するには、タブバーを使用します。セクションを切り替えると、新しいセクションの ViewNavigator コンテナに切り替わります。表示が変更され、新しい ViewNavigator のスタックの最上位に現在ある View オブジェクトが表示されます。

セクションは、TabbedViewNavigator.selectedIndex プロパティを使用してプログラムで変更することもできます。このプロパティには、選択したビューナビゲーターの 0 から始まるインデックスが格納されています。

モバイルアプリケーションでのユーザー入力の処理

モバイルアプリケーションでは、デスクトップやブラウザーアプリケーションとは異なるユーザー入力の処理が必要になります。AIR 用に構築されたデスクトップアプリケーション、または Flash Player 用に構築されたブラウザーアプリケーションでは、主要な入力デバイスはマウスおよびキーボードです。モバイルデバイスの場合、主要な入力デバイスはタッチスクリーンです。モバイルデバイスは何らかのキーボードを備えている場合も多く、デバイスによっては 5 方向（左、右、上、下、選択）の入力方法も備えています。

mx.core.UIComponent クラスは、アプリケーションで使用する入力タイプをコンポーネントで設定するのに使用する interactionMode スタイルプロパティを定義します。Halo と Spark テーマの場合、デフォルト値は mouse で、マウスが主要な入力デバイスであることを示しています。Mobile テーマの場合、デフォルト値は touch で、主要な入力デバイスがタッチスクリーンであることを示しています。

モバイルアプリケーションでのハードウェアキーのサポート

ViewNavigatorApplication コンテナまたは TabbedViewNavigatorApplication コンテナで定義されているアプリケーションは、デバイスの戻るハードウェアキーとメニューハードウェアキーに応答します。戻るキーを押すと、アプリケーションは前のビューにナビゲートします。前のビューがない場合は、アプリケーションが終了し、デバイスのホームスクリーンが表示されます。

戻るボタンを押すと、アプリケーションのアクティブビューが backKeyPressed イベントを受信します。戻るキーのアクションをキャンセルするには、backKeyPressed イベントのイベントハンドラーで preventDefault() を呼び出します。

メニューボタンを押すと、現在のビューに ViewMenu コンテナが定義されている場合は、そのコンテナが表示されます。ViewMenu コンテナは、View コンテナの下部にあるメニューを定義します。各 View コンテナは、そのビュー固有のメニューを定義します。

メニューキーを押すと、現在の View コンテナが menuKeyPressed イベントを送出します。メニューボタンのアクションをキャンセルして、ViewMenu が表示されないようにするには、menuKeyPressed イベントのイベントハンドラーの preventDefault() メソッドを呼び出します。

詳しくは、68 ページの「モバイルアプリケーションでのメニューの定義」を参照してください。

モバイルアプリケーションでのハードウェアキーボードイベントの処理

Flex で構築したモバイルアプリケーションでは、ユーザーによってモバイルデバイスのハードウェアキーが押されたことを検出できます。例えば、Android デバイスで、ホームボタン、戻るボタンまたはメニューボタンが押されたことを検出できます。

ハードウェアキーが押されたことを検出するには、KEY_UP または KEY_DOWN イベントのイベントハンドラーを作成します。通常、イベントハンドラーは、Application、ViewNavigatorApplication または TabbedViewNavigatorApplication コンテナで定義されているアプリケーションオブジェクトに追加します。

Stage オブジェクトでは、アプリケーションの描画領域を定義します。各アプリケーションには、Stage オブジェクトが 1 つあります。したがって、アプリケーションコンテナは、実際には Stage オブジェクトの子コンテナになります。

Stage.focus プロパティは、現在キーボードフォーカスが設定されているコンポーネントを示します。どのコンポーネントにもフォーカスが設定されていない場合は null になります。キーボードフォーカスが設定されているコンポーネントとは、ユーザーがキーボードを操作したときに、イベントの通知を受け取るコンポーネントです。そのため、Stage.focus がアプリケーションオブジェクトに設定されている場合は、アプリケーションオブジェクトのイベントハンドラーが呼び出されます。

モバイルデバイスでは、別のアプリケーションによってアプリケーションが中断されることがあります。例えば、モバイルデバイスでアプリケーションを実行しているときに電話がかかってきたり、ユーザーによって別のアプリケーションに切り替えられたりすることがあります。ユーザーがアプリケーションの切り替えを戻したときには、Stage.focus プロパティに null が設定されます。そのため、アプリケーションオブジェクトに割り当てられたイベントハンドラーは、キーボードに反応しません。

モバイルアプリケーションでは Stage.focus プロパティが null になる可能性があるので、Stage オブジェクト自体でキーボードイベントをリスニングして、必ずアプリケーションでイベントが認識されるようにしてください。次の例は、キーボードイベントハンドラーを Stage オブジェクトに割り当てています。

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

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

            // Add the hardware key event handlers to the stage.
            protected function appCompleteHandler(event:FlexEvent):void {
                stage.addEventListener("keyDown", handleButtons, false,1);
                stage.addEventListener("keyUp", handleButtons, false, 1);
            }

            // Event handler to handle hardware keyboard keys.
            protected function handleButtons(event:KeyboardEvent):void
            {
                if (event.keyCode == Keyboard.HOME) {
                    // Handle Home button.
                }
                else if (event.keyCode == Keyboard.BACK) {
                    // Hanlde back button.
                }
            }
        ]]>
    </fx:Script>
</s:ViewNavigatorApplication>
```

モバイルアプリケーションでのマウスイベントとタッチイベントの処理

AIR では、異なる入力タイプを示すために異なるイベントを生成します。これには次のようなイベントが含まれます。

マウスイベント マウスまたはタッチスクリーンによって生成されたユーザーインタラクションによって生成されたイベント。マウスイベントには、mouseOver、mouseDown および mouseUp があります。

タッチイベント タッチスクリーンへの指でのタッチなど、ユーザーによるデバイスへの接触を検出するデバイスで生成されるイベント。タッチイベントには、touchTap、touchOver および touchMove があります。ユーザーがタッチスクリーンでデバイスを操作する場合は、通常、指またはポインティングデバイスを使用してスクリーンにタッチします。

ジェスチャーイベント 2 本の指でタッチスクリーンを同時に押すなど、マルチタッチインタラクションで生成されるイベント。ジェスチャーイベントには、gesturePan、gestureRotate および gestureZoom があります。例えば、一部のデバイスでは、ピンチジェスチャーを使用してイメージをズームアウトできます。

マウスイベントのプリセットサポート

Flex フレームワークおよび Flex コンポーネントセットには、マウスイベントのプリセットのサポートはありますが、タッチイベントやジェスチャーイベントのサポートはありません。例えば、ユーザーはタッチスクリーンを使用して、モバイルアプリケーションの Flex コンポーネントを操作します。コンポーネントは、mouseDown や mouseOver などのマウスイベントに応答しますが、タッチイベントやジェスチャーイベントには応答しません。

例えば、ユーザーがタッチスクリーンを押して Flex の Button コントロールを選択したとします。Button コントロールは、mouseUp および mouseDown イベントを使用して、ユーザーがこのコントロールを操作したことを通知します。Scroller コントロールは、mouseMove および mouseUp イベントを使用して、ユーザーが表示をスクロールしていることを示します。


アドビのデベロッパーエバンジェリストの Paul Trani がタッチイベントとジェスチャーイベントの処理について、Touch Events and Gesture on Mobile で説明しています。

AIR によって生成されたイベントの制御

flash.ui.Multitouch.inputMode プロパティは、AIR および Flash Player で生成されるイベントを制御します。flash.ui.Multitouch.inputMode プロパティの値は、次のうちのいずれかです。

- MultitouchInputMode.NONE：AIR ではマウスイベントを送出しますが、タッチイベントまたはジェスチャーイベントは送出しません。
- MultitouchInputMode.TOUCH_POINT：AIR ではマウスイベントおよびタッチイベントを送出しますが、ジェスチャーイベントは送出しません。このモードでは、Flex フレームワークは、MultitouchInputMode.NONE で受信するのと同じマウスイベントを受信します。
- MultitouchInputMode.GESTURE：AIR ではマウスイベントおよびジェスチャーイベントを送出しますが、タッチイベントは送出しません。このモードでは、Flex フレームワークは、MultitouchInputMode.NONE で受信するのと同じマウスイベントを受信します。

リストに示すように、flash.ui.Multitouch.inputMode プロパティの設定に関係なく、AIR は常にマウスイベントを送出します。したがって、Flex コンポーネントは、タッチスクリーンを使用して実行されたユーザーインタラクションに常に応答できます。

Flex では、アプリケーションの flash.ui.Multitouch.inputMode プロパティにどの値でも使用できます。したがって、Flex コンポーネントは、タッチイベントおよびジェスチャーイベントに応答しませんが、どのイベントにも応答するようにアプリケーションに機能を追加できます。例えば、Button コントロールにイベントハンドラーを追加して、touchTap、touchOver、touchMove イベントなどのタッチイベントを処理できます。

『ActionScript 3.0 開発ガイド』には、様々なデバイスでのユーザー入力処理の概要と、タッチ、マルチタッチおよびジェスチャー入力操作に関する説明があります。詳しくは、以下を参照してください。

- ユーザー操作の基礎
- タッチ入力、マルチタッチ入力およびジェスチャ入力

モバイルアプリケーションとスプラッシュ画面の定義

モバイルアプリケーションコンテナの作成

モバイルアプリケーションの最初のタグは、通常は次のいずれかです。

- <s:ViewNavigatorApplication> タグは、セクションが 1 つのみのモバイルアプリケーションを定義します。

- <s:TabbedViewNavigatorApplication> タグは、複数のセクションがあるモバイルアプリケーションを定義します。

タブレット用のアプリケーションを開発する際は、電話機の場合ほどスクリーンサイズの制限は重要ではありません。したがって、タブレットの場合は、アプリケーションを小さなビューで構造化する必要がありません。代わりに、標準の Spark の Application コンテナを使用し、サポートされているモバイルコンポーネントとスキンを使用して、アプリケーションを作成できます。

注意：どのようなモバイルアプリケーションを作成する際にも、電話機の場合でも、Spark の Application コンテナを使用できます。ただし、Spark の Application コンテナでは、ビューのナビゲーション、データのパーシスタンス、デバイスの戻るボタンとメニューボタンはサポートされません。詳しくは、34 ページの「モバイルアプリケーションコンテナと Spark の Application コンテナの相違」および About the Application container を参照してください。

モバイルアプリケーションのコンテナには、次のデフォルトの特性があります。

特性	**Spark の ViewNavigatorApplication コンテナと TabbedViewNavigatorApplication コンテナ**
デフォルトサイズ	利用可能なスクリーン領域をすべて使用する 100 %の高さと 100 %の幅。
子レイアウト	アプリケーションのビューを構成する個々の View コンテナごとに定義されます。
デフォルトのパディング	0 ピクセル。
スクロールバー	なし。アプリケーションコンテナのスキンにスクロールバーを追加すると、ユーザーがアプリケーション全体をスクロールできるようになります。このスクロールは、アプリケーションの ActionBar 領域とタブバー領域も対象となります。通常、ビューのこれらの領域をスクロールする必要はありません。そのため、スクロールバーは、アプリケーションコンテナのスキンではなく、アプリケーションの個々の View コンテナに追加します。

モバイルアプリケーションコンテナと Spark の Application コンテナの相違

Spark のモバイルアプリケーションコンテナには、Spark の Application コンテナと同じ機能が多数用意されています。例えば、モバイルアプリケーションコンテナにスタイルを適用する方法は、Spark の Application コンテナに適用する方法と同じです。

Spark のモバイルアプリケーションコンテナには、Spark の Application コンテナとは異なるいくつかの特性があります。

- **パーシスタンスのサポート**

  データ記憶域をサポートし、ディスクとの間でデータの書き込みと読み込みを行います。パーシスタンスによって、ユーザーはモバイルアプリケーションを中断できます。例えば、電話の呼び出しに応答し、通話が終了したときにアプリケーションの状態を復元できます。

- **ビューナビゲーションのサポート**

  ViewNavigatorApplication コンテナは、複数のビュー間のナビゲーションを制御する単一の ViewNavigator コンテナを自動的に作成します。

  TabbedViewNavigatorApplication コンテナは、複数のセクション間のナビゲーションを制御する単一の TabbedViewNavigator コンテナを自動的に作成します。

- **デバイスの戻るボタンとメニューボタンのサポート**

  ユーザーが戻るボタンを押すと、スタックにある前のビューが表示されます。メニューボタンを押すと、現在のビューに ViewMenu コンテナが定義されている場合は、そのコンテナが表示されます。

Spark の Application コンテナについて詳しくは、About the Application container を参照してください。

アプリケーションレベルのイベントの処理

NativeApplication クラスは AIR アプリケーションを表します。このクラスは、アプリケーションの情報やアプリケーション全体の機能を提供し、アプリケーションレベルのイベントを送出します。ユーザーのモバイルアプリケーションに相当する NativeApplication クラスのインスタンスには、NativeApplication.nativeApplication という静的なプロパティを使用してアクセスできます。

例えば、NativeApplication クラスは、ユーザーのモバイルアプリケーションで処理できる invoke および exiting というイベントを定義します。次の例は、exiting イベントのイベントハンドラーを定義する NativeApplication クラスを示しています。

```
<?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>
```

ViewNavigator には、ViewNavigatorApplication.navigator プロパティを使用してアクセスすることに注意してください。

アプリケーションへのスプラッシュ画面の追加

Spark の Application コンテナは、ViewNavigatorApplication コンテナと TabbedViewNavigatorApplication コンテナの基本クラスです。Spark テーマで使用する場合、Spark の Application コンテナでは、アプリケーションの SWF ファイルのダウンロードおよび初期化の進捗が表示されるアプリケーションプリローダーがサポートされます。

Mobile テーマで使用する場合は、代わりにスプラッシュ画面を表示できます。スプラッシュ画面は、アプリケーションの起動時に表示されます。

注意：デスクトップアプリケーションでスプラッシュ画面を使用するには、Application.preloader プロパティを spark.preloaders.SplashScreen に設定します。また、アプリケーションのライブラリパスに frameworks¥libs¥mobile¥mobilecomponents.swc を追加します。


ブロガーの Joseph Labrecque 氏が、AIR for Android Splash Screen with Flex というブログを公開しています。


ブロガーの Brent Arnold 氏が、Android アプリケーションへのスプラッシュ画面の追加に関するビデオを公開しています。

画像ファイルからのスプラッシュ画面の追加

スプラッシュ画面を画像ファイルから直接読み込むことができます。スプラッシュ画面を設定するには、アプリケーションクラスの splashScreenImage、splashScreenScaleMode および splashScreenMinimumDisplayTime プロパティを使用します。

例えば、次の例では、レターボックス形式を使用して JPG ファイルからスプラッシュ画面を読み込みます。

```
<?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>
```

カスタムコンポーネントからのスプラッシュ画面の追加

前の節の例では、JPG ファイルを使用してスプラッシュ画面を定義しました。このメカニズムの欠点は、アプリケーションが実行されるモバイルデバイスの性能に関係なく、同じ画像が使用されてしまうことです。

モバイルデバイスのスクリーン解像度とサイズは様々です。スプラッシュ画面に 1 つの画像を使用するのではなく、その代わりにカスタムコンポーネントを定義できます。そのコンポーネントでモバイルデバイスの性能を判別し、適切な画像をスプラッシュ画面に使用します。

カスタムコンポーネントを定義するには、SplashScreenImage クラスを使用します。次に例を示します。

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

    <!-- Default splashscreen image. -->
    <s:SplashScreenImageSource
        source="@Embed('../assets/logoDefault.jpg')"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Portrait.jpg')"
        dpi="240"
        aspectRatio="portrait"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Landscape.jpg')"
        dpi="240"
        aspectRatio="landscape"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo160.jpg')"
        dpi="160"
        aspectRatio="portrait"
        minResolution="960"/>
</s:SplashScreenImage>
```

コンポーネントの定義内で、SplashScreenImageSource クラスを使用して、スプラッシュ画面の各画像を定義します。SplashScreenImageSource.source プロパティで画像ファイルを指定します。SplashScreenImageSource の dpi、aspectRatio および minResolution プロパティで、画像の表示に必要なモバイルデバイスの性能を定義します。

例えば、最初の SplashScreenImageSource 定義では、画像の source プロパティのみを指定しています。dpi、aspectRatio および minResolution プロパティは設定されていないので、この画像はすべてのデバイスで使用できます。したがって、ここでは、デバイスの機能と一致する他の画像がない場合に表示されるデフォルトの画像を定義しています。

2 つ目と 3 つ目の SplashScreenImageSource 定義では、縦向きモードまたは横向きモードで 240 DPI のデバイス用の画像を指定しています。

最後の SplashScreenImageSource 定義では、縦向きモードで 160 DPI、かつ最低解像度が 960 ピクセルのデバイス用の画像を指定しています。minResolution プロパティの値は、Stage.stageWidth と Stage.stageHeight プロパティの大きい方の値と比較されます。2 つの値のうち大きい方の値が、minResolution プロパティ以上である必要があります。

次のモバイルアプリケーションで、このコンポーネントを使用します。

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

SplashScreenImage クラスでは、デバイスの性能に最も一致する画像が自動的に判別されます。この一致確認は、各 SplashScreenImageSource 定義の dpi、aspectRatio および minResolution プロパティに基づいて行われます。

最も一致する対象を判別する手順は、次のとおりです。

1. モバイルデバイスの設定と一致するすべての SplashScreenImageSource 定義を判別します。次の場合に一致したと見なされます。
   - **a** SplashScreenImageSource 定義で、その設定が明示的に定義されていない場合。例えば、dpi プロパティの設定がない場合は、すべてのデバイスの DPI. と一致します。
   - **b** dpi または aspectRatio プロパティについては、プロパティとモバイルデバイスの対応する設定が完全に一致する必要があります。
   - **c** minResolution プロパティについては、Stage.stageWidth と Stage.stageHeight プロパティの大きい方が minResolution 以上である場合に、プロパティはデバイス上の設定と一致します。
2. 2 つ以上の SplashScreenImageSource 定義がデバイスと一致する場合は、
   - **a** 明示された設定が最多数のものが選択されます。例えば、SplashScreenImageSource 定義で dpi と aspectRatio プロパティの両方が指定されている定義は、dpi プロパティのみが指定されている定義よりも一致が優先されます。
   - **b** 引き続き 2 つ以上の一致が存在する場合は、minResolution の値が最も高いものが選択されます。
   - **c** 引き続き 2 つ以上の一致が存在する場合は、コンポーネント内で最初に定義されているものが選択されます。

スプラッシュ画面の画像の明示的な選択

SplashScreenImage.getImageClass() メソッドは、モバイルデバイスの性能と最も一致する SplashScreenImageSource 定義を判別します。このメソッドをオーバーライドして、独自のカスタムロジックを追加できます。次に例を示します。

この例では、iOS のスプラッシュ画面用の SplashScreenImageSource 定義を追加しています。getImageClass() メソッドのオーバーライド部分で、最初にアプリケーションが iOS で実行されているか判別します。該当する場合、iOS 専用の画像を表示します。

アプリケーションが iOS で実行されていない場合は、super.getImageClass() メソッドを呼び出します。このメソッドでは、デフォルトの実装を使用して、表示する SplashScreenImageSource インスタンスを判別します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\myComponents\MyIOSSplashScreen.mxml -->
<s:SplashScreenImage xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <fx:Script>
        <![CDATA[
            // Override getImageClass() to return an image for iOS.
          override public function getImageClass(aspectRatio:String, dpi:Number, resolution:Number):Class {
                // Is the application running on iOS?
                if (Capabilities.version.indexOf("IOS") == 0)
                    return iosImage.source;

                return super.getImageClass(aspectRatio, dpi, resolution);
            }
        ]]>
    </fx:Script>
    <!-- Default splashscreen image. -->
    <s:SplashScreenImageSource
        source="@Embed('../assets/logoDefault.jpg')"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Portrait.jpg')"
        dpi="240"
        aspectRatio="portrait"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Landscape.jpg')"
        dpi="240"
        aspectRatio="landscape"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo160.jpg')"
        dpi="160"
        aspectRatio="portrait"
        minResolution="960"/>
    <!-- iOS splashscreen image. -->
    <s:SplashScreenImageSource id="iosImage"
        source="@Embed('../assets/logoIOS.jpg')"/>
</s:SplashScreenImage>
```

モバイルアプリケーションでのビューの定義

通常、モバイルアプリケーションでは複数の画面またはビューを定義します。ユーザーがアプリケーションをナビゲートするのに応じて、様々なビューに切り替わります。

ナビゲーションは、アプリケーションのユーザーにわかりやすいものにします。つまり、ユーザーが 1 つのビューから別のビューに移動する際には、前のビューに戻れるようにする必要があります。「ホーム」ボタンや、他の最上位のナビゲーション補助（ユーザーが他の場所からアプリケーションの場所に移動できる機能）を定義できます。

モバイルアプリケーションのビューを定義するには、View コンテナを使用します。モバイルアプリケーションのビュー間のナビゲーションを制御するには、ViewNavigator コンテナを使用します。

pushView() を使用したビューの変更

新規のビューをスタックにプッシュするには、ViewNavigator.pushView() メソッドを使用します。ViewNavigator には、ViewNavigatorApplication.navigator プロパティを使用してアクセスします。ビューをプッシュすると、アプリケーションの表示が新しいビューに切り替わります。

pushView() メソッドの構文は、次のとおりです。

```
pushView(viewClass:Class,
    data:Object = null,
    context:Object = null,
    transition:spark.transitions:ViewTransitionBase = null):void
```

ここで、各項目の説明は次のとおりです。

- viewClass には、ビューのクラス名を指定します。通常、このクラスはビューを定義する MXML ファイルに相当します。
- data には、ビューに渡すデータを指定します。このオブジェクトは、新しいビューの View.data プロパティに書き込まれます。
- context には、ViewNavigator.context プロパティに書き込む任意のオブジェクトを指定します。新規のビューを作成すると、このプロパティが参照され、この値に基づいてアクションが実行されます。例えば、context の値に基づいて、様々な方法でデータが表示されます。
- transition には、ビューが新規のビューに切り替わる際に再生するトランジションを指定します。ビューのトランジションについて詳しくは、87 ページの「モバイルアプリケーションでのトランジションの定義」を参照してください。

単一の Object を渡すための data 引数の使用

新規のビューに必要なデータを格納した単一の Object を渡すには、data 引数を使用します。その後、次の例に示すように、View.data プロパティを使用して、オブジェクトにアクセスできます。

```
<?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>
```

この例では、EmployeeView は EmployeeView.mxml ファイルに定義されています。このビューは、data プロパティを使用して、渡された Object から従業員の姓名および従業員の ID にアクセスします。

View.data プロパティは、View オブジェクトに対する add イベント時点までには確実に有効になります。View コンテナのライフサイクルについて詳しくは、46 ページの「Spark の ViewNavigator コンテナと View コンテナのライフサイクル」を参照してください。

アプリケーションの最初のビューに対するデータの引き渡し

ViewNavigatorApplication.firstView プロパティと ViewNavigator.firstView プロパティでは、アプリケーションの最初のビューを定義します。最初のビューにデータを渡すには、ViewNavigatorApplication.firstViewData プロパティまたは ViewNavigator.firstViewData プロパティを使用します。

ビューに対するデータの引き渡し

次の例では、ViewNavigatorApplication コンテナを使用して、モバイルアプリケーションを定義します。ViewNavigatorApplication コンテナは、アプリケーションによって定義された各ビューのナビゲートに使用する ViewNavigator クラスの単一のインスタンスを自動的に作成します。

```
<?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>
```

この例では、ActionBar コントロールのナビゲーション領域に「ホーム」ボタンを定義します。「ホーム」ボタンを選択すると、スタックからすべてのビューがポップされて最初のビューに戻ります。次の図は、このアプリケーションを示しています。

次の例に示すように、EmployeeMainView.mxml ファイルでは、アプリケーションの最初のビューを定義します。

```
<?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>
```

このビューには、ユーザーが従業員名を選択できる List コントロールを定義します。名前を選択すると、change イベントのイベントハンドラーによって、別のビューのインスタンスが EmployeeView というスタックにプッシュされます。EmployeeView のインスタンスがプッシュされると、アプリケーションでは EmployeeView ビューへの切り替えが処理されます。

この例の pushView() メソッドでは、新しいビューおよび新しいビューに渡すデータを定義する Object の 2 つの引数が使用されます。この例では、List コントロールで現在選択されているアイテムに対応するデータオブジェクトを渡します。

次の例は、EmployeeView の定義を示しています。

```
<?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>
```

EmployeeView には、List コントロールのデータプロバイダーからの 3 つのフィールドが表示されます。EmployeeView は、View.data プロパティを使用して、渡されたデータにアクセスします。

ブロガーの Steve Mathews 氏が、Passing data between Views にクックブックエントリを公開しています。

ビューからのデータの受け取り

ViewNavigator.popView() メソッドは、現在のビューからスタックにある前のビューに制御を戻します。popView() メソッドを実行すると、現在のビューが破棄され、スタックにある前のビューが復元されます。前の View の復元には、その data プロパティをスタックからリセットすることも含まれます。

作成時に送出されるイベントなど、ビューのライフサイクルについて詳しくは、46 ページの「Spark の ViewNavigator コンテナと View コンテナのライフサイクル」を参照してください。

新しいビューは、非アクティブ化された時点の元の data オブジェクトが設定されて復元されます。したがって、通常は元の data オブジェクトを使用して古いビューから新しいビューにデータを渡すことはしません。代わりに、古いビューの createReturnObject() メソッドをオーバーライドします。createReturnObject() メソッドは、単一の Object を返します。

返されるオブジェクトの種類

createReturnObject() メソッドによって返される Object は、ViewNavigator.poppedViewReturnedObject プロパティに書き込まれます。poppedViewReturnedObject プロパティのデータのタイプは ViewReturnObject です。

ViewReturnObject には、context と object の 2 つのプロパティを定義します。object プロパティには、createReturnObject() メソッドによって返される Object が格納されます。context プロパティには、pushView() を使用してナビゲーションスタックにビューがプッシュされたときにビューに渡された context 引数の値が格納されています。

poppedViewReturnedObject プロパティは、add イベントがビューで受信されるまでに、新しいビューに確実に設定されます。poppedViewReturnedObject.object プロパティが null の場合、データは返されません。

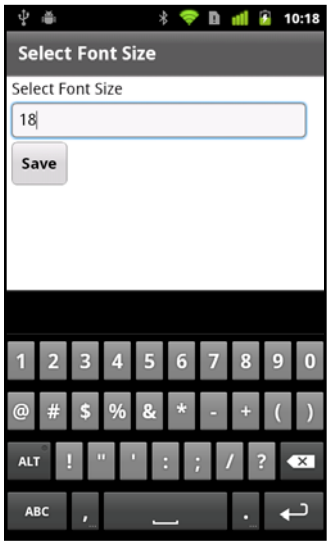
例：ビューに対するデータの引き渡し

次の SelectFont.mxml の例は、フォントサイズを設定できるビューを示しています。createReturnObject() メソッドのオーバーライドにより、値は Number として返されます。前のビューから渡された data プロパティの fontSize フィールドによって、TextInput コントロールの初期値が設定されます。

```
<?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>
```

次の図は、SelectFont.mxml によって定義されるビューを示しています。



次の例の MainFontView.mxml というビューでは、SetFont.mxml で定義されているビューを使用しています。MainFontView.mxml ビューでは、次のことが定義されています。

- SetFont.mxml で定義されているビューに切り替える ActionBar の Button コントロール。
- View.data プロパティが null かどうかを最初に判断する add イベントのイベントハンドラー。null の場合は、イベントハンドラーが data.fontSize フィールドを View.data プロパティに追加します。

  data プロパティが null でない場合は、イベントハンドラーがフォントサイズを data.fontSize フィールドの値に設定します。

```
<?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>
```

縦方向と横方向のアプリケーションの設定

デバイスの方向が変わると、モバイルデバイスによって、アプリケーションの方向が自動的に設定されます。異なる方向にアプリケーションを設定するために、Flex では、縦方向と横方向に対応する portrait と landscape の 2 つのビューステートを定義します。これらのビューステートを使用して、方向に基づいたアプリケーションの特性を設定します。

次の例では、ビューステートを使用して、現在の方向に基づいて Group コンテナの layout プロパティを制御します。

```
<?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>
```

この例は、検索ビューを定義しています。Group コンテナは、入力検索テキストと「検索」ボタンのレイアウトを制御します。縦方向モードの場合、Group コンテナでは垂直方向のレイアウトが使用されます。レイアウトを横方向モードに変更すると、Group コンテナで水平方向のレイアウトが使用されるようになります。

レイアウトモードをサポートするためのカスタムスキンの定義

モバイルアプリケーションには、カスタムスキンクラスを定義できます。縦方向および横方向のレイアウトがサポートされるスキンでは、portrait と landscape のビューステートを処理する必要があります。

ユーザーがデバイスを回転したときにレイアウトの方向が変更されないようにアプリケーションを設定できます。そのためには、アプリケーションの -app.xml で終わる XML ファイルを編集し、次のプロパティを設定します。

- レイアウトの方向がアプリケーションによって変更されないようにするには、<autoOrients> プロパティを false に設定します。
- 方向を設定するには、<aspectRatio> プロパティを portrait または landscape に設定します。

Spark の ViewNavigator コンテナのオーバーレイモードの設定

デフォルトでは、モバイルアプリケーションのタブバーと ActionBar コントロールによって、アプリケーションのビューでは使用できない領域が定義されます。つまり、コンテンツには、モバイルデバイスのフルスクリーンサイズを使用できません。

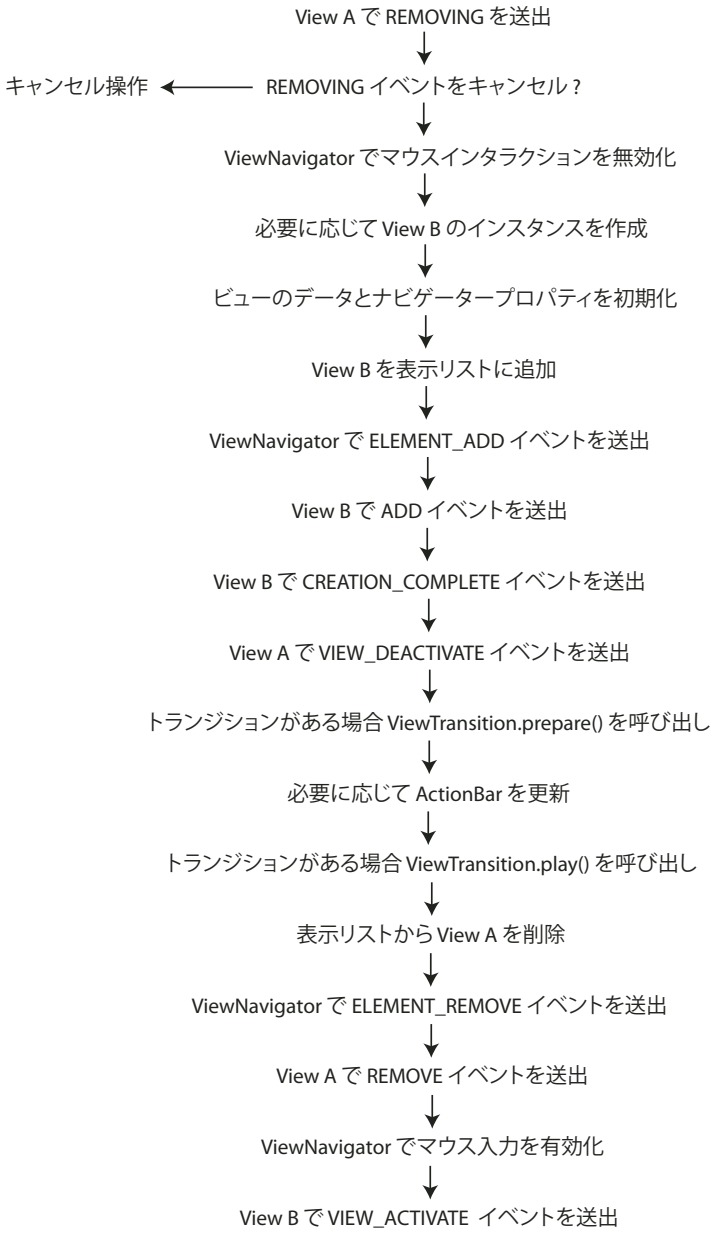
ただし、これらのコンポーネントのデフォルトのレイアウトは、ViewNavigator.overlayControls プロパティを使用して変更できます。overlayControls プロパティを true に設定すると、アプリケーションのコンテンツ領域は、スクリーンの幅と高さ全体に広がります。ActionBar コントロールとタブバーは、部分的に透明になるアルファ値を使用してコンテンツ領域に重ねられます。

ViewNavigator コンテナの spark.skins.mobile.ViewNavigatorSkin というスキンクラスでは、異なる値の overlayControls プロパティを処理するためのビューステートを定義します。overlayControls プロパティが true の場合は、現在のステート名に「AndOverlay」が追加されます。例えば、ViewNavigator のスキンは、デフォルトで「portrait」ステートになります。overlayControls プロパティが true の場合は、ナビゲーターのスキンステートが「portraitAndOverlay」に変更されます。

Spark の ViewNavigator コンテナと View コンテナのライフサイクル

ユーザーがモバイルアプリケーションでビューを別のビューに切り替えると、Flex により一連の操作が実行されます。ビューの切り替えプロセスの様々な段階で、Flex によってイベントが送出されます。これらのイベントを監視して、プロセスの途中でアクションを実行できます。例えば、removing イベントを使用して、あるビューから別のビューへの切り替えをキャンセルできます。

次の図は、View A という現在のビューから View B という別のビューに切り替えるプロセスを示しています。



モバイルアプリケーションでのタブの定義

アプリケーションのセクションの定義

TabbedViewNavigatorApplication コンテナを使用して、複数のセクションがあるモバイルアプリケーションを定義します。TabbedViewNavigatorApplication コンテナは、TabbedViewNavigator コンテナを自動的に作成します。TabbedViewNavigator コンテナには、アプリケーションのセクション間のナビゲーションをサポートするタブバーを作成します。

各 ViewNavigator コンテナでは、アプリケーションの異なるセクションを定義します。ViewNavigator コンテナを指定するには、TabbedViewNavigatorApplication コンテナの navigators プロパティを使用します。

次の例では、3 つの ViewNavigator タグに対応する 3 つのセクションを定義します。各 ViewNavigator には、セクションに移動したときに表示される最初のビューを定義します。

```
<?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>
```

注意：navigators 子タグは、TabbedViewNavigator のデフォルトのプロパティなので、MXML に指定する必要はありません。

各 ViewNavigator には個別のナビゲーションスタックが確保されます。したがって、pushView() や popView() などの ViewNavigator メソッドは、現在アクティブなセクションが対象となります。モバイルデバイスの戻るボタンは、現在の ViewNavigator のスタックにある前のビューに制御を戻します。ビューを変更しても、現在のセクションは変わりません。

セクションのナビゲーションについて、アプリケーションに特定のロジックを追加する必要はありません。TabbedViewNavigator コンテナは、セクションのナビゲーションを制御するタブバーをアプリケーションの下部に自動的に作成します。

必須ではありませんが、現在のセクションのプログラムによる制御を追加することもできます。セクションをプログラムで変更するには、TabbedViewNavigator.selectedIndex プロパティを目的のセクションのインデックスに設定します。セクションのインデックスは 0 から始まります。アプリケーションの最初のセクションはインデックス 0 にあり、2 番目はインデックス 1 にあるとういうように設定されます。

Flex のアドビ認定エキスパートの Brent Arnold 氏が、ViewNavigator のナビゲーションスタックの使用に関するビデオを公開しています。

アドビのエバンジェリストの Holly Schinsky が、モバイルアプリケーションでのタブ間のデータの受け渡し方法を Flex Mobile Development - Passing Data Between Tabs で説明しています。

video2brain の Creating a Tabbed View Navigator Application で公開されている TabbedViewNavigator コンテナに関するビデオをご覧ください。

セクション変更イベントの処理

セクションが変更されると、TabbedViewNavigator コンテナが次のイベントを送出します。

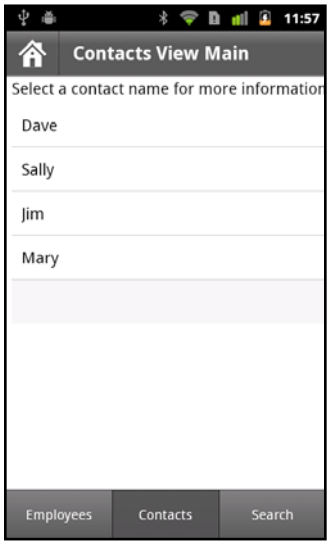
- セクションが変更される直前に changing イベントが送出されます。セクションの変更を防ぐには、changing イベントのイベントハンドラーの preventDefault() メソッドを呼び出します。
- セクションが変更された直後に change イベントが送出されます。

複数のセクションがある ActionBar の設定

ActionBar コントロールは ViewNavigator に関連付けられています。そのため、セクションの ViewNavigator を定義する際に、各セクションの ActionBar を設定できます。次の例では、アプリケーションの 3 つの異なるセクションを定義する各 ViewNavigator コンテナに対して別々に ActionBar を設定します。

```
<?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>
```

次の図は、タブバーで「Contacts」タブが選択された状態のこのアプリケーションを示しています。



別の方法として、アプリケーションの各ビューに ActionBar を定義することもできます。このようにすると、アプリケーションのどこで使用しても、各ビューで同じ ActionBar コンテンツが使用されます。

タブバーの制御

ビューでのタブバーコントロールの非表示化

ビューでは、View.tabBarVisible プロパティを false に設定することで、タブバーを非表示にできます。デフォルトでは、tabBarVisible プロパティは true に設定されており、タブバーが表示されます。

TabbedViewNavigator.hideTabBar() メソッドと TabbedViewNavigator.showTabBar() メソッドを使用して、表示を制御することもできます。

Flex のアドビ認定エキスパートの Brent Arnold 氏が、タブバーの非表示に関するビデオを公開しています。

TabbedViewNavigator コンテナのタブバーへのエフェクトの適用

デフォルトでは、タブバーの表示と非表示のエフェクトとしてスライドエフェクトが使用されます。タブバーでは、現在選択されているタブを変更する際に、エフェクトは使用されません。

タブバーの表示と非表示のデフォルトエフェクトを変更するには、TabbedViewNavigator.createTabBarHideEffect() メソッドと TabbedViewNavigator.createTabBarShowEffect() メソッドをオーバーライドします。タブバーを非表示にした後、タブバーの visible プロパティと includeInLayout プロパティを false に設定するのを忘れないでください。

モバイルアプリケーションでの複数ペインの作成

SplitViewNavigator は、モバイルデバイスの 1 つのスクリーンに 2 つ以上の子ビューナビゲーターを表示するスキナブルコンテナです。各ビューナビゲーターは、SplitViewNavigator コンテナによって管理される別々のペインに表示されます。

SplitViewNavigator コンテナの子には、ViewNavigatorBase を拡張する任意のコンポーネントを指定できます。そのため、ViewNavigator および TabbedViewNavigator コンテナを子として使用できます。

注意：スクリーン領域に同時に複数のペインを表示する必要があるので、タブレットでは SplitViewNavigator のみを使用することをお勧めします。

デフォルトでは、SplitViewNavigator は、子に対応するペインを水平方向にレイアウトします。垂直方向のレイアウトを使用するように指定したり、代わりにカスタムレイアウトを定義したりすることもできます。

SplitViewNavigator コンテナの作成

タブレットデバイスの一般的なインターフェイスパターンは、マスター／詳細のパターンです。このパターンでは、マスターペインと詳細ペインの 2 つのメインコンテンツ領域にスクリーンを分割します。通常、ユーザーはマスターペインを操作して、詳細ペインのコンテンツ表示を操作します。

各ペインは SplitViewNavigator の子に対応します。子は ViewNavigator または TabbedViewNavigator コンテナです。子はビューナビゲーターであるので、各ペインのビューナビゲーターには、その固有のビュースタックとアクションバーが含まれます。

次の図は、マスターペインと詳細ペインを持つ、アプリケーションの SplitViewNavigator コンテナを示しています。

この例では、左側のマスターペインに、一連のアドビ製品を表示する Spark List コントロールが含まれます。右側の詳細ペインには、現在マスターペインで選択されている製品の追加情報が表示されます。

次に、この例のメインアプリケーションファイルを示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSplitVNSimple.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <s:SplitViewNavigator width="100%" height="100%">
        <s:ViewNavigator width="256" height="100%"
            firstView="views.MasterCategory"/>
        <s:ViewNavigator width="100%" height="100%"
            firstView="views.DetailView"/>
    </s:SplitViewNavigator>
</s:Application>
```

SplitViewNavigator は、Application または TabbedViewNavigatorApplication コンテナの子に設定することができます。この例では、SplitViewNavigator は、Application コンテナの唯一の子です。SplitViewNavigator で高さと幅を 100 %に指定して、デバイスのスクリーンの全領域を占有していることに注目してください。

この例では、SplitViewNavigator の子は ViewNavigator コンテナです。最初の ViewNavigator でマスターペインを定義し、2 つ目で詳細ペインを定義します。

注意：SplitViewNavigator は、3 つ以上の子を持つこともできます。そのため、3 つや 4 つ、さらにそれ以上のペインを持つ SplitViewNavigator も作成できます。

SplitViewNavigator コンテナのペインおよびビューへのアクセス

SplitViewNavigator コンテナでは、自分の子へのアクセスに使用するメソッドとプロパティが定義されます。例えば、SplitViewNavigator.numViewNavigators プロパティを使用して、SplitViewNavigator の子ビューナビゲーターの数を決定します。

SplitViewNavigator.getViewNavigatorAt() メソッドを使用して、子のインデックスに基づいて SplitViewNavigator の子にアクセスします。前述の例の場合、マスターペインの ViewNavigator のインデックスは 0、詳細ペインの ViewNavigator のインデックスは 1 です。

注意：SplitViewNavigator コンテナは、getElementAt() メソッドと getElementIndex() メソッドを継承します。これらのメソッドは、SplitViewNavigator で使用しないでください。代わりに getViewNavigatorAt() を使用してください。

個々のペインの ViewNavigator への参照から、SplitViewNavigator でそのペインの個別のビューにアクセスできます。

子から SplitViewNavigator コンテナにアクセスするには、子の parentNavigator プロパティを使用します。例えば、ViewNavigator.parentNavigator には、親の SplitViewNavigator コンテナへの参照が含まれます。

View コンテナは、View.navigator プロパティを使用して、親のビューナビゲーターにアクセスします。そのため、View.navigator.parentNavigator を使用して、ビューから SplitViewNavigator にアクセスできます。

前述の例では、マスターペインの ViewNavigator で、最初のビューとして MasterCategory を指定しています。このビューは、MasterCategory.mxml ファイルで次のように定義されています。

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

    <fx:Script>
        <![CDATA[
            import spark.components.SplitViewNavigator;
            import spark.components.ViewNavigator;
            import spark.events.IndexChangeEvent;

            protected function myList_changeHandler(event:IndexChangeEvent):void {
                // Create a reference to the SplitViewNavigator.
                var splitNavigator:SplitViewNavigator = navigator.parentNavigator as SplitViewNavigator;
                // Create a reference to the ViewNavigator for the Detail frame.
                var detailNavigator:ViewNavigator = splitNavigator.getViewNavigatorAt(1) as ViewNavigator;
                // Change the view of the Detail frame based on the selected List item.
                detailNavigator.pushView(DetailView, myList.selectedItem);
            }
        ]]>
    </fx:Script>

    <s:List width="100%" height="100%" id="myList"
            change="myList_changeHandler(event);">
        <s:dataProvider>
            <s:ArrayCollection>
                <fx:Object Product="Adobe AIR" Price="11.99"
                           Image="@Embed(source='../assets/air_icon_sm.jpg')"
                           Description="Try AIR." Link="air"/>
                <fx:Object Product="Adobe ColdFusion" Price="11.99"
                           Image="@Embed(source='../assets/coldfusion_icon_sm.jpg')"
                           Description="Try ColdFusion." Link="coldfusion"/>
                <fx:Object Product="Adobe Flash Player" Price="11.99"
                           Image="@Embed(source='../assets/flashplayer_icon_sm.jpg')"
                           Description="Try Flash." Link="flashplayer"/>
```

```
                <fx:Object Product="Adobe Flex" Price="Free"
                           Image="@Embed(source='../assets/flex_icon_sm.jpg')"
                           Description="Try Flex." Link="flex.html"/>
                <fx:Object Product="Adobe LiveCycleDS" Price="11.99"
                           Image="@Embed(source='../assets/livecycleds_icon_sm.jpg')"
                           Description="Try LiveCycle DS." Link="livcycle"/>
                <fx:Object Product="Adobe LiveCycle ES2" Price="11.99"
                           Image="@Embed(source='../assets/livecyclees_icon_sm.jpg')"
                           Description="Try LiveCycle ES." Link="livcycle"/>
            </s:ArrayCollection>
        </s:dataProvider>
        <s:itemRenderer>
            <fx:Component>
                <s:IconItemRenderer
                    labelField="Product"
                    iconField="Image"/>
            </fx:Component>
        </s:itemRenderer>
    </s:List>
</s:View>
```

MasterCategory.mxml で、アドビ製品の情報が含まれる単一の List コントロールを定義しています。List コントロールは、カスタムアイテムレンダラーを使用して、各製品のラベルとアイコンを表示します。アイテムレンダラーの定義について詳しくは、Using a mobile item renderer with a Spark list-based control を参照してください。

マスターペインの List コントロールは change イベントを使用して、ユーザーアクションに反応して詳細ペインを更新します。イベントハンドラーは、最初に SplitViewNavigator コンテナへの参照を取得します。この参照から、詳細フレームの ViewNavigator への参照を取得します。

最後に、イベントハンドラーは、詳細フレームの ViewNavigator の push() メソッドを呼び出します。push() メソッドは 2 つの引数を持ちます。この引数は、ViewNavigator のスタックにプッシュするビューと、選択された List アイテムの情報が含まれるオブジェクトです。

SplitViewNavigator コンテナの詳細ペインの更新

前述の例の詳細ペインには、マスターペインの List コントロールで選択されたアイテムの情報が表示されます。詳細ペインには DetailView という名前が付けられ、DetailView.mxml ファイルで次のように定義されています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\DetailView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Detail">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[

            // Use navigateToURL() to open a link to the product page.
            protected function label1_clickHandler(event:MouseEvent):void {
                var destinationURL:String = "http://www.adobe.com/products/" + data.Link;
                navigateToURL(new URLRequest(destinationURL));
            }
        ]]>
    </fx:Script>


    <s:VGroup width="461" height="670">
        <s:Image source="{data.Image}"
            height="176" width="169"
            horizontalCenter="0" top="0"/>
        <s:Label text="{data.Product}"
            fontSize="24" fontWeight="bold"
            top="100" left="0"/>
        <s:Label text="Price: {data.Price}"
            top="125" left="0"/>
        <s:TextArea y="174"
            width="100%" height="20%"
            contentBackgroundColor="0xCC6600"
            text="{data.Description}"/>
        <s:Label text="Click for more information"
            color="#0000FF"
            click="label1_clickHandler(event)"/>
    </s:VGroup>
</s:View>
```

マスターペインは、List コントロールで選択されたアイテムに対応するオブジェクトを DetailView.mxml ファイルに渡します。詳細ペインは、View.data プロパティを使用して、このデータにアクセスします。続いて、詳細ペインは、製品の画像と情報を表示し、その製品のさらに詳細な情報が含まれるページへのハイパーリンクを作成します。

View コンテナへのデータの引き渡しについて詳しくは、39 ページの「ビューに対するデータの引き渡し」を参照してください。

デバイスの向きに基づいたペインの表示

タブレット用のアプリケーションを開発する場合は、タブレットの向きに基づいて別々のレイアウトを使用できます。例えば、横向きモードでは、タブレットは横長のスクリーン領域になり、複数のペインを簡単に表示できます。縦向きレイアウトでは、スクリーンの幅が狭くなるので、ペインを非表示にするように選択できます。

SplitViewNavigator コンテナでは、autoHideFirstViewNavigator プロパティが定義されます。このプロパティを使用して、それぞれの向きの最初のペインの表示を制御します。デフォルトでは、autoHideFirstViewNavigator は false であるので、向きに関係なく、コンテナに最初のペインが表示されます。

autoHideFirstViewNavigator を true に設定すると、コンテナでは、横向きモードの場合は最初のペインが表示され、縦向きモードの場合は最初のペインが非表示になります。SplitViewNavigator コンテナで最初のペインを非表示にするには、関連するビューナビゲーターの visible プロパティを false に設定します。

縦向きモードで最初のペインが非表示の場合、ペインを開くには、SplitViewNavigator.showFirstViewNavigatorInPopUp() メソッドを使用します。このメソッドを呼び出すと、Callout コンテナに最初のペインが開かれます。Callout コンテナは、アプリケーションの最前面に表示されるポップアップコンテナです。次に図を示します。

この例では、SplitViewNavigator の詳細ペインのアクションバーに「Show Navigator」というラベルのボタンを追加します。コンテナの最初のペインが非表示になっている場合、ユーザーはこのボタンを選択して、マスターペインを開きます。

注意：SkinnablePopUpContainer や、SkinnablePopUpContainer のサブクラスに最初のペインを開くには、SplitViewNavigator のカスタムスキンを作成します。

Callout が既に開かれている場合、showFirstViewNavigatorInPopUp() メソッドは無視されます。デバイスの向きが横向きモードに戻されると、Callout は自動的に閉じられ、最初のペインが再表示されます。

Callout コンテナの外側をクリックすると、コンテナは閉じられます。また、SplitViewNavigator.hideViewNavigatorPopUp() メソッドを呼び出して、コンテナを閉じることもできます。

Callout コンテナについて詳しくは、76 ページの「モバイルアプリケーションへの Callout コンテナの追加」を参照してください。

Callout コンテナに表示するペインへのアクションバーの追加

次に示すメインアプリケーションファイルでは、SplitViewNavigator の autoHideFirstViewNavigator プロパティを true に設定します。この例では、ビューステートを使用して、デバイスが縦向きモードになったときに、詳細ペインのアクションバーにボタンを追加します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSVNOrient.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    resize="resizeHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.ResizeEvent;

            // Update the state based on the orientation of the device.
            protected function resizeHandler(event:ResizeEvent):void {
                currentState = aspectRatio;
            }
        ]]>
    </fx:Script>

    <s:states>
        <s:State name="portrait"/>
        <s:State name="landscape"/>
    </s:states>

    <s:SplitViewNavigator id="splitNavigator"
            width="100%" height="100%"
            autoHideFirstViewNavigator="true">

        <s:ViewNavigator width="256" height="100%"
            firstView="views.MasterCategoryOrient"/>
        <s:ViewNavigator width="100%" height="100%"
            firstView="views.DetailView">
            <s:actionContent.portrait>
                <s:Button id="navigatorButton"
                    label="Show Navigator"
                    click="splitNavigator.showFirstViewNavigatorInPopUp(navigatorButton);"/>
            </s:actionContent.portrait>
        </s:ViewNavigator>
    </s:SplitViewNavigator>
</s:Application>
```

このアプリケーションでは、Application コンテナに resize イベントのイベントハンドラーを追加します。Flex では、タブレットの向きが変更されたときに、resize イベントが送出されます。resize イベントのイベントハンドラーで、現在の向きに基づいてアプリケーションのビューステートを設定します。ビューステートについて詳しくは、View states を参照してください。

詳細ペインのビューナビゲーターは、現在のステートを使用して、アクションバーの Button コントロールの外観を制御します。横向きモードの場合は、マスターペインが表示されるので、ボタンを非表示にします。

縦向きモードの場合は、マスターペインが非表示になると、詳細ペインのアクションバーに Button コントロールが表示されます。この場合、ユーザーは Button コントロールを選択して、マスターペインが含まれる Callout を開きます。

Button コントロールへの参照を、引数として showFirstViewNavigatorInPopUp() メソッドに渡します。この引数では Callout コンテナのホストを指定するので、その結果、Button コントロールを基準にして Callout が配置されます。

ユーザーアクションに対応して SplitViewNavigator の Callout を閉じる

Callout コンテナの外側をクリックすると、コンテナは閉じられます。ただし、デフォルトでは、Callout コンテナの内側をクリックしても、コンテナは閉じられません。

この例では、ユーザーによって List アイテムが選択されたときに、SplitViewnavigator.hideViewNavigatorPopUp() メソッドを呼び出して Callout を閉じます。このメソッドは、List コントロールの change イベントのイベントハンドラーで呼び出します。次に例を示します。

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

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.components.SplitViewNavigator;
            import spark.components.ViewNavigator;
            import spark.events.IndexChangeEvent;

            protected function myList_changeHandler(event:IndexChangeEvent):void {
                // Create a reference to the SplitViewNavigator.
                var splitNavigator:SplitViewNavigator = navigator.parentNavigator as SplitViewNavigator;
                // Create a reference to the ViewNavigator for the Detail frame.
                var detailNavigator:ViewNavigator = splitNavigator.getViewNavigatorAt(1) as ViewNavigator;
                // Change the view of the Detail frame based on the selected List item.
                detailNavigator.pushView(DetailView, myList.selectedItem);

                // If the Master is open in a callout, close it.
                // Otherwise, this method does nothing.
                splitNavigator.hideViewNavigatorPopUp();
            }
        ]]>
    </fx:Script>

    <s:List width="100%" height="100%" id="myList"
            change="myList_changeHandler(event);">
        <s:dataProvider>
            <s:ArrayCollection>
                <fx:Object Product="Adobe AIR" Price="11.99"
                           Image="@Embed(source='../assets/air_icon_sm.jpg')"
                           Description="Try AIR." Link="air"/>
                <fx:Object Product="Adobe ColdFusion" Price="11.99"
                           Image="@Embed(source='../assets/coldfusion_icon_sm.jpg')"
                           Description="Try ColdFusion." Link="coldfusion"/>
                <fx:Object Product="Adobe Flash Player" Price="11.99"
                           Image="@Embed(source='../assets/flashplayer_icon_sm.jpg')"
                           Description="Try Flash." Link="flashplayer"/>
                <fx:Object Product="Adobe Flex" Price="Free"
                           Image="@Embed(source='../assets/flex_icon_sm.jpg')"
                           Description="Try Flex." Link="flex.html"/>
                <fx:Object Product="Adobe LiveCycleDS" Price="11.99"
                           Image="@Embed(source='../assets/livecycleds_icon_sm.jpg')"
                           Description="Try LiveCycle DS." Link="livcycle"/>
                <fx:Object Product="Adobe LiveCycle ES2" Price="11.99"
                           Image="@Embed(source='../assets/livecyclees_icon_sm.jpg')"
                           Description="Try LiveCycle ES." Link="livcycle"/>
            </s:ArrayCollection>
        </s:dataProvider>
        <s:itemRenderer>
            <fx:Component>
                <s:IconItemRenderer
                    labelField="Product"
                    iconField="Image"/>
            </fx:Component>
        </s:itemRenderer>
    </s:List>
</s:View>
```

SplitViewNavigator コンテナのパーシスタンスの実装

モバイルデバイスのアプリケーションは、テキストメッセージ、電話の呼び出し、他のモバイルアプリケーションなど、他のアクションによって頻繁に中断されます。ユーザーは通常、中断されたアプリケーションが再起動されたときに中断前のアプリケーションの状態が復元されることを期待します。パーシスタンスメカニズムを使用すると、デバイスでは、アプリケーションの前の状態を復元できます。詳しくは、115 ページの「モバイルアプリケーションでのパーシスタンスの有効化」を参照してください。

SplitViewNavigator では、ViewNavigatorBase 基本クラスから継承した loadViewData() メソッドと saveViewData() メソッドが実装されます。そのため、SplitViewNavigator では、ナビゲーションスタックをシリアライズ／デシリアライズしたり、それぞれの子ナビゲーターのデータを表示したりすることができます。

ただし、アプリケーションが中断された場合は、手動でこれらのメソッドを呼び出す必要があります。次に例を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSplitVNPersist.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    initialize="initializeHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.managers.PersistenceManager;

            // Create an instance of the PersistenceManager.
            public var persistenceManager:PersistenceManager;

            // Event handler to initialize SplitViewNavigator.
            protected function initializeHandler(event:FlexEvent):void {

                // Register the event handler for the deactivate event.
                NativeApplication.nativeApplication.addEventListener(Event.DEACTIVATE, onDeactivate);

                persistenceManager = new PersistenceManager();
                persistenceManager.load();

                var data:Object = persistenceManager.getProperty("navigatorState");
                if (data)
                    splitNavigator.loadViewData(data);
            }

            // Event handler to save SplitViewNavigator on application deactivate event.
            protected function onDeactivate(event:Event):void {
                persistenceManager.setProperty("navigatorState", splitNavigator.saveViewData());
                persistenceManager.save();
            }
        ]]>
    </fx:Script>

    <s:SplitViewNavigator id="splitNavigator" width="100%" height="100%">
        <s:ViewNavigator width="256" height="100%"
            firstView="views.MasterCategory"/>
        <s:ViewNavigator width="100%" height="100%"
            firstView="views.DetailView"/>
    </s:SplitViewNavigator>
</s:Application>
```

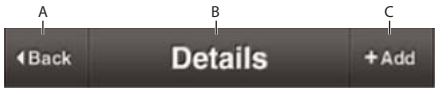
モバイルアプリケーションでのナビゲーションコントロール、タイトルコントロールおよびアクションコントロールの定義

ActionBar コントロールの設定

ViewNavigator コンテナには ActionBar コントロールを定義します。ActionBar コントロールには、タイトル、ナビゲーションコントロールおよびアクションコントロール用の標準的な領域が用意されています。この領域では、アプリケーションの任意の場所または特定のビューでユーザーがアクセスできるグローバルなコントロールを定義できます。例えば、ActionBar コントロールを使用して、「ホーム」ボタンや「検索」ボタン、その他のオプションなどを追加できます。

単一セクション、つまり ViewNavigator コンテナが 1 つのみあるモバイルアプリケーションでは、すべてのビューが同じアクションバーを共有します。複数のセクション、つまり複数の ViewNavigator コンテナがあるモバイルアプリケーションでは、各セクションに独自のアクションバーを定義します。

ActionBar コントロールを使用して、アクションバー領域を定義します。ActionBar コントロールでは、次の図に示すように 3 つの異なる領域を定義します。



A. ナビゲーション領域 **B**. タイトル領域 **C**. アクション領域

ActionBar の領域

- **ナビゲーション領域**

  ユーザーがセクションをナビゲートできるようにするコンポーネントを格納します。例えば、ナビゲーション領域には「ホーム」ボタンを定義できます。

  navigationContent プロパティを使用して、ナビゲーション領域に表示するコンポーネントを定義します。navigationLayout プロパティを使用して、ナビゲーション領域のレイアウトを定義します。

- **タイトル領域**

  タイトルのテキストを指定する文字列またはコンポーネントを格納します。コンポーネントを指定すると、タイトル文字列は指定できません。

  title プロパティを使用して、タイトル領域に表示する文字列を指定します。titleContent プロパティを使用して、タイトル領域に表示するコンポーネントを定義します。titleLayout プロパティを使用して、タイトル領域のレイアウトを定義します。titleContent プロパティに値を指定すると、ActionBar スキンでは title プロパティは無視されます。

- **アクション領域**

  ユーザーがビューで実行できるアクションを定義するコンポーネントを格納します。例えば、アクション領域の一部として「検索」または「更新」ボタンを定義できます。

  actionContent プロパティを使用して、アクション領域に表示するコンポーネントを定義します。actionLayout プロパティを使用して、アクション領域のレイアウトを定義します。

説明したとおりにナビゲーション領域、タイトル領域およびアクション領域を使用することをお勧めしますが、これらの領域に配置するコンポーネントに制限はありません。

ViewNavigatorApplication、ViewNavigator または View コンテナの ActionBar プロパティの設定

ViewNavigatorApplication コンテナ、ViewNavigator コンテナまたは個々の View コンテナには、ActionBar コントロールのコンテンツを定義するプロパティを設定できます。View コンテナが最も優先され、ViewNavigator コンテナと ViewNavigatorApplication コンテナが順に続きます。そのため、ViewNavigatorApplication コンテナに設定したプロパティはアプリケーション全体に適用されますが、ViewNavigator または View コンテナでそれらのプロパティをオーバーライドできます。

注意：ActionBar コントロールは ViewNavigator に関連付けられるので、モバイルアプリケーションの単一セクションごとに固有になります。したがって、TabbedViewNavigatorApplication コンテナや TabbedViewNavigator コンテナから ActionBar を設定することはできません。

例 : アプリケーションレベルでの Spark の ActionBar コントロールのカスタマイズ

次の例は、モバイルアプリケーションのメインアプリケーションファイルを示しています。

```
<?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>
```

この例では、ActionBar コントロールのナビゲーションコンテンツ領域に「ホーム」ボタンを定義し、アクションコンテンツ領域に「更新」ボタンを定義します。

次の例では、アプリケーションの最初のビューを定義する MobileViewHome という View コンテナを定義します。View コンテナでは「Home View」というタイトル文字列を定義します。このタイトル文字列は、ActionBar コントロールのナビゲーションコンテンツ領域またはアクションコンテンツ領域でオーバーライドすることはできません。

```
<?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>
```

例：View コンテナでの ActionBar コントロールのカスタマイズ

この例では、ViewNavigatorApplication コンテナのナビゲーション領域に「ホーム」ボタンを定義する 1 つのセクションがあるメインアプリケーションファイルを使用します。アクション領域に「検索」ボタンも定義します。

```
<?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>
```

このアプリケーションの最初のビューは MobileViewHomeOverride というビューです。MobileViewHomeOverride ビューには、「検索」ページを定義する第 2 の View コンテナにナビゲートする Button コントロールを定義します。

```
<?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>
```

「検索」ページを定義する View コンテナでは、次に示すように、ActionBar コントロールのタイトル領域とアクション領域をオーバーライドします。

```
<?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>
```

次の図は、このビューの ActionBar コントロールを示しています。

「検索」ビューは ActionBar コントロールのナビゲーション領域をオーバーライドしないので、ナビゲーション領域には「ホーム」ボタンが引き続き表示されます。

ActionBar コントロールの非表示化

View.actionBarVisible プロパティを false に設定すると、ビューの ActionBar コントロールを非表示にできます。デフォルトでは、actionBarVisible プロパティは true に設定されており、ActionBar コントロールが表示されます。

次の例に示すように、ViewNavigator によって制御されるすべてのビューの ActionBar コントロールを非表示にするには、ViewNavigator.hideActionbar() メソッドを使用します。

```
<?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>
```

ActionBar に対するカスタムエフェクトは、ActionBar が非表示になるときにも、表示されるときにも定義できます。デフォルトでは、表示または非表示になるときに、アニメーションエフェクトが ActionBar で使用されます。デフォルトのエフェクトを変更するには、ViewNavigator.createActionBarHideEffect() メソッドと ViewNavigator.createActionBarShowEffect() メソッドをオーバーライドします。ActionBar を非表示にするエフェクトを再生した後に、ビューのレイアウトに含まれなくなるように、visible プロパティと includeInLayout プロパティを false に設定します。

モバイルアプリケーションでのスクロールバーの使用

モバイルアプリケーションでスクロールバーを使用する際の考慮事項

通常、スクリーンの表示領域に内容を表示しきれない場合、アプリケーションにはスクロールバーが表示されます。Scroller コントロールを使用して、アプリケーションにスクロールバーを追加します。Spark List コントロールなど、一部のコンポーネントではスクロールがサポートされるため、Scroller コンポーネントを使用する必要はありません。詳しくは、Scrolling Spark containers を参照してください。

スクロールバーのヒット領域とは、マウスを合わせてスクロールするスクリーンの領域のことです。デスクトップまたはブラウザーベースのアプリケーションでは、スクロールバーの表示領域がヒット領域です。モバイルアプリケーションでは、スクリーンの表示領域に内容を表示しきれない場合でも、スクロールバーは非表示です。スクロールバーを非表示にすることで、アプリケーションはスクリーンの幅と高さをすべて使用できます。

モバイルアプリケーションでは、ユーザーが Button コントロールを選択するなどしてコントロールを操作する場合と、スクリーンをスクロールしたい場合を識別する必要があります。モバイルアプリケーションのスクロールバーで考慮が必要な 1 つの点は、ユーザーインタラクションへの応答で Flex コンポーネントの外観が頻繁に変わることです。

例えば、ユーザーが Button コントロールを押すと、そのボタンの外観が変わり、選択されたことが示されます。ユーザーがボタンを離すと、そのボタンの外観は未選択の状態に戻ります。ただし、スクロール時に、ボタンが表示されている画面の上に触れたとしても、ボタンの外観は変更しないようにする必要があります。

アドビの技術者である Steven Shongrunden がスクロールバーの操作例を Saving scroll position between views in a mobile Flex Application に示しています。

スクロールの用語

モバイルアプリケーションでのスクロールについての説明では、次の用語を使用しています。

コンテンツ Group コンテナや List コントロールなどのスクロール可能なコンポーネントにおけるコンポーネント全体の領域。スクリーンサイズやアプリケーションのレイアウトによっては、コンテンツのサブセットしか表示されないことがあります。

ビューポート 現在表示されているコンポーネントのコンテンツ領域のサブセット。

ドラッグ タッチジェスチャーの一種で、ユーザーがスクロール可能な領域にタッチして指を動かし、そのジェスチャーに従ってコンテンツを移動する場合に使用されます。

速度 ドラッグジェスチャーの移動の速度と方向。X 軸と Y 軸に沿ってピクセル / ミリ秒の単位で測定されます。

スロー ドラッグジェスチャーの一種で、ドラッグジェスチャーが一定の速度になったところで、ユーザーが指を離し、スクロール可能なコンテンツをそのまま移動させる場合に使用されます。

バウンス ドラッグまたはスローのジェスチャーでは、スクロール可能なコンポーネントのビューポートをコンポーネントのコンテンツの外側に移動できます。この場合、ビューポートには空の領域が表示されます。指を離すと（つまりスローの速度が 0 になると）、ビューポートはバウンドしながらその停止地点に戻され、ビューポートにコンテンツが表示されます。この移動は、ビューポートが停止地点に近づくにつれてゆっくりした移動になるので、停止はスムーズに行われます。

モバイルアプリケーションでのスクロールモード

List や Scroller などのスクロール可能なコンポーネントでは、コンポーネントの pageScrollingEnabled および scrollSnappingMode プロパティの設定に基づいて、様々なタイプのスクロールがサポートされます。これらのプロパティは、interactionMode スタイルが touch に設定されている場合にのみ有効です。

次の表で、スクロールモードについて説明します。

pageScrollingEnabled	scrollSnappingMode	モード
false（デフォルト）	none（デフォルト）	デフォルトでは、スクロールはピクセルベースです。最終スクロール位置は、ドラッグまたはスローのジェスチャーに基づいたピクセル位置です。例えば、List コントロールをスクロールしたとします。指を離すと、List アイテムが欠けた状態で表示されていても、スクロールは終了します。

pageScrollingEnabled	scrollSnappingMode	モード
false	leadingEdge、center、trailingEdge	スクロールはピクセルベースですが、scrollSnappingMode の値に基づいて、コンテンツは最終位置にスナップされます。 例えば、scrollSnappingMode に leadingEdge の値が設定されている List を垂直方向にスクロールしたとします。List コントロールは、先頭のリストエレメントとリストの先頭が同じ位置になる最終スクロール位置にスナップされます。
true	none	スクロールはページベースです。スクロール可能なコンポーネントのビューポートのサイズによって、ページのサイズが決まります。ジェスチャーに関係なく、一度に 1 ページのみスクロールできます。 コンポーネントの表示領域の 50 %以上をスクロールすると、ページは次のページにスクロールされます。50 %未満をスクロールした場合は、コンポーネントは現在のページのまま変わりません。あるいは、スクロールの速度が十分に速い場合は、次のページが表示されます。速度が十分に速くない場合は、コンポーネントは現在のページのまま変わりません。 コンテンツサイズがビューポートサイズの倍数に一致しない場合、ビューポートにリストがぴったり収まるように、最後のページに追加のパディングが追加されます。
true	leadingEdge、center、trailingEdge	スクロールはページベースですが、scrollSnappingMode の値に基づいて、コンポーネントは最終位置にスナップされます。スナップモードが必ず遵守されるようにするため、必ずしもスクロールの間隔はページのサイズと完全に一致するとは限りません。

モバイルアプリケーションでのスクロールの例

次の例では、モバイルアプリケーションで Scroller コンポーネントを使用して、Group コンテナをラップします。Group コンテナは、大きな画像が含まれる Image コントロールを子として持ちます。Group コンテナを Scroller でラップして、画像をスクロールできるようにします。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SparkMobilePixelScrollerHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">
    <s:Scroller width="200" height="200">
        <s:Group>
            <s:Image width="300" height="400"
                source="@Embed(source='../assets/logo.jpg')"/>
        </s:Group>
    </s:Scroller>
</s:View>
```

この例では、pageScrollingEnabled および scrollSnappingMode プロパティのすべての設定を省略していることに注意してください。したがって、この例では、デフォルトのピクセルスクロールモードが使用されるので、画像内の任意のピクセル位置にスクロールできます。

次の例は、pageScrollingEnabled および scrollSnappingMode プロパティを設定する List コントロールを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SparkMobilePageScrollHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Adobe Product List">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;

            protected function myList_changeHandler(event:IndexChangeEvent):void {
                navigator.pushView(views.ProductPricelView,myList.selectedItem);
            }

        ]]>
    </fx:Script>

    <s:List id="myList" labelField="Product"
        height="200" width="100%"
        borderVisible="true"
        scrollSnappingMode="leadingEdge"
        pageScrollingEnabled="true"
        change="myList_changeHandler(event);">
        <s:dataProvider>
            <s:ArrayCollection>
                <fx:Object Product="Adobe AIR" Price="11.99"/>
                <fx:Object Product="Adobe BlazeDS" Price="11.99"/>
                <fx:Object Product="Adobe ColdFusion" Price="11.99"/>
                <fx:Object Product="Adobe Flash Player" Price="11.99"/>
                <fx:Object Product="Adobe Flex" Price="Free"/>
                <fx:Object Product="Adobe LiveCycleDS" Price="11.99"/>
                <fx:Object Product="Adobe LiveCycle ES2" Price="11.99"/>
                <fx:Object Product="Open Source Media Framework"/>
                <fx:Object Product="Adobe Photoshop" Price="11.99"/>
                <fx:Object Product="Adobe Illustrator" Price="11.99"/>
                <fx:Object Product="Adobe Reader" Price="11.99"/>
                <fx:Object Product="Adobe Acrobat" Price="11.99"/>
                <fx:Object Product="Adobe InDesign" Price="Free"/>
                <fx:Object Product="Adobe Connect" Price="11.99"/>
                <fx:Object Product="Adobe Dreamweaver" Price="11.99"/>
                <fx:Object Product="Open Framemaker"/>
            </s:ArrayCollection>
        </s:dataProvider>
    </s:List>
</s:View>
```

この例では、スナップ設定が leadingEdge のページスクロールを使用します。そのため、List をスクロールすると、一度に 1 ページずつスクロールできます。ページが変更されると、コントロールは、先頭のリストエレメントとリストの先頭が同じ位置になる最終スクロール位置にスナップされます。

StageText でのスクロールの考慮事項

StageText では、標準のテキストフィールドコントロールではなく、ネイティブテキスト入力をモバイルアプリケーションで使用できます。ただし、スクロール可能なコンテナは、StageText を使用する TextInput コントロールや TextArea コントロールなどのテキスト入力コントロールを保持できません。そのため、スクロール可能なコンテナでテキスト入力コントロールを使用するには、StageText を使用しないようにコントロールのスキンを再設定します。

Flex には、StageText に依存しない TextInput コントロールおよび TextArea コントロールが用意されています。スクロール可能なコンテナでは、これらのコントロールが設定された次のスキンを使用します。

- spark.skins.mobile.TextInputSkin スキン（StageText を使用しない TextInput）。
- spark.skins.mobile.TextAreaSkin スキン（StageText を使用しない TextArea）。

次に、スクロール可能なコンテナで TextInput コントロールおよび TextArea コントロールを使用する View コンテナの例を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMobileStageTextScrollHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <!-- Create CSS class selectors that reference the skins
         that do not rely on StageText. -->
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        .myTextInputStyle {
            skinClass: ClassReference("spark.skins.mobile.TextInputSkin");
        }
        .myTextAreaStyle {
            skinClass: ClassReference("spark.skins.mobile.TextAreaSkin");
        }
    </fx:Style>

    <!-- Apply the class selectors to the TextInput and TextArea controls. -->
    <s:Scroller width="100%" height="100%">
        <s:VGroup height="250" width="100%"
                  paddingTop="10" paddingLeft="5" paddingRight="10">
            <s:HGroup verticalAlign="middle">
                <s:Label text="Text Input 1: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
                <s:Label text="Text Input 2: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
```

```
                <s:Label text="Text Input 3: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
                <s:Label text="Text Input 4: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
                <s:Label text="TextArea 1: "
                    fontWeight="bold"/>
                <s:TextArea width="225" height="100"
                    styleName="myTextAreaStyle "/>
            </s:HGroup>
        </s:VGroup>
    </s:Scroller>
</s:View>
```

イベントとスクロールバー

Flex コンポーネントは、イベントを使用してユーザーインタラクションが発生したことを示します。次に、ユーザーインタラクションに対して、コンポーネントの外観を変更するか、その他のアクションを実行できます。

アプリケーション開発者は、イベントを使用してユーザーインタラクションを処理します。例えば、ユーザーがボタンを選択する操作に対しては、Button コントロールの click イベントを使用してイベントハンドラーを実行するのが一般的です。また、ユーザーがリストからアイテムを選択する場合は、List コントロールの change イベントを使用してイベントハンドラーを実行します。

Flex のスクロールメカニズムは、mouseDown イベントに依存しています。つまり、スクロールメカニズムは、mouseDown イベントをリスニングして、スクロール操作を開始するかどうかを判断します。

スクロール操作としてのユーザージェスチャーの解釈

スクロール可能なコンテナに複数の Button コントロールが含まれるアプリケーションがあるとします。

1. ユーザーが指で Button コントロールを押します。ボタンは mouseDown イベントを送出します。
2. Flex では、定義済みの時間だけ、ユーザーインタラクションへの応答が遅延されます。遅延時間によって、ユーザーがスクリーンをスクロールしようとしたのではなく、ボタンを選択したことが確定します。

   遅延時間内に、ユーザーが定義済みの長さを超えて指を移動した場合、Flex ではそのジェスチャーがスクロールアクションと解釈されます。スクロールと解釈されるには、ユーザーは指を約 0.08 インチ移動する必要があります。この距離は、252 DPI のデバイスでは約 20 ピクセルになります。

   遅延時間を超える前に指を移動したので、Button コントロールではインタラクションが認識されません。そのボタンはイベントを送出せず、外観も変わりません。
3. 遅延時間を超えると、Button コントロールでユーザーインタラクションが認識されます。そのボタンの外観が変わり、選択されたことが示されます。

   遅延する時間を設定するには、コントロールの touchDelay プロパティを使用します。デフォルト値は 100 ミリ秒です。touchDelay プロパティを 0 に設定すると、遅延は発生せず、すぐにスクロールが開始されます。
4. 遅延時間を超え、Flex でマウスイベントが送出された後に、ユーザーが 20 ピクセルを超えて指を移動したとします。Button コントロールは通常の状態に戻り、スクロールアクションが開始されます。

この場合、遅延時間を超えているので、ボタンの外観は変わっています。ただし、遅延時間を超えた後でも、ユーザーが 20 ピクセルを超えて指を移動すると、Flex ではそのジェスチャーがスクロールアクションと解釈されます。

注意：Flex コンポーネントでは、マウスイベント以外に多数の各種イベントがサポートされています。コンポーネントを使用する場合は、これらのイベントに対してアプリケーションがどのように応答するかを決定します。mouseDown イベントが発生した場合、ユーザーが意図する動作は不明確です。コンポーネントを操作しようとしている場合もあれば、スクロールしようとしている場合もあります。mouseDown イベントにはこのような不明確さがあるので、代わりに、click イベントまたは mouseUp イベントをリスニングすることをお勧めします。

モバイルアプリケーションでのスクロールイベントの処理

スクロール操作の開始を通知するために、mouseDown イベントを送出するコンポーネントは、touchInteractionStarting というバブリングイベントを送出します。そのイベントがキャンセルされない場合、コンポーネントは touchInteractionStart というバブリングイベントを送出します。

コンポーネントが touchInteractionStart イベントを検出した場合、そのコンポーネントがユーザージェスチャーに応答しないようにする必要があります。例えば、Button コントロールが touchInteractionStart イベントを検出した場合は、最初の mouseDown イベントに基づいて設定したビジュアルインジケーターをオフにします。

コンポーネントがスクロールの開始を許可しない場合、コンポーネントは touchInteractionStarting イベントのイベントハンドラーの preventDefault() メソッドを呼び出すことができます。

スクロール操作が完了したら、mouseDown イベントを送出したコンポーネントが、touchInteractionEnd というバブリングイベントを送出します。

最初のタッチポイントに基づくスクロール動作

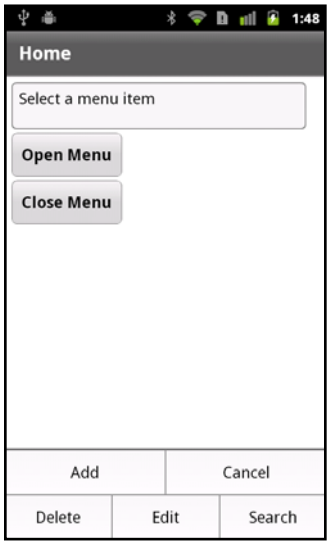
次の表で、最初のタッチポイントの場所に基づいてスクロールを処理する方法について説明します。

選択したアイテム	動作
空白、 編集不可テキスト、 選択不可テキスト	どのコンポーネントもジェスチャーを認識しません。Scroller は、ユーザーが 20 ピクセルを超えてタッチポイントを移動するのを待ってから、スクロールを起動します。
List コントロール内のアイテム	遅延時間を超えると、選択したアイテムのアイテムレンダラーにより、選択された状態に表示が変わります。ただし、ユーザーが 20 ピクセルを超えて指を移動した場合はいつでも、アイテムの外観が通常の状態に変わり、スクロールが開始されます。
Button、 CheckBox、 RadioButton、 DropDownList	遅延時間を超えると、mouseDown ステートが表示されます。ただし、ユーザーが 20 ピクセルを超えてタッチポイントを移動すると、アイテムの外観が通常の状態に変わり、スクロールが開始されます。
List アイテムレンダラー内の Button コンポーネント	アイテムレンダラーではコントロールを強調表示しません。Button や Scroller は、通常の Button の場合と同様にジェスチャーを処理します。

モバイルアプリケーションでのメニューの定義

ViewMenu コンテナは、モバイルアプリケーションの View コンテナの下部にあるメニューを定義します。各 View コンテナは、そのビュー固有のメニューを定義します。

次の図は、アプリケーションの ViewMenu コンテナを示しています。



ViewMenu コンテナでは、単一階層のメニューボタンで構成されるメニューを定義します。つまり、サブメニューがあるメニューは作成できません。

ViewMenu コンテナの子は ViewMenuItem コントロールとして定義されます。各 ViewMenuItem コントロールは、メニュー内の 1 つのボタンを表します。

ViewMenu コンテナでのユーザーインタラクション

モバイルデバイスのハードウェアメニューキーを使用して、メニューを開きます。プログラムで開くこともできます。

1 つのメニューボタンを選択すると、メニュー全体が閉じます。ユーザーがメニューボタンを選択すると、ViewMenuItem コントロールは click イベントを送出します。

メニューが開いているときに、デバイスの戻るボタンまたはメニューボタンを押すとメニューが閉じます。スクリーンのメニュー以外の場所を押してもメニューが閉じます。

キャレットとは、現在フォーカスされているメニューボタンです。キャレットを変更するには、デバイスの 5 種類のコントロールまたは矢印キーを使用します。デバイスの Enter キーまたは 5 種類のコントロールを押してキャレット項目を選択し、メニューを閉じます。

モバイルアプリケーションでのメニューの作成

View.viewMenuItems プロパティを使用して、ビューのメニューを定義します。次の例に示すように、View.viewMenuItems プロパティには ViewMenuItem コントロールの Vector を指定します。

```
<?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>
```

この例では、View.viewMenuItems プロパティを使用して 5 つのメニュー項目を追加します。各メニュー項目は 1 つの ViewMenuItem コントロールで表されています。各 ViewMenuItem コントロールで、label プロパティを使用して、その項目のメニューに表示されるテキストを指定します。

ViewMenu コンテナは明示的に定義しないことに注意してください。View コンテナでは、ViewMenu コンテナのインスタンスが自動的に作成されて、ViewMenuItem コントロールが格納されます。

ViewMenuItem コントロールの icon **スタイルの使用**

ViewMenuItem コントロールでは、画像を組み込むのに使用できる icon スタイルプロパティを定義します。icon スタイルは、label プロパティとともに使用することもできます。

ViewMenuItem コントロールの click **イベントの処理**

各 ViewMenuItem コントロールでは、click イベントのイベントハンドラーも定義します。ユーザーが項目を選択すると、ViewMenuItem コントロールは click イベントを送出します。前述の例では、すべてのメニュー項目で同じイベントハンドラーを使用しています。ただし、click イベントごとに異なるイベントハンドラーを定義することもできます。

ViewMenuItem コントロールのプログラムでのオープン

デバイスのハードウェアメニューキーを使用して、メニューを開きます。さらに、このアプリケーションでは、メニューをプログラムで開いたり閉じたりするために、2 つの Button コントロールを定義します。

メニューをプログラムで開くには、アプリケーションコンテナの viewMenuOpen プロパティを true に設定します。メニューを閉じるには、このプロパティを false に設定します。viewMenuOpen プロパティは ViewNavigatorApplicationBase クラスに定義されます。このクラスは、ViewNavigatorApplication および TabbedViewNavigatorApplication コンテナの基本クラスです。

ViewMenu および ViewMenuItem コンポーネントへのスキンの適用

スキンを使用して、ViewMenu および ViewMenuItem コンポーネントの外観を制御します。ViewMenu のデフォルトのスキンクラスは spark.skins.mobile.ViewMenuSkin です。ViewMenuItem のデフォルトのスキンクラスは spark.skins.mobile.ViewMenuItemSkin です。

ブロガーの Daniel Demmel 氏が、ViewMenu コントロールを Gingerbread ブラックのようなスキンにする方法に関するブログを公開しています。

スキンクラスでは、スキンのステート（normal、closed、disabled など）を使用してスキンの外観を制御します。また、スキンでトランジションを定義して、ビューステートが変更されたときのメニューの外観を制御します。

詳しくは、154 ページの「モバイルのスキン適用の基本」を参照してください。

ViewMenu コンテナのレイアウトの設定

ViewMenuLayout クラスでは、ビューメニューのレイアウトを定義します。メニュー項目の数に応じて、メニューを複数の行で構成できます。

ViewMenuItem のレイアウト規則

ViewMenuLayout クラスの requestedMaxColumnCount プロパティで、1 行に表示するメニュー項目の最大数を定義します。デフォルトでは、requestedMaxColumnCount プロパティは 3 に設定されます。

次の規則に従って、ViewMenuLayout クラスがレイアウトを実行する方法を定義します。

- requestedMaxColumnCount プロパティがデフォルト値の 3 に設定され、定義するメニュー項目が 3 つ以下の場合、すべてのメニュー項目は 1 行に表示されます。各メニュー項目は同じサイズで表示されます。

  定義するメニュー項目が 4 つ以上の場合、つまり、メニュー項目数が requestedMaxColumnCount プロパティで指定した数より多い場合、ViewMenu コンテナでは複数の行が作成されます。

- メニュー項目の数が requestedMaxColumnCount プロパティの値で等分できる場合、各行には同じ数のメニュー項目が表示されます。各メニュー項目は同じサイズで表示されます。

  例えば、requestedMaxColumnCount プロパティがデフォルト値の 3 に設定され、6 つのメニュー項目を定義するとします。この場合、メニューは 2 行に表示され、各行に 3 つのメニュー項目が表示されます。

- メニュー項目の数が requestedMaxColumnCount プロパティの値で等分できない場合、各行には異なる数のメニュー項目が表示されます。メニュー項目のサイズは、その行に表示されるメニュー項目の数によって決まります。

  例えば、requestedMaxColumnCount プロパティがデフォルト値の 3 に設定され、8 つのメニュー項目を定義するとします。この場合、メニューは 3 行に表示されます。最初の行には 2 つのメニュー項目が表示されます。2 行目と 3 行目には、それぞれ 3 つのメニュー項目が表示されます。

カスタム ViewMenuItem レイアウトの作成

ViewMenuLayout クラスには、メニュー項目間の間隔、および各行に表示するメニュー項目のデフォルト数を変更できるプロパティが含まれています。独自のレイアウトクラスを作成することで、メニューのカスタムレイアウトを作成することもできます。

デフォルトでは、spark.skins.mobile.ViewMenuSkin クラスで ViewMenu コンテナのスキンを定義します。カスタマイズした ViewMenuLayout クラスを ViewMenu コンテナに適用するには、ViewMenu コンテナの新規のスキンクラスを定義します。

デフォルトの ViewMenuSkin クラスには、次の例に示すように、contentGroup という名前の Group コンテナの定義が含まれています。

```
...
    <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>
...
```

独自のスキンクラスを定義する場合も、contentGroup という名前のコンテナを定義する必要があります。このコンテナで layout プロパティを使用して、カスタマイズしたレイアウトクラスを指定します。

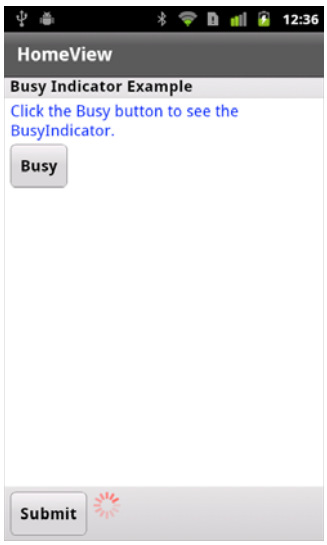
次の例に示すように、カスタムスキンクラスをアプリケーションに適用できます。

```
<?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>
```

モバイルアプリケーションの長時間かかるアクティビティに対するビジーインジケーターの表示

Spark の BusyIndicator コントロールは、12 個のスポークが付いた回転するスピナーを表示します。BusyIndicator コントロールを使用すると、長時間かかる操作が進行中であることを視覚的に示すことができます。

次の図は、「Submit」ボタンの横にある Spark の Panel コンテナのコントロールバー領域の BusyIndicator コントロールを示しています。



長時間かかる操作が進行している間、BusyIndicator コントロールが表示されるようにします。操作が完了したら、このコントロールを非表示にします。

例えば、イベントハンドラーに BusyIndicator コントロールのインスタンスを作成し、そのイベントハンドラーが長時間かかる操作を開始するとします。イベントハンドラーでは、addElement() メソッドを呼び出してこのコントロールをコンテナに追加します。処理が完了したら、removeElement() を呼び出して BusyIndicator コントロールをコンテナから削除します。

また、このコントロールの visible プロパティを使用して、コントロールの表示と非表示を切り替えることもできます。次の例では、View コンテナ内で Spark の Panel コンテナのコントロールバー領域に BusyIndicator コントロールを追加します。

```
<?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>
```

この例では、BusyIndicator コントロールの visible プロパティが最初に false に設定されているので、コントロールは非表示です。Busy ボタンをクリックすると、visible プロパティが true に設定されて、コントロールが表示されます。

BusyIndicator コントロールは、表示されている場合のみ回転します。したがって、visible プロパティを false に設定すると、コントロールで処理サイクルは実行されません。

注意：visible プロパティを false に設定した場合、コントロールは非表示になりますが、コントロール自体は親コンテナのレイアウトに含まれたままです。レイアウトからこのコントロールを除外するには、visible プロパティと includeInLayout プロパティを false に設定します。

Spark の BusyIndicator コントロールでは、スキンがサポートされていません。ただし、スタイルを使用して、スピナーの色と回転間隔を設定できます。前述の例では、symbolColor プロパティを使用して、このインジケーターの色を設定しています。

モバイルアプリケーションへのトグルスイッチの追加

Spark ToggleSwitch コントロールでは、単純なバイナリスイッチを定義します。このコントロールは、サムと、サムをスライドするトラックで構成されます。

ToggleSwitch コントロールは、ToggleButton や CheckBox コントロールに似ています。これらのすべてのコントロールでは、選択と選択解除のいずれかの値を選択できます。

次の図は、アプリケーションの ToggleSwitch コントロールを示しています。

ToggleSwitch コントロールには、選択と選択解除の 2 つの位置があります。サムを左に移動すると、コントロールは選択解除の位置になります。サムを右に移動すると、選択の位置になります。図では、スイッチは選択解除の位置にあります。

コントロール内の任意の場所をクリックすると、位置が切り替わります。トラックに沿ってサムをスライドして、位置を変更することもできます。サムを離すと、サムの位置から近い方の位置（選択または選択解除）にサムが移動します。

デフォルトでは、ラベルの「OFF」が選択解除の位置に対応し、「ON」が選択の位置に対応します。

ToggleSwitch コントロールの作成

次に、前述の図に示した ToggleSwitch コントロールを定義する View コンテナを示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\ToggleSwitchSimpleHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingTop="10" paddingLeft="5"/>
    </s:layout>
    <s:ToggleSwitch id="ts"
        slideDuration="1000"/>
    <s:Form>
        <s:FormItem label="Toggle Label: ">
            <s:Label text="{ts.selected ? 'ON' : 'OFF'}"/>
        </s:FormItem>
        <s:FormItem label="Toggle Position: ">
            <s:Label text="{ts.thumbPosition}"/>
        </s:FormItem>
    </s:Form>
</s:View>
```

この例では、サムの位置に基づいて、最初の Label コントロールに「ON」または「OFF」を表示します。2 つ目の Label コントロールでは、現在のサムの位置を 0.0（選択解除）から 1.0（選択）の間の値で表示します。

また、この例では、slideDuration スタイルに 1000 を設定します。このスタイルでは、選択と選択解除の位置の間でサムがスライドされたときのサムのアニメーションの継続時間（ミリ秒）を決定します。

次に、メインアプリケーションファイルを示します。

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

ToggleSwitch コントロールのデフォルトコールアウトの変更

前述の例では、ToggleSwitch コントロールで、選択解除と選択のラベルのデフォルト値である「OFF」（選択解除）と「ON」（選択）を使用しています。コントロールのラベルやその他の視覚的な特性をカスタマイズするには、spark.skins.mobile.ToggleSwitchSkin のサブクラスとしてスキンクラスを定義するか、独自のスキンクラスを作成します。

次のスキンクラスでは、ラベルを「Yes」と「No」に変更します。

```
package skins
// components\mobile\skins\MyToggleSwitchSkin.as
{
    import spark.skins.mobile.ToggleSwitchSkin;

    public class MyToggleSwitchSkin extends ToggleSwitchSkin
    {
        public function MyToggleSwitchSkin()
        {
            super();
            // Set properties to define the labels
            // for the selected and unselected positions.
            selectedLabel="Yes";
            unselectedLabel="No";
        }
    }
}
```

次の View コンテナでは、このスキンクラスを使用します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\ToggleSwitchSkinHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingTop="10" paddingLeft="5"/>
    </s:layout>

    <s:ToggleSwitch id="ts"
        slideDuration="1000"
        skinClass="skins.MyToggleSwitchSkin"/>

    <s:Form>
        <s:FormItem label="Toggle Label: ">
            <s:Label text="{ts.selected ? 'Yes' : 'No'}"/>
        </s:FormItem>
        <s:FormItem label="Toggle Position: ">
            <s:Label text="{ts.thumbPosition}"/>
        </s:FormItem>
    </s:Form>
</s:View>
```

モバイルアプリケーションへの Callout コンテナの追加

モバイルアプリケーションでは、Callout は、アプリケーションの最前面にポップアップされるコンテナです。このコンテナは 1 つ以上のコンポーネントを保持することができ、様々な種類のレイアウトをサポートします。

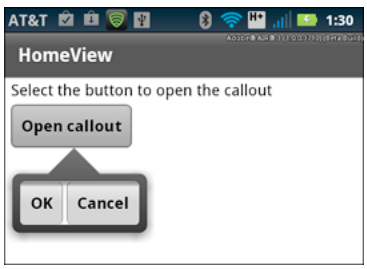
Callout コンテナには、モーダルと非モーダルの設定があります。モーダルコンテナは、コンテナが閉じられるまで、このコンテナがキーボードとマウスのすべての入力を受け取ります。非モーダルコンテナを使用した場合は、コンテナが開いている間も、アプリケーションの他のコンポーネントが入力を受け取ることができます。

Flex では、CalloutButton と Callout の 2 つのコンポーネントが用意されており、これらを使用して、モバイルアプリケーションに Callout コンテナを追加できます。

CalloutButton コントロールを使用した Callout コンテナの作成

CalloutButton コントロールでは、Callout コンテナを作成する簡単な方法が提供されます。このコンポーネントを使用すると、コールアウト（吹き出し）の中に表示されるコンポーネントを定義して、コンテナのレイアウトを設定できます。

ユーザーがモバイルアプリケーションで CalloutButton コントロールを選択すると、このコントロールによって Callout コンテナが開かれます。Flex では、次の図に示すように、Callout コンテナから CalloutButton コントロールへの矢印が自動的に描画されます。



次の例は、前述の図に示した CalloutButton を作成するモバイルアプリケーションを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutButtonSimpleHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <s:Label text="Select the button to open the callout"/>

    <s:CalloutButton id="myCB"
        horizontalPosition="end"
        verticalPosition="after"
        label="Open callout">
        <s:calloutLayout>
            <s:HorizontalLayout/>
        </s:calloutLayout>

        <!-- Define buttons that appear in the callout. -->
        <s:Button label="OK"
            click="myCB.closeDropDown();"/>
        <s:Button label="Cancel"
            click="myCB.closeDropDown();"/>
    </s:CalloutButton>
</s:View>
```

CalloutButton コントロールで、Callout コンテナの中に表示される 2 つの Button コントロールを定義します。また、CalloutButton コントロールで、Callout コンテナのレイアウトとして HorizontalLayout を使用するように指定します。デフォルトでは、コンテナは BasicLayout を使用します。

CalloutButton でのコールアウトコンテナのオープンとクローズ

Callout コンテナは、ユーザーが CalloutButton コントロールを選択するか、CalloutButton.openDropDown() メソッドが呼び出されると開きます。horizontalPosition および verticalPosition プロパティでは、CalloutButton コントロールを基準にして、Callout コンテナの位置を決定します。例については、85 ページの「Callout コンテナのサイズと位置の指定」を参照してください。

CalloutButton で開かれる Callout コンテナは、常に非モーダルです。つまり、コールアウトが開いている間も、アプリケーションの他のコンポーネントは入力を受け取ることができます。モーダルなコールアウトを作成するには、Callout コンテナを使用します。

この Callout コンテナは、Callout コンテナの外をクリックするか、CalloutButton.closeDropDown() メソッドが呼び出されるまで、開いた状態が続きます。次の例では、Callout コンテナ内の 2 つの Button コントロールの click イベントのイベントハンドラーで、closeDropDown() メソッドを呼び出します。

Callout コンテナを使用したコールアウトの作成

CalloutButton コントロールは、Callout コンテナと、コールアウトを開いたり閉じたりするために必要なすべてのロジックを 1 つのコントロールにカプセル化します。この場合、CalloutButton コントロールは、Callout コンテナのホストと呼ばれます。

また、モバイルアプリケーションでは Callout コンテナも使用できます。Callout コンテナの利点は、単一のホストに関連付けられないので、アプリケーション内のどの場所からでも再利用できることです。

Callout コンテナを開くには、Callout.open() メソッドと Callout.close() メソッドを使用します。通常は、イベントに応答する形で使用します。open() メソッドを呼び出すときに、オプションの引数を渡して、Callout コンテナをモーダルにするように指定できます。デフォルトでは、Callout コンテナは非モーダルです。

Callout コンテナの位置は、ホストコンポーネントが基準になります。horizontalPosition および verticalPosition プロパティで、ホストを基準にしたコンテナの位置を決定します。例については、85 ページの「Callout コンテナのサイズと位置の指定」を参照してください。

Callout コンテナはポップアップであるので、アプリケーションの通常の MXML レイアウトコードの一部としては作成しません。その代わりに、Callout コンテナは、MXML ファイルでカスタム MXML コンポーネントとして定義します。

次の例では、アプリケーションの comps ディレクトリ内の MyCallout.mxml ファイルで、Callout コンテナを定義します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\comps\MyCallout.mxml -->
<s:Callout xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    horizontalPosition="start"
    verticalPosition="after">

    <s:VGroup
        paddingTop="10" paddingLeft="5" paddingRight="10">

        <s:HGroup verticalAlign="middle">
            <s:Label text="First Name: "
                fontWeight="bold"/>
            <s:TextInput width="225"/>
        </s:HGroup>

        <s:HGroup verticalAlign="middle">
            <s:Label text="Last Name: "
                fontWeight="bold"/>
            <s:TextInput width="225"/>
        </s:HGroup>

        <s:HGroup>
            <s:Button label="OK" click="close();"/>
            <s:Button label="Cancel" click="close();"/>
        </s:HGroup>
    </s:VGroup>
</s:Callout>
```

MyCallout.mxml では、ユーザーが姓と名を入力できる簡単なポップアップを定義しています。ボタンで、click イベントに応答して close() メソッドを呼び出し、コールアウトを閉じていることに注目してください。

次の例は、click イベントに応答して MyCallout.mxml を開く View コンテナを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutSimpleHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import comps.MyCallout;

            // Event handler to open the Callout component.
            protected function button1_clickHandler(event:MouseEvent):void {
                var myCallout:MyCallout = new MyCallout();
                // Open as a modal callout.
                myCallout.open(calloutB, true);
            }
        ]]>
    </fx:Script>

    <s:Label text="Select the button to open the callout"/>
    <s:Button id="calloutB"
        label="Open Callout container"
        click="button1_clickHandler(event);"/>
</s:View>
```

最初に、アプリケーションに MyCallout.mxml コンポーネントを読み込みます。click イベントに応答して、calloutB という名前のボタンが、MyCallout.mxml のインスタンスを作成し、open() メソッドを呼び出します。

open() メソッドでは、2 つの引数を指定します。最初の引数は、calloutB がコールアウトのホストコンポーネントであることを指定しています。その結果、コールアウトは、calloutB の位置を基準にしてアプリケーションに配置されます。2 番目の引数は true で、モーダルなコールアウトを作成しています。

インライン Callout コンテナの定義

Callout コンテナは、必ずしも別のファイルに定義する必要はありません。次の例では、<fx:Declaration> タグを使用して、View コンテナのインラインコンポーネントとして定義します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutInlineHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            // Event handler to open the Callout component.
            protected function button1_clickHandler(event:MouseEvent):void {
                var myCallout:MyCallout = new MyCallout();
                // Open as a modal callout.
                myCallout.open(calloutB, true);
            }
        ]]>
    </fx:Script>
```

```
    <fx:Declarations>
        <fx:Component className="MyCallout">
            <s:Callout
                horizontalPosition="end"
                verticalPosition="after">
                <s:VGroup
                    paddingTop="10" paddingLeft="5" paddingRight="10">
                    <s:HGroup verticalAlign="middle">
                        <s:Label text="First Name: "
                                 fontWeight="bold"/>
                        <s:TextInput width="225"/>
                    </s:HGroup>
                    <s:HGroup verticalAlign="middle">
                        <s:Label text="Last Name: "
                                 fontWeight="bold"/>
                        <s:TextInput width="225"/>
                    </s:HGroup>
                    <s:HGroup>
                        <s:Button label="OK" click="close();"/>
                        <s:Button label="Cancel" click="close();"/>
                    </s:HGroup>
                </s:VGroup>
            </s:Callout>
        </fx:Component>
    </fx:Declarations>

    <s:Label text="Select the button to open the callout"/>
    <s:Button id="calloutB"
        label="Open Callout container"
        click="button1_clickHandler(event);"/>
</s:View>
```

Callout コンテナからのデータの引き渡し

Callout コンテナの close() メソッドを使用して、メインアプリケーションにデータを返します。close() メソッドには次のシグニチャがあります。

```
public function close(commit:Boolean = false, data:*):void
```

ここで、各項目の説明は次のとおりです。

- commit は、返されたデータをアプリケーションがコミットする必要がある場合は true になります。
- data には、返されるデータを指定します。

close() メソッドを呼び出すと、close イベントが送出されます。close イベントに関連するイベントオブジェクトは、タイプ spark.events.PopUpEvent のオブジェクトです。PopUpEvent クラスでは、commit と data の 2 つのプロパティが定義され、これらのプロパティには close() メソッドに対応する引数の値が含まれます。close イベントのイベントハンドラーで、これらのプロパティを使用して、コールアウトから返されたデータを調べます。

Callout コンテナは、SkinnablePopUpContainer クラスのサブクラスです。SkinnablePopUpContainer クラスでも、メインアプリケーションにデータを返すために同じメカニズムを使用します。SkinnablePopUpContainer コンテナからデータを返す例については、Passing data back from the Spark SkinnablePopUpContainer container を参照してください。

次の例では、前述した Callout コンポーネントを変更して、姓と名の値が返されるようにします。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\comps\MyCalloutPassBack.mxml -->
<s:Callout xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    horizontalPosition="start"
    verticalPosition="after">

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;

            public var retData:String = new String();

            // Event handler for the click event of the OK button.
            protected function clickHandler(event:MouseEvent):void {
                //Create the return data.
                retData = firstName.text + " " + lastName.text;
                // Close the Callout.
                // Set the commit argument to true to indicate that the
                // data argument contains a valid value.
                close(true, retData);
            }

        ]]>
    </fx:Script>

    <s:VGroup
        paddingTop="10" paddingLeft="5" paddingRight="10">
        <s:HGroup verticalAlign="middle">
            <s:Label text="First Name: "
                     fontWeight="bold"/>
            <s:TextInput id="firstName" width="225"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Last Name: "
                     fontWeight="bold"/>
            <s:TextInput id="lastName" width="225"/>
        </s:HGroup>
        <s:HGroup>
            <s:Button label="OK" click="clickHandler(event);"/>
            <s:Button label="Cancel" click="close();"/>
        </s:HGroup>
    </s:VGroup>
</s:Callout>
```

この例では、ユーザーの「OK」の選択に応答して、String を作成して姓と名を返します。

続いて、View コンテナは Callout の close イベントを使用して、返されたデータを表示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutPassBackDataHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import comps.MyCalloutPassBack;
            import spark.events.PopUpEvent;

            public var myCallout:MyCalloutPassBack = new MyCalloutPassBack();

            // Event handler to open the Callout component.
            protected function clickHandler(event:MouseEvent):void {
                // Add an event handler for the close event to check for
                // any returned data.
                myCallout.addEventListener('close', closeHandler);
                // Open as a modal callout.
                myCallout.open(calloutB, true);
            }

            // Handle the close event from the Callout.
            protected function closeHandler(event:PopUpEvent):void {
                // If commit is false, no data is returned.
                if (!event.commit)
                    return;

                // Write the returned Data to the TextArea control.
                myTA.text = String(event.data);

                // Remove the event handler.
                myCallout.removeEventListener('close', closeHandler);
            }

        ]]>
    </fx:Script>

    <s:Label text="Select the button to open the callout"/>
    <s:Button id="calloutB"
        label="Open Callout container"
        click="clickHandler(event);"/>
    <s:TextArea id="myTA"/>
</s:View>
```

Callout への ViewNavigator の追加

Callout コンテナ内で ViewNavigator を使用することができます。ViewNavigator を使用すると、コールアウトにアクションバーと複数のビューを追加できます。

例えば、次の View は、MyCalloutPassBackVN ファイルで定義されている Callout コンテナを開きます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutPassBackDataHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import comps.MyCalloutPassBackVN;
            import spark.events.PopUpEvent;

            public var myCallout:MyCalloutPassBackVN = new MyCalloutPassBackVN();

            // Event handler to open the Callout component.
            protected function clickHandler(event:MouseEvent):void {
                myCallout.addEventListener('close', closeHandler);
                myCallout.open(calloutB, true);
            }

            // Handle the close event from the Callout.
            protected function closeHandler(event:PopUpEvent):void {
                if (!event.commit)
                    return;

                myTA.text = String(event.data);
                myCallout.removeEventListener('close', closeHandler);
            }
        ]]>
    </fx:Script>

    <s:Label text="Select the Open button to open the callout"/>
    <s:TextArea id="myTA"/>
    <s:actionContent>
        <s:Button id="calloutB" label="Open"
            click="clickHandler(event);"/>
    </s:actionContent>
</s:View>
```

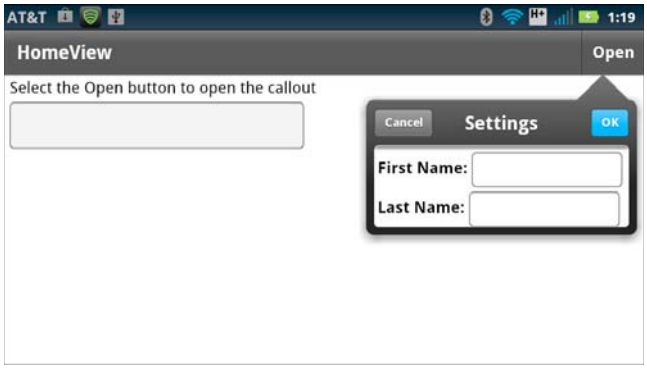
MyCalloutPassBackVN.mxml ファイルで、ViewNavigator コンテナを保持する Callout コンテナを定義します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\comps\MyCalloutVN.mxml -->
<s:Callout xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    contentBackgroundAppearance="none"
    horizontalPosition="start"
    verticalPosition="after">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexMouseEvent;
            import views.SettingsView;

            protected function done_clickHandler(event:MouseEvent):void {
                // Create an instance of SettingsView, and
                // initialize it as a copy of the current View of the ViewNavigator.
                var settings:SettingsView = (viewNav.activeView as SettingsView);

                // Create the String to represent the returned data.
                var retData:String = new String();
                // Initialze the String from the current View.
                retData = settings.firstName.text + " " + settings.lastName.text;
                // Close the Callout and return thhe data.
                this.close(true, retData);
            }
        ]]>
    </fx:Script>

    <s:ViewNavigator id="viewNav" width="100%" height="100%" firstView="views.SettingsView">
        <s:navigationContent>
            <s:Button label="Cancel" click="close(false)"/>
        </s:navigationContent>
        <s:actionContent>
            <s:Button id="done" label="OK" emphasized="true" click="done_clickHandler(event);"/>
        </s:actionContent>
    </s:ViewNavigator>
</s:Callout>
```

MyCalloutPassBackVN.mxml で、ViewNavigator の最初のビューが SettingsView であることを指定します。SettingsView で、ユーザーの姓と名の TextInput コントロールを定義します。ユーザーが「OK」を選択すると、Callout を閉じて、返されたデータを MyCalloutPassBackVN に返します。

注意：ViewNavigator が Callout コンテナに表示される際に、ActionBar に透明な背景色が設定されます。この例では、Callout コンテナで contentBackgroundAppearance を none に設定します。このように設定すると、Callout のデフォルトの白い contentBackgroundColor が透明な ActionBar の領域に表示されません。

次の図は、Callout が開いているアプリケーションを示しています。



次に、SettingsView.mxml を示します。

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

    <s:VGroup
        paddingTop="10" paddingLeft="5" paddingRight="10">
        <s:HGroup verticalAlign="middle">
            <s:Label text="First Name: "
                fontWeight="bold"/>
            <s:TextInput id="firstName" width="225"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Last Name: "
                fontWeight="bold"/>
            <s:TextInput id="lastName" width="225"/>
        </s:HGroup>
    </s:VGroup>
</s:View>
```

注意：Callout コンテナ内の ViewNavigator で定義された ActionBar は、背景が透明になります。デフォルトでは、ある View から別の View へのトランジションは Callout 内に正しく表示されます。ただし、デフォルト以外のトランジション（CrossFadeViewTransition や ZoomViewTransition など）を指定した場合は、2 つのビューの ActionBar 領域が重なる可能性があります。この問題を回避するには、ActionBar と Callout のカスタムスキンクラスを作成し、そのクラスで透明でない背景を使用します。

Callout コンテナのサイズと位置の指定

CalloutButton コントロールと Callout コンテナでは、horizontalPosition と verticalPosition の 2 つのプロパティを使用して、Callout コンテナの位置をそのホストを基準に指定します。これらのプロパティには、"before"、"start"、"middle"、"end"、"after" および "auto"（デフォルト）を指定できます。

例えば、これらのプロパティを次のように設定します。

```
horizontalPosition="before"
verticalPosition="after"
```

Callout コンテナは、ホストコンポーネントの左下に開かれます。次のように設定した場合、

```
horizontalPosition="middle"
verticalPosition="middle"
```

Callout コンテナは、ホストコンポーネントの最前面に、コールアウトの中心とホストコンポーネントの中心を揃えて開かれます。

コールアウトからホストへの矢印の描画

horizontalPosition と verticalPosition プロパティの 5 つの組み合わせを除いて、すべての場合に、コールアウトにはホストを指す矢印が描画されます。矢印が表示されない位置は、コールアウトをホストの中央に重ねて中心に配置した場合と、隅に配置した場合です。次の組み合わせでは、矢印は表示されません。

```
// Centered
horizontalPosition="middle"
verticalPosition="middle"

// Upper-left corner
horizontalPosition="before"
verticalPosition="before"

// Lower-left corner
horizontalPosition="before"
verticalPosition="after"

// Upper-right corner
horizontalPosition="after"
verticalPosition="before"

// Lower-right corner
horizontalPosition="after"
verticalPosition="after"
```

また、Callout コンテナの場合、horizontalPosition と verticalPosition プロパティによって、読み取り専用の Callout.arrowDirection プロパティの値が決定されます。ホストを基準にした Callout コンテナの位置によって、arrowDirection プロパティの値が決定されます。設定される可能性がある値は、"up"、"left" などです。

Callout.arrow スキンパーツは、arrowDirection プロパティの値を使用して、コールアウトの位置に基づいて矢印を描画します。

Callout コンテナに使用されるメモリの管理

Callout コンテナを使用する場合の 1 つの考慮事項は、Callout で使用されるメモリをどのように管理するかということです。例えば、アプリケーションで使用されるメモリを減らす場合は、Callout を開くときに毎回そのインスタンスを作成します。この場合、Callout は閉じられたときに破棄されます。ただし、必ず、Callout へのすべての参照（特にイベントハンドラー）が削除されていることを確認してください。削除されていない場合、Callout は破棄されません。

別の方法としては、Callout コンテナが比較的小さい場合に、アプリケーションで同じ Callout を繰り返し再利用することもできます。この設定では、アプリケーションは Callout の単一インスタンスを作成します。このインスタンスを再利用し、Callout は使用されていない間もメモリに残しておきます。この設定では、Callout が開かれるたびにアプリケーションで再作成する必要がないので、アプリケーションの実行時間が短縮されます。

CalloutButton コントロールでのメモリーの管理

CalloutButton コントロールで使用される Callout の設定を行うには、CalloutButton.calloutDestructionPolicy プロパティを設定します。"auto" の値は、Callout が閉じられたときに破棄されるようにコントロールを設定します。"never" の値は、Callout をメモリにキャッシュするようにコントロールを設定します。

Callout コンテナでのメモリーの管理

Callout コンテナでは、calloutDestructionPolicy プロパティは定義されません。代わりに、Callout コンテナのインスタンスをアプリケーションでどのように作成するかによって、メモリの使用を管理します。次の例では、Callout コンテナを開くときに、毎回インスタンスを作成します。

```
protected function button1_clickHandler(event:MouseEvent):void {
    // Create a new instance of the callout container every time you open it.
    var myCallout:MyCallout = new MyCallout();
    myCallout.open(calloutB, true);
}
```

別の方法としては、Callout コンテナの単一インスタンスを定義しておいて、Callout コンテナを開くときに毎回再利用することもできます。

```
// Create a single instance of the callout container.
public var myCallout:MyCallout = new MyCallout();

protected function button1_clickHandler(event:MouseEvent):void {
    myCallout.open(calloutB, true);
}
```

モバイルアプリケーションでのトランジションの定義

Spark のビュートランジションでは、ある View コンテナから別のコンテナへの切り替えがスクリーン上で発生したときに、その切り替えを表示する方法を定義します。トランジションは、ビューの変更時にアニメーションを適用することにより動作します。トランジションを使用して、人を引きつけるモバイルアプリケーションのインターフェイスを作成してください。

デフォルトでは、既存の View コンテナがスクリーン外にスライドし、新しいビューがスクリーン内にスライドします。この切り替えは、カスタマイズすることもできます。例えば、View コンテナにいくつかのフィールドのみを表示するフォームを定義し、後に続く View コンテナにその他のフィールドを表示します。ビューからビューにスライドする代わりに、反転またはフェードによるトランジションを使用できます。

Flex には次のビュートランジションクラスが用意されており、View コンテナを切り替える際に使用できます。

トランジション	説明
CrossFadeViewTransition	既存のビューと新しいビューとのクロスフェードトランジションを実行します。既存のビューがフェードアウトし、新しいビューがフェードインします。
FlipViewTransition	既存のビューと新しいビューとの反転トランジションを実行します。反転の方向とタイプを定義できます。
SlideViewTransition	既存のビューと新しいビューとのスライドトランジションを実行します。既存のビューがスライドアウトし、新しいビューがスライドインします。スライドする方向とタイプを制御できます。Flex で使用されるデフォルトのビュートランジションです。
ZoomViewTransition	既存のビューと新しいビューとのズームトランジションを実行します。既存のビューをズームアウトするか、新しいビューをズームインできます。

注意：モバイルアプリケーションのビュートランジションは、標準の Flex のトランジションとは関係ありません。標準の Flex のトランジションでは、ステートの変更時に再生されるエフェクトを定義します。View のトランジションは、ViewNavigator コンテナのナビゲーション操作によってトリガーされます。View のトランジションは、MXML には定義できず、ステートの操作も行いません。

トランジションの適用

アクティブな View コンテナを切り替える際のトランジションを適用します。View コンテナを切り替えるとビュートランジションが発生するので、ViewNavigator コンテナを介してトランジションを制御します。

例えば、現在のビューを変更するには、ViewNavigator クラスの次のメソッドを使用できます。

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

これらのすべてのメソッドでは、ビューを切り替える際に再生するトランジションを定義するためのオプションの引数が使用されます。

デバイスのハードウェアナビゲーションキー（戻るボタンなど）を使用して、現在のビューを切り替えることもできます。ハードウェアキーを使用してビューを切り替えると、ViewNavigator では、ViewNavigator.defaultPopTransition プロパティおよび ViewNavigator.defaultPushTransition プロパティで定義されているデフォルトのトランジションが使用されます。デフォルトでは、これらのプロパティは SlideViewTransition クラスを使用するように指定されます。

次の例は、defaultPopTransition および defaultPushTransition プロパティを初期化して FlipViewTransition を使用するメインアプリケーションファイルを示しています。

```
<?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>
```

最初のビューである EmployeeMainViewTrans.mxml は CrossFadeViewTransition を定義しています。次に、EmployeeView への変更時に、CrossFadeViewTransition を引数として pushView() メソッドに渡します。

```
<?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>
```

次の例に示すように、EmployeeView は EmployeeView.mxml ファイルに定義されています。

```
<?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>
```

ActionBar コントロールへのトランジションの適用

デフォルトでは、1 つのビューから別のビューへのトランジションに ActionBar は含まれていません。代わりに、ActionBar コントロールでは、指定されているトランジションに関係なく、ビューの変更時にスライドトランジションが使用されます。ビューの変更時のトランジションに ActionBar を含めるには、トランジションクラスの transitionControlsWithContent プロパティを true に設定します。

トランジションでのイージングクラスの使用

トランジションは、加速フェーズ、減速フェーズの順に 2 つのフェーズで再生されます。トランジションの加速と減速のプロパティは、イージングクラスを使用して変更できます。イージングを使用すると、現実的な加速と減速の割合を作成できます。イージングクラスを使用して、バウンスエフェクトを作成したり、その他の動作タイプを制御したりすることもできます。

Spark のイージングクラスは、spark.effects.easing パッケージで提供されます。このパッケージには、バウンス、線状、正弦波のイージングなど、最も一般的なタイプのイージングのクラスが格納されています。これらのクラスの使用について詳しくは、Using Spark easing classes を参照してください。

次の例は、前の節に定義されているアプリケーションに対する変更を示しています。このバージョンでは、FlipViewTransition に Bounce イージングクラスを追加します。

```
<?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>
```

バウンスの表示は、必ずデバイスの戻るボタンを使用して行ってください。

ビュートランジションのライフサイクル

ビュートランジションには、準備フェーズと実行フェーズの 2 つの主要なフェーズがあります。

transition クラスの 3 つのメソッドでは、準備フェーズを定義します。これらのメソッドは次の順序で呼び出されます。

1 captureStartValues()

このメソッドが呼び出されるときには、ViewNavigator は新しいビューを作成済みですが、新しいビューを検証していないか、ActionBar コントロールとタブバーのコンテンツを更新していません。このメソッドを使用して、トランジションで役割を果たすコンポーネントの開始値を取得します。

2 captureEndValues()

このメソッドが呼び出されるときには、新しいビューは完全に検証済みで、ActionBar コントロールとタブバーには新しいビューのステートが反映されています。トランジションでは、このメソッドを使用して新しいビューから必要な値を取得できます。

3 prepareForPlay()

このメソッドでは、トランジションのコンポーネントのアニメーション化に使用するエフェクトインスタンスを初期化できます。

実行フェーズは、ViewNavigator がトランジションの play() メソッドを呼び出すと開始されます。この時点では、新しいビューが作成されて検証され、ActionBar コントロールとタブバーが初期化されています。トランジションで start イベントが送出され、エフェクトの play() メソッドの呼び出しにより、準備フェーズで作成されたエフェクトのインスタンスが起動されます。

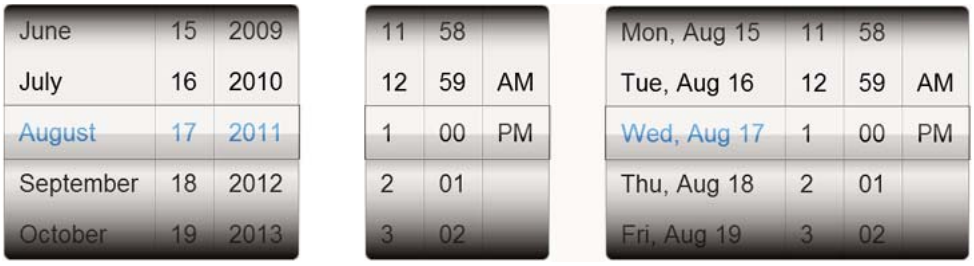
ビュートランジションが完了すると、トランジションで end イベントが送出されます。end イベントを送出するためにユーザーが呼び出す transitionComplete() メソッドは、ViewTransitionBase というトランジション基本クラスによって定義されます。完了イベントを送出する前に、トランジションで一時オブジェクトをクリーンアップし、作成したリスナーを削除することが重要です。

transitionComplete() メソッドを呼び出すと、ViewNavigator はビュー変更プロセスを終了し、トランジションを未初期化ステートにリセットします。

モバイルアプリケーションでの日付と時刻の選択

DateSpinner コントロールを使用すると、ユーザーはモバイルアプリケーションで日付と時刻を選択できます。一般的なモバイルユーザーインターフェイスである横並びのスクロールホイールを使用します。各ホイールには、日付と時刻の個々のパーツが表示されます。

基本的な 3 つのタイプの DateSpinner コントロールを使用できます。次の図は、3 つのタイプの DateSpinner コントロールを示しています。



A. 日付 B. 時刻 C. 日付と時刻

次の表で、DateSpinner のタイプについて説明します。

タイプ	定数（同等の文字列）	説明
日付	DateSelectorDisplayMode.DATE（「date」）	年、月、日を表示します。次に例を示します。 `|| June || 11 || 2011 ||` 日付はデフォルトのタイプです。DateSpinner コントロールの displayMode プロパティを設定しない場合、Flex によってプロパティが date に設定されます。 現在の日付は、accentColor スタイルプロパティで定義された色でハイライト表示されます。 サポートされている最も過去の日付は、1601 年 1 月 1 日です。サポートされている最も未来の日付は、9999 年 12 月 31 日です。
時刻	DateSelectorDisplayMode.TIME（「time」）	時と分を表示します。12 時間形式の時刻を使用するロケールの場合は、AM/PM のインジケーターも表示されます。次に例を示します。 `|| 2 || 57 || PM ||` 現在の時刻はハイライト表示されません。 DateSpinner コントロールでは、秒は表示できません。 12 時間形式と 24 時間形式を切り替えることはできません。DateSpinner では、現在のロケールで標準的な形式が使用されます。
日付と時刻	DateSelectorDisplayMode.DATE_AND_TIME（「dateAndTime」）	日付と時、分を表示します。12 時間形式の時刻を使用するロケールの場合は、AM/PM のインジケーターも表示されます。次に例を示します。 `|| Mon Jun 13 || 2 || 57 || PM ||` 現在の日付は、accentColor スタイルプロパティで定義された色でハイライト表示されます。現在の時刻はハイライト表示されません。 DateSpinner コントロールでは、秒は表示できません。 月の名前（英語）は短縮形で表示されます。年は表示されません。

DateSpinner コントロールは、複数の SpinnerList コントロールによって構成されます。各 SpinnerList には、DateSpinner コントロールの特定部分の有効値のリストが表示されます。例えば、日付を表示する DateSpinner コントロールには 3 つの SpinnerList が含まれ、1 つは年、1 つは月、1 つは日が表示されます。時刻のみを表示する DateSpinner には、2 つまたは 3 つの SpinnerList が含まれ、1 つは時、1 つは分、1 つはオプションで AM/PM（時刻の表示が 12 時間単位の場合）が表示されます。

DateSpinner コントロールのタイプの変更

DateSpinner のタイプを選択するには、コントロールの displayMode プロパティの値を設定します。displayMode プロパティには、DateSelectionDisplayMode クラスで定義されている定数、またはそれと同等の文字列を設定できます。

次の例を使用すると、別の DateSpinner タイプに切り替えることができます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerTypes.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Types">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <s:ComboBox id="modeList" selectedIndex="0" change="ds1.displayMode=modeList.selectedItem.value">
        <s:ArrayList>
            <fx:Object value="date" label="Date"/>
            <fx:Object value="time" label="Time"/>
            <fx:Object value="dateAndTime" label="Date and Time"/>
        </s:ArrayList>
    </s:ComboBox>
    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>

    <s:Label text="{ds1.selectedDate}"/>

</s:View>
```

ユーザーが DateSpinner コントロールを操作すると、スピナーはリスト内の一番近いアイテムにスナップされます。停止中は、スピナーの選択は変更されません。

DateSpinner コントロールの選択値の他のコントロールへのバインド

DateSpinner コントロールの selectedDate プロパティをモバイルアプリケーションの他のコントロールにバインドすることができます。selectedDate プロパティは、Date オブジェクトへのポインターであるので、この方法によって、Date オブジェクトのメソッドにアクセスできます。

次の例では、年、月、日を Label コントロールにバインドします。

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

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var dayArrayC:ArrayCollection = new ArrayCollection();

            [Bindable]
            private var selectedDateProperty:Date;

            private function initAC():void {
                dayArrayC.addItem("Sunday");
                dayArrayC.addItem("Monday");
                dayArrayC.addItem("Tuesday");
                dayArrayC.addItem("Wednesday");
                dayArrayC.addItem("Thursday");
                dayArrayC.addItem("Friday");
                dayArrayC.addItem("Saturday");
            }

        ]]>
    </fx:Script>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>
    <fx:Binding source="ds1.selectedDate" destination="selectedDateProperty"/>

    <s:Label id="label1" text="Day: {dayArrayC.getItemAt(ds1.selectedDate.day)}"/>
    <s:Label id="label2" text="Day of month: {selectedDateProperty.getDate()}"/>
    <s:Label id="label3" text="Month: {ds1.selectedDate.getMonth() + 1}"/>
    <s:Label id="label4" text="Year: {selectedDateProperty.getFullYear()}"/>

</s:View>
```

プログラムによる DateSpinner コントロールの日付の選択

プログラムで DateSpinner コントロールの日付を変更するには、新しい Date オブジェクトを selectedDate プロパティの値に割り当てます。

次の例では、ユーザーに年、月、日を入力するように指示します。ボタンをクリックすると、DateSpinner は新しい日付に変更されます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerProgrammaticSelection.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Programmatic Selection"
        creationComplete="init()">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.components.calendarClasses.DateSelectorDisplayMode;

            private function init():void {
                // change event is dispatched when DateSpinner changes from user interaction
                ds1.addEventListener("change", spinnerEventHandler);
                // valueCommit event is dispatched when DateSpinner programmatically changes
                ds1.addEventListener("valueCommit", spinnerEventHandler);
            }

            private function b1_clickHandler(e:Event):void {
                ds1.selectedDate = new Date(ti3.text,ti1.text,ti2.text);
            }

            protected function spinnerEventHandler(event:Event):void {
                eventLabel.text = event.type;
            }
        ]]>
    </fx:Script>

    <s:TextInput id="ti1" prompt="Enter a Month"/>
    <s:TextInput id="ti2" prompt="Enter a Day"/>
    <s:TextInput id="ti3" prompt="Enter a Year"/>
    <s:Button id="b1" label="Go!" click="b1_clickHandler(event)"/>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>

    <s:Label id="eventLabel"/>

</s:View>
```

プログラムによって日付が変更されたときには、DateSpinner コントロールで change と valueCommit の両方のイベントが送出されます。ユーザーインタラクションによって日付が変更されたときには、DateSpinner コントロールで change イベントが送出されます。

選択されている日付がプログラムによって変更された場合、選択した値が表示に適用されますが、間の値のアニメーションは表示されません。

DateSpinner コントロールの日付範囲の制限

minDate および maxDate プロパティを使用して、DateSpinner コントロールでユーザーが選択可能な日付を制限できます。これらのプロパティでは、Date オブジェクトを取得します。minDate プロパティより前の日付、および maxDate プロパティより後の日付には、DateSpinner コントロールでアクセスできません。また、「date」モードでは無効な年は表示されず、「dateAndTime」モードでは無効な日付は表示されません。

次の例では、使用可能な日付範囲が異なる 2 つの DateSpinner コントロールを作成します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/MinMaxDates.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Min/Max Dates">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Min date today, max date October 31, 2012. -->
    <s:Label text="{dateSpinner1.selectedDate}"/>
    <s:DateSpinner id="dateSpinner1"
        displayMode="{DateSelectorDisplayMode.DATE}"
        minDate="{new Date()}"
        maxDate="{new Date(2012,9,31)}"/>
    <!-- Min date 3 days ago, max date 7 days from now. -->
    <s:Label text="{dateSpinner2.selectedDate}"/>
    <s:DateSpinner id="dateSpinner2"
        displayMode="{DateSelectorDisplayMode.DATE}"
        minDate="{new Date(new Date().getTime() - 1000*60*60*24*3)}"
        maxDate="{new Date(new Date().getTime() + 1000*60*60*24*7)}"/>
</s:View>
```

1 つの最小日付と 1 つの最大日付のみ設定できます。日付の配列や複数の選択範囲は設定できません。

また、minDate および maxDate プロパティを使用して、「time」モードの DateSpinner に制限をかけることもできます。次の例では、時刻の選択を AM 8 時から PM 2 時までに制限します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/MinMaxTime.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Min/Max Time">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Limit time selection to between 8am and 2pm -->
    <s:DateSpinner id="dateSpinner1"
                   displayMode="{DateSelectorDisplayMode.TIME}"
                   minDate="{new Date(0,0,0,8,0)}"
                   maxDate="{new Date(0,0,0,14,0)}"/>

</s:View>
```

DateSpinner コントロールのイベントへの応答

DateSpinner コントロールでは、ユーザーによって日付が変更されたときに change イベントが送出されます。この change イベントの target プロパティには、DateSpinner への参照が含まれているので、選択されている日付へのアクセスに使用できます。次に例を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerChangeEvent.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Change Event">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;

            private var dayArray:Array = new Array(
                "Sunday","Monday","Tuesday",
                "Wednesday","Thursday","Friday","Saturday");

            private function ds1_changeHandler(e:Event):void {
                // Optionally cast the DateSpinner's selectedDate as a Date
                var d:Date = new Date(e.currentTarget.selectedDate);
                ta1.text = "You selected:";
                ta1.text += "\n Day of Week: " + dayArray[d.day];
                ta1.text += "\n Year: " + d.fullYear;
                // Month is 0-based in ActionScript, so add 1:
                ta1.text += "\n Month: " + int(d.month + 1);
                ta1.text += "\n Day: " + d.date;
            }
        ]]>
    </fx:Script>

    <s:DateSpinner id="ds1"
        displayMode="{DateSelectorDisplayMode.DATE}"
        change="ds1_changeHandler(event)"/>

    <s:TextArea id="ta1" height="200" width="350"/>
</s:View>
```

change イベントの送出（および selectedDate プロパティの値の更新）は、すべてのスピナーでユーザーインタラクションによる回転が止まったときにのみ実行されます。

プログラムによって行われた日付の変更をキャプチャするには、value_commit イベントをリスニングします。

DateSpinner コントロールの分の間隔の変更

DateSpinner コントロールに表示される分の間隔は、minuteStepSize プロパティを使用して変更できます。このプロパティは、displayMode が「time」または「dateAndTime」に設定されている DateSpinner コントロールに対してのみ適用されます。例えば、minuteStepSize プロパティに 10 を設定した場合、DateSpinner コントロールの分のスピナーには、0、10、20、30、40 および 50 の値が表示されます。

次の例を使用すると、minuteStepSize プロパティの値を設定できます。分のスピナーは、それに応じて更新されます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerMinuteInterval.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Minute Interval">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>
    <s:Label text="Select an interval:"/>

    <s:ComboBox id="intervalList" selectedIndex="0"
        change="ds1.minuteStepSize=intervalList.selectedItem.value">
        <s:ArrayList>
            <fx:Object value="1" label="1"/>
            <fx:Object value="2" label="2"/>
            <fx:Object value="3" label="3"/>
            <fx:Object value="4" label="4"/>
            <fx:Object value="5" label="5"/>
            <fx:Object value="6" label="6"/>
            <fx:Object value="10" label="10"/>
            <fx:Object value="12" label="12"/>
            <fx:Object value="15" label="15"/>
            <fx:Object value="20" label="20"/>
            <fx:Object value="30" label="30"/>
        </s:ArrayList>
    </s:ComboBox>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.TIME}"/>
</s:View>
```

minuteStepSize プロパティの有効な値は、60 を割り切れる数である必要があります。60 を割り切れない値（例えば 25）を指定した場合、minuteStepSize プロパティには 1 の値がデフォルト設定されます。

分の間隔を指定したときに、現在の時刻が分のスピナーの値に当てはまらない場合は、DateSpinner コントロールの現在の選択値が一番近い間隔の値に切り捨てられます。例えば、時刻が 10:29 で、minuteStepSize が 15 の場合、10:15 という値が minDate の設定に違反していないとすれば、時刻は 10:15 に切り捨てられます。

DateSpinner コントロールの外観のカスタマイズ

DateSpinner コントロールでは、fontSize や color、letterSpacing など、ほとんどのテキストスタイルがサポートされます。さらに、accentColor という新しいスタイルプロパティも追加されます。このスタイルは、スピナーリストでの現在の日付や時刻の色を変更します。次の例では、この色を赤色に設定します。

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

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Acceptable style formats are color_name (e.g., 'red') or
        hex colors (e.g., '0xFF0000') -->
    <s:DateSpinner id="dateSpinner1" accentColor="0xFF0000"
        displayMode="{DateSelectorDisplayMode.DATE}"/>
</s:View>
```

DateSpinner コントロールでは、textAlign プロパティはサポートされません。テキストの配置はコントロールで設定します。

DateSpinner コントロールのその他の外観をカスタマイズするには、コントロールのカスタムスキンを作成するか、CSS を使用して一部のサブコンポーネントを変更します。

DateSpinnerSkin クラスは、DateSpinner のサイズ設定を制御します。DateSpinner コントロール内の各スピナーは、それぞれ個別の SpinnerListSkin を使用した SpinnerList オブジェクトです。1 つの DateSpinner コントロール内のすべてのスピナーは、1 つの SpinnerListContainer の子です。SpinnerListContainer は、それぞれ個別のスキン SpinnerListContainerSkin を持っています。

DateSpinner コントロールの height プロパティを明示的に設定できます。width プロパティを設定した場合、コントロールは、要求した幅に合わせてサイズ変更された領域の中央に配置されます。

DateSpinner コントロール内のスピナーの設定を変更するために、SpinnerList、SpinnerListContainer および SpinnerListItemRenderer の CSS タイプセレクターも使用できます。例えば、SpinnerList タイプセレクターは、スピナーのパディングプロパティを制御します。

次の例では、スピナーのパディングを変更します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/DateSpinnerExamples2.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.CustomSpinnerListSkinExample">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        s|SpinnerListItemRenderer {
            paddingTop: 7;
            paddingBottom: 7;
            paddingLeft: 5;
            paddingRight: 5;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

モバイルアプリケーションでは、タイプセレクターは、アプリケーションの子ビューではなく、最上位のアプリケーションファイルに含める必要があります。ビュー内のスタイルブロックに SpinnerListItemRenderer タイプセレクターを設定しようとすると、Flex でコンパイラー警告がスローされます。

SpinnerListContainerSkin クラスを拡張して、DateSpinner コントロールのスピナーの外観をより詳細にカスタマイズできます。次の例では、SpinnerListContainer にカスタムスキンを適用します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/DateSpinnerExamples3.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                            xmlns:s="library://ns.adobe.com/flex/spark"
                            firstView="views.CustomSpinnerListSkinExample">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        /* Change SpinnerListContainer for all DateSpinner controls */
        s|SpinnerListContainer {
            skinClass: ClassReference("customSkins.CustomSpinnerListContainerSkin");
        }

        /* Change padding for all DateSpinner controls */
        s|SpinnerListItemRenderer {
            paddingTop: 7;
            paddingBottom: 7;
            paddingLeft: 5;
            paddingRight: 5;
            fontSize: 12;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

次の CustomSpinnerListContainerSkin クラスでは、「選択インジケーター」のサイズを縮小して、スピナーの行に新しいサイズのフォントとパディングをより詳細に反映します。

```
// datespinner/customSkins/CustomSpinnerListContainerSkin.as
package customSkins {
    import mx.core.DPIClassification;

    import spark.skins.mobile.SpinnerListContainerSkin;
    import spark.skins.mobile.supportClasses.MobileSkin;
    import spark.skins.mobile160.assets.SpinnerListContainerBackground;
    import spark.skins.mobile160.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile160.assets.SpinnerListContainerShadow;
    import spark.skins.mobile240.assets.SpinnerListContainerBackground;
    import spark.skins.mobile240.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile240.assets.SpinnerListContainerShadow;
    import spark.skins.mobile320.assets.SpinnerListContainerBackground;
    import spark.skins.mobile320.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile320.assets.SpinnerListContainerShadow;

    public class CustomSpinnerListContainerSkin extends SpinnerListContainerSkin
    {
        public function CustomSpinnerListContainerSkin() {
            super();

            switch (applicationDPI)
            {
                case DPIClassification.DPI_320:
                {
                    borderClass = spark.skins.mobile320.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
```

```
spark.skins.mobile320.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile320.assets.SpinnerListContainerShadow;

                    cornerRadius = 10;
                    borderThickness = 2;
                    selectionIndicatorHeight = 80; // was 120
                    break;
                }
                case DPIClassification.DPI_240:
                {
                    borderClass = spark.skins.mobile240.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile240.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile240.assets.SpinnerListContainerShadow;

                    cornerRadius = 8;
                    borderThickness = 1;
                    selectionIndicatorHeight = 60; // was 90
                    break;
                }
                default: // default DPI_160
                {
                    borderClass = spark.skins.mobile160.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile160.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile160.assets.SpinnerListContainerShadow;

                    cornerRadius = 5;
                    borderThickness = 1;
                    selectionIndicatorHeight = 40; // was 60

                    break;
                }
            }
        }
    }
}
```

モバイルコンポーネントへのスキンの適用について詳しくは、154 ページの「モバイルのスキン適用の基本」を参照してください。

DateSpinner コントロールでのローカライズされた日付と時刻の使用

DateSpinner コントロールでは、アプリケーションを実行中のデバイスでサポートされるすべてのロケールがサポートされます。例えば、ロケールを ja-JP に設定すると、DateSpinner の日付表示は、日本語ロケールの標準形式に変更されます。

DateSpinner コントロールで直接 locale プロパティを設定したり、コンテナ（例えば Application）でこのプロパティを設定したりすることができます。DateSpinner コントロールは、このプロパティの値を継承します。locale プロパティを使用してロケールをオーバーライドしていない場合、デフォルトのロケールは、アプリケーションを実行中のデバイスのロケールになります。

次の例では、デフォルトのロケールを「ja-JP」に設定します。ロケールを選択して、DateSpinner の形式を次のように変更できます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/LocalizedDateSpinner.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Localized DateSpinner" locale="ja_JP">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            private function localeChangeHandler():void {
                ds1.setStyle('locale',localeSelector.selectedItem);
            }
        ]]>
    </fx:Script>

    <s:ComboBox id="localeSelector" change="localeChangeHandler()">
        <s:ArrayList>
            <fx:String>en-US</fx:String>
            <fx:String>en-UK</fx:String>
            <fx:String>es-AR</fx:String>
            <fx:String>he-IL</fx:String>
            <fx:String>ko-KR</fx:String>
            <fx:String>ja-JP</fx:String>
            <fx:String>vi-VN</fx:String>
            <fx:String>zh-CN</fx:String>
            <fx:String>zh-TW</fx:String>
        </s:ArrayList>
    </s:ComboBox>
    <s:DateSpinner id="ds1" displayMode="dateAndTime"/>

</s:View>
```

モバイルアプリケーションでのスピナーリストの使用

SpinnerList コンポーネントは、一般的にモバイルアプリケーションでのデータの選択に使用する特殊な List です。デフォルトでは、ユーザーがリストアイテムをスクロールしているときに、リストの最後に達すると、アイテムがラップアラウンドします。SpinnerList コントロールは、一般的にモバイルアプリケーションの数値ステッパーコンポーネントとして使用します。

次の図は、標準的な SpinnerList コントロールがモバイルアプリケーションでどのように表示されるかを示しています。

SpinnerList コントロール

SpinnerList は、回転する円柱状のドラムのように機能します。ユーザーは、上方向または下方向にスローしてリストを回転したり、リストを上方向または下方向にドラッグしたり、リスト内のアイテムをクリックしたりできます。

通常は、SpinnerList コントロールを SpinnerListContainer コントロールでラップします。このクラスでは、SpinnerList のほとんどのクロームを提供してレイアウトを定義します。クロームには、ボーダーやシャドウ、選択インジケーターの外観などが含まれます。

SpinnerList のデータはリストとして保存されます。このデータは、SpinnerListItemRenderer を使用してスピナーにレンダリングされます。アイテムレンダラーをオーバーライドして、リストアイテムの外観やコンテンツをカスタマイズすることができます。

DateSpinner コントロールは、SpinnerList コントロールのセットでカスタムアイテムレンダラーを使用する 1 つの例です。

現時点では、SpinnerList コントロールのアイテムを無効化するには、コントロール全体を無効化する以外に方法はありません。この制約は、DateSpinner コントロールには当てはまりません。DateSpinner コントロールでは、無効な日付の範囲を設定する追加ロジックが用意されています。

スピナーリストのデータの定義

SpinnerList コントロールのデータを定義するには、次のいずれの方法で定義します。

- SpinnerList コントロールの dataProvider プロパティでデータをインラインで定義します。
- <s:SpinnerList> タグの子タグとしてデータを定義します。
- ActionScript または MXML でデータを定義し、そのデータを SpinnerList コントロールにバインドします。このデータは、外部サービスや埋め込みリソース（XML ファイルなど）、その他のデータソースを使用して取得できます。
- Flash Builder のサービスウィザードを使用して、SpinnerList コントロールをデータサービス操作にバインドします。Flash Builder でのデータ中心型アプリケーションの構築について詳しくは、データサービスへの接続を参照してください。

SpinnerList コントロールでは、IList インターフェイスをデータプロバイダーとして実装するクラスを取得できます。このようなクラスには、ArrayCollection、ArrayList、NumericDataProvider および XMLListCollection クラスがあります。

SpinnerList コントロールのインスタンス作成時にデータプロバイダーを定義しない場合は、SpinnerList は 1 行の空の行で表示されます。データプロバイダーを追加すると、SpinnerList はサイズが変更され、デフォルトの 5 つのアイテムがリストに表示されます。

次の例では、SpinnerList コントロールのデータを <s:SpinnerList> タグの子タグで定義します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListComplexDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Complex Data Provider">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <s:Label text="Select a person:"/>

    <s:SpinnerListContainer>
        <s:SpinnerList id="peopleList" width="200" labelField="name">
            <s:ArrayList>
                <fx:Object name="Friedeman Friese" companyID="14266"/>
                <fx:Object name="Stephen Glenn" companyID="14266"/>
                <fx:Object name="Reiner Knizia" companyID="11233"/>
                <fx:Object name="Alan Moon" companyID="11543"/>
                <fx:Object name="Klaus Teuber" companyID="13455"/>
                <fx:Object name="Dale Yu" companyID="14266"/>
            </s:ArrayList>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:Label text="Selected ID: {peopleList.selectedItem.companyID}"/>

</s:View>
```

次の例では、SpinnerList のデータを <s:SpinnerList> で定義します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListInlineDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Inline Data Provider">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <!-- Create data provider inline. -->
        <s:SpinnerList id="smallList" dataProvider="{new ArrayList([1,5,10,15,30])}"
                       wrapElements="false" typicalItem="44"/>
    </s:SpinnerListContainer>

    <s:Label text="Selected Item: {smallList.selectedItem}"/>

</s:View>
```

次の例では、SpinnerList のデータを ActionScript で定義します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListBasicDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Basic Data Provider"
        creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            public var daysOfWeek:ArrayList;

            private function initApp():void {
                daysOfWeek = new ArrayList(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]);
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="daysList" width="100" dataProvider="{daysOfWeek}"/>
    </s:SpinnerListContainer>

    <s:Label text="Selected Day: {daysList.selectedItem}"/>

</s:View>
```

ActionScript で複合オブジェクトをデータとして使用する場合は、SpinnerList に正しいラベルが表示されるように labelField プロパティを指定します。次に例を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListComplexASDP.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Complex Data Provider in AS" creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            private var myAC:ArrayList;
```

```
            private function initApp():void {
                myAC = new ArrayList([
                    {name:"Alan Moon",id:42},
                    {name:"Friedeman Friese",id:44},
                    {name:"Dale Yu",id:45},
                    {name:"Stephen Glenn",id:47},
                    {name:"Reiner Knizia",id:48},
                    {name:"Klaus Teuber",id:49}
                ]);
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="peopleList" dataProvider="{myAC}"
                       width="200"
                       labelField="name"/>
    </s:SpinnerListContainer>
    <s:Label text="Selected ID: {peopleList.selectedItem.id}"/>
</s:View>
```

また、便利な NumericDataProvider クラスを使用して、SpinnerList コントロールに数値データを提供することもできます。このクラスを使用すると、一連の数値データと最小値、最大値およびステップの量を簡単に定義できます。

次の例では、SpinnerList コントロールのデータソースとして NumericDataProvider クラスを使用します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/MinMaxSpinnerList.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Min/Max SpinnerLists"
        backgroundColor="0x000000">

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer top="10" left="10">
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="23" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="59" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="59" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100"
                       dataProvider="{new ArrayList(['AM','PM'])}"
                       wrapElements="false"/>
    </s:SpinnerListContainer>
</s:View>
```

stepSize プロパティの値には、負の数値も設定できます。この場合は、最大値が最初に表示される値になります。スピナーは最大値から始まり、最小値に向かってステップします。

スピナーリスト内のアイテムの選択

SpinnerList コントロールでは、一度に 1 つのアイテムのみの選択がサポートされます。選択されたアイテムは、常にコンポーネントの中央に表示され、デフォルトでは選択インジケーターの下に表示されます。SpinnerList が回転中でないときは、必ずアイテムが選択されている必要があります。無効化されているアイテムやアイテムがない行は選択できません。

SpinnerList コントロールで現在選択されているアイテムを取得するには、コントロールの selectedIndex または selectedItem プロパティにアクセスします。

SpinnerList コントロールで現在選択されているアイテムを設定するには、selectedIndex または selectedItem プロパティの値を設定します。通常、これらのプロパティは <s:SpinnerList> タグで設定して、SpinnerList の作成時にアイテムが選択されるようにします。

SpinnerList の selectedIndex または selectedItem プロパティの値を明示的に設定しない場合、デフォルトで選択されるアイテムは、リスト内の最初のアイテムになります。

selectedIndex または selectedItem プロパティを使用して、スピナーの選択アイテムをプログラムによって変更することもできます。これらのいずれかのプロパティを設定すると、コントロールではそのアイテムにスナップされます。この場合、スピナーでそのアイテムに対するアニメーション（つまり回転）は行われません。

次の例では、SpinnerList コントロールをカウントダウンのタイマーとして使用します。Timer オブジェクトでは、selectedIndex プロパティの値を毎秒変更することで、スピナーの選択アイテムを変更します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListCountdownTimer.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Countdown Timer"
        creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            private var myTimer:Timer;

            private function initApp():void {
                myTimer = new Timer(1000, 0); // 1 second
                myTimer.addEventListener(TimerEvent.TIMER, changeSpinner);
                myTimer.start();
            }
            private function changeSpinner(e:Event):void {
                secList.selectedIndex = secList.selectedIndex - 1;
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer left="50" top="50">
        <s:SpinnerList id="secList" width="100" selectedIndex="60">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="60" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
    </s:SpinnerListContainer>
</s:View>
```

スピナーリストでのユーザーインタラクションとイベント

SpinnerList コントロールの選択アイテムが変更されたときには、change イベントと valueCommit イベントが送出されます。通常、これはスワイプなど、ユーザーインタラクションに対する反応です。ユーザーがアイテムにタッチして選択した場合は、change イベントと valueCommit イベントに加えて、さらに click イベントも送出されます。

プログラムによって選択アイテムが変更された場合は、valueCommit イベントのみが送出されます。

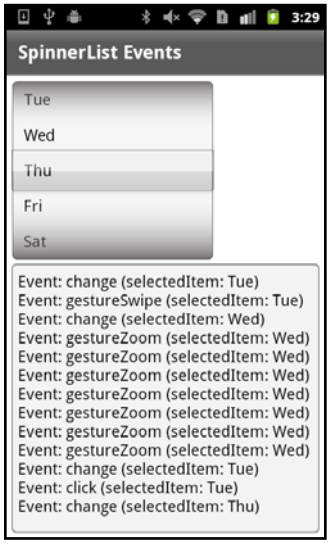
SpinnerList コントロールが回転しているときに、通過した各アイテムのイベントは送出されません。change や valueCommit などのイベントは、新しいアイテムで停止されたときにのみ送出されます。

データプロバイダーを使用して SpinnerList コントロールの最初のインスタンスが作成されたときには、change イベントと valueCommit イベントの両方が送出されます。

次の例は、SpinnerList コントロールを使用しているときに送出される一般的なイベントを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListEvents.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="SpinnerList Events"
        creationComplete="initApp()">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"
                          paddingRight="10" paddingBottom="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            public var daysOfWeek:ArrayList;

            private function initApp():void {
                daysOfWeek = new ArrayList(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]);
            }

            private function eventHandler(e:Event):void {
               ta1.text += "Event: " + e.type + " (selectedItem: " + e.currentTarget.selectedItem + ")\n";
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="daysList" width="300"
            dataProvider="{daysOfWeek}"
            change="eventHandler(event)"
            gestureSwipe="eventHandler(event)"
            click="eventHandler(event)"
            gestureZoom="eventHandler(event)"
        />
    </s:SpinnerListContainer>
    <s:TextArea id="ta1" width="100%" height="100%"/>
</s:View>
```

次の図は、SpinnerList コントロールを操作した後の出力を示しています。



SpinnerList コントロールのイベント

スピナーリストのラップアラウンド設定

デフォルトでは、SpinnerList コントロールのデータプロバイダーのアイテム数が、スピナーに表示されるアイテム数よりも少ない場合、スピナーはラップアラウンドせず、リストの最後のアイテムで停止します。そうでない場合は、ユーザーが最後のアイテムを通り過ぎると、スピナーはリストの最初に戻ります。

リストに表示されるデフォルトのアイテム数は 5 です。アイテムの数を変更する場合は、カスタムスキンを作成します。詳しくは、112 ページの「スピナーリストのカスタムスキンの作成」を参照してください。

wrapElements プロパティの値は、SpinnerList コントロールでリスト内の最後のアイテムに到達した後に、もう一度最初のアイテムから開始するかどうかを決定します。wrapElements を false に設定した場合、リストのアイテム数や表示されるアイテム数に関係なく、スピナーはリストの最後に達すると停止します。

wrapElements プロパティを true に設定した場合、スピナーはもう一度最初のアイテムから開始されますが、これは、リストに含まれるアイテム数が、表示可能なアイテム数よりも 1 つ以上多い場合にのみ実行されます。例えば、SpinnerList の表示アイテムが 5 つで、リストにアイテムが 4 つしか存在しない場合は、wrapElements プロパティの設定にかかわらず、リストはラップアラウンドしません。

SpinnerList のデフォルトのラップアラウンド動作は、wrapElements プロパティを true または false に設定してオーバーライドできます。

次の例を使用すると、wrapElements プロパティの値を切り替えることができます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListWrapElements.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Wrap Elements">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="smallList" typicalItem="45"
                       dataProvider="{new ArrayList([1,5,6,10,15,30])}"
                       wrapElements="{cb1.selected}"/>
    </s:SpinnerListContainer>

    <!-- By default, cause the elements to be wrapped by setting this to true -->
    <s:CheckBox id="cb1" label="Wrap Elements" selected="true"/>

</s:View>
```

一般的に、ユーザーは、一度にリストに表示される数よりもアイテムが多い場合、リストがラップアラウンドすることを望みます。スピナーで表示可能な数よりもアイテムが少ない場合は、一般的に、リストがラップアラウンドしないことを望みます。

スピナーリストのスタイルの設定

SpinnerList コントロールでは、Spark のモバイルテーマに共通するすべてのテキストスタイルのサポートを制御します。これらのスタイルには、fontSize、fontWeight、color、textDecoration および配置のプロパティがあります。これらのスタイルプロパティは、MXML または CSS のコントロールで直接設定できます。また、これらのプロパティが親コンテナで設定された場合、SpinnerList はこれらのプロパティを継承します。

また、SpinnerListItemRenderer スタイルプロパティを変更することで、SpinnerList のパディングプロパティを定義できます。

次の例では、SpinnerList タイプセレクターでテキスト関連のスタイルプロパティを設定し、SpinnerListItemRenderer タイプセレクターでパディングプロパティを設定します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/SpinnerListExamples2.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                            xmlns:s="library://ns.adobe.com/flex/spark"
                            firstView="views.SpinnerListStyles">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|SpinnerList {
            textAlign: right;
            fontSize: 13;
            fontWeight: bold;
            color: red;
        }
        s|SpinnerListItemRenderer {
            paddingTop: 5;
            paddingBottom: 5;
            paddingRight: 5;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

モバイルアプリケーションでタイプセレクターを使用する場合は、最上位のアプリケーションファイルで <fx:Style> ブロックを定義します。そうしない場合、コンパイラーで警告がスローされ、スタイルは適用されません。

SpinnerList コントロールでは、accentColor、backgroundAlpha、backgroundColor および chromeColor スタイルプロパティはサポートされません。

スピナーリストのカスタムスキンの作成

SpinnerList コントロールまたは SpinnerListContainer コントロールのカスタムスキンを作成できます。通常、そうするには、カスタムスキンクラスのベースとして SpinnerListSkin または SpinnerListContainerSkin のソースをコピーします。

通常は、カスタム SpinnerList スキンを作成して、SpinnerList コントロールやそのコンテナの次の点を変更します。

- 現在選択されているアイテム（selectionIndicator）を囲むボックスのサイズまたはシェイプを変更します。これは、カスタム SpinnerListContainerSkin クラスを作成して行います。
- 各行の高さ（rowHeight）を定義します。これは、カスタム SpinnerListSkin クラスを作成して行います。
- 表示される行数（requestedRowCount）を定義します。これは、カスタム SpinnerListSkin クラスを作成して行います。
- コンテナの外観（角丸の半径やボーダーの太さなど）を定義します。これは、カスタム SpinnerListContainerSkin クラスを作成して行います。

カスタム SpinnerListSkin およびカスタム SpinnerListContainerSkin の例については、98 ページの「DateSpinner コントロールの外観のカスタマイズ」を参照してください。

スピナーリストでの画像の使用

SpinnerList コントロールでは、テキストラベルの代わりに、画像を使用できます。そうするには、SpinnerList のアイテムレンダラーとして IconItemRenderer を定義します。

IconItemRenderer オブジェクトで画像を使用するには、画像を埋め込むか、実行時に画像を読み込みます。モバイルユーザーのために、画像を埋め込んでデータのネットワーク利用を最小限に抑える方法が適切である場合があります。

次の例では、SpinnerList コントロールで埋め込み画像を使用します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListEmbeddedImage.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Embedded Images">

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
            [Embed(source="../../assets/product_icons/flex_50x50.gif")]
            [Bindable]
            public var icon0:Class;
            [Embed(source="../../assets/product_icons/acrobat_reader_50x50.gif")]
            [Bindable]
            public var icon1:Class;
            [Embed(source="../../assets/product_icons/coldfusion_50x50.gif")]
            [Bindable]
            public var icon2:Class;
            [Embed(source="../../assets/product_icons/flash_50x50.gif")]
            [Bindable]
            public var icon3:Class;
            [Embed(source="../../assets/product_icons/flash_player_50x50.gif")]
            [Bindable]
            public var icon4:Class;
            [Embed(source="../../assets/product_icons/photoshop_50x50.gif")]
            [Bindable]
            public var icon5:Class;

            // Return an ArrayList of icons for each spinner
            private function getIconList():ArrayList {
                var a:ArrayList = new ArrayList();
                a.addItem({icon:icon0});
                a.addItem({icon:icon1});
                a.addItem({icon:icon2});
                a.addItem({icon:icon3});
                a.addItem({icon:icon4});
                a.addItem({icon:icon5});
                return a;
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="productList1" width="90" dataProvider="{getIconList()}" selectedIndex="0">
            <s:itemRenderer>
                <fx:Component>
```

```
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
        <s:SpinnerList id="productList2" width="90" dataProvider="{getIconList()}" selectedIndex="2">
            <s:itemRenderer>
                <fx:Component>
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
        <s:SpinnerList id="productList3" width="90" dataProvider="{getIconList()}" selectedIndex="1">
            <s:itemRenderer>
                <fx:Component>
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
    </s:SpinnerListContainer>
</s:View>
```

次の図は、モバイルデバイス上のこのアプリケーションの表示を示しています。

埋め込み画像を使用した SpinnerList コントロール

第 4 章：アプリケーションのデザインとワークフロー

モバイルアプリケーションでのパーシスタンスの有効化

モバイルデバイスのアプリケーションは、テキストメッセージ、電話の呼び出し、他のモバイルアプリケーションなど、他のアクションによって頻繁に中断されます。ユーザーは通常、中断されたアプリケーションが再起動されたときに中断前のアプリケーションの状態が復元されることを期待します。パーシスタンスメカニズムを使用すると、デバイスでは、アプリケーションの前の状態を復元できます。

Flex フレームワークには、モバイルアプリケーションで使用できるパーシスタンスが 2 種類用意されています。インメモリパーシスタンスでは、ユーザーがアプリケーションをナビゲートしたときにビューのデータが保存されます。セッションパーシスタンスでは、ユーザーがアプリケーションを終了して再起動した場合にデータが復元されます。セッションパーシスタンスは、モバイルアプリケーションでは重要です。これは、モバイル用のオペレーティングシステムはアプリケーションをいつでも（メモリが少ない場合など）終了できるためです。

ブロガーの Steve Mathews 氏が、Simple data persistence in a Flex mobile application にクックブックエントリを公開しています。

ブロガーの Holly Schinsky 氏が、Flex Mobile Data Handling でパーシスタンスとデータ処理に関するブログを公開しています。

インメモリパーシスタンス

View コンテナでは、View.data プロパティを使用してインメモリパーシスタンスがサポートされています。既存のビューの data プロパティは、選択セクションが変更された場合や、新規ビューが ViewNavigator スタックにプッシュされて既存のビューが破棄された場合に、自動的に保存されます。制御がそのビューに戻り、ビューが再インスタンス化されてアクティブ化されると、ビューの data プロパティが復元されます。したがって、インメモリパーシスタンスを使用すると、実行時にビューの状態情報を維持できます。

セッションパーシスタンス

セッションパーシスタンスでは、アプリケーションの実行間で、アプリケーションの状態情報が維持されます。セッションパーシスタンスを実装するには、ViewNavigatorApplication コンテナと TabbedViewNavigatorApplication コンテナで persistNavigatorState プロパティを定義します。persistNavigatorState を true に設定すると、セッションパーシスタンスを有効にできます。デフォルトでは、persistNavigatorState は false です。

セッションパーシスタンスを有効にすると、FxAppCache というローカル共有オブジェクトを使用して、アプリケーションの状態がディスクに書き込まれます。アプリケーションで spark.managers.PersistenceManager のメソッドを使用して、このローカル共有オブジェクトに追加情報を書き込むこともできます。

ViewNavigator のセッションパーシスタンス

ViewNavigator コンテナでは、アプリケーションの終了時にビュースタックの状態をディスクに保存することにより、セッションパーシスタンスがサポートされています。この保存には、現在の View の data プロパティが含まれます。

アプリケーションが再起動されると、ViewNavigator のスタックが再初期化されて、アプリケーションの終了時に表示されていたものと同じビューとコンテンツが表示されます。スタックには各ビューの data プロパティのコピーが格納されているため、ビューがアクティブになったときに、スタックにある以前のビューを再作成できます。

TabbedViewNavigator のセッションパーシスタンス

TabbedViewNavigator コンテナの場合、アプリケーションの終了時に、セッションパーシスタンスではタブバーで現在選択されているタブを保存します。タブは、ViewNavigator とタブを定義するビュースタックに対応します。この保存には、現在の View の data プロパティが含まれます。したがって、アプリケーションが再起動されると、アクティブなタブとそれに関連付けられている ViewNavigator が、アプリケーション終了時に設定されていた状態に設定されます。

注意：TabbedViewNavigatorApplication コンテナによって定義されるアプリケーションの場合は、現在の ViewNavigator のスタックのみが保存されます。したがって、アプリケーションが再起動されると、現在の ViewNavigator の状態のみが復元されます。

セッションパーシスタンスのデータの表示

Flex で使用されるパーシスタンスメカニズムでは暗号化も保護もされません。したがって、保持されたデータは、別のプログラムやユーザーが解析できる形式で格納されます。ユーザー認証情報などの重要な情報は、このメカニズムを使用して保持しないでください。より保護機能の高い独自のパーシスタンスマネージャーを作成することもできます。詳しくは、118 ページの「パーシスタンスメカニズムのカスタマイズ」を参照してください。

セッションパーシスタンスの使用

次の例では、persistNavigatorState プロパティを true に設定して、アプリケーションのセッションパーシスタンスを有効にしています。

```
<?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>
```

このアプリケーションでは、EmployeeMainView.mxml を最初のビューとして使用しています。EmployeeMainView.mxml では、ユーザー名を選択できるようにする List コントロールが定義されています。

```
<?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>
```

セッションパーシスタンスを表示するには、アプリケーションを開き、List コントロールの「Dave」を選択して、EmployeeView.mxml ビューにナビゲートします。

```
<?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>
```

EmployeeView.mxml ビューに、「Dave」に関するデータが表示されます。次に、アプリケーションを終了します。アプリケーションを再起動すると、アプリケーションを終了したときと同じデータが表示された EmployeeView.mxml ビューが再び表示されます。

ローカル共有オブジェクト内のデータへのアクセス

ローカル共有オブジェクト内の情報はキー値のペアとして保存されています。setPropery() や getProperty() などの PeristenceManager のメソッドは、このキーを使用して、ローカル共有オブジェクト内の関連する値にアクセスします。

setProperty() メソッドを使用すると、独自のキーと値のペアをローカル共有オブジェクトに書き込むことができます。setProperty() メソッドには次のシグニチャがあります。

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

getProperty() メソッドを使用すると、特定のキーの値にアクセスできます。getProperty() メソッドには次のシグニチャがあります。

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

persistNavigatorState プロパティが true のときは、パーシスタンスマネージャーによって、アプリケーションの終了時に次の 2 つのキーと値のペアがローカル共有オブジェクトに自動的に保存されます。

- applicationVersion

  application.xml ファイルに記述されているアプリケーションのバージョン。

- navigatorState

  現在の ViewNavigator のスタックに対応するナビゲーターのビューステート。

手動パーシスタンスの実行

persistNavigatorState プロパティが true の場合は、Flex によってセッションパーシスタンスが自動的に実行されます。persistNavigatorState プロパティが false のときにも、アプリケーションデータは保持できます。その場合は、PeristenceManager のメソッドを使用して、独自のパーシスタンスメカニズムを実装してください。

setProperty() メソッドと getProperty() メソッドを使用すると、ローカル共有オブジェクトに対して情報の書き込みおよび読み取りができます。load() メソッドを呼び出すと、PeristenceManager を初期化できます。save() を呼び出すと、データをディスクに書き込めます。

注意：persistNavigatorState プロパティが false の場合、Flex では、現在の ViewNavigator のビュースタックが、アプリケーションの終了時に自動的に保存されることも、アプリケーションの起動時に復元されることもありません。

パーシスタンスイベントの処理

カスタムのパーシスタンスメカニズムを開発するには、モバイルアプリケーションコンテナの次のイベントを使用します。

- navigatorStateSaving
- navigatorStateLoading

navigatorStateSaving イベントのハンドラーの preventDefault() メソッドを呼び出すと、アプリケーションの状態のディスクへの保存をキャンセルできます。navigatorStateLoading イベントのイベントハンドラーの preventDefault() メソッドを呼び出すと、再起動時のアプリケーションの読み込みをキャンセルできます。

パーシスタンスメカニズムのカスタマイズ

セッションパーシスタンスを有効にすると、アプリケーションを終了したときに表示されていたビューが表示されます。アプリケーションの状態を完全に復元できるように、ビューの data プロパティまたは共有オブジェクトなどの他の場所に十分な情報を保存する必要があります。

例えば、復元されたビューで、ビューの data プロパティに基づいて計算を実行する必要がある場合があります。次に、アプリケーションで、アプリケーションの再起動を認識して、必要な計算を実行する必要があります。1 つの方法として、View の serializeData() メソッドと deserializePersistedData() メソッドをオーバーライドして、アプリケーションの終了または再起動時に独自のアクションを実行することもできます。

セッションパーシスタンスのプリセットのデータ型のサポート

パーシスタンスメカニズムでは、Number、String、Array、Vector、Object、uint、int および Boolean などのすべてのプリセットのデータ型が自動的にサポートされます。これらのデータ型は、パーシスタンスメカニズムによって自動的に保存されます。

セッションパーシスタンスのカスタムクラスサポート

多くのアプリケーションでは、カスタムクラスを使用して、データが定義されています。プリセットのデータ型によって定義されたプロパティがカスタムクラスに含まれている場合は、パーシスタンスメカニズムによって、自動的にクラスを保存および読み込みできます。ただし、flash.net.registerClassAlias() メソッドを呼び出して、まずパーシスタンスメカニズムにクラスを登録する必要があります。通常は、パーシスタンスストレージが初期化されるか、パーシスタンスストレージにデータが保存される前に、アプリケーションの preinitialize イベントで、このメソッドを呼び出します。

プリセットのデータ型以外のデータ型を使用する複雑なクラスを定義する場合は、String などのサポートされている型にそのデータを変換する必要があります。また、プライベート変数をクラスで定義している場合は、プライベート変数も自動的には保存されません。パーシスタンスメカニズムで複雑なクラスをサポートするには、flash.utils.IExternalizable インターフェイスをクラスに実装する必要があります。このインターフェイスでは、writeExternal() メソッドと readExternal() メソッドをクラスに実装して、クラスのインスタンスの保存と復元を行う必要があります。

モバイルアプリケーションでの複数のスクリーンサイズと DPI 値のサポート

複数のスクリーンサイズと DPI 値をサポートするためのガイドライン

プラットフォームに依存しないアプリケーションを開発するには、様々な出力デバイスがあることに注意する必要があります。デバイスには様々なスクリーンサイズや解像度、および様々な DPI 値や密度があります。

Flex エンジニアの Jason SJ が、解像度に依存しないモバイルアプリケーションを作成するための 2 つのアプローチについてブログで説明しています。

用語集

解像度とは、高さのピクセル数 x 幅のピクセル数のことで、デバイスがサポートする合計ピクセル数のことです。

DPI とは、平方インチ当たりのドット数のことで、デバイススクリーンのピクセルの密度のことです。DPI という用語は、PPI（pixels per inch）と同じ意味です。

DPI に対する Flex のサポート

Flex の次の機能を使用すると、解像度や DPI に依存しないアプリケーションを作成するプロセスを簡素化できます。

スキン モバイルコンポーネントのための DPI を認識するスキン。デフォルトのモバイルスキンでは、多くのデバイスの解像度に合わせて拡大／縮小するための追加のコーディングが必要ありません。

applicationDPI カスタムスキンのデザインサイズを定義するプロパティ。このプロパティを特定の DPI に設定して、ユーザーが異なる DPI 値のデバイスでアプリケーションを実行したとします。Flex では、使用中のデバイスの DPI に合わせてアプリケーションのすべての内容が拡大／縮小されます。

デフォルトのモバイルスキンは DPI に依存しないスキンです。DPI スケーリングを使用する場合もしない場合もあります。したがって、静的なサイズまたはカスタムスキンのコンポーネントを使用しない場合は、通常、applicationDPI プロパティを設定する必要はありません。

動的なレイアウト

異なる解像度に対応するには、動的なレイアウトを使用するのが効果的です。例えば、コントロールの幅を 100 %に設定すると、解像度が 480x854 の場合も 480x800 の場合も、解像度に関係なく常にスクリーンの幅全体に表示されるようになります。

applicationDPI プロパティの設定

密度に依存しないアプリケーションを作成する際には、ルートアプリケーションタグにターゲット DPI を設定できます。（モバイルアプリケーションの場合、ルートタグは、<s:ViewNavigatorApplication>、<s:TabbedViewNavigatorApplication> または <s:Application> です。）

applicationDPI プロパティの値は、ターゲットデバイスのおおよその解像度に従って、160、240 または 320 に設定します。次に例を示します。

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

applicationDPI プロパティを設定すると、実行時にターゲットデバイスの実際の解像度（runtimeDPI）と比較される際に、アプリケーションに対する拡大／縮小が効率的に定義されます。例えば、applicationDPI プロパティを 160 に設定し、ターゲットデバイスの runtimeDPI が 160 の場合、スケール比率は 1（拡大／縮小なし）となります。applicationDPI プロパティを 240 に設定した場合、スケール比率は 1.5（すべての表示を 150 %に拡大）になります。320 の場合、スケール比率は 2 になるので、すべての表示が 200 %に拡大されます。

スケール比率が整数でない場合、補間により、線が不明確になるなどの好ましくないアーティファクトが表示される場合があります。

DPI スケーリングの無効化

アプリケーションの DPI スケーリングを無効にする場合は、applicationDPI プロパティの値を設定しません。

applicationDPI と runtimeDPI について

次の表では、異なる解像度でアプリケーションを操作する際に必要な Application クラスの 2 つのプロパティについて説明します。

プロパティ	説明
applicationDPI	アプリケーションのターゲット解像度または DPI。 このプロパティに値を指定すると、ルートアプリケーションにスケール比率が適用されます。そのため、1 つの DPI 値に対してデザインされたアプリケーションが、DPI 値が異なる別のデバイスでも美しく表示されるように拡大／縮小されます。 スケール比率は、このプロパティと runtimeDPI プロパティの値を比較して計算されます。このスケール比率は、プリローダー、ポップアップおよびステージ上のすべてのコンポーネントを含めたアプリケーション全体に適用されます。 値が指定されていない場合、このプロパティは runtimeDPI プロパティと同じ値を返します。 このプロパティは、ActionScript では設定できません。MXML でのみ設定できます。このプロパティの値を実行時に変更することはできません。
runtimeDPI	アプリケーションを現在実行しているデバイスの解像度または DPI 値。 DPIClassification クラスで定義された定数の 1 つに端数処理した Capabilities.screenDPI プロパティの値を返します。 このプロパティは読み取り専用です。

解像度や DPI に依存しないアプリケーションの作成

解像度や DPI に依存しないアプリケーションは、次の特性を備えています。

画像 ベクトル形式の画像は、ターゲットデバイスの実際の解像度に合わせてスムーズに拡大／縮小されます。一方、ビットマップの場合はスムーズに拡大／縮小できない可能性があります。このような場合は、MultiDPIBitmapSource クラスを使用して、デバイスの解像度に応じた様々な解像度のビットマップを読み込めます。

テキスト テキストのフォントサイズ（テキスト自体ではなく）は、解像度に合わせて拡大／縮小されます。

レイアウト 動的レイアウトを使用して、拡大／縮小時にアプリケーションが適切に表示されるようにします。一般に、絶対値を使用してピクセル境界を指定するような制約のあるレイアウトの使用は避けてください。制約を使用する必要がある場合は、applicationDPI プロパティの値を使用して拡大／縮小を考慮します。

拡大／縮小 scaleX および scaleY プロパティを Application オブジェクトで使用しないでください。applicationDPI プロパティを設定すると、Flex で拡大／縮小が処理されます。

スタイル スタイルシートを使用して、ターゲットデバイスの OS およびアプリケーションの DPI 設定のスタイルプロパティをカスタマイズできます。

スキン Flex のモバイルテーマのスキンは、アプリケーションの DPI 値を使用して、実行時に使用するアセットを判断します。FXG ファイルで定義された視覚的なスキンアセットはすべて、ターゲットデバイスに合うように調整されます。

アプリケーションサイズ アプリケーションの高さと幅は明示的に設定しないでください。また、カスタムコンポーネントまたはポップアップのサイズを計算する場合は、stageWidth プロパティと stageHeight プロパティを使用しないでください。代わりに、SystemManager.screen プロパティを使用します。

実行時 DPI の判断

アプリケーションが起動されると、アプリケーションは Capabilities.screenDPI という Flash Player のプロパティから runtimeDPI プロパティの値を取得します。このプロパティは、DPIClassification クラスで定義された定数の 1 つにマップされます。例えば、232 DPI で動作している Droid は、240 の実行時 DPI 値にマップされます。デバイスの DPI 値は、DPIClassification 定数（160、240 または 320）といつも正確に一致するわけではありません。ターゲット値の範囲に基づいて、定数の分類にマップされます。

次のようにマップされます。

DPIClassification 定数	160 DPI	240 DPI	320 DPI
デバイスの実際の DPI	<200	>=200 かつ <280	>=280

これらのマッピングをカスタマイズして、デフォルトの動作をオーバーライドしたり、DPI 値を正しく取得できないデバイスを調整することができます。詳しくは、128 ページの「デフォルト DPI のオーバーライド」を参照してください。

自動サイズ変更の有無の選択

（applicationDPI プロパティの値を設定して）自動サイズ変更を使用するかどうかは、便利さを優先するか、ピクセルレベルで視覚的な忠実度を優先するかの兼ね合いで決めます。applicationDPI プロパティを設定して、アプリケーションを自動サイズ変更する場合は、applicationDPI をターゲットとするスキンが使用されます。スキンは、デバイスの実際の密度に合うように拡大／縮小されます。アプリケーションの他のアセットとレイアウト位置も拡大／縮小されます。

自動サイズ変更を使用する場合に、1 つの DPI 値をターゲットとするスキンやアセットを作成している場合は、通常次のようにします。

- 指定する applicationDPI をターゲットとするスキンとビュー／コンポーネントのレイアウトのセットを 1 つ作成します。
- アプリケーションのスキンまたは他の場所で使用するビットマップアセットの複数のバージョンを作成し、MultiDPIBitmapSource クラスを使用して指定します。自動サイズ変更を使用する場合、スキンのベクトルアセットとテキストで密度を意識する必要はありません。
- アプリケーションでは 1 つのターゲット DPI 値のみが考慮されているので、@media ルールをスタイルシートで使用しないでください。
- 様々な密度のデバイスでアプリケーションをテストして、拡大／縮小されたアプリケーションが各デバイスで適切に表示されることを確認します。特に、整数ではない比率で拡大／縮小されるデバイスを確認してください。例えば、applicationDPI が 160 の場合は、240 DPI のデバイスでアプリケーションをテストしてください。

自動サイズ変更をしない（applicationDPI プロパティを設定しない）場合は、applicationDPI の値を取得します。このプロパティを使用して、デバイスの実際の DPI の分類を判断し、次のようにして実行時にアプリケーションを適合させます。

- 実行時 DPI の各分類を対象にして、スキンとレイアウトのセットを複数作成するか、様々な密度に動的に適合するスキンとレイアウトのセットを 1 つ作成します。（Flex プリセットのスキンでは、後者の方法を採用しています。各スキンクラスが、applicationDPI プロパティを確認して自分自身を適切に設定します。）
- @media ルールをスタイルシートで使用すると、デバイスの DPI の分類に基づいて CSS ルールにフィルターをかけることができます。通常は、各 DPI 値に対して、フォントサイズとパディング値をカスタマイズします。

- 様々な密度のデバイスでアプリケーションをテストして、スキンとレイアウトが正しく調整されることを確認します。

DPI に基づいたスタイルの選択

Flex では、CSS を使用して、ターゲットの OS とアプリケーション DPI 値に基づいたスタイルを適用できます。スタイルシートの @media ルールを使用してスタイルを適用します。@media ルールは CSS 仕様の一部です。Flex ではこのルールを拡張し、application-dpi および os-platform という追加のプロパティを挿入します。これらのプロパティを使用して、アプリケーションの DPI とアプリケーションが実行されているプラットフォームに基づいて、スタイルを選択的に適用します。

次の例では、Spark Button コントロールのデフォルトの fontSize スタイルプロパティを 12 に設定します。デバイスで 240 DPI が使用され、Android オペレーティングシステムで実行されている場合、fontSize プロパティは 10 となります。

```
<?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>
```

application-dpi プロパティの値

CSS の application-dpi プロパティは、ルートアプリケーションに設定されている applicationDPI スタイルプロパティの値と比較されます。CSS の application-dpi プロパティに有効な値は次のとおりです。

- 160
- 240
- 320

application-dpi でサポートされている各値には、DPIClassification クラスに対応する定数があります。

os-platform プロパティの値

CSS の os-platform プロパティは、Flash Player の flash.system.Capabilities.version プロパティの値に相当します。CSS の os-platform プロパティに有効な値は次のとおりです。

- Android
- iOS
- Macintosh
- Linux
- QNX
- Windows

大文字と小文字が異なっても一致します。

どのエントリも一致しない場合、サポートされるプラットフォームのリストの最初の 3 文字を比較して、二次的な一致を探します。

application-dpi および os-platform プロパティのデフォルト

application-dpi プロパティまたは os-platform プロパティが含まれている式を明示的に定義していない場合は、すべての式が一致していると見なされます。

@media ルールの演算子

@media ルールには「and」および「not」の一般的な演算子を使用できます。カンマ区切りのリストもサポートされています。式をカンマで区切ると、「or」条件になります。

「not」演算子使用する際には、「not」を式の最初のキーワードにする必要があります。この演算子は、「not」に続くプロパティだけでなく式全体を否定します。バグ SDK-29191 があるので、「not」演算子の後には、「all」などのメディアタイプを式の前に指定する必要があります。

次の例は、これらの一般的な演算子の使用方法を示しています。

```
<?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>
```

DPI に基づいたビットマップアセットの選択

通常、ビットマップ画像のアセットは、デザインされている解像度でのみ最適にレンダリングされます。この制限により、複数の解像度を対象としたアプリケーションをデザインする場合は、多少の困難を伴う可能性があります。解決策としては、複数（異なる解像度ごとに 1 つ）のビットマップを作成して、アプリケーションの runtimeDPI プロパティの値に従って、適切なビットマップを読み込みます。

Spark の BitmapImage コンポーネントと Image コンポーネントには、Object タイプの source プロパティがあります。このプロパティがあるので、どのアセットを使用するかを定義したクラスを渡すことができます。この場合は、runtimeDPI プロパティの値に従って異なるソースをマップするように、MultiDPIBitmapSource クラスを渡します。

次の例では、DPI に従って異なる画像を読み込みます。

```
<?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>
```

デスクトップアプリケーションで MultiDPIBitmapSource が指定されている BitmapImage クラスおよび Image クラスを使用すると、ソースには source160dpi プロパティが使用されます。

Button コントロールの icon プロパティにも、クラスが引数として使用されます。したがって、MultiDPIBitmapSource オブジェクトを Button のアイコンのソースとして使用することもできます。アイコンのソースは、次の例に示すようにインラインで定義できます。

```
<?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).toString();
            }
        ]]>
    </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>
```

次の例に示すように、<fx:Declarations> ブロックでアイコンを宣言し、データバインディングを使用してソースを割り当てて、アイコンを定義することもできます。

```
<?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>
```

runtimeDPI プロパティが null または空の文字列（""）の source***XXX***dpi プロパティにマップしている場合、Flash Player では、次に高い解像度のプロパティがソースとして使用されます。その値も null または空の場合は、次に低い解像度が使用されることになります。その値も null または空の場合は、null がソースとして割り当てられて、画像は表示されません。つまり、画像を特定の DPI で表示しないという明示的な指定はできません。

DPI に基づいたスキンアセットの選択

デフォルトモバイルスキンのコンストラクターのロジックでは、applicationDPI プロパティの値に基づいてアセットが選択されます。これらのクラスでは、ターゲット DPI 値に最も近いアセットが選択されます。DPI スケーリングを使用する場合もしない場合も機能するカスタムスキンをデザインする場合は、applicationDPI プロパティを使用し、runtimeDPI プロパティは使用しません。

例えば、spark.skins.mobile.ButtonSkin クラスでは、次のように、特定の DPI 値用にデザインされている FXG アセットを選択する switch/case 文を使用します。

```
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;
    }
}
```

FXG アセットを条件付きで選択することに加え、モバイルのスキンクラスでは、レイアウトの間隔やレイアウトのパディングなど、他のスタイルプロパティ値も設定します。これらの設定は、ターゲットデバイスの DPI に基づいています。

applicationDPI を設定しない場合

applicationDPI プロパティを設定していない場合、スキンは runtimeDPI プロパティを使用するようにデフォルト設定されます。このメカニズムでは、runtimeDPI プロパティではなく、applicationDPI プロパティの値に基づくスキンに対して、DPI スケーリングを使用する場合もしない場合も適切なリソースが使用されるようになります。

カスタムスキンを作成している場合は、applicationDPI 設定を無視するように選択できます。その結果、ターゲットデバイスの DPI に合うように拡大／縮小されるスキンになります。ただし、アセットがその DPI 値用に特にデザインされていない場合は、最適な表示にならない可能性があります。

CSS の applicationDPI を使用する場合

カスタムスキンを作成せずに、CSS の @media セレクターで applicationDPI プロパティの値を使用して、モバイルアプリケーションやタブレットアプリケーションで使用されるスタイルをカスタマイズします。詳しくは、122 ページの「DPI に基づいたスタイルの選択」を参照してください。

スケール比率と現在の DPI の手動による判断

モバイルアプリケーションやタブレットアプリケーションがターゲットデバイスの DPI 値に基づいて選択するアセットを手動で指定するには、実行時にスケール比率を計算します。スケール比率は、runtimeDPI プロパティの値を applicationDPI スタイルプロパティの値で除算して計算します。

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

計算したスケール比率を使用して、アセットを手動で選択できます。次の例では、ビットマップアセットの場所を DPI 値ごとに定義します。次に、その場所から画像を読み込みます。

```
<?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>
```

スケール比率を使用するビューは次のとおりです。

```
<?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>
```

デフォルト DPI のオーバーライド

アプリケーションの DPI 値を設定すると、実行しているデバイスからの DPI 値に基づいてアプリケーションが拡大／縮小されます。場合によっては、デバイスから不適切な DPI 値が報告されることや、カスタムの拡大／縮小方法を採用してデフォルトの DPI 選択方法をオーバーライドすることがあります。

デフォルトの DPI マッピングをオーバーライドすることで、アプリケーションのデフォルトの拡大／縮小動作をオーバーライドできます。例えば、デバイスから 160 ではなく 240 の DPI であると不適切な情報が届く場合は、このデバイスを見つけて、160 の DPI として分類するカスタムマッピングを作成できます。

特定のデバイスの DPI 値をオーバーライドするには、Application クラスの runtimeDPIProvider プロパティが RuntimeDPIProvider クラスのサブクラスを指し示すようにします。サブクラスで、runtimeDPI という getter をオーバーライドし、カスタム DPI マッピングを提供するロジックを追加します。UIComponent などのフレームワークの他のクラスに依存関係を追加しないでください。このサブクラスは Player API にのみ呼び出すことができます。

次の例では、Capabilities.os プロパティが「Mac 10.6.5」に該当するデバイスについて、カスタム DPI マッピングを設定します。

```
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;
    }
}
}
```

次のアプリケーションでは、DPITestClass を使用して、拡大／縮小に使用する実行時の DPI 値を決定しています。この例では、ViewNavigatorApplication クラスの runtimeDPIProvider プロパティを指しています。

```
<?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>
```

次に、RuntimeDPIProvider クラスのサブクラスについて別の例を示します。この例では、カスタムクラスがデバイスのスクリーンの幅と高さの解像度をチェックして、デバイスが不適切な DPI 値を報告しているかどうかを判断します。

```
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;
    }
}
}
```

第 5 章：テキスト

モバイルアプリケーションでのテキストの使用

モバイルアプリケーションでのテキストに対するガイドライン

Spark のテキストコントロールの中には、モバイルアプリケーションで使用するために最適化されたコントロールがあります。可能な場合は、次のテキストコントロールを使用してください。

- Spark TextArea
- Spark TextInput
- Spark Label

ユーザーインタラクションが可能なテキストコントロール（TextArea および TextInput）では、基盤となる入力メカニズムとして StageText クラスを使用します。StageText は、基盤となる OS のネイティブテキストコントロールにフックされます。そのため、これらのテキストコントロールは、通常の Flex のコントロールではなく、ネイティブコントロールのように動作します。

デバイスでサポートされている場合に StageText を使用すると次のような利点があります。

- ソフトキーボードのネイティブのパフォーマンスとルックアンドフィール
- 自動補完
- 自動訂正
- タッチベースのテキスト選択
- カスタマイズ可能なソフトキーボード
- キー制限

アドビのエバンジェリスト Christian Cantrell が StageText ベースのコントロールを使用する場合の長所と短所について説明しています。

TextField ベースのバージョンの TextArea コントロールおよび TextInput コントロールも使用できます。これらのバージョンを使用してフォントを埋め込んだり、StageText ベースのバージョンでは使用できない他の機能を使用したりできます。

テキストコントロール用のスキン

モバイルアプリケーションを作成するときは、常にモバイルテーマが自動的に適用されます。そのため、Spark の TextInput コントロールおよび TextArea コントロールでは、次の StageText ベースのモバイルスキンがデフォルトで使用されます。

- StageTextAreaSkin
- StageTextInputSkin

StageTextAreaSkin クラスおよび StageTextInputSkin クラスはモバイルアプリケーション用に最適化されており、StageTextSkinBase クラスに基づいています。これらのスキンは、ネイティブテキスト入力クラスのラッパーとして機能します。ただし、TextField ベースでないスキンの次の機能はサポートしていません。

TextField ベースのコントロールではサポートされる機能	TextField ベースのコントロールでもサポートされない機能
フォームのスクロール テキスト測定 クリッピング 埋め込みフォント 分数のアルファ値 Flash Text Engine（FTE） keyUp や keyDown などの低レベルのキーボードイベントへのアクセス	Layout Framework（TLF）テキスト 双方向性とミラーリング Compact Font Format（CFF） テキストレンダリングに対する RichEditableText HTML テキスト

これらの制限の一部は、TextField ベースのバージョンを使用すれば回避できます。TextField ベースのバージョンのテキスト入力コントロールを使用するには、次の例に示すように、スキンクラスが TextField バージョン（TextInputSkin および TextAreaSkin）を指すようにします。

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin" text="TextField-based Skin"/>
```

Spark の Label コントロールではスキンを使用しませんが、TLF も使用しません。

モバイルアプリケーションでの TLF

通常、モバイルアプリケーションでは、Text Layout Framework（TLF）を使用するテキストコントロールの使用を避けてください。TextArea および TextInput コントロールのモバイルスキンはモバイルアプリケーション用に最適化されているので、デスクトップ用および Web ベースのこれらのコントロールの場合のように TLF を使用しないでください。TLF は、テキストをレンダリングするための豊富なコントロールを実現するアプリケーションで使用します。

次のテキストコントロールは TLF を使用しており、モバイルアプリケーション用に最適化されていないので、モバイルアプリケーションでは使用しないでください。

- Spark RichText
- Spark RichEditableText

ソフトキーボードでの入力

ユーザーが入力用のテキストコントロールをフォーカスすると、キーボードがないモバイルデバイスの場合はソフトキーボードが表示されます。ソフトキーボードで使用できるキーおよび他のプロパティはある程度制御できます。例えば、自動訂正と自動キャピタライズを有効にして、事前に決定されている数種類のキーボードレイアウトの中から選択できます。

詳しくは、141 ページの「モバイルアプリケーションでのソフトキーボードの使用」を参照してください。

テキスト入力コントロールでのスクロール

TextInput コントロールおよび TextArea コントロールのデフォルトのモバイルスキンでは、フォームのスクロールはサポートされません。つまり、コントロールをスクロールする必要があるフォームやビューでは、これらのコントロールを表示できません。表示すると、コントロールが実装されている方法がビジュアルアーティファクトとして表示されます。

スクロールするコンテナでテキスト入力コントロールを使用するには、StageText ベースのスキンではなく、TextField ベースのスキンを使用してください。詳しくは、65 ページの「StageText でのスクロールの考慮事項」を参照してください。

テキスト入力コントロールでのトランジション

アニメーションを滑らかにするため、ランタイムは、アニメーションを再生するときには常に、StageText コントロールをコントロールからキャプチャしたビットマップに置き換えます。そのため、トランジションアニメーションの開始時にわずかな遅延が発生する場合があり、アニメーションの開始時と終了時に若干のビジュアルアーティファクトが発生することもあります。

わずかな遅延は、コンポーネント内のテキストのビットマップ表現をキャプチャするための時間です。StageText ベースのコントロールの範囲と数が増えると、この遅延は長くなります。遅延を減らすには、大きい、または多数の StageText ベースコンポーネントをアニメーション化しないようにします。

テキスト入力コントロールでのポップアップ

最上位のポップアップの外側にある StageText ベースのテキスト入力は、ポップアップが表示されるときに、ビットマップ表現に置き換えられます。結果として次のことがいえます。

- 非モーダルポップアップではなく、モーダルポップアップを使うようにします。モーダルポップアップが表示されると、ポップアップの外側のコンポーネントはインタラクティビティを失うものと予想されます。このような場合、StageText のビットマップへの置き換えはあまり目立ちません。
- StageText ベースのコンポーネントは、IFocusManagerContainer インターフェイスを実装しているポップアップの内部でのみ使用する必要があります。たとえば、SkinnableContainer またはその派生型の 1 つをポップアップの基礎として使用します。
- 下位レイヤー内のテキストコンポーネントをアクティブのままにする必要があるときは、Callout コンテナを使用します。テキストコンポーネントがコールアウトを所有している場合、テキストコンポーネントはアクティブのままになり、コールアウトの矢印がそのテキストコンポーネントを指すように設定します。これは、テキストコンポーネントをまだ使用できることの、ユーザーに対する自然な目印になります。
- 複数のポップアップを同時に使用しないようにします。複数のポップアップが同時に表示されると、最上位のポップアップだけがインタラクティブになります。しかし、ポップアップがオーバーラップしていない場合、ユーザーにはどのポップアップが最上位なのか判断できません。
- 非モーダルポップアップがテキストコントロールとオーバーラップするときは、ほぼ全体がオーバーラップするようにポップアップを配置します。テキストが見えなければ、テキストコントロールがまだインタラクティブであるとユーザーが考えることはまずありません。

モバイルアプリケーションでの埋め込みフォント

StageText を使用するテキスト入力コントロールには、埋め込みフォントを使用できません。代わりに、TextField ベースのスキンをテキスト入力コントロールに使用してください。Label コントロールも CFF ベースのフォントが必要な FTE を使用しているので、埋め込みフォントを使用できません。CFF フォントは、モバイルアプリケーションでのパフォーマンスがよくありません。

詳しくは、153 ページの「モバイルアプリケーションでの埋め込みフォント」を参照してください。

StageText コントロールクラスの階層

StageText クラス（TextInput および TextArea）には、複雑なクラス階層があります。基本クラス自体の階層は次のようになっています。

```
    UIComponent
        |
SkinnableComponent
        |
SkinnableTextBase
        |
TextInput/TextArea
```

Spark のすべてのクラスと同様に、スキンクラスには独自の階層があります。

```
    UIComponent
        |
    MobileSkin    StyleableStageText
        |              |
StageTextSkinBase: textDisplay
        |
StageTextInputSkin/StageTextAreaSkin
```

基本スキンクラスの textDisplay プロパティには、ネイティブテキスト入力へのフックが StyleableStageText オブジェクトとして用意されています。このクラスには、StageText ベースのテキスト入力コントロールで使用できるスタイルを定義する役割もあります。

モバイルアプリケーションでの Label コントロールの使用

Spark Label コントロールは、編集不可で選択不可の単一行のテキストに最適です。

次の例では、モバイルアプリケーションで簡単な• Label コントロールを使用しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleLabel.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Simple Label">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="This is a simple Label control."/>

</s:View>
```

Label コントロールは FTE を使用しますが、FTE は、TextInput、TextArea などモバイルアプリケーション用に最適化されているテキストコントロールよりも効率的ではありません。ただし、Label コントロールは TLF を使用しないので、通常は TLF を実装する RichText や RichEditableText などのコントロールより効率的です。

通常、Spark Label コントロールをモバイルアプリケーションで使用する場合は慎重に使用します。スキンやアイテムレンダラーでは、Spark Label コントロールを使用しないでください。ActionScript ベースのアイテムレンダラーを作成するときは、テキストのレンダリングに StyleableTextField クラスを使用します。MXML ベースのコンポーネントの場合でも、Label を使用できます。

Label コントロールは CFF を使用するので、モバイルアプリケーションでフォントを埋め込む場合には、Label コントロールを使用しないでください。代わりに、TextField ベースのバージョンの TextArea コントロールを使用してください。詳しくは、153 ページの「モバイルアプリケーションでの埋め込みフォント」を参照してください。

モバイルアプリケーションでの TextArea コントロールの使用

Spark TextArea コントロールは、ユーザーが複数行のテキストを入力および編集できるテキスト入力コントロールです。このクラスは、モバイルアプリケーション用に最適化されています。

TextArea コントロールのデフォルトの動作では、基盤となる OS のネイティブメソッドへのフックを提供するソフトキーボードが使用されます。そのため、自動訂正、自動補完およびソフトキーボードのカスタマイズなどの機能がサポートされます。

次の例では、モバイルアプリケーションで TextArea コントロールを使用しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleTextArea.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Simple TextArea">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            // Note the use of \n to add line feeds/carriage returns
            // and \" to add quotation marks.
            [Bindable]
           public var myText:String ="\"Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.\"\n\n\"Ut wisi enim ad minim
veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.\"";
        ]]>
    </fx:Script>

    <!-- Basic TextArea control with multiple lines of text. -->
    <s:TextArea id="myTA" height="75%" text="{myText}"
                paddingLeft="20" paddingTop="20"
                paddingRight="20" paddingBottom="20"/>

</s:View>
```

モバイルアプリケーションでは、TextArea コントロールのスキンとして、StageTextAreaSkin クラスがデフォルトで使用されます。このスキンは、テキストのレンダリングに、RichEditableText クラスではなく、StyleableStageText クラスを使用します。そのため、TextArea コントロールでは TLF がサポートされません。モバイル以外のスキンの TextArea コントロールで使用できるスタイルのサブセットのみがサポートされます。

インタラクティブではない複数行のテキストブロックを使用する必要がある場合は、TextArea コントロールの editable プロパティを false に設定します。（ランタイムで selectable プロパティが無効になります。）borderVisible プロパティを false に設定して、ボーダーを削除することもできます。contentBackgroundColor プロパティおよび contentBackgroundAlpha プロパティを設定すれば、背景色を変更できます。

次の例では、アプリケーションの背景に溶け込む、インタラクティブでないテキストブロックを作成しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/BlockOfText.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Block of Text">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <s:HGroup>
        <s:Image source="@Embed(source='../../assets/myImage.jpg')" width="30%"/>
        <!-- Create a multi-line block of text. -->
        <s:TextArea width="65%"
                editable="false"
                borderVisible="false"
                contentBackgroundColor="0xFFFFFF"
                contentBackgroundAlpha="0"
                height="400"
                text="Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat."/>
    </s:HGroup>

</s:View>
```

TextArea コントロールは TLF をサポートしていないので、textFlow、content または selectionHighlighting プロパティを使用できません。さらに、次のメソッドも使用できません。

- getFormatOfRange()
- setFormatOfRange()

モバイルアプリケーションでの TextInput コントロールの使用

Spark TextInput コントロールは、ユーザーが単一行のテキストを入力および編集できるテキスト入力コントロールです。このクラスは、モバイルアプリケーション用に最適化されています。

TextInput コントロールのデフォルトの動作では、基盤となる OS のネイティブメソッドへのフックを提供するソフトキーボードが使用されます。そのため、自動訂正、自動補完およびソフトキーボードのカスタマイズなどの機能がサポートされます。

次の例は、モバイルアプリケーションでプロンプトテキストとカスタムフォーカスリングを使用する TextInput コントロールを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleTextInput.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Simple TextInput">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout
            paddingTop="20"
            paddingLeft="20"
            paddingRight="20"/>
    </s:layout>

    <s:TextInput
        prompt="Enter text here"
        focusColor="green"
        focusThickness="5"
        focusAlpha=".1"/>
    <s:TextInput
        prompt="Enter text here, too"
        focusColor="red"
        focusThickness="5"
        focusAlpha=".1"/>
</s:View>
```

モバイルアプリケーションでは、TextInput コントロールのスキンとして StageTextInputSkin クラスがデフォルトで使用されます。このスキンは、テキストのレンダリングに、RichEditableText クラスではなく、StyleableStageText クラスを使用します。そのため、TextInput コントロールでは TLF がサポートされません。モバイル以外のスキンの TextInput コントロールで使用できるスタイルのサブセットのみがサポートされます。

モバイルアプリケーションでの RichText および RichEditableText コントロールの使用

モバイルアプリケーションでは、RichText および RichEditableText コントロールを使用しないことをお勧めします。これらのコントロールにはモバイルスキンが用意されておらず、モバイルアプリケーション用に最適化されていません。これらのコントロールを使用する場合は、TLF を使用するので処理が重くなります。

MX テキストコントロール

モバイルアプリケーションでは、MX の Text や Label などの MX テキストコントロールは使用できません。代わりに、Spark の同等機能を使用してください。

モバイルアプリケーションでのテキスト入力コントロールのスタイルの設定

TextInput および TextArea コントロールは、モバイルテーマのスタイルのサブセットのみをサポートします。これらのスタイルは、StyleableStageText クラスに定義されています。

モバイルアプリケーションの TextInput および TextArea でサポートされるのは次のスタイルのみです。

- color
- contentBackgroundAlpha
- contentBackgroundColor
- フォーカスリングのスタイル：focusAlpha、focusBlendMode、focusColor および focusThickness

- fontFamily
- fontStyle
- fontSize
- fontWeight
- locale
- パディングのスタイル：paddingBottom、paddingLeft、paddingRight および paddingTop
- showPromptWhenFocused
- textAlign

モバイルアプリケーションでは、Label コントロールで、これらのスタイルに加えて textDecoration スタイルがサポートされます。

fontFamily スタイルの使用

デフォルトのモバイルスキンでは、fontFamily プロパティにおいてフォントのカンマ区切りリストはサポートされません。代わりに 1 つのフォントのみを指定でき、デバイス上に存在するフォントとのマッピングがランタイムによって試みられます。

例えば、「Arial」を指定した場合、そのフォントが使用可能であれば、デバイスにおいて Arial でテキストがレンダリングされます。そのフォントを使用できない場合は、どの種類のフォントで代替するのが最善かをランタイムが推測して決定します。ランタイムで認識できないフォント名を指定すると、デバイスではデフォルトフォントでテキストがレンダリングされます。通常、モバイルデバイスのデフォルトは sans-serif フォントです。

_sans、_serif または _typewriter を指定すると、それぞれ sans-serif、serif または code フォントをモバイルデバイスで常に取得できます。

次の例は、fontFamily スタイルの値に基づいてテキストがどのようにレンダリングされるかを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/FontFamilyExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="The fontFamily style">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <s:TextInput prompt="This is _sans" fontSize="14" fontFamily="_sans"/>
    <s:TextInput prompt="This is _serif" fontSize="14" fontFamily="_serif"/>
    <s:TextInput prompt="This is _typewriter" fontSize="14" fontFamily="_typewriter"/>
    <s:TextInput prompt="This is Arial" fontSize="14" fontFamily="arial"/>
    <s:TextInput prompt="This is Times" fontSize="14" fontFamily="times"/>
    <s:TextInput prompt="This is Times New Roman" fontSize="14" fontFamily="Times New Roman"/>
    <!-- Try a gibberish font name to see what the device's default font is: -->
    <s:TextInput prompt="This is bugblatter" fontSize="14" fontFamily="bugblatter"/>
</s:View>
```

テキスト入力コントロールのカスタムモバイルスキンの作成

MXML と ActionScript を使用すれば、モバイルアプリケーションのテキスト入力コントロールの視覚的な外観と動作の一部を制御できます。例えば、TextArea と TextInput コントロールのボーダーのカラーを設定したり、ボーダーの外観を切り替えたりできます。テキストコントロールの特定の部分の外観を変更するには、カスタムスキンの作成が必要になる場合もあります。

モバイルテーマのデフォルトの TextInput スキンおよび TextArea スキンは、StageTextAreaSkin と StageTextInputSkin クラスで定義されています。これらのスキンは、ほとんどのレイアウトとクロームのロジックを StageTextSkinBase クラスから取得しています。

カスタムスキンを作成するには、新しい外観を定義するカスタム StageTextSkinBase クラスを作成します。次に、このカスタムクラスを拡張するカスタムの StageTextAreaSkin クラスまたは StageTextInputSkin クラスを作成します。

モバイルテーマのカスタムスキンの作成について詳しくは、154 ページの「モバイルのスキン適用の基本」を参照してください。

テキスト入力コントロールのキーの制限

テキスト入力コントロールに対して StageText ベースのスキンを使用する場合は、TextInput コントロールまたは TextArea コントロールの restrict プロパティを使用して、使用可能な文字を制限できます。

restrict プロパティのデフォルト値は null なので、デフォルトでは、ユーザーはすべての文字を入力できます。

restrict プロパティには、使用できる文字の文字列を指定できます。範囲を指定するには、ハイフン（-）を使用します。空白、カンマまたは他の文字を定義に含める場合以外は、これらの文字を使用して範囲を区切らないでください。次の例は、制限の構文のいくつかの使用例を示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/RestrictStrings.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Examples of restrict">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="20" paddingLeft="20" paddingRight="20"/>
    </s:layout>

    <s:TextInput prompt="Alpha-numeric only" restrict="a-zA-Z0-9"/>
    <s:TextInput prompt="Numbers only" restrict="0-9"/>
    <s:TextInput prompt="All chars, only uppercase alpha" restrict="^a-z"/>
    <!-- ASCII chars 32 (space) through 126 (tilde) only: -->
    <s:TextInput prompt="" restrict="\u0020-\u007E"/>
    <s:TextInput prompt="All chars but not the caret or hyphen" restrict="^\^\-"/>
</s:View>
```

restrict プロパティの文字列値は左から右に読み取られます。キャレット（^）に続く文字はすべて使用できなくなります。次に例を示します。

```
ta1.restrict = "A-Z^Q"; // All uppercase alpha characters, but exclude Q
```

文字列の最初の文字がキャレット（^）の場合、そのキャレットに続く文字以外はすべて使用可能になります。次に例を示します。

```
ta1.restrict = "^a-z"; // All characters, but exclude lowercase alpha
```

バックスラッシュ文字（円記号）を使用して、特殊文字をエスケープできます。例えば、キャレット文字の使用を制限するには、次のようにします。

```
ta1.restrict = "^\^"; // All characters, but exclude the caret
```

¥u を使用して ASCII のキーコードを入力できます。次に例を示します。

```
ta1.restrict = "\u0020-\u007E"; // ASCII chars 32 through 126 only
```

モバイルアプリケーションでのテキストに対するユーザーインタラクション

テキストコントロールでスワイプなどのジェスチャーを使用できます。次の例では、スワイプイベントをリスニングし、スワイプが発生した方向を示します。

```
<?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()">
    <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():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>
```

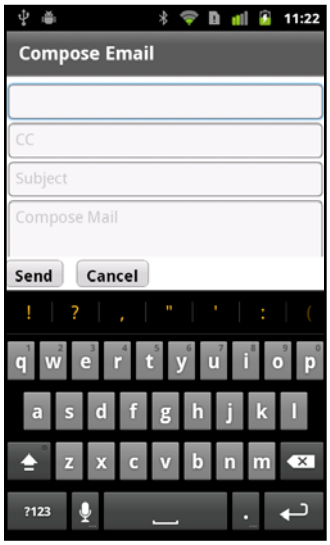
タッチしてドラッグするジェスチャーでは常にテキストが選択されます（テキストコントロールが選択可能または編集可能な場合のみ）。場合によっては、ユーザーがテキストコントロール上で、タッチしてドラッグまたはスワイプジェスチャーを実行したときに、テキストが選択されないようにする必要があります。この場合は、selectable プロパティと editable プロパティを false に設定するか、またはスワイプイベントハンドラーで selectRange(0,0) メソッドを呼び出して選択範囲をリセットします。

テキストが Scroller の内側にある場合、Scroller は、ジェスチャーがテキストコンポーネントの外側で行われている場合にのみスクロールします。

モバイルアプリケーションでのソフトキーボードの使用

多くのデバイスにはハードウェアキーボードがありません。ハードウェアキーボードがないデバイスでは、必要なときにスクリーン上に開くキーボードが使用されます。スクリーンキーボードまたは仮想キーボードとも呼ばれるソフトキーボードは、ユーザーが情報を入力した後や、ユーザーが操作をキャンセルしたときに閉じます。

次の図は、ソフトキーボードを使用したアプリケーションを示しています。



ソフトキーボードには、起動元のコンポーネントに基づく、異なる機能セットがあります。

- ネイティブ機能：デフォルトのテキスト入力コントロールである TextArea および TextInput で使用されるキーボードは、自動訂正、自動補完、カスタムキーボードレイアウトなどの機能のネイティブインターフェイスにフックされています。テキスト入力コントロールのデフォルトの StageText ベースのスキンクラスには、すべての機能セットのサポートが組み込まれています。すべてのネイティブ機能をサポートしていないデバイスもあります。
- 制限付き：TextArea および TextInput 以外のコントロールで使用されるか、TextArea および TextInput で TextField ベースのスキンを使用する際に使用されるキーボードです。制限付きの機能セットでは、自動訂正、自動補完、カスタムキーボードレイアウトなどのネイティブ OS 機能はサポートされません。

スクリーンの一部がキーボードとして使用されるので、Flex では狭いスクリーン領域でもアプリケーションが引き続き機能するようにする必要があります。例えば、ユーザーが TextInput コントロールを選択すると、ソフトキーボードが開きます。Flex では、キーボードが開くと、アプリケーションのサイズを使用可能なスクリーン領域に合わせて自動的にサイズ変更します。次に、選択した TextInput コントロールがキーボードの上に表示されるように位置変更されます。

ブロガーの Peter Elst 氏が、Controlling the soft keyboard in Flex Mobile applications というブログを公開しています。

モバイル Flex アプリケーションでのソフトキーボードのオープン

モバイルアプリケーションでソフトキーボードを開くには、次の 3 つの方法があります。

- TextInput や TextArea などのテキスト入力コントロールがあるコントロールにフォーカスを設定する
- コントロールの needsSoftKeyboard プロパティを true に設定して、そのコントロールにフォーカスを設定する
- コントロールで requestSoftKeyboard() メソッドを呼び出す（iOS 以外）

キーボードは、次のいずれかのアクションが発生するまで開いたままになります。

- ユーザーが、テキスト入力を受け取らないコントロールにフォーカスを移動した場合。これは、ユーザーが別のテキスト入力コントロールを手動でポイントした場合、またはユーザーがキーボードの Return キーを押したためにアプリケーションで別のコントロールにフォーカスが移動した場合に発生する可能性があります。

  フォーカスが別のテキスト入力コントロールに移動した場合や needsSoftKeyboard が true に設定されているコントロールに移動した場合、キーボードは開いたままになります。
- ユーザーがデバイスの戻るボタンを押して入力をキャンセルした場合。
- インタラクティブでないコントロールにプログラムでフォーカスを変更するか、stage.focus を null に設定した場合。

ユーザーへのテキスト入力コントロールの提示

ユーザーにテキスト入力コントロールを提示すると、editable プロパティが false に設定されていない限り、ユーザーがそのコントロールにフォーカスしたときにソフトキーボードが表示されます。

TextInput コントロールおよび TextArea コントロールのデフォルトの動作では、テキストのレンダリングに StageText クラスが使用されます。そのため、これらのコントロールに対して表示されるキーボードでは、自動訂正、自動キャピタライズ、キーボードタイプなどのネイティブ機能がサポートされます。一部の機能がサポートされないデバイスもあります。

テキスト入力コントロールに TextField ベースのスキンを使用するようにスキンクラスを変更すると、機能が制限されます。キーボード自身は同じですが、ネイティブ機能はサポートされません。

needsSoftKeyboard **プロパティの設定**

Button コントロールや ButtonBar コントロールなど、入力以外のコントロールを設定してソフトキーボードを開くこともできます。テキスト入力コントロール以外のコントロールがフォーカスを受け取ったときにキーボードを開くには、コントロールの needsSoftKeyboard プロパティを true に設定します。すべての Flex コンポーネントは、InteractiveObject クラスからこのプロパティを継承します。

TextInput および TextArea 以外のコントロール用に開くキーボードでは、自動キャピタライズ、自動訂正、カスタマイズ可能なキーボードタイプなどのネイティブ機能はサポートされません。

注意：テキスト入力コントロールは、フォーカスを受け取ると常にキーボードを開きます。テキスト入力コントロールでは needsSoftKeyboard プロパティが無視され、テキスト入力コントロールに対してこのプロパティが影響しないように設定されます。

requestSoftKeyboard() **メソッドの呼び出し**

ソフトキーボードをプログラムで起動するには、requestSoftKeyboard() メソッドを呼び出します。このメソッドを呼び出すオブジェクトでは、needsSoftKeyboard プロパティを true に設定しておく必要もあります。このメソッドは、メソッドを呼び出したオブジェクトにフォーカスを変更して、ハードウェアキーボードがデバイスにない場合は、ソフトキーボードを起動します。

requestSoftKeyboard() は InteractiveObject クラスで定義されています。そのため、InteractiveObject のサブクラスである任意のコンポーネントでこのメソッドを呼び出せます。

TextArea コントロールまたは TextInput コントロールで requestSoftKeyboard() メソッドを呼び出すと、自動訂正や自動キャピタライズなどのネイティブキーボード機能がサポートされます（デバイスで機能がサポートされている場合）。

requestSoftKeyboard() メソッドは、iOS デバイス上では機能しません。

ソフトキーボードでのネイティブ機能の使用

モバイルアプリケーションの TextInput コントロールおよび TextArea コントロールのスキンは、StageTextInputSkin クラスおよび StageTextAreaSkin クラスで定義されています。これらのスキンは、StageText クラスを使用してテキストをレンダリングし、ソフトキーボードのネイティブ機能にフックしています。中でも、次に挙げる特長が注目に値します。

- 自動訂正

- 自動キャピタライズ
- カスタム Return キーラベル
- カスタムキーボードタイプ

テキスト入力コントロールでは、テキストレンダリング用に TextField ベースのスキンを使用することもできます。これらのスキンでは、ネイティブ機能はサポートされません。ただし、これらのスキンには、フォームのスクロール、フォントの埋め込み、keyUp イベントおよび keyDown イベントへのアクセス、クリッピング、テキスト測定、分数のアルファ値のサポートなどの、基盤となるテキストコントロールに対する追加機能は提供されています。

TextField ベースのスキンを使用するには、テキスト入力コントロールの skinClass プロパティが TextInputSkin クラスおよび TextAreaSkin クラスを指すように設定します。次に例を示します。

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin"/>
<s:TextArea skinClass="spark.skins.mobile.TextAreaSkin"/>
```

Flex モバイルアプリケーションのソフトキーボードでの自動訂正の使用

自動訂正とは、スペルミスを修正し、ユーザーの入力に対して予測入力の適用を試みる OS の動作のことです。デバイスに応じて、テキスト上のバブル、ソフトキーボードの拡張などの方法で、この動作を実装できます。

テキスト入力コントロールの autoCorrect プロパティを true に設定すれば、モバイルアプリケーションのソフトキーボードで自動訂正を使用できます。これがデフォルト値です。

次の例では、自動訂正のオンとオフを切り替えることができるようにしています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/AutoCorrectionExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Auto-Correction">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextInput prompt="Enter your text" autoCorrect="{myCB.selected}"/>
    <s:CheckBox id="myCB" label="Enable auto-correct" enabled="true"/>

</s:View>
```

自動訂正をサポートしていないデバイスもあります。autoCorrect プロパティをサポートしていないデバイスでこのプロパティを有効または無効にしても、ランタイムではこの値は無視され、デバイスのデフォルト動作が使用されます。

Flex モバイルアプリケーションのソフトキーボードでの自動キャピタライズの使用

自動キャピタライズとは、ユーザーがテキストを入力するときに、特定の単語や英文字を大文字にするようにテキスト入力コントロールに指示する設定のことです。例えば、すべての文字を大文字にしたり、各文の最初の単語の初めの英文字のみを自動的に大文字にしたりすることができます。モバイルアプリケーションでテキストを入力する際に、大文字と小文字を気にしなくてよいのは、ユーザーにとって便利です。

ソフトキーボードで自動キャピタライズを使用するには、テキスト入力コントロールの autoCapitalize プロパティの値を設定します。設定できる値は、none、word、sentence および all です。設定できる値は、AutoCapitalize クラスに定義されています。デフォルト値は none です。

次の例では、自動キャピタライズに様々な値を選択できるようにしています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/AutoCapitalizeExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Auto-Capitalization">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a capitalization setting:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="capTypeList" width="300" labelField="name" fontSize="12">
            <s:ArrayCollection>
                <fx:Object name="All" value="all"/>
                <fx:Object name="None" value="none"/>
                <fx:Object name="Sentence" value="sentence"/>
                <fx:Object name="Word" value="word"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput autoCapitalize="{capTypeList.selectedItem.value}"/>

</s:View>
```

自動キャピタライズをサポートしていないデバイスもあります。autoCapitalize プロパティをサポートしていないデバイスでこの値を設定しても、ランタイムではこの値は無視され、デバイスのデフォルトが使用されます。

Flex モバイルアプリケーションのソフトキーボードのタイプの変更

モバイルアプリケーションのソフトキーボードのタイプは、SoftKeyboardType クラスに定義されています。キーボードのタイプは、テキスト入力コントロールの softKeyboardType プロパティで選択します。

email や contact など、ほとんどのキーボードの違いはわずかです。例えば、email キーボードでは、マイクロフォンが「@」記号で置き換えられている以外は contact キーボードとすべて同じキーになっています。url キーボードでは、「/」記号が使用されています。ただし、number キーボードは例外です。このキーボードは、数字と演算子を中心にした電卓のような表示になっています。

次の例は、使用可能な様々な種類のソフトキーボードを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/KeyboardTypes.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Keyboard Types">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a keyboard type:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="keyboardTypeList" width="300" labelField="name">
            <s:ArrayCollection>
                <fx:Object name="Contact" value="contact"/>
                <fx:Object name="Default" value="default"/>
                <fx:Object name="Email" value="email"/>
                <fx:Object name="Number" value="number"/>
                <fx:Object name="Punctuation" value="punctuation"/>
                <fx:Object name="URL" value="url"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput softKeyboardType="{keyboardTypeList.selectedItem.value}" text=""/>

</s:View>
```

一部の種類のソフトキーボードがサポートされていないデバイスもあります。サポートされていない種類を指定すると、ランタイムではその値は無視され、デバイスのデフォルトが使用されます。

Flex モバイルアプリケーションのソフトキーボードの Return キーラベルの変更

ソフトキーボードがポップアップ表示されてユーザーがテキストを入力する際には、入力が完了し、次のフィールドに移動するか、または入力したデータを送信する必要があることをユーザーが示す方法が必要です。ソフトキーボードでは通常、Return キーでこの処理を行います。このキーはテキスト入力には文字を入力しませんが、ユーザーのテキスト入力が完了したことをテキスト入力コントロールに通知します。

Return キーに設定できるラベルは ReturnKeyLabel クラスに定義されています。設定できる値は default、done、go、next および search です。Return キーのラベルは、テキスト入力コントロールの returnKeyLabel プロパティで指定します。

次の例では、様々な Return キーのラベルを選択できるようにしています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/ReturnKeyLabels.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Return Key Labels">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a return key label:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="returnKeyLabelList" width="300" labelField="name">
            <s:ArrayCollection>
                <fx:Object name="Default" value="default"/>
                <fx:Object name="Done" value="done"/>
                <fx:Object name="Go" value="go"/>
                <fx:Object name="Next" value="next"/>
                <fx:Object name="Search" value="search"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput returnKeyLabel="{returnKeyLabelList.selectedItem.value}" text=""/>

</s:View>
```

Return キーの種類が異なっても、イベントやインタラクションに違いはありません。returnKeyLabel プロパティを変更しても、キーのラベルが変更されるだけです。

Return キーのラベルの設定をサポートしていないデバイスもあります。returnKeyLabel プロパティをサポートしていないデバイスでこの値を設定しても、ランタイムではこの値は無視され、デバイスのデフォルトが使用されます。

モバイルアプリケーションのソフトキーボードでのイベントの使用

モバイルデバイスでのソフトキーボードの操作は、デスクトップや Web ベースのアプリケーションでのキーボードの操作とは異なります。次の表に、ソフトキーボードの操作に関連したイベントの一覧を示します。

イベント	送出される時期
enter	ユーザーが Return キーを押したとき。
keyDown と keyUp	StageText ベースのスキンの場合は、一部のキーについてのみ、それが押されたときと放されたとき。TextField ベースのスキンの場合は、すべてのキーについて、それが押されたときと放されたとき。 これらのイベントは、すべてのデバイスのすべてのキーについて送出されるわけではありません。キーボードを起動するコントロールに TestField ベースのスキンを使用している場合以外は、これらのメソッドに依存してソフトキーボードによるキー入力を取得することはしないでください。
softKeyboardActivating	キーボードが開く直前。
softKeyboardActivate	キーボードが開いた直後。
softKeyboardDeactivate	キーボードが閉じた後。

ユーザーがソフトキーボードでの操作を完了した時点を特定するには、テキスト入力コントロールで FlexEvent.ENTER イベントをリスニングできます。Return キーが押されると、コントロールからこのイベントが送出されます。enter イベントをリスニングすることにより、検証やフォーカスの変更を実行したり、最近入力されたテキストに対して他の操作を実行したりできます。

一部のケースでは、enter イベントが送出されません。この制限は、ビューの最後のテキスト入力コントロールでソフトキーボードを使用する際に Android デバイスで発生します。この問題を回避するには、ビューの最後のテキスト入力コントロールの returnKeyLabel プロパティを go、next または search に設定します。

次の例では、ユーザーがソフトキーボードで Next キーを押すと、1 つのフィールドから次のフィールドにフォーカスが変更されます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/UseNextLikeTab.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Change Focus">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private function changeField(ti:TextInput):void {
                // Before changing focus to a new control, set the stage's focus to null:
                stage.focus = null;

                // Set focus on the TextInput that was passed in:
                ti.setFocus();
            }
        ]]>
    </fx:Script>

    <s:HGroup>
        <s:Label text="1:" paddingTop="15"/>
        <s:TextInput id="ti1" prompt="First Name"
                     width="80%"
                     returnKeyLabel="next"
                     enter="changeField(ti2)"/>
    </s:HGroup>
    <s:HGroup>
        <s:Label text="2:" paddingTop="15"/>
        <s:TextInput id="ti2" prompt="Middle Initial"
                     width="80%"
                     returnKeyLabel="next"
                     enter="changeField(ti3)"/>
    </s:HGroup>
    <s:HGroup>
        <s:Label text="3:" paddingTop="15"/>
        <s:TextInput id="ti3" prompt="Last Name"
                     width="80%"
                     returnKeyLabel="next"
                     enter="changeField(ti1)"/>
    </s:HGroup>

</s:View>
```

TextInput コントロールおよび TextArea コントロールのデフォルトのソフトキーボードをユーザーが操作する際には、キーの小さなサブセットに対してのみ、keyUp および keyDown イベントが送出されます。すべてのキーの個々のキーの押下を取得するには、change イベントを使用します。change イベントは、テキスト入力コントロールの内容が変更されると必ず送出されます。この方法の短所は、押されたキーのプロパティにアクセスできず、キーの押下ロジックを自分で記述する必要があることです。

次の例は、最後に押されたキーの文字コードを表示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/CompareMobileKeyPresses.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Keyboard Events">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private var storedValueOfText:String = null;

            // Compare the new text against the stored value to see what key was pressed.
            private function compareKey(e:Event):void {
                var key:String = "";
                if (storedValueOfText == null) {
                  key = ti1.text.charCodeAt(0).toString(); // Capture the first key pressed.
                } else {
                    // Compare the stored value against the current value and extract the difference.
                    for (var i:int = 0; i<ti1.text.length; i++) {
                        if (ti1.text.charAt(i) == storedValueOfText.charAt(i)) {
                            // Do nothing if they're equal.
                        } else {
                            key = ti1.text.charCodeAt(i).toString();
                        }
                    }
                }
                ti2.text = "The '" + key + "' key was pressed.";
                storedValueOfText = ti1.text;
            }
        ]]>
    </fx:Script>

    <s:TextInput id="ti1" change="compareKey(event)"/>
    <s:TextInput id="ti2" editable="false"/>
</s:View>
```

この例は、テキスト入力フィールドの最後にカーソルがあった場合に、最後に押されたキーのみを示します。

TextField ベースのコントロールのソフトキーボードをユーザーが操作する際には、keyUp および keyDown などのイベントがすべてのキーに対して機能します。次の例では、keyUp ハンドラーを使用して現在のキーを取得し、キーコードに基づいて Label コントロールにスタイルを適用しています。requestSoftKeyboard() メソッドでは TextInput コントロールや TextArea コントロールではなく Label コントロールに対するキーボードを起動するので、アプリケーションでは TextField フィールドベースのキーボードが使用されます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/RequestSoftKeyboardExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="requestSoftKeyboard()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private function handleButtonClick():void {
                myLabel.requestSoftKeyboard();
            }
            /* Ok to use keyUp handler on limited screen keyboard. */
            private function handleKeys(event:KeyboardEvent):void {
                var c:int;
                switch(event.keyCode) {
                    case(82): // 82 = "r"
                        c = 0xFF0000;
                        break;
                    case(71): // 71 = "g"
                        c = 0x00FF00;
                        break;
                    case(66): // 66 = "b"
                        c = 0x0000FF;
                        break;
                }
                event.currentTarget.setStyle("color",c);
            }
        ]]>
    </fx:Script>
    <s:Label id="myLabel" text="This is a label." needsSoftKeyboard="true" keyUp="handleKeys(event)"/>
    <s:Button id="b1" label="Click Me" click="handleButtonClick()"/>

</s:View>
```

ソフトキーボードをプログラムで閉じるには、stage.focus を null に設定します。ソフトキーボードを閉じて、別のコントロールにフォーカスを設定するには、stage.focus を null に設定してから、ターゲットのコントロールにフォーカスを設定します。また、別のコントロールの requestSoftKeyboard() メソッドを呼び出して、別のコントロールにフォーカスを変更してソフトキーボードを開くこともできます。

ソフトキーボードの一部のプロパティには、イベントハンドラーからアクセスできます。ソフトキーボードのサイズと場所にアクセスするには、次の例に示すように、flash.display.Stage クラスの softKeyboardRect プロパティを使用します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/SoftKeyboardEvents.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Soft Keyboard Events">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextInput prompt="Enter your text"
             softKeyboardActivate="ta1.text+=stage.softKeyboardRect + '\n'"
             softKeyboardDeactivate="ta1.text+=stage.softKeyboardRect + '\n'"
             softKeyboardActivating="ta1.text+=stage.softKeyboardRect + '\n'"/>
    <s:TextArea id="ta1" width="100%" height="100%" editable="false"/>

</s:View>
```

ソフトキーボードに対するアプリケーションの設定

ソフトキーボードをサポートするために、アプリケーションでは、キーボードが開いたときに次のアクションを実行します。

- キーボードがアプリケーションに重ならないように、使用可能な残りのスクリーン領域に合わせてアプリケーションをサイズ変更します。
- コントロールが表示されるように、フォーカスがあるテキスト入力コントロールの親コンテナをスクロールします。

ソフトキーボード用のシステムの設定

フルスクリーンモードで実行されるアプリケーションでは、ソフトキーボードはサポートされません。そのため、app.xml ファイルで、<fullScreen> 属性が false（デフォルト値）に設定されていることを確認してください。

アプリケーションのレンダリングモードが CPU モードに設定されていることを確認してください。レンダリングモードは、アプリケーションの app.xml 記述ファイルの <renderMode> 属性で制御されています。<renderMode> 属性が、gpu ではなく、cpu（デフォルト値）に設定されていることを確認してください。

注意：<renderMode> 属性は、デフォルトでは app.xml ファイルに含まれていません。この設定を変更するには、<initialWindow> 属性のエントリとして、この属性を追加します。この属性が app.xml ファイル含まれていない場合は、デフォルト値の cpu になります。

ソフトキーボードが開いたときの親コンテナのスクロール

アプリケーションのサイズ変更の動作は、Application コンテナの resizeForSoftKeyboard プロパティによって決まります。resizeForSoftKeyboard プロパティが false（デフォルト値）の場合、キーボードをアプリケーションの最前面に表示できます。値が true の場合は、キーボードのサイズを差し引いたサイズにアプリケーションのサイズが変更されます。

スクロールをサポートするには、StageText ベースのスキンではなく TextField ベースのスキンをテキスト入力コントロールで使用する必要があります。そうするには、TextInput コントロールまたは TextArea コントロールの skinClass プロパティが、TextInputSkin クラスまたは TextAreaSkin クラスをそれぞれ指すようにします。

スクロールするには、Scroller コンポーネントのテキスト入力コンポーネントの親コンテナを折り返します。キーボードを開くコンポーネントがフォーカスを取得すると、Scroller は、コンポーネントをビュー内に自動的にスクロールします。このコンポーネントは、Scroller コンポーネントのネストされた複数の子コンテナである場合もあります。

親コンテナは、GroupBase クラスまたは SkinnableContainer クラス、あるいは GroupBase または SkinnableContainer のサブクラスである必要があります。フォーカスを取得するコンポーネントは、IVisualElement インターフェイスを実装し、フォーカス可能である必要があります。

Scroller コンポーネントの親コンポーネントを折り返すことで、キーボードが表示されているときにコンテナをスクロールできます。例えば、1 つのコンテナに、複数のテキスト入力コントロールが配置されているとします。次に、それぞれのテキスト入力コントロールをスクロールして、データを入力します。

キーボードが閉じたとき、使用可能なスクリーン領域よりもコンテナのほうが小さい場合があります。使用可能なスクリーン領域よりもコンテナのほうが小さい場合は、Scroller によってスクロール位置が 0（コンテナの 1 番上）に戻されます。

次の例は、複数の TextInput コントロールと単一の Scroller コンポーネントが格納されている View コンテナを示しています。

```
<?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">
            <!-- Use TextField-based skins so that scrolling works. -->
            <s:TextInput prompt="To" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextInput prompt="CC" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextInput prompt="Subject" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextArea height="400" width="100%" prompt="Compose Mail"
skinClass="spark.skins.mobile.TextAreaSkin"/>
        </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>
```

VGroup コンテナは TextInput コントロールの親コンテナです。Scroller は、TextInput の各コントロールがフォーカスを取得したときに、そのコントロールがキーボードより上方に表示されるように VGroup を折り返します。

Scroller コンポーネントについて詳しくは、Scrolling Spark containers を参照してください。

ソフトキーボードが開いたときのアプリケーションのサイズ変更

Application コンテナの resizeForSoftKeyboard プロパティが true の場合、キーボードが開かれると、使用可能なスクリーン領域に合わせてアプリケーションのサイズが変更されます。キーボードが閉じると、アプリケーションは、そのサイズを元に戻します。

次の例は、resizeForSoftKeyboard プロパティを true に設定してアプリケーションのサイズ変更をサポートするアプリケーションのメインアプリケーションファイルを示しています。

```
<?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>
```

アプリケーションのサイズ変更を有効にするには、アプリケーションの app.xml 記述ファイルの <softKeyboardBehavior> 属性が none に設定されていることを確認します。<softKeyboardBehavior> 属性のデフォルト値は none です。このデフォルトでは、Stage 全体を移動するように AIR が設定され、フォーカスされているテキストコンポーネントが表示されます。

ソフトキーボードイベントは十分に信頼性があり、ほとんどの場合は自動的な動作に任せることができますが、ソフトキーボードによって隠れるような位置に重要な UI 要素を配置するとソフトキーボードイベントが失敗するので、そのようなことは避けてください。たとえば、“OK”、“ ログイン ”、“ 送信 ” などのボタンをビューの下半分に配置することは避けます。

StageText ベースのコントロールがアニメーション化されると、またはアニメーションに加わると、ランタイムは、テキストが他のアイテムと同期して移動するように、コントロールを一時的にビットマップに置き換えます。コントロールがフォーカスを持っている場合、これによりコントロールは一時的にフォーカスを失います。場合により、ソフトキーボードは非表示になるか、または非表示にならずに softKeyboardDeactivate イベントをトリガーします。

このような動作は、特に、ソフトキーボードによって発生するサイズの変化がアニメーション化されるポップアップ内の StageText ベースのコンポーネントにプログラムでフォーカスを設定しているときに、問題になる可能性があります。これを避けるには、ソフトキーボードによってサイズが変化しないポップアップで StageText ベースのコンポーネントを使用してみてください。

サイズが変化するポップアップで StageText ベースのコンポーネントを使用する必要がある場合は、最初にソフトキーボードを表示し、ポップアップのアニメーションが完了するのを待ってから、プログラムで StageText ベースのコンポーネントにフォーカスを設定してください。または、プログラムでポップアップの内部にフォーカスを設定しないようにします。

ソフトキーボードと共に使用するポップアップの設定

Callout コンテナは、モバイルアプリケーションの最前面にポップアップとして表示されます。Callout コンテナは、キーボード入力を受け取る 1 つ以上のコンポーネントを保持することができます。Callout コンテナについて詳しくは、77 ページの「Callout コンテナを使用したコールアウトの作成」を参照してください。

Callout の親クラスである SkinnablePopUpContainer コンテナのプロパティを使用して、ポップアップでのキーボードとのインタラクションを設定します。

moveForSoftKeyboard true（デフォルト値）の場合、ポップアップは、キーボードが開かれるとキーボードの上に移動します。

resizeForSoftKeyboard true（デフォルト値）の場合、ポップアップは、キーボードが開かれると、キーボードの上の使用可能なスペースに合わせてサイズ変更されます。

SkinnablePopUpContainer の moveForSoftKeyboard または resizeForSoftKeyboard プロパティを true に設定した場合、ソフトキーボードの表示が変化するたびにコンテナが移動またはサイズ変更する可能性があります。これらの自動動作のどちらによっても、他のコンポーネントのサイズ変化や移動が発生することがあります。

このようなシナリオを避ける最も簡単な方法は、resizeForSoftKeyboard を true に設定しないことです。ソフトキーボードに対してアプリケーションのサイズが変化しなければ、ソフトキーボードによってコンポーネントがユーザーの指先から移動してしまうことはありません。ただし、このようにすると、ソフトキーボードによってアプリケーションの UI の一部が隠れる可能性があります。

アプリケーションのサイズを自動的に変更する必要がある場合は、インタラクティブなコンポーネントを、ソフトキーボードの表示の変化によってユーザーの指先から移動しないように配置してください。次のような方法を利用できます。

- ボタン、チェックボックス、その他の小さいターゲットを、下揃えの制約を設定して配置したり、パーセント指定の高さで他のコンポーネントの下に配置したりしないようにします。
- キーボードが表示または非表示になるときに、最下部にあるコンポーネントの上端が動かないように、レイアウトを設計します。
- ソフトキーボードを非表示にするための組み込みの情報が不足しているプラットフォームでは、固定の UI 要素を使用して行います。ソフトキーボードを非表示にする専用のボタンを用意したり、または単にテキストコンポーネントからフォーカスを除去するためにタップする十分な大きさの固定の空白領域を設けます。
- 移動する可能性のあるコンポーネントを縦に配置しないでください。縦に配置すると、ソフトキーボードの表示が変化したときにジェスチャーがターゲットを誤って検索する場合があります。

モバイルアプリケーションでの埋め込みフォント

モバイルアプリケーションには、一部制限はありますが、フォントを埋め込むことができます。

Label コントロールは FTE を使用（したがって、CFF フォントを使用）するので、モバイルアプリケーションにフォントを埋め込む場合は、TextField ベースのスキンが設定された TextArea または TextInput コントロールを使用します。StageText ベースのスキンにはフォントを埋め込めません。通常、モバイルアプリケーションでは FTE の使用を避けます。

次の例に示すように、CSS で embedAsCFF を false に設定して、TextField ベースのスキンを適用します。

```
<?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;
            skinClass: ClassReference("spark.skins.mobile.TextAreaSkin");
        }
    </fx:Style>
</s:ViewNavigatorApplication>
```

EmbeddingFontView ビューの TextArea コントロールでは、タイプセレクターを適用します。

```
<?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:layout>
        <s:VerticalLayout paddingTop="20" paddingLeft="20" paddingRight="20"/>
    </s:layout>
    <s:TextArea id="ta1"
                width="100%"
                styleName="customStyle"
                text="This is a TextArea control that uses an embedded font."/>
    <s:TextArea id="ta2"
                width="100%"
                text="This TextArea control does not use an embedded font."/>
</s:View>
```

クラスセレクター（s|TextArea など）を使用してスタイルを適用する（またはフォントを埋め込む）場合は、メインアプリケーションファイルにクラスセレクターを定義します。モバイルアプリケーションのビュー内でクラスセレクターを定義することはできません。

詳しくは、Embed fonts を参照してください。

第 6 章：スキン

モバイルのスキン適用の基本

デスクトップスキンとモバイルスキンの比較

モバイルのスキンは、対応するデスクトップのスキンより軽量なスキンです。したがって、両者には多くの相違点があります。例えば、モバイルスキンには次の特徴があります。

- モバイルスキンは ActionScript で記述されています。ActionScript のみで記述されたスキンは、モバイルデバイスで最高のパフォーマンスを実現します。
- モバイルスキンは spark.skins.mobile.supportClasses.MobileSkin クラスを拡張します。Skin クラスを拡張する SparkSkin クラスとは異なり、このクラスは UIComponent クラスを拡張します。
- モバイルスキンのグラフィックアセットには、パフォーマンスを上げるため、コンパイル済みの FXG または単純な ActionScript 描画が使用されます。これに対し、デスクトップアプリケーションのスキンでは、通常、多くの描画に MXML グラフィックが使用されます。
- モバイルスキンではスキンステートを宣言する必要がありません。モバイルスキンは ActionScript で記述されているため、ステートは手続き的に実装する必要があります。
- モバイルスキンでステートのトランジションはサポートされていません。
- モバイルスキンは手動でレイアウトします。モバイルスキンはグループを拡張しないため、Spark レイアウトがサポートされません。そのため、子は ActionScript で手動で配置されます。
- モバイルスキンでは一部のスタイルがサポートされません。モバイルテーマでは、モバイルスキンのパフォーマンスまたはその他の相違に基づいて一部のスタイルが省略されます。


パフォーマンス関連の相違に加えて、Flash Builder では、一部のモバイルスキンファイルの使用方法も異なります。ライブラリプロジェクトで使用されるモバイルテーマについては、特に異なります。ブロガーの Jeffry Houser 氏が、これに対する修正方法を公開しています。

モバイルのホストコンポーネント

モバイルのスキンでは、通常、パブリックの hostComponent プロパティを宣言します。このプロパティは省略可能ですが、指定することをお勧めします。hostComponent プロパティは、スキンを使用するコンポーネントと同じタイプにする必要があります。例えば、ActionBarSkin では、ActionBar タイプの hostComponent を宣言します。

```
public var hostComponent:ActionBar;
```

Flex では、コンポーネントで初めてスキンをロードするときに、hostComponent プロパティの値が設定されます。

デスクトップのスキンと同様に、ホストコンポーネントを使用して、スキンの適用先コンポーネントのプロパティとメソッドにアクセスできます。例えば、ホストコンポーネントのパブリックプロパティにアクセスしたり、スキンクラス内からイベントリスナーをホストコンポーネントに追加できます。

モバイルのスタイル

モバイルスキンでは、デスクトップスキンでサポートされるスタイルプロパティのサブセットがサポートされます。モバイルテーマでは、このスタイルのセットを定義します。

次の表は、モバイルテーマを使用している場合に、コンポーネントに使用できるスタイルプロパティを示しています。

スタイルプロパティ	プロパティをサポートしているコンポーネント	継承あり／継承なし
accentColor	Button、ActionBar、ButtonBar	継承あり
backgroundAlpha	ActionBar	継承なし
backgroundColor	Application	継承なし
borderAlpha	List	継承なし
borderColor	List	継承なし
borderVisible	List	継承なし
chromeColor	ActionBar、Button、ButtonBar、CheckBox、HSlider、RadioButton	継承あり
color	テキストがあるすべてのコンポーネント	継承あり
contentBackgroundAlpha	TextArea、TextInput	継承あり
contentBackgroundColor	TextArea、TextInput	継承あり
focusAlpha	フォーカス可能なすべてのコンポーネント	継承なし
focusBlendMode	フォーカス可能なすべてのコンポーネント	継承なし
focusColor	フォーカス可能なすべてのコンポーネント	継承あり
focusThickness	フォーカス可能なすべてのコンポーネント	継承なし
locale	すべてのコンポーネント	継承あり
paddingBottom	TextArea、TextInput	継承なし
paddingLeft	TextArea、TextInput	継承なし
paddingRight	TextArea、TextInput	継承なし
paddingTop	TextArea、TextInput	継承なし
selectionColor	ViewMenuItem	継承あり

テキストベースのコンポーネントは、fontFamily、fontSize、fontWeight などの標準のテキストスタイルもサポートします。

モバイルテーマがスタイルプロパティをサポートしているかどうかを確認するには、ActionScript リファレンスガイド でコンポーネントの説明を参照してください。これらのスタイル制限の多くは、テキストベースのモバイルコンポーネントでは TLF（Text Layout Framework）を使用しないために課されています。代わりに、モバイルスキンでは、TLF ベースのテキストコントロールが、より軽量なコンポーネントで置換されています。詳しくは、131 ページの「モバイルアプリケーションでのテキストの使用」を参照してください。

モバイルテーマでは、rollOverColor、cornerRadius、および dropShadowVisible スタイルプロパティはサポートされません。

Flex エンジニアの Jason SJ が、モバイルアプリケーションのスタイルとテーマについてブログで説明しています。

モバイルのスキンパーツ

モバイルのスキンでは、スキンパーツに関して、デスクトップのスキンと同じスキン規約に従う必要があります。コンポーネントに必須のスキンパーツがある場合は、モバイルスキンで適切なタイプのパブリックプロパティを宣言する必要があります。

例外

すべてのスキンパーツが必須というわけではありません。例えば、Spark の Button には、iconDisplay や labelDisplay というオプションのスキンパーツがあります。したがって、モバイルの ButtonSkin クラスには、BitmapImage タイプの iconDisplay プロパティを宣言できます。StyleableTextField タイプの labelDisplay プロパティを宣言することもできます。

labelDisplay パーツには、使用するスタイルがすべて継承テキストスタイルであるため、id プロパティを設定しません。また、StyleableTextField は UIComponent でないため、id プロパティがありません。iconDisplay パーツはスタイルをサポートしていません。したがって、id プロパティも設定しません。

高度な CSS を使用したスタイルの設定

高度な CSS の id セレクターを使用してスキンパーツにスタイルを設定する場合は、スキンにもスキンパーツの id プロパティを設定する必要があります。例えば、ActionBar の titleDisplay スキンパーツには、高度な CSS を使用してスタイルを設定できるように id プロパティを設定します。次に例を示します。

```
@namespace s "library://ns.adobe.com/flex/spark";
s|ActionBar #titleDisplay {
    color:red;
}
```

モバイルテーマ

モバイルテーマでは、モバイルアプリケーションでサポートするスタイルが決まります。モバイルテーマで使用できるスタイルの数は、Spark テーマのサブセットです（若干追加があります）。モバイルテーマでサポートされているスタイルの一覧は、154 ページの「モバイルのスタイル」で確認できます。

モバイルアプリケーションのデフォルトのテーマ

モバイルアプリケーションのテーマは、themes¥Mobile¥mobile.swc ファイルに定義されます。このファイルには、モバイルアプリケーションのグローバルスタイルと、各モバイルコンポーネントのデフォルト設定を定義します。このテーマファイルのモバイルスキンは、spark.skins.mobile.* パッケージで定義されています。このパッケージには、MobileSkin 基本クラスが含まれます。

mobile.swc テーマファイルは、Flash Builder モバイルプロジェクトにデフォルトで含まれていますが、SWC ファイルはパッケージエクスプローラーに表示されません。

Flash Builder で新規のモバイルプロジェクトを作成すると、このテーマがデフォルトで適用されます。

テーマの変更

テーマを変更するには、theme コンパイラー引数を使用して、新しいテーマを指定します。次に例を示します。

```
-theme+=myThemes/NewMobileTheme.swc
```

テーマについて詳しくは、About themes を参照してください。

Flex エンジニアの Jason SJ が、モバイルアプリケーションのテーマを作成してオーバーレイする方法をブログで説明しています。

モバイルのスキンステート

MobileSkin クラスは、UIComponent クラスのステートメカニズムを上書きします。デスクトップアプリケーションのビューステートの実装は使用しません。したがって、モバイルスキンでは、ホストコンポーネントのスキンステートの内、スキンが実装しているステートのみを宣言します。モバイルスキンでは、ステート名のみに基づいてステートを手続き的に変更します。これとは対照的に、デスクトップスキンでは、使用されているかどうかに関わりなく、すべてのステートを宣言する必要があります。デスクトップスキンでは、mx.states.* パッケージ内のクラスも使用してステートを変更します。

ほどんどのモバイルスキンでは、対応するデスクトップのスキンより少ない数のステートを実装します。例えば、spark.skins.mobile.ButtonSkin クラスは、up、down および disabled というステートを実装します。spark.skins.spark.ButtonSkin では、これらのステートに加え、over ステートを実装します。このステートはタッチデバイスでは一般に使用されないので、モバイルスキンでは over ステートの動作は定義しません。

commitCurrentState() メソッド

モバイルスキンのクラスには、commitCurrentState() メソッドでのステートの動作を定義します。カスタムスキンクラスの commitCurrentState() メソッドを編集することにより、モバイルスキンに動作を追加し、追加のステートをサポートできるようにします。

currentState プロパティ

スキンの外観は、currentState プロパティの値によって異なります。例えば、モバイルの ButtonSkin クラスでは、currentState プロパティの値によって、ボーダークラスとして使用される FXG クラスが決まります。

```
if (currentState == "down")
    return downBorderSkin;
else
    return upBorderSkin;
```

currentState プロパティについて詳しくは、Create and apply view states を参照してください。

モバイルのグラフィック

通常、モバイルスキンのグラフィックアセットにはコンパイル済みの FXG が使用されます。これに対し、デスクトップアプリケーションのスキンでは、通常、多くの描画に MXML グラフィックが使用されます。

ビットマップの埋め込みグラフィック

クラスでは、通常は適切に拡大／縮小されるビットマップの埋め込みグラフィックを使用できます。ただし、ビットマップは、様々な画面密度に対して常に適切に拡大／縮小されるとは限りません。異なるアセットをいくつか、画面密度ごとに作成するほうが、適切に拡大／縮小できます。

デフォルトのモバイルテーマのグラフィック

デフォルトのモバイルテーマのモバイルスキンでは、ターゲットデバイスの DPI 用に最適化された FXG グラフィックが使用されます。スキンでは、ルートアプリケーションの applicationDPI プロパティの値に基づいてグラフィックがロードされます。例えば、DPI が 320 のデバイスに CheckBox コントロールがロードされる場合、CheckBoxSkin クラスでは、spark.skins.mobile320.assets.CheckBox_up.fxg グラフィックが upIconClass プロパティに使用されます。DPI が 160 の場合は、spark.skins.mobile160.assets.CheckBox_up.fxg グラフィックが使用されます。

次のデスクトップの例に、DPI が異なる CheckBox スキンで使用されるさまざまなグラフィックを示します。

```
<?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>
```

モバイルアプリケーションでの解像度と DPI について詳しくは、119 ページの「モバイルアプリケーションでの複数のスクリーンサイズと DPI 値のサポート」を参照してください。

ActionScript のスキンでは、キャッシュされたベクトル形式の画像をビットマップとして使用することもできます。唯一の欠点は、アルファトランジションなど、ピクセルの再描画を必要とするトランジションを使用できないことです。詳しくは、www.adobe.com/jp/devnet/air/flex/articles/writing_multiscreen_air_apps.html を参照してください。

モバイルアプリケーションのスキンの作成

モバイルスキンをカスタマイズする場合は、モバイルスキンのカスタムクラスを作成します。場合によっては、モバイルスキンクラスで使用するアセットも編集します。

モバイルスキンクラスを編集する際は、ステートに基づいたインタラクションの変更、新規スタイルのサポートの実装、およびスキンに対する子コンポーネントの追加または削除を実行できます。通常は、既存のスキンのソースコードを使用して編集を開始し、新しいクラスとして保存します。

モバイルスキンで使用されるアセットを編集することで、サイズ、色、グラデーション、背景など、スキンの視覚的なプロパティを変更することもできます。この場合、スキンで使用される FXG アセットも編集できます。モバイルスキンで使用されるソース *.fxg ファイルは、spark/skins/mobile/assets ディレクトリにあります。

モバイルスキンの視覚的なプロパティのすべてが *.fxg ファイルに定義されているわけではありません。例えば、Button スキンの背景色は ButtonSkin クラスの chromeColor スタイルプロパティで定義されます。FXG アセットでは定義されません。この場合、背景色を変更するには、スキンクラスを編集することになります。

モバイルスキンクラスの作成

モバイルスキンのカスタムクラスを作成する場合、最も簡単な方法は、既存のモバイルスキンのクラスをベースとして使用する方法です。そのクラスを変更し、カスタムスキンとして使用します。

カスタムスキンクラスを作成するには：

1. プロジェクトにディレクトリを作成します（例えば、customSkins など）。このディレクトリは、カスタムスキンのパッケージ名になります。パッケージの作成は必須ではありませんが、独立したパッケージにカスタムスキンを整理しておくことをお勧めします。
2. 新規のディレクトリにスキンのカスタムクラスを作成します。CustomButtonSkin.as など、新しいクラスに任意の名前を指定します。
3. 新しいクラスのベースとして使用するスキンクラスの内容をコピーします。例えば、ButtonSkin を基本クラスとして使用する場合は、spark.skins.mobile.ButtonSkin ファイルの内容を、スキンの新しいカスタムクラスにコピーします。
4. 新しいクラスを編集します。例えば、次の最小限の変更を CustomButtonSkin クラスに追加します。
   - パッケージの場所の変更

     ```
     package customSkins
     //was: package spark.skins.mobile
     ```

   - クラス宣言内のクラスの名前を変更します。また、スキンの基本クラスではなく、新しいスキンの基となるクラスを拡張します。

     ```
     public class CustomButtonSkin extends ButtonSkin
     // was: public class ButtonSkin extends ButtonSkinBase
     ```

   - コンストラクター内のクラス名の変更

     ```
     public function CustomButtonSkin()
     //was: public function ButtonSkin()
     ```

5. カスタムスキンクラスを変更します。例えば、追加のステートや新規の子コンポーネントのサポートを追加します。また、一部のグラフィックアセットはスキンクラス自体に定義されているので、そのいくつかのアセットを変更できます。

   スキンクラスを読みやすくするには、上書きしていないメソッドをカスタムスキンから削除する方法が一般的です。

   次のカスタムスキンクラスは、ButtonSkin を拡張して、drawBackground() メソッドをカスタムロジックで置き換えています。背景の塗りの線形グラデーションを円形グラデーションに置換します。

```
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 アプリケーションで、166 ページの「カスタムモバイルスキンの適用」に記載されているいずれかのメソッドを使用して、カスタムスキンを適用します。次の例は、コンポーネントタグの skinClass プロパティを使用して、customSkins.CustomButtonSkin スキンを適用しています。

```
<?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>
```

モバイルスキンのライフサイクルのメソッド

カスタムスキンクラスを作成する場合は、次の UIComponent メソッドに習熟する必要があります。継承、保護されたこれらのメソッドは、スキンの子およびメンバーを定義し、表示リストにある他のコンポーネントとのインタラクションを支援します。

- createChildren() - スキンに必要なグラフィックまたはテキストの子オブジェクトを作成します。
- commitProperties() - コンポーネントデータを必要に応じてスキンにコピーします。
- measure() - 可能な限り効率的にスキンを測定し、スキンの measuredWidth プロパティと measuredHeight プロパティに結果を格納します。
- updateDisplayList() - グラフィックおよびテキストの位置とサイズを設定します。必要な ActionScript 描画を行います。このメソッドは、スキンで drawBackground() メソッドと layoutContents() メソッドを呼び出します。

これらのメソッドの使用について詳しくは、Implementing the component を参照してください。

モバイルスキンでカスタマイズするための一般的なメソッド

多くのモバイルスキンでは、次のメソッドが実装されています。

- layoutContents() — ドロップシャドウやラベルなど、スキンの子を配置します。モバイルスキンクラスでは、HorizontalLayout や VerticalLayout などの Spark レイアウトはサポートされていません。layoutContents() などのメソッドにスキンの子を手動でレイアウトします。
- drawBackground() — スキンに背景をレンダリングします。一般的に、chromeColor スタイル、backgroundColor スタイル、または contentBackgroundColor スタイルを、スキンの形に基づいて描画するなどの使用方法があります。また、applyColorTransform() メソッドを使用するなどして、着色にも使用できます。
- commitCurrentState() - モバイルスキンのステートの動作を定義します。サポートされているステートを追加または削除したり、このメソッドを編集して既存のステートの動作を変更したりできます。このメソッドは、ステートが変化したときに呼び出されます。ほどんどのスキンクラスは、このメソッドをオーバーライドします。詳しくは、156 ページの「モバイルのスキンステート」を参照してください。

カスタム FXG アセットの作成

モバイルスキンの視覚的なアセットの大半は、FXG を使用して定義されています。FXG とは、静的なグラフィックを定義するための宣言型の構文です。FXG ドキュメントの書き出しには、Adobe Fireworks、Adobe Illustrator、Adobe Catalyst などのグラフィックツールを使用できます。その後、モバイルスキンで FXG ドキュメントを使用できるようになります。テキストエディターで FXG ドキュメントを作成することもできます。ただし、複雑なグラフィックを最初から記述するには困難を伴う可能性があります。

モバイルスキンでは、通常、FXG ファイルを使用してスキンのステートを定義します。例えば、CheckBoxSkin クラスでは、次の FXG ファイルを使用して、ボックスやチェックマークの記号の外観を定義します。

- CheckBox_down.fxg
- CheckBox_downSymbol.fxg
- CheckBox_downSymbolSelected.fxg
- CheckBox_up.fxg
- CheckBox_upSymbol.fxg
- CheckBox_upSymbolSelected.fxg

これらのファイルをグラフィックエディターで開くと、次のように表示されます。

チェックボックスのステート（down、downSymbol、downSymbolSelected、up、upSymbol および upSymbolSelected）

複数の解像度用の FXG ファイル

多くのモバイルスキンには、FXG グラフィックファイルが 3 セット（デフォルトのターゲット解像度ごとに 1 セット）含まれています。例えば、6 つの CheckBoxSkin クラスすべての異なるバージョンが、spark/skins/mobile160、spark/skins/mobile240、および spark/skins/mobile320 ディレクトリにあります。

カスタムスキンを作成するには、次のいずれかを行います。

- デフォルトのスキンのいずれか（通常 160 DPI）をベースとして使用します。アプリケーションオブジェクトで applicationDPI プロパティを設定することにより、アプリケーションを実行しているデバイスに合わせてカスタムスキンを拡大／縮小するロジックを追加します。
- 最適な表示を実現するため、カスタムスキンの 3 つのバージョンすべて（160、240、および 320 DPI）を作成してください。

モバイルスキンによっては、グラフィカルアセットに FXG ファイルを 1 セットだけ使用し、DPI 固有のグラフィックは持たない場合もあります。これらのアセットは spark/skins/mobile/assets ディレクトリに格納されます。例えば、ViewMenuItem スキンや TabbedViewNavigator ボタンバースキンには DPI 固有のバージョンがないため、すべての FXG アセットがこのディレクトリに格納されます。

FXG ファイルのカスタマイズ

既存の FXG ファイルを開いてカスタマイズすることも、FXG ファイルを作成し、Adobe Illustrator などのグラフィックエディターからファイルを書き出すこともできます。FXG ファイルを編集した後は、それらをスキンクラスに適用します。

FXG ファイルを変更してカスタムスキンを作成するには：

1. 159 ページの「モバイルスキンクラスの作成」の説明に従って、カスタムスキンクラスを作成し、customSkins ディレクトリに格納します。
2. customSkins ディレクトリの下にサブディレクトリを作成します（例えば、assets など）。サブディレクトリの作成は省略可能ですが、FXG ファイルとスキンクラスの編成に役立ちます。
3. assets ディレクトリにファイルを作成し、既存の FXG ファイルの内容をコピーします。例えば、CustomCheckBox_upSymbol.fxg というファイルを作成します。spark/skins/mobile160/assets/CheckBox_upSymbol.fxg の内容を新しい CustomCheckBox_upSymbol.fxg ファイルにコピーします。
4. 新しい FXG ファイルを変更します。例えば、グラデーションのエントリが挿入された「X」で、チェックを描画するロジックを置換します。

```
<?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 カスタムスキンクラスで、新しい FXG クラスを読み込み、プロパティに適用します。例えば、CustomCheckBox クラスで次の操作を実行します。

1 新しい FXG ファイルを読み込みます。

```
//import spark.skins.mobile.assets.CheckBox_upSymbol;
import customSkins.assets.CustomCheckBox_upSymbol;
```

2 新しいアセットをカスタムスキンクラスに追加します。例えば、新しい FXG アセットを指し示すように upSymbolIconClass プロパティの値を変更します。

```
upSymbolIconClass = CustomCheckBox_upSymbol;
```

完成したカスタムスキンクラスは、次のようになります。

```
// 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
        }
    }
}
```

スキンの FXG アセットの操作および最適化について詳しくは、Optimizing FXG を参照してください。

アプリケーションでの FXG ファイルの表示

FXG ファイルは XML で記述されているので、最終的な製品がどのように表示されるかを想定することが難しい可能性があります。FXG ファイルをコンポーネントとして追加し、Spark コンテナにラップすることで、FXG ファイルを読み込んでレンダリングする Flex のアプリケーションを記述できます。

アプリケーションに FXG ファイルをコンポーネントとして追加するには、アプリケーションのソースパスにソースファイルの位置を追加します。例えば、モバイル FXG アセットを Web ベースのアプリケーションで表示するには、ソースパスにモバイルテーマを追加します。これにより、コンパイラーで FXG ファイルが検出できるようになります。

次のデスクトップの例では、CheckBox コンポーネントをモバイルアプリケーションで使用した場合に、さまざまな FXG アセットをレンダリングします。この例をコンパイルする場合は、frameworks¥projects¥mobiletheme¥src¥ ディレクトリを、コンパイラーの source-path 引数に追加します。

```
<?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>
```

カスタムモバイルスキンでのテキストの使用

モバイルスキンにテキストをレンダリングするには、StyleableStageText クラスまたは StyleableTextField クラスを使用します。これらのテキストクラスは、モバイルアプリケーション用に最適化されています。

StyleableStageText では、TextInput コントロールおよび TextArea コントロールのネイティブテキスト入力にアクセスできます。ここでは、UIComponent クラスが拡張されて、IEditableText インターフェイスおよび ISoftKeyboardHintClient インターフェイスが実装されています。

ネイティブ入力にアクセスする必要がない場合は、TextInput と TextArea コントロールで StyleableTextField も使用されます。また、ActionBar や Button などの入力以外のテキストコントロールでも使用されます。このクラスは TextField クラスを拡張し、ISimpleStyleClient インターフェイスおよび IEditableText インターフェイスを実装します。

モバイルアプリケーションでのテキストコントロールの使用について詳しくは、131 ページの「モバイルアプリケーションでのテキストの使用」を参照してください。

モバイルスキンでの TLF

パフォーマンス上の理由から、モバイルスキンでは、TLF を使用するクラスの使用をできるだけ避けてください。Spark Label コンポーネントを使用する場合など、FTE. を使用するクラスを使用できる場合もあります。

モバイルスキンでの htmlText

モバイルアプリケーションでは htmlText プロパティを使用できません。

テキストを使用するジェスチャー

タッチしてドラッグするジェスチャーでは、常にテキストが選択されます（テキストが選択可能または編集可能な場合）。テキストが Scroller の内側にある場合、Scroller は、ジェスチャーがテキストコンポーネントの外側で行われている場合にのみスクロールします。これらのジェスチャーは、テキストが編集可能かつ選択可能な場合にのみ機能します。

テキストを編集可能かつ選択可能にする方法

テキストを編集可能かつ選択可能にするには、editable プロパティと selectable プロパティを true に設定します。

```
textDisplay.editable = true;
textDisplay.selectable = true;
```

双方向性

StyleableStageText クラスまたは StyleableTextField クラスのテキストでは、双方向性はサポートされません。

カスタムモバイルスキンの適用

カスタムスキンは、デスクトップアプリケーションのコンポーネントにカスタムスキンを適用する場合と同様の方法で、モバイルコンポーネントに適用できます。

ActionScript でのスキンの適用

```
// Call the setStyle() method:
myButton.setStyle("skinClass", "MyButtonSkin");
```

MXML でのスキンの適用

```
<!-- Set the skinClass property: -->
<s:Button skinClass="MyButtonSkin"/>
```

CSS でのスキンの適用

```
// Use type selectors for mobile skins, but only in the root document:
s|Button {
    skinClass: ClassReference("MyButtonSkin");
}
```

または

```
// Use class selectors for mobile skins in any document:
.myStyleClass {
    skinClass: ClassReference("MyButtonSkin");
}
```

カスタムモバイルスキンの適用例

次の例では、カスタムモバイルスキンをモバイルコンポーネントに適用する 3 つのメソッドすべてを示しています。

```
<?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>
```

CSS のタイプセレクターを使用してカスタムスキンを適用する場合は、ルートモバイルアプリケーションファイルにタイプセレクターを設定します。モバイルビューにタイプセレクターを設定することはできません。これはカスタムコンポーネントの場合と同じです。モバイルアプリケーションのビューまたはドキュメントには、引き続きクラスセレクターを使用して ActionScript、MXML または CSS にスタイルを設定できます。

第 7 章：テストとデバッグ

起動設定の管理

Flash Builder では、モバイルアプリケーションの実行時またはデバッグ時に起動設定が使用されます。アプリケーションをデスクトップ上で起動するか、またはコンピューターに接続したデバイス上で起動するかを指定できます。

起動設定を作成するには、次の手順に従います。

1 実行／実行構成を選択して、実行構成ダイアログボックスを開きます。

デバッグの構成ダイアログボックスを開くには、実行／デバッグの構成を選択します。169 ページの「デバイスでのモバイルアプリケーションのテストとデバッグ」を参照してください。

実行構成またはデバッグの構成には、Flash Builder ツールバーの「実行」ボタンまたは「デバッグ」ボタンのドロップダウンリストからアクセスすることもできます。

2 「モバイルアプリケーション」ノードを展開します。ダイアログボックスのツールバーの「新規の起動構成」ボタンをクリックします。

3 ドロップダウンリストでターゲットプラットフォームを指定します。

4 起動方法を指定します。

- **デスクトップ上**

  指定したデバイス設定に従って、AIR Debug Launcher（ADL）を使用してデスクトップ上でアプリケーションを実行またはデバッグします。この起動方法は、デバイスでのアプリケーションの実行に対する真の意味でのエミュレーションではありません。ただし、アプリケーションのレイアウトを表示して、アプリケーションを操作できます。169 ページの「ADL を使用したアプリケーションのプレビュー」を参照してください。

  デバイスの設定を編集するには「設定」をクリックします。169 ページの「デスクトッププレビュー用のデバイス情報の設定」を参照してください。

- **デバイス上**

  デバイスにアプリケーションをインストールして実行します。

  Google Android プラットフォームの場合、Flash Builder によってアプリケーションがデバイス上にインストールされ、アプリケーションが起動されます。Flash Builder は、コンピューターの USB ポートに接続されているデバイスにアクセスします。詳しくは、169 ページの「デバイスでのモバイルアプリケーションのテストとデバッグ」を参照してください。

  Windows では、Android デバイスをコンピューターに接続するために USB ドライバーが必要です。詳しくは、19 ページの「Android デバイス用の USB デバイスドライバーのインストール（Windows）」を参照してください。

5 起動ごとにアプリケーションデータを消去するかどうかを指定します（該当する場合）。

デスクトップでのモバイルアプリケーションのテストとデバッグ

最初のテストやデバッグの場合、またはモバイルデバイスがない場合は、AIR Debug Launcher（ADL）を使用して、デスクトップでアプリケーションをテストおよびデバッグできます。

モバイルアプリケーションを最初にテストまたはデバッグする前に、起動設定を定義します。ターゲットプラットフォームを指定し、起動方法として「デバイス上」を指定します。168 ページの「起動設定の管理」を参照してください。

デスクトッププレビュー用のデバイス情報の設定

デバイス設定のプロパティによって、ADL および Flash Builder デザインモードでのアプリケーションの表示方法が決まります。

15 ページの「デバイス設定の設定」に、サポートされる設定を一覧表示します。デバイス設定は、デバイス上のアプリケーションの外観には影響を与えません。

画面密度

開発デスクトップ上でアプリケーションをプレビューしたり、Flash Builder のデザインモードでアプリケーションのレイアウトを表示したりすることができます。Flash Builder では、240 DPI の画面密度を使用します。プレビュー中のアプリケーションの外観は、デバイスでサポートするピクセル密度の違いによって、デバイス間で異なる場合があります。

ADL を使用したアプリケーションのプレビュー

デスクトップ上でアプリケーションをプレビューする場合、Flash Builder では ADL を使用してアプリケーションが起動されます。ADL には、デバイスのボタンをエミュレートするための対応するショートカットが含まれたデバイスメニューが用意されています。

例えば、デバイスの戻るボタンをエミュレートするには、デバイス／戻るを選択します。デバイスの回転をエミュレートするには、デバイス／ Rotate Left またはデバイス／ Rotate Right を選択します。自動方向変更を選択していない場合、Rotate オプションは無効化されます。

デバイスでのリストのスクロールをエミュレートするには、リスト内をドラッグします。

Flex のアドビ認定エキスパートの Brent Arnold 氏が、ADL を使用してモバイルアプリケーションをデスクトップでプレビューする方法についてのビデオチュートリアルを公開しています。

デバイスでのモバイルアプリケーションのテストとデバッグ

Flash Builder を使用して、自分の開発デスクトップまたはデバイスからモバイルアプリケーションをテストまたはデバッグできます。

アプリケーションは、定義した起動設定に基づいてテストおよびデバッグします。Flash Builder では、アプリケーションの実行とデバッグの両方に起動設定を使用します。Flash Builder を使用してデバイス上でアプリケーションをデバッグすると、アプリケーションのデバッグバージョンがデバイスにインストールされます。

注意： リリースビルドをデバイスに書き出す場合は、アプリケーションの非デバッグバージョンをインストールします。非デバッグバージョンは、デバッグには適していません。

詳しくは、起動設定の管理を参照してください。

Google Android デバイスでのアプリケーションのデバッグ

Android デバイスでデバッグが可能なのは、Android 2.2 以降です。

デバッグは、次のいずれかのシナリオで実行できます。

USB 経由のデバッグ USB 接続経由でアプリケーションをデバッグするには、USB ポート経由でデバイスをホストコンピューターに接続します。USB 経由でデバッグする場合、Flash Builder では、デバッグが始まる前にアプリケーションを必ずパッケージ化してから、そのアプリケーションをデバイスにインストールして起動します。デバッグセッション全体を通して、デバイスがホストコンピューターの USB ポートに接続されていることを確認してください。

ネットワーク経由のデバッグ ネットワーク経由でアプリケーションをデバッグする場合は、デバイスとホストコンピューターを同じネットワークに接続する必要があります。デバイスとホストコンピューターは、Wi-Fi、イーサネットまたは Bluetooth 経由でネットワークに接続できます。

ネットワーク経由でデバッグする場合、Flash Builder では、接続されているデバイスにインストール済みのアプリケーションをデバッグできます。アプリケーションを再インストールする必要はありません。パッケージ化の間とアプリケーションをデバイスにインストールしている間のみ、USB ポート経由でデバイスをホストコンピューターに接続します。デバッグの間は、USB ポートからデバイスを取り外せます。ただし、デバッグセッション全体を通して、デバイスとホストコンピューターの間にネットワーク接続があることを確認してください。

アプリケーションのデバッグの準備

USB 経由またはネットワーク経由のデバッグを開始する前に、次の手順に従います。

1 （Windows）適切な USB ドライバーがインストールされていることを確認します。

Windows に、Android USB ドライバーをインストールします。詳しくは、Android SDK ビルドに付属のマニュアルを参照してください。詳しくは、19 ページの「Android デバイス用の USB デバイスドライバーのインストール（Windows）」を参照してください。

2 デバイスで USB デバッグが有効になっていることを確認します。

デバイスの「Settings」で、Applications ／ Development に移動し、「USB debugging」を有効にします。

接続されているデバイスのチェック

デバイス上のモバイルアプリケーションを実行またはデバッグするとき、Flash Builder は接続されているデバイスをチェックします。Flash Builder は、ただ 1 つのデバイスが接続されてオンラインになっていることを検出すると、アプリケーションをデプロイして起動します。それ以外の場合は、デバイスを選択ダイアログボックスを表示します。次のような場合です。

- 接続されたデバイスが見つからない
- 接続されたデバイスが 1 つだけ検出されたが、オフラインであるか、OS バージョンがサポートされていない
- 複数の接続されたデバイスが検出された

複数のデバイスが検出された場合、デバイスを選択ダイアログボックスに、デバイスとそのステート（オンラインかオフライン）が一覧表示されます。起動するデバイスを選択します。

デバイスを選択ダイアログボックスでは、OS のバージョンと AIR のバージョンが一覧表示されます。Adobe AIR がデバイスにインストールされていない場合は、自動的にインストールされます。

ネットワークデバッグの設定

この手順は、ネットワーク経由でアプリケーションをデバッグする場合のみ実行してください。

ネットワーク経由のデバッグの準備

ネットワーク経由でアプリケーションをデバッグする前に、次の手順に従います。

1 Windows で、ポート 7935（Flash Player デバッガーのポート）とポート 7（echo/ping のポート）を開きます。

手順について詳しくは、Microsoft TechNet の記事を参照してください。

Windows Vista では、Windows ファイアウォール／設定の変更／詳細設定の「ワイヤレス ネットワーク接続」チェックボックスの選択を解除します。

2 デバイスの Settings ／ Wireless and Network で、ワイヤレス設定を行います。

プライマリネットワークインターフェイスの選択

ホストコンピューターは、同時に複数のネットワークインターフェイスに接続できます。ただし、デバッグに使用するプライマリネットワークインターフェイスを選択できます。このインターフェイスを選択するには、Android APK パッケージファイルにホストアドレスを追加します。

1 Flash Builder で、環境設定を開きます。

2 Flash Builder ／ターゲットプラットフォームを選択します。

ダイアログボックスに、ホストコンピューターで使用可能なすべてのネットワークインターフェイスが一覧表示されます。

3 Android APK パッケージに埋め込むネットワークインターフェイスを選択します。

選択したネットワークインターフェイスが、デバイスからアクセスできることを確認します。接続が確立されているにもかかわらず、選択したネットワークインターフェイスにデバイスからアクセスできない場合は、ホストコンピューターの IP アドレスを求めるダイアログボックスが表示されます。

アプリケーションのデバッグ

1 USB ポートまたはネットワーク接続を経由して、デバイスに接続します。

2 実行／デバッグの構成を選択して、デバッグ用の起動設定を設定します。

- 「起動方法」で「デバイス上」を選択します。
- 「USB 経由でデバッグ」または「ネットワーク経由のデバッグ」を選択します。

  最初にネットワーク経由でアプリケーションをデバッグするときに、USB 経由でデバイスにアプリケーションをインストールできます。そうするためには、「アプリケーションを USB 経由のデバイスにインストール」を選択し、USB ポート経由でホストコンピューターにデバイスを接続します。

  アプリケーションをインストールしたときに、後続のデバッグセッションを USB 経由で接続したくない場合は、「アプリケーションを USB 経由のデバイスにインストール」の選択を解除します。

- （オプション）起動ごとにアプリケーションデータを消去

  デバッグセッションごとにアプリケーションの状態を維持する場合は、このオプションを選択します。このオプションは、アプリケーションで sessionCachingEnabled が True に設定されている場合にのみ適用されます。

3 「デバッグ」を選択して、デバッグセッションを開始します。

デバッガーが起動し、アプリケーションが開始するのを待機します。デバッガーがデバイスとの接続を確立すると、デバッグセッションが開始します。

ネットワーク経由のデバイスでデバッグを行おうとすると、アプリケーションで IP アドレスの指定を求めるダイアログボックスが表示されることがあります。ダイアログボックスには、デバッガーが接続できなかったことが示されます。デバイスがネットワークに正しく接続されていること、および Flash Builder を実行しているコンピューターにそのネットワークから接続できることを確認してください。

注意： 企業やホテルなどのゲストネットワークでは、デバイスとコンピューターが同じネットワークに存在していても、両者が接続できないことがあります。

ネットワーク経由でデバッグを行う場合に、アプリケーションが既にデバイスにインストールされているときは、ホストコンピューターの IP アドレスを入力することにより、デバッグを開始してください。


Flex のアドビ認定エキスパートの Brent Arnold 氏が、USB を介した Android デバイスのデバッグに関するビデオチュートリアルを公開しています。

関連項目

Debug and Package Apps for Devices（ビデオ）

Apple iOS デバイスでのアプリケーションのデバッグ

Apple iOS デバイス上のアプリケーションをデバッグするには、iOS デバイス上のデバッグ iOS パッケージ（IPA ファイル）を手動でデプロイおよびインストールします。Apple iOS プラットフォームでは、自動デプロイはサポートされていません。

重要：アプリケーションを iOS デバイス上でデバッグする前に、21 ページの「iOS アプリケーションの作成、デバッグまたはデプロイのための準備」で説明されている手順を行ってください。

1. Apple iOS デバイスを、開発コンピューターに接続します。
2. iOS デバイスで iTunes を起動します。

   注意：iOS デバイスにアプリケーションをインストールしたり、iOS デバイスのデバイス ID を取得したりするには、iTunes が必要です。
3. Flash Builder で、実行／デバッグの構成を選択します。
4. デバッグの構成ダイアログボックスで、次の手順に従います。
   a. デバッグするアプリケーションを選択します。
   b. ターゲットプラットフォームに、「Apple iOS」を選択します。
   c. 起動方法に、「デバイス上」を選択します。
   d. 次のいずれかのパッケージ化の方法を選択します。

      標準 この方法を使用すると、Apple iOS のデバイスで実行できるリリース品質のバージョンのアプリケーションをパッケージ化できます。この方法を使用したアプリケーションのパフォーマンスは、最終リリースパッケージのパフォーマンスと同等なので、Apple App Store に送信できます。

      ただし、この方法でデバッグ iOS（IPA）ファイルを作成すると数分かかります。

      高速 この方法を使用すると、IPA ファイルを迅速に作成してから、デバイス上でこのファイルを実行およびデバッグできます。この方法は、アプリケーションのテスト目的に適しています。この方法を使用したアプリケーションのパフォーマンスは、リリース品質ではなく、Apple App Store への登録には適していません。
   e. 「設定」をクリックして、適切なコード署名証明書、プロビジョニングファイル、およびパッケージの内容を選択します。
   f. 「ネットワークデバッグの設定」をクリックして、デバッグ iOS パッケージに追加するネットワークインターフェイスを選択します。

      注意：ホストコンピューターは、同時に複数のネットワークインターフェイスに接続できます。ただし、デバッグに使用するプライマリネットワークインターフェイスを選択できます。
   g. 「デバッグ」をクリックします。Flash Builder に、パスワードを求めるダイアログボックスが表示されます。P12 証明書のパスワードを入力します。

デバッグ IPA ファイルが生成され、bin-debug フォルダに格納されます。

5 iOS デバイスで、次の手順に従います。

1 （オプション）iTunes で、File ／ Add To Library を選択し、Apple から取得したモバイルプロビジョニングプロファイルファイル（.mobileprovision というファイル名拡張子）を参照します。

2 iTunes で、File ／ Add To Library を選択し、手順 4 で生成したデバッグ IPA ファイルを参照します。

3 File ／ Sync を選択し、iOS デバイスを iTunes に同期します。

4 Flash Builder で、デバッグ IPA ファイルで指定されたホストアドレスへの接続が試みられます。アプリケーションがホストアドレスに接続できない場合、ホストコンピューターの IP アドレスを求めるダイアログボックスが表示されます。

注意：デバッグ IPA パッケージを最後に生成してから、自分のコードやアセットを変更していない場合は、パッケージ化がスキップされ、アプリケーションがデバッグされます。つまり、デバイスにインストールされたアプリケーションを起動して「デバッグ」をクリックすると、Flash Builder デバッガーに接続できます。この方法により、アプリケーションを毎回パッケージ化することなく、繰り返しデバッグを行うことができます。

第 8 章：デバイスへのインストール

Google Android デバイスへのアプリケーションのインストール

プロジェクトの開発、テストおよびデプロイメントのフェーズで、アプリケーションをデバイスに直接インストールできます。

Flash Builder を使用して、アプリケーションを Android デバイスに直接インストールできます。Adobe AIR がインストールされていないデバイスにパッケージをインストールする場合、Flash Builder では AIR が自動的にインストールされます。

1 Google Android デバイスを開発コンピューターに接続します。

Flash Builder は、コンピューターの USB ポートに接続されているデバイスにアクセスします。必要な USB デバイスのドライバーが設定されていることを確認してください。18 ページの「Google Android デバイスの接続」を参照してください。

2 Flash Builder で、実行／実行構成を選択します。実行構成ダイアログボックスで、デプロイするモバイルアプリケーションを選択します。

3 起動構成方法に、「デバイス上」を選択します。

4 (オプション）起動ごとにアプリケーションデータを消去するかどうかを指定します。

5 「適用」をクリックします。

Flash Builder によって、Android デバイスにアプリケーションがインストールおよび起動されます。Adobe AIR がインストールされていないデバイスにパッケージをインストールした場合、Flash Builder では AIR が自動的にインストールされます。

Flex のアドビ認定エキスパートの Brent Arnold 氏が、Android デバイスのアプリケーションの設定と実行に関するビデオチュートリアルを公開しています。

Apple iOS デバイスへのアプリケーションのインストール

Apple iOS プラットフォームでは自動デプロイがサポートされていないため、iOS デバイスでは、アプリケーション（IPA ファイル）を手動でインストールします。

重要：iOS デバイスにアプリケーションをインストールする前に、Apple iOS 開発用証明書（P12 形式）および開発版のプロビジョニングプロファイルが必要です。21 ページの「iOS アプリケーションの作成、デバッグまたはデプロイのための準備」で説明されている手順を必ず実行してください。

1 Apple iOS デバイスを、開発コンピューターに接続します。

2 開発コンピューターで、iTunes を起動します。

注意：iOS デバイスにアプリケーションをインストールしたり、iOS デバイスの Unique Device Identifier（UDID）を取得したりするには、iTunes が必要です。

3 Flash Builder で、実行／実行構成を選択します。

4 実行構成ダイアログボックスで、次の手順に従います。

a インストールするアプリケーションを選択します。

b ターゲットプラットフォームに、「Apple iOS」を選択します。

c 起動方法に、「デバイス上」を選択します。

d 次のいずれかのパッケージ化の方法を選択します。

標準 この方法を使用すると、Apple iOS のデバイスで実行できるリリース品質のバージョンのアプリケーションをパッケージ化できます。

標準のパッケージ化の方法では、パッケージ化の前に、アプリケーションの SWF ファイルのバイトコードが ARM 命令に変換されます。パッケージ化の前に追加の変換手順があるため、この方法でアプリケーション（IPA）ファイルを作成すると数分かかります。標準の方法では、高速の方法よりも時間がかかります。ただし、標準の方法を使用したアプリケーションのパフォーマンスは、リリース品質なので、Apple App Store への登録に適しています。

高速 この方法を使用すると、IPA ファイルを迅速に作成できます。

高速のパッケージ化の方法では、バイトコードの変換がバイパスされて、アプリケーションの SWF ファイルとアセットがプリコンパイル済みの AIR ランタイムと単純にバンドルされます。高速のパッケージ化の方法は、標準の方法よりも高速です。ただし、高速の方法を使用したアプリケーションのパフォーマンスは、リリース品質ではないので、Apple App Store への登録には適していません。

注意：標準と高速のパッケージ化の方法の間には、ランタイムや機能上の違いはありません。

e 「設定」をクリックして、適切なコード署名証明書、プロビジョニングファイル、およびパッケージの内容を選択します。

f 「実行」をクリックします。Flash Builder に、パスワードを求めるダイアログボックスが表示されます。P12 証明書のパスワードを入力します。

IPA ファイルが生成され、bin-debug フォルダに格納されます。

5 開発用コンピューターで、次の手順に従います。

1 iTunes で、File ／ Add To Library を選択し、Apple から取得したモバイルプロビジョニングプロファイルファイル（.mobileprovision というファイル名拡張子）を参照します。

モバイルプロビジョニングプロファイルファイルを iTunes にドラッグ& ドロップすることもできます。

2 iTunes で、File ／ Add To Library を選択し、手順 4 で生成した IPA ファイルを参照します。

IPA ファイルを iTunes にドラッグ& ドロップすることもできます。

3 File ／ Sync を選択し、iOS デバイスを iTunes に同期します。

アプリケーションが iOS デバイスにデプロイされ、起動できるようになりました。

第 9 章：パッケージ化と書き出し

モバイルアプリケーションのパッケージ化とオンラインストアへの書き出し

モバイルアプリケーションのリリースビルドをパッケージ化して書き出すには、Flash Builder のリリースビルドの書き出し機能を使用します。リリースビルドとは通常、Android Market、Amazon Appstore、Apple App store などのオンラインストアにアップロードするアプリケーションの最終バージョンのことです。

アプリケーションを書き出す際に、デバイスにアプリケーションをインストールすることもできます。書き出し時にデバイスがコンピューターに接続されていると、Flash Builder はアプリケーションをデバイスにインストールします。プラットフォーム固有のアプリケーションパッケージを書き出して、後でデバイスにインストールすることもできます。作成されたパッケージは、ネイティブアプリケーションと同じ方法でデプロイしてインストールできます。

Android Market や Amazon App Store への Android アプリケーションの書き出しについて詳しくは、176 ページの「Android APK のリリース用パッケージの書き出し」を参照してください。

Apple App store への iOS アプリケーションの書き出しについて詳しくは、177 ページの「Apple iOS のリリース用パッケージの書き出し」を参照してください。

キャプティブランタイムアプリケーションの書き出し

リリースビルドの書き出し機能を使用してモバイルアプリケーションを書き出す場合、アプリケーションパッケージ内に AIR ランタイムを埋め込むように選択できます。

この場合、ユーザーは、AIR がインストールされていないデバイスでもアプリケーションを実行できます。パッケージの書き出し先のプラットフォームに応じて、キャプティブランタイムまたは共有ランタイムを使用できます。

Android APK のリリース用パッケージの書き出し

モバイルアプリケーションを書き出す前に、Android の権限をカスタマイズできます。アプリケーション記述ファイルで、設定を手動でカスタマイズします。これらの設定は、bin-debug¥**app_name**-app.xml ファイルの <android> ブロックにあります。詳しくは、AIR アプリケーション記述ファイルを参照してください。

後でデバイスにインストールするためにアプリケーションを書き出す場合は、デバイスの OS のプロバイダーが提供しているツールを使用して、アプリケーションパッケージをインストールします。

1. Flash Builder で、プロジェクト／リリースビルドの書き出しを選択します。
2. 書き出すプロジェクトとアプリケーションを選択します。
3. プロジェクトを書き出すターゲットプラットフォームと場所を選択します。
4. プラットフォーム固有のアプリケーションパッケージを書き出して署名します。

   アプリケーションは、各ターゲットプラットフォーム用に電子署名を使用してパッケージ化することも、デスクトップ用に電子署名済みの AIR アプリケーションとしてパッケージ化することもできます。

また、後で署名できる AIRI 中間ファイルとしてアプリケーションを書き出すこともできます。このオプションを選択した場合は、後で AIR の adt コマンドラインツールを使用して AIRI を APK ファイルとしてパッケージ化します。次に、プラットフォーム固有のツール（例えば、Android SDK では adb を使用）を使用して、デバイスに APK ファイルをインストールします。コマンドラインツールを使用してアプリケーションをパッケージ化する方法について詳しくは、178 ページの「コマンドラインを使用した作成、テスト、デプロイ」を参照してください。

5 「パッケージ化設定」ページでは、電子証明書、パッケージのコンテンツおよびネイティブエクステンションを選択できます。

デプロイメント また、デバイスへのアプリケーションのインストールも行う場合は、デプロイメントページをクリックし、「接続されているデバイスにアプリケーションをインストールして起動」を選択します。コンピューターの USB ポートに 1 つ以上のデバイスが接続されていることを確認します。

- **キャプティブランタイムアプリケーションの書き出し**

  アプリケーションパッケージの書き出し時に、APK ファイル内に AIR ランタイムを埋め込む場合は、このオプションを選択します。この場合、ユーザーは、AIR がインストールされていないデバイスでもアプリケーションを実行できます。

- **共有ランタイムを使用するアプリケーションを書き出し**

  アプリケーションパッケージの書き出し時に、APK ファイル内に AIR ランタイムを埋め込まない場合は、このオプションを選択します。ユーザーのデバイスに AIR がインストールされていない場合に、アプリケーションパッケージで必要な Adobe AIR をダウンロードするための URL を選択または指定できます。

  デフォルト URL は Android Market を指しています。ただし、デフォルト URL をオーバーライドして、Amazon Appstore の場所を指す URL を選択するか、独自の URL を入力できます。

電子署名「電子署名」タブをクリックし、アプリケーション発行者の ID を表す電子証明書を作成または参照します。選択した証明書に対してパスワードを指定することもできます。

証明書を作成する場合、証明書は自己署名になります。商用の署名付き証明書は、証明書のプロバイダーから取得できます。AIR アプリケーションへのデジタル署名を参照してください。

パッケージのコンテンツ（オプション）「パッケージのコンテンツ」タブをクリックして、パッケージに含めるファイルを指定します。

ネイティブエクステンション（オプション）アプリケーションパッケージに含めるネイティブエクステンションを選択します。

ネイティブエクステンションについて詳しくは、13 ページの「ネイティブエクステンションの使用」を参照してください。

6 「終了」をクリックします。

Flash Builder によって、最初のパネルで指定したディレクトリ（デフォルトはプロジェクトの最上位ディレクトリ）に、***ApplicationName***.apk が作成されます。書き出し中にデバイスがコンピューターに接続されていた場合は、アプリケーションがデバイスにインストールされます。

Apple iOS のリリース用パッケージの書き出し

iOS パッケージは、アドホックで配布するため、または Apple App Store に登録するために作成および書き出しが可能です。

重要：iOS パッケージを書き出す前に、必要な証明書と配布プロビジョニングプロファイルを Apple から取得していることを確認してください。そうするには、21 ページの「iOS アプリケーションの作成、デバッグまたはデプロイのための準備」で説明されている手順を行います。

1 Flash Builder で、プロジェクト／リリースビルドの書き出しを選択します。

2 書き出し先のターゲットプラットフォームとして「Apple iOS」を選択して、IPA パッケージに署名します。

「次へ」をクリックします。

3 Apple から取得した P12 証明書と配布プロビジョニングプロファイルを選択します。

4 「パッケージ化設定」ページでは、プロビジョニング証明書、電子証明書、パッケージのコンテンツおよびネイティブエクステンションを選択できます。

デプロイメント デフォルトでは、iOS パッケージを書き出すと、AIR ランタイムが IPA ファイル内に埋め込まれます。

電子署名 Apple から取得した P12 証明書と配布プロビジョニングプロファイルを選択します。

次のいずれかのパッケージタイプを選択できます。

- **限定的な配布用の Ad hoc パッケージ** アプリケーションを制限付きで配布する場合
- **Apple App Store への最終リリースパッケージ** アプリケーションを Apple App Store に送信する場合

パッケージのコンテンツ（オプション）「パッケージのコンテンツ」タブをクリックして、パッケージに含めるファイルを指定します。

ネイティブエクステンション（オプション）アプリケーションパッケージに含めるネイティブエクステンションを選択します。

ネイティブエクステンションが iOS5 SDK の機能を使用している場合は、iOS SDK の場所を選択します。詳しくは、15 ページの「iOS5 ネイティブエクステンションのサポート」を参照してください。

5 「終了」をクリックします。

Flash Builder で、パッケージ設定を検証してから、アプリケーションをコンパイルします。パッケージ化が完了すると、接続した Apple iOS デバイスに IPA ファイルをインストールしたり、Apple App Store に送信したりできます。

AIR 開発ツール（ADT）を使用して IPA ファイルをパッケージ化するには、『Adobe AIR アプリケーションの構築』の「iOS パッケージ」を参照してください。

コマンドラインを使用した作成、テスト、デプロイ

モバイルアプリケーションは、Flash Builder を使用しなくても作成できます。代わりに、mxmlc、adl および adt コマンドラインツールを使用します。

次に、コマンドラインツールを使用してモバイルアプリケーションを開発し、デバイスにインストールする一般的なプロセスを示します。各手順については、後でより詳しく説明します。

1 mxmlc ツールを使用してアプリケーションをコンパイルします。

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

この手順では、「airmobile」に設定した configname パラメーターを渡す必要があります。

2 adl ツールを使用して、AIR Debug Launcher（ADL）でアプリケーションをテストします。

```
adl MyMobileApp-app.xml -profile mobileDevice
```

この手順では、アプリケーション記述ファイルを作成し、そのファイルを引数として adl ツールに渡す必要があります。また、mobileDevice プロファイルも指定します。

3 adt ツールを使用してアプリケーションをパッケージ化します。

```
adt -package -target apk SIGN_OPTIONS MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

この手順では、最初に証明書を作成する必要があります。

4 アプリケーションをモバイルデバイスにインストールします。アプリケーションを Android デバイスにインストールするには、adb ツールを使用します。

```
adb install -r MyMobileApp.apk
```

この手順では、まずモバイルデバイスを USB 経由でコンピューターに接続する必要があります。

5 モバイルアプリケーションをオンラインストアにデプロイします。

mxmlc を使用したモバイルアプリケーションのコンパイル

mxmlc コマンドラインコンパイラーを使用して、モバイルアプリケーションをコンパイルできます。mxmlc を使用するには、configname パラメーターに airmobile という値を渡します。次に例を示します。

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

+configname=airmobile を渡すことによって、airmobile-config.xml ファイルを使用することをコンパイラーに指示します。このファイルは、sdk¥frameworks ディレクトリにあります。このファイルは次のタスクを実行します。

- mobile.swc テーマを適用します。
- 次のライブラリパス変更を行います。
  - ライブラリパスから libs¥air を削除します。モバイルアプリケーションでは、Window クラスおよび WindowedApplication クラスはサポートされません。
  - ライブラリパスから libs¥mx を削除します。モバイルアプリケーションでは、MX コンポーネント（チャートを除く）はサポートされません。
  - ライブラリパスに libs¥mobile を追加します。
- ns.adobe.com¥flex¥mx 名前空間と www.adobe.com¥2006¥mxml 名前空間が削除されます。モバイルアプリケーションでは、MX コンポーネント（チャートを除く）はサポートされません。
- アクセシビリティを無効にします。
- RSL エントリを削除します。モバイルアプリケーションでは RSL はサポートされません。

mxmlc コンパイラーによって SWF ファイルが生成されます。

adl を使用したモバイルアプリケーションのテスト

モバイルアプリケーションをテストするには、AIR Debug Launcher（ADL）を使用できます。ADL を使用してアプリケーションを実行およびテストする場合は、アプリケーションをあらかじめパッケージ化してデバイスにインストールしておく必要がありません。

adl ツールを使用したデバッグ

ADL では、トレースステートメントとランタイムエラーが標準出力に出力されますが、ブレークポイントやその他のデバッグ機能はサポートされていません。複雑なデバッグ問題については、Flash Builder などの統合開発環境を使用できます。

adl ツールの起動

コマンドラインから adl ツールを起動するには、モバイルアプリケーションのアプリケーション記述ファイルを渡し、profile パラメーターを mobileDevice に設定します。次に例を示します。

```
adl MyMobileApp-app.xml -profile mobileDevice
```

mobileDevice プロファイルでは、モバイルデバイスにインストールされているアプリケーションの一連の機能が定義されます。mobileDevice プロファイルについて詳しくは、様々なプロファイルの機能を参照してください。

アプリケーション記述の作成

アプリケーションのコンパイルに Flash Builder を使用しなかった場合は、アプリケーション記述ファイルを手動で作成します。作成のベースとして、¥sdk¥samples¥descriptor-sample.xml ファイルを使用できます。通常、少なくとも次の変更を加える必要があります。

- <initialWindow><content> エレメントがモバイルアプリケーションの SWF ファイルの名前を指し示すようにします。

```
<initialWindow>
    <content>MyMobileApp.swf</content>
    ...
</initialWindow>
```

- アプリケーションのタイトルを変更します。これは、このタイトルが、モバイルデバイス上のアプリケーションのアイコンの下に表示される内容であるためです。タイトルを変更するには、<name><text> エレメントを編集します。

```
<name>
    <text xml:lang="en">MyMobileApp by Nick Danger</text>
</name>
```

- <android> ブロックを追加して、アプリケーションに対する Android 固有の権限を設定します。デバイスで使用するサービスにもよりますが、多くの場合、次の権限を使用できます。

```
<application>
    ...
    <android>
        <manifestAdditions>
            <![CDATA[<manifest>
                <uses-permission android:name="android.permission.INTERNET"/>
            </manifest>]]>
        </manifestAdditions>
    </android>
</application>
```

記述ファイルでは、アプリケーションの高さと幅、アイコンファイルの場所、バージョン管理情報、およびインストール場所に関するその他の詳細も設定できます。

アプリケーション記述ファイルの作成および編集について詳しくは、AIR アプリケーション記述ファイルを参照してください。

adt を使用したモバイルアプリケーションのパッケージ化

AIR 開発ツール（ADT）を使用して、コマンドラインでモバイルアプリケーションをパッケージ化します。adt ツールによって、Android モバイルデバイスにデプロイできる APK ファイルを作成できます。

証明書の作成

APK ファイルを作成する前に、証明書を作成します。開発目的の場合は、自己署名証明書を使用できます。自己署名証明書は、adt ツールを使用して作成できます。次に例を示します。

```
adt -certificate -cn SelfSign -ou QE -o "Example" -c US 2048-RSA newcert.p12 password
```

adt ツールによって、カレントディレクトリに newcert.p12 ファイルが作成されます。アプリケーションをパッケージ化するときに、この証明書を adt に渡します。実稼働アプリケーションには自己署名証明書を使用しないでください。自己署名証明書は、限られた保証しかユーザーに提供しません。公認された認証機関が発行する証明書を使用して AIR インストールファイルに署名する方法について詳しくは、AIR アプリケーションへの署名を参照してください。

パッケージファイルの作成

Android に APK ファイルを作成するには、証明書などのアプリケーションに関する詳細を adt ツールに渡します。次に例を示します。

```
adt -package -target apk -storetype pkcs12 -keystore newcert.p12 -keypass password MyMobileApp.apk
MyMobileApp-app.xml MyMobileApp.swf
```

adt ツールの出力は、**appname**.apk ファイルです。

iOS 用のパッケージ化

iOS モバイルアプリケーションをパッケージ化するには、プロビジョニングファイルとともに、Apple から開発者証明書を取得する必要があります。これには、Apple の developer program への参加が必要です。詳しくは、21 ページの「iOS アプリケーションの作成、デバッグまたはデプロイのための準備」を参照してください。

Flex のエバンジェリスト Piotr Walczyszyn 氏が、iOS デバイス用のアプリケーションを ADT と ANT を使用してパッケージ化する方法について説明しています。

ブロガーの Valentin Simonov 氏が、iOS でアプリケーションを発行する方法についての追加情報を提供しています。

adb によるデバイスへのモバイルアプリケーションのインストール

Android を実行しているモバイルデバイスにアプリケーション（APK ファイル）をインストールするには、Android Debug Bridge（adb）を使用します。adb ツールは、Android SDK に含まれています。

デバイスのコンピューターへの接続

adb を実行して APK ファイルをモバイルデバイスにインストールする前に、デバイスをコンピューターに接続します。Windows および Linux システムの場合、デバイスの接続には USB ドライバーが必要です。

使用しているデバイス用の USB ドライバーのインストールについて詳しくは、Using Hardware Devices を参照してください。

接続されたデバイスへのアプリケーションのインストール

デバイスをコンピューターに接続した後、アプリケーションをデバイスにインストールできます。adb ツールを使用してアプリケーションをインストールするには、install オプションを使用し、APK ファイルの名前を渡します。次に例を示します。

```
adb install -r MyMobileApp.apk
```

アプリケーションが既にインストールされている場合は、-r オプションを使用してアプリケーションを上書きします。このオプションを使用しない場合は、新しいバージョンをモバイルデバイスにインストールするたびに、アプリケーションをアンインストールする必要があります。

関連項目

Android Debug Bridge

オンラインストアへのアプリケーションのデプロイ

Android Market、Amazon Appstore、Apple の App store などのオンラインアプリケーションストアに、アプリケーションをデプロイできます。

Lee Brimlow 氏が、Android アプリケーション用の新規 AIR を Android Market にデプロイする方法について説明しています。

Christian Cantrell 氏が、アプリケーションを Amazon Appstore for Android にデプロイする方法について説明しています。