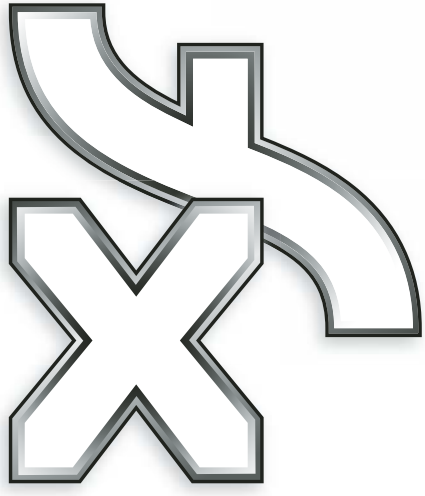


Adobe

Adobe Flex 2

開発ガイド

Flex™ 2 開発ガイド

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Part Number: 90069414 (12/06)

Part Number: 90070488 (12/06)

目 次

第 2 部 : FLEX アプリケーションのユーザーインターフェイスの作成

第 1 章

Flex マニュアルについて

1

『Flex 2 開発ガイド』は、Adobe® Flex™ 2 を使用してインターネットアプリケーションを開発するときの参考書です。本マニュアルは、Flex について学習中であるか、または Flex プログラミングの知識を深めたいと考えているアプリケーション開発者の方を対象としています。本マニュアルを通じて、Flex に用意されたアプリケーション開発のための各種ツールについて、基礎知識を身に付けることができます。

目次

本マニュアルの使い方

本マニュアルには、Flex アプリケーションの開発に役立つ情報が記載されています。本マニュアルは、Flex の基本操作を実施したことがある場合や、『Flex 2 ファーストステップガイド』を既に読んでいる場合に最も効果的です。『Flex 2 ファーストステップガイド』には、Flex の紹介が記載されており、本マニュアルの利用に役立つ基本的な知識を身につけることができます。

『Flex 2 開発ガイド』は、以下の 7 部で構成されます。

各部のタイトル	説明
第 1 部の「Flex プログラミング言語の使用」	MXML と ActionScript の使用方法について説明します。
第 2 部の「Flex アプリケーションのユーザーインターフェイスの作成」	Flex コンポーネントを使用してアプリケーションのユーザーインターフェイスを作成する方法について説明します。
第 3 部の「ユーザーインターフェイスの カスタマイズ」	アプリケーションに追加の機能を付加してユーザーの操作性を向上させる方法について説明します。
第 4 部の「Flex プログラミングに関する トピック」	よりインタラクティブで表現力の高いアプリケーションを作成するための高度なプログラミング手法を紹介します。
第 5 部の「Flex データ機能」	Flex データ表現およびデータ機能の使用方法について説明します。
第 6 部の「データアクセスと相互接続性」	外部データを操作するための Flex 機能について説明します。これには Adobe® Flex™ データサービス機能などがあります。
第 7 部の「チャートコンポーネント」	チャートコンポーネントの使用方法について説明します。

Flex マニュアルへのアクセス

Flex マニュアルは、すべてのユーザーを対象としています。

マニュアルセット

Flex マニュアルセットには、以下のマニュアルが含まれています。

マニュアル名	説明
Flex 2 開発ガイド	ダイナミックな Web アプリケーションの開発方法について説明します。
Flex 2 ファーストステップガイド	Flex の機能とアプリケーション開発手順の概要について説明します。
Flex 2 アプリケーションの構築および展開ガイド	Flex アプリケーションの構築およびデプロイ方法について説明します。
Flex 2 コンポーネントの作成と拡張	Flex コンポーネントの作成および拡張方法について説明します。
既存の Flex アプリケーションの Flex 2 への移行	移行プロセスの概要や、Flex と ActionScript の変更点に関する詳しい説明を記載しています。
Flex Builder 2 ユーザーガイド	あらゆるレベルの Flex Builder ユーザーを対象として、Adobe® Flex™ Builder™ 2 の全機能について解説しています。
Adobe Flex 2 リファレンスガイド	Flex API の説明、シンタックス、用途、およびコード例を記載しています。

オンラインマニュアルの参照

すべての Flex マニュアルは、HTML および PDF (Adobe® Portable Document Format) ファイル形式でオンラインで提供されています。Adobe の Web サイトより入手いただけます。Adobe® Flex™ Builder™ の [ヘルプ] メニューからもアクセスできます。

表記規則

本マニュアルでは、次の表記規則を使用しています。

- イタリック体で表記されている部分は、置き換える必要のある値 (フォルダパスの一部など) です。
- `Code font` で表記されている部分はコードです。
- *`Code font italic`* で表記されている部分はパラメータです。
- **ボールド体**で表記されている部分は入力されたとおりの文字列です。

第 1 部

Flex プログラミング言語の使用

1

第 I 部では、Adobe Flex 2 プログラミング言語の MXML と ActionScript の使用方法について説明します。

次のトピックが含まれます。

第 2 章

MXML でのアプリケーション開発

2

MXML は XML 言語の一種で、Adobe Flex アプリケーションのユーザーインターフェイスコンポーネントをレイアウトするために使用します。MXML は、たとえばサーバーサイドのデータソースへのアクセスや、ユーザーインターフェイスコンポーネントとサーバーサイドのデータソースの間のデータバインディングなど、アプリケーションの非ビジュアルな側面を宣言的に定義するために使用します。このトピックでは MXML について説明し、Flex アプリケーション開発での MXML の使い方を示します。

MXML のシンタックスの詳細については、41 ページ、第 3 章の「MXML のシンタックス」を参照してください。

目次

MXML について

Flex アプリケーションは、MXML と ActionScript の 2 つの言語を組み合わせて記述します。MXML は XML マークアップ言語の一種で、ユーザーインターフェイスコンポーネントをレイアウトするために使用します。MXML は、たとえばサーバー上のデータソースへのアクセスや、ユーザーインターフェイスコンポーネントとサーバー上のデータソースの間のデータバインディングなど、アプリケーションの非ビジュアルな側面を宣言的に定義するために使用します。

HTML と同様に、MXML にはユーザーインターフェイスを定義するためのタグが用意されています。HTML の経験があれば、MXML に違和感を覚えることはないでしょう。しかし、MXML は HTML よりも構造化されており、提供されるタグセットも豊富です。たとえば、MXML には、Web サービス接続、データバインディング、およびアニメーションエフェクトを提供する非ビジュアルコンポーネント用のタグだけでなく、データグリッド、ツリー、タブナビゲータ、アコーディオン、メニューなどのビジュアルコンポーネント用のタグも用意されています。さらに、MXML タグとして参照するカスタムコンポーネントを使用して、MXML を拡張することもできます。

MXML と HTML の最も大きな違いの 1 つは、MXML で定義されたアプリケーションは SWF ファイルにコンパイルされて Adobe® Flash® Player でレンダリングされる点です。これにより、ページベースの HTML アプリケーションよりも多機能でダイナミックなユーザーインターフェイスを提供できます。

MXML アプリケーションは 1 つのファイルに記述することも、また複数のファイルに記述することもできます。MXML では、MXML および ActionScript ファイルに記述されたカスタムコンポーネントもサポートされます。

単純なアプリケーションの記述

MXML ファイルは普通の XML ファイルなので、いろいろな開発環境を選択できます。MXML コードは、単純なテキストエディタ、XML 専用エディタ、またはテキスト編集をサポートする統合開発環境 (IDE) で記述できます。Flex には、アプリケーション開発に使用できる Adobe Flex Builder という専用の IDE が付属しています。

次の例は、<mx:Application> タグと 2 つの子タグ <mx:Panel> および <mx:Label> だけを含む単純な "Hello World" アプリケーションです。<mx:Application> タグは Application コンテナを定義するタグで、常に Flex アプリケーションのルートタグになります。<mx:Panel> タグは、タイトルバー、タイトル、ステータスメッセージ、境界線、および子のコンテンツ領域が含まれる Panel コンテナを定義します。<mx:Label> タグは Label コントロールを表します。これはテキストを表示する非常に単純なユーザーインターフェイスコンポーネントです。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <mx:Label text="Hello World!" fontWeight="bold" fontSize="24"/>

  </mx:Panel>
</mx:Application>
```

このコードを "hello.mxml" という名前のファイルに保存します。MXML ファイルの名前は、小文字の拡張子 .mxml で終了する必要があります。

次の図は、Web ブラウザウィンドウに表示された "Hello World" アプリケーションです。

XML エンコーディングについて

ドキュメントの1行目では、XML バージョンのオプションの宣言を指定しています。MXML ファイルのエンコード方法を指定するエンコーディング情報を含めることをお勧めします。多くのエディタでは、一連のファイルエンコーディングオプションからエンコード情報を選択できるようになっています。北米で使用されているオペレーティングシステムでは、ISO-8859-1エンコーディング形式が主に使用され、ほとんどのプログラムでこの形式がデフォルトで使用されています。UTF-8 エンコーディング形式を使用すると、最大限のプラットフォーム互換性を確保できます。UTF-8 では、ファイル内のすべての文字に一意な数値が付けられます。UTF-8 は、プラットフォーム、プログラム、および言語のいずれにも依存しません。エンコーディング形式を指定する場合は、使用するファイルエンコーディングと一致する必要があります。次に示すのは、UTF-8 エンコーディング形式を指定する XML 宣言タグの例です。

```
<?xml version="1.0" encoding="utf-8"?>
```

<mx:Application> タグについて

<mx:Application> タグは、Flex アプリケーションのルートタグであるだけでなく、Application コンテナも表します。" コンテナ " は、他のコンポーネントを含み、その子コンポーネントの配置に関する組み込みレイアウト規則を持つユーザーインターフェイスコンポーネントです。デフォルトでは、Application コンテナの子は上から下に垂直にレイアウトされます。Application コンテナ内に別の種類のコンテナ (上記の Panel コンテナなど) を配置することで、他の規則に従ってユーザーインターフェイスコンポーネントを配置することもできます。詳細については、127 ページ、第 6 章の「Flex ビジュアルコンポーネントの使用」を参照してください。

MXML タグプロパティについて

MXML タグのプロパティ (たとえば <mx:Label> タグの text、fontWeight、fontSize プロパティなど) を使用すると、コンポーネントの初期状態を宣言的に設定できます。実行時にコンポーネントの状態を変更するには、<mx:Script> タグ内で ActionScript コードを使用します。詳細については、55 ページ、第 4 章の「ActionScript の使用」を参照してください。

MXML の SWF ファイルへのコンパイル

アプリケーションはコンパイルされた SWF ファイルとしてデプロイできます。また、Adobe Flex データサービス がある場合は、アプリケーションを MXML および ActionScript ファイルのセットとしてデプロイすることも可能です。

Flex Builder を使用している場合は、Flex Builder 内でアプリケーションを SWF ファイルにコンパイルして実行します。アプリケーションが正常に実行されたら、SWF ファイルを Web サーバーまたはアプリケーションサーバーのディレクトリにコピーしてデプロイします。ユーザーは次の形式で HTTP 要求を送信して、デプロイされた SWF ファイルにアクセスします。

```
http://hostname/path/filename.swf
```

Flex にはコマンドライン MXML コンパイラの mxmlc もあり、これを使用して MXML ファイルをコンパイルすることもできます。コマンドラインから mxmlc を使用して "hello.mxml" をコンパイルする例を次に示します。

```
cd flexInstallDir/bin
mxmlc --show-actionscript-warnings=true --strict=true c:¥appDir¥hello.mxml
```

この例では、flexInstallDir は Flex のインストールディレクトリ、appDir は "hello.mxml" を含むディレクトリを表します。生成された SWF ファイル (hello.swf) は "hello.mxml" と同じディレクトリに書き込まれます。

mxmlc の詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 9 章の「Flex コンパイラの使用」を参照してください。Flash Player のデバッガバージョンの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 11 章の「ログ機能」を参照してください。

MXML タグと ActionScript クラスの関係

Flex は ActionScript クラスライブラリとして実装されています。クラスライブラリには、コンポーネント (コンテナおよびコントロール)、マネージャクラス、データサービスクラス、およびその他の機能に関するクラスが含まれています。アプリケーションを開発するときは、MXML および ActionScript 言語とクラスライブラリを組み合わせて使用します。

MXML タグは ActionScript クラスまたはクラスのプロパティに対応します。Flex は、MXML タグを解析し、対応する ActionScript オブジェクトを含む SWF ファイルをコンパイルします。たとえば、Flex には Flex Button コントロールを定義する ActionScript Button クラスがあります。MXML では、次の MXML ステートメントを使用して Button コントロールを作成します。

```
<mx:Button label="Submit"/>
```

MXML タグを使用してコントロールを宣言する場合は、そのクラスのインスタンスオブジェクトを作成します。この MXML ステートメントは、Button オブジェクトを作成し、Button オブジェクトの label プロパティを "Submit" というストリングに初期化します。

ActionScript クラスに対応する MXML タグは、ActionScript クラスと同じ命名規則を使用します。クラス名は大文字で始まり、クラス名の中でも大文字で単語が分けられます。すべての MXML タグ属性は、ActionScript オブジェクトのプロパティ、オブジェクトに適用されるスタイル、またはオブジェクトのイベントリスナーに対応します。Flex クラスライブラリおよび MXML タグのシンタックスの詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

Flex アプリケーション構造について

MXML アプリケーションは 1 つのファイルに記述することも、また複数のファイルに記述することもできます。通常は <mx:Application> タグを含むメインファイルを定義します。メインファイルの中から、MXML、ActionScript、またはその両方の組み合わせで記述した別のファイルを参照できます。

一般的なコーディング方法として、Flex アプリケーションを機能単位つまりモジュールに分割し、ウォッチモジュールが個別のタスクを実行するようにします。Flex では、アプリケーションを別々の MXML ファイルおよび ActionScript ファイルに分割できます。これらの各ファイルはそれぞれ異なるモジュールに対応します。アプリケーションをモジュールに分割すると、次のような多くのメリットが得られます。

開発の容易さ　異なる開発者または開発グループが、互いに独立してモジュールを開発およびデバッグできます。

再利用性　異なるアプリケーションでモジュールを再利用できるので、重複した作業を行う必要がなくなります。

保守性　アプリケーションを単一のファイルで開発した場合よりも迅速にエラーを隔離しデバッグできます。

Flex では、モジュールは、MXML または ActionScript のいずれかで実装されるカスタムコンポーネントに相当します。これらのカスタムコンポーネントは他のカスタムコンポーネントを参照できます。Flex ではコンポーネント参照のネストレベルに制限はありません。アプリケーションの必要に応じてコンポーネントを定義できます。

アプリケーションの開発

MXML での開発は、HTML、JSP (JavaServer Pages)、ASP (Active Server Pages)、CFML (ColdFusion Markup Language) といった他の Web アプリケーションファイルと同様、反復的なプロセスに基づきます。Flex アプリケーションの開発は、任意のテキストエディタを開いて、XML タグを入力し、ファイルを保存して、そのファイルの URL を Web ブラウザで開くという作業の繰り返しです。

Flex にはコードデバッグ用のツールも用意されています。詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 12 章の「コマンドラインデバッガの使用」を参照してください。

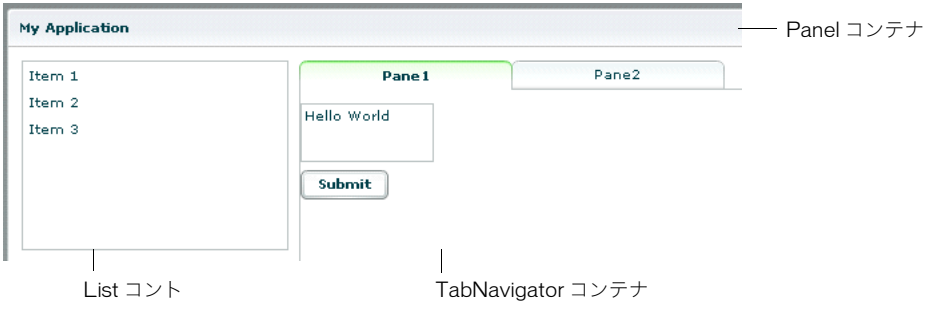
コンテナを使用したユーザーインターフェイスのレイアウト

Flex のモデルビューデザインパターンでは、ユーザーインターフェイスコンポーネントがビューを表します。MXML 言語は、コントロールとコンテナの 2 種類のユーザーインターフェイスコンポーネントをサポートしています。コントロールとは、ボタンやテキストフィールド、リストボックスなどのフォームエレメントです。コンテナとは、コントロールや他のコンテナを収めるための、画面上の矩形の領域です。

コンテナコンポーネントは、ユーザーインターフェイスのレイアウトと、アプリケーションを通じたユーザーナビゲーションの制御に使用します。レイアウトコンテナの例には、子コンポーネントを水平にレイアウトするための HBox コンテナ、子コンポーネントを垂直にレイアウトするための VBox コンテナ、子コンポーネントを行と列にレイアウトするための Grid コンテナなどがあります。ナビゲータコンテナの例には、タブ付きパネルを作成するための TabNavigator コンテナ、折りたたみ可能なパネルを作成するための Accordion ナビゲータコンテナ、パネルを重ねてレイアウトするための ViewStack ナビゲータコンテナなどがあります。

Container クラスは、すべての Flex コンテナクラスの基本クラスです。Container クラスを拡張したコンテナには、子コンポーネントをレイアウトするための独自の機能が追加されています。コンテナタグの一般的なプロパティには、id、width、height などがあります。標準の Flex コンテナの詳細については、461 ページ、第 13 章の「コンテナについて」を参照してください。

次の図は、ユーザーインターフェイスの左側に List コントロール、右側に TabNavigator コンテナを含む Flex アプリケーションを示します。どちらのコントロールも Panel コンテナの中にあります。



このアプリケーションを実装するコードを次に示します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10">

    <mx:HBox>
      <!-- 3 つのアイテムからなるリスト -->
```

```
        <mx:List>
          <mx:dataProvider>
            <mx:Array>
              <mx:String>Item 1</mx:String>
              <mx:String>Item 2</mx:String>
              <mx:String>Item 3</mx:String>
            </mx:Array>
          </mx:dataProvider>
        </mx:List>

        <!-- TabNavigator の 1 番目のペイン -->
        <mx:TabNavigator borderStyle="solid">
          <mx:VBox label="Pane1" width="300" height="150">
            <mx:TextArea text="Hello World"/>
            <mx:Button label="Submit"/>
          </mx:VBox>

          <!-- TabNavigator の 2 番目のペイン -->
          <mx:VBox label="Pane2" width="300" height="150">
            <!-- ストックビューを挿入する -->
          </mx:VBox>

        </mx:TabNavigator>
      </mx:HBox>
    </mx:Panel>
</mx:Application>
```

List コントロールと TabNavigator コンテナは HBox コンテナ内に含まれているので、水平方向にレイアウトされます。TabNavigator コンテナ内のコントロールは VBox コンテナ内に含まれているので、垂直方向にレイアウトされます。

ユーザーインターフェイスコンポーネントのレイアウトの詳細については、127 ページ、第 6 章の「Flex ビジュアルコンポーネントの使用」を参照してください。

ユーザーインターフェイスコントロールの追加

Flex には、Button コントロール、TextInput コントロール、ComboBox コントロールなど、さまざまなユーザーインターフェイスコンポーネントが用意されています。コンテナを使用してアプリケーションのレイアウトとナビゲーションを定義したら、次にユーザーインターフェイスコントロールを追加します。

次の例では HBox (Horizontal Box) コンテナを使用し、その中に子コントロールとして TextInput コントロールと Button コントロールを配置しています。HBox コンテナの子は水平方向にレイアウトされます。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
  <mx:HBox>
    <mx:TextInput id="myText"/>
    <mx:Button click="storeZipInDatabase(myText.text)"/>
  </mx:HBox>

</mx:Application>
```

コントロールタグの一般的なプロパティには、id、width、height、fontSize、color などのプロパティ、click や change などのイベントのイベントリスナー、showEffect や rollOverEffect などのエフェクトトリガがあります。標準の Flex コントロールの詳細については、241 ページ、第 9 章の「コントロールの使用」を参照してください。

MXML タグの id プロパティの使用

いくつかの例外を除き (54 ページの「MXML タグの規則」を参照)、MXML タグはオプションの id プロパティを持ちます。このプロパティは、MXML ファイル内で一意にする必要があります。タグが id プロパティを持つ場合は、ActionScript で対応するオブジェクトを参照できます。

次の例では、Web サービス要求の結果が writeToLog 関数によりトレースされます。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
...

  <mx:VBox>
    <mx:TextInput id="myText" text="Hello World!" />
    <mx:Button id="mybutton" label="Get Weather" click="writeToLog();"/>
  </mx:VBox>
  <mx:Script>
    <![CDATA[
      private function writeToLog():void {
        trace(myText.text);
      }
    ]]>
  </mx:Script>
</mx:Application>
```

このコードをコンパイルすると、myText という名前のパブリック変数が自動生成され、その中に TextInput インスタンスへの参照が格納されます。この自動生成された変数を ActionScript で使用することにより、コンポーネントのインスタンスにアクセスできます。つまり、任意の ActionScript クラスまたはスクリプトブロックの中で TextInput コントロールの id インスタンス参照を使用して、TextInput コントロールのインスタンスを明示的に参照することが可能です。コンポーネントのインスタンスを参照することで、プロパティを変更したり、メソッドを呼び出したりすることができます。

MXML ファイル内のそれぞれの id 値は一意なので、ファイル内のすべてのオブジェクトは同じフラットな名前空間に属します。myVBox.myText.text のようなドット表記で親を参照してオブジェクトを修飾する必要はありません。

詳細については、61 ページの「Flex コンポーネントの参照」を参照してください。

XML 名前空間の使用

XML ドキュメント内では、タグは名前空間に関連付けられます。XML 名前空間を使用すると、同じ XML ドキュメント内の XML タグの複数のセットを参照できます。MXML タグの xmlns プロパティは、XML 名前空間を指定します。デフォルトの名前空間を使用するには、接頭辞を省略します。追加のタグを使用するには、タグ接頭辞と名前空間を指定します。たとえば、次の <mx:Application> タグ内の xmlns プロパティは、MXML 名前空間内のタグに対して接頭辞 "mx:" を使用することを指定します。この MXML 名前空間の URI (Universal Resource Identifier) は http://www.adobe.com/2006/mxml です。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

XML 名前空間では、MXML 名前空間に含まれていないカスタムタグを使用することができます。次に、CustomBox という名前のカスタムタグを含むアプリケーションの例を示します。名前空間値 containers.boxes.* は、CustomBox という名前の MXML コンポーネントが "containers/boxes" ディレクトリ内にあることを示します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:MyComps="containers.boxes.*">

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10">

    <MyComps:CustomBox/>

  </mx:Panel>
</mx:Application>
```

"containers/boxes" ディレクトリは、アプリケーションファイルを格納しているディレクトリのサブディレクトリであっても、または "flex-config.xml" ファイル内で定義されている ActionScript ソースパスディレクトリのサブディレクトリであってもかまいません。同じファイルがこの両方の場所に存在する場合は、アプリケーションファイルディレクトリ内のファイルが使用されます。接頭辞名は任意ですが、宣言どおりに使用する必要があります。

SWC ファイルに含まれているコンポーネントを使用する場合は、SWC ファイルがそれを使用する MXML ファイルと同じディレクトリ内にある場合でも、パッケージ名と名前空間が一致する必要があります。SWC ファイルは、Flex コンポーネントのアーカイブファイルです。SWC ファイルを使用することで、Flex 開発者は互いに効率よくコンポーネントをやり取りできます。単一のファイルを交換するだけで済み、MXML ファイルや ActionScript ファイル、イメージなどのリソースファイルを逐一送信する必要はありません。また、SWF ファイルが SWC ファイル内にコンパイルされているため、コードを不用意に露出するのを避けることができます。SWC ファイルの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 9 章の「Flex コンパイラの使用」を参照してください。

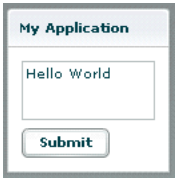
MXML を使用したランタイムコードのトリガ

Flex アプリケーションは、たとえばユーザーが Button コントロールをクリックしたときなどに、ランタイムイベントに基づいて動作します。ランタイムイベントを処理するためのコードから構成される " イベントリスナー " を、MXML タグのイベントプロパティに指定できます。たとえば、<mx:Button> タグには click イベントプロパティがあり、実行時に Button コントロールがクリックされたときに実行される ActionScript コードをここに指定できます。イベントリスナーのコードが単純であれば、イベントプロパティに直接指定できます。複雑なコードを使用するには、<mx:Script> タグ内に定義した ActionScript 関数の名前を指定します。

次の例は、Button コントロールと TextArea コントロールを 1 つずつ含むアプリケーションです。Button コントロールの click プロパティには、TextArea コントロールの text プロパティの値を "Hello World" というテキストに設定する単純なイベントリスナーが指定されています。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10">

    <mx:TextArea id="textarea1"/>
    <mx:Button label="Submit" click="textarea1.text='Hello World';"/>

  </mx:Panel>
</mx:Application>
```

次の図は、Web ブラウザウィンドウに表示されたアプリケーションです。



次の例は、<mx:Script> タグ内の ActionScript 関数でイベントリスナーを指定したアプリケーションのコードです。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
      private function hello():void {
        textarea1.text="Hello World";
      }
    ]]>
  </mx:Script>
```

```
  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <mx:TextArea id="textarea1"/>
    <mx:Button label="Submit" click="hello();"/>

  </mx:Panel>
</mx:Application>
```

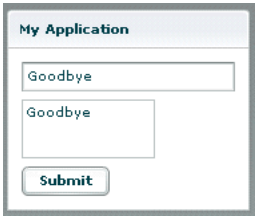
MXML 内での ActionScript の使用については、55 ページ、第 4 章の「ActionScript の使用」を参照してください。

コンポーネント間でのデータのバインディング

Flex で異なるコンポーネントのプロパティを互いにバインディングするためのシンタックスは単純です。次の例のように、中かっこ ({ }) 内に値を指定することで、TextArea コントロールの text プロパティを TextInput コントロールの text プロパティにバインドできます。アプリケーションが初期化されると、両方のコントロールに "Hello" というテキストが表示されます。ユーザーが Button コントロールをクリックすると、両方のコントロールに "Goodbye" というテキストが表示されます。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <mx:TextInput id="textinput1" text="Hello"/>
    <mx:TextArea id="textarea1" text="{textinput1.text}"/>
    <mx:Button label="Submit" click="textinput1.text='Goodbye';"/>

  </mx:Panel>
</mx:Application>
```

次の図は、ユーザーが [Submit] ボタンをクリックした後に Web ブラウザウィンドウに表示されたアプリケーションです。



中かっこ ({ }) を使用したシンタックスの代わりに、<mx:Binding> タグを使用することもできます。この場合は、タグ内にバインド元とバインド先を指定します。データバインディングの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

RPC サービスの使用

リモートプロシージャコール (RPC) サービスを使用すると、アプリケーションがリモートサーバーとやり取りしてデータの提供を受けたり、アプリケーションからサーバーにデータを送信できます。

Flex は、ローカルおよびリモートのサーバーサイドロジックへのアクセスを提供する各種 RPC サービスとやり取りするように設計されています。たとえば、Flex アプリケーションは、SOAP (Simple Object Access Protocol) を使用する Web サービス、Flex と同じアプリケーションサーバー上にある Java オブジェクト (AMF を使用)、または XML を返す HTTP URL に接続できます。AMF は Flash Remoting MX で使用されているプロトコルです。

データアクセスを提供する MXML コンポーネントのことを "RPC コンポーネント " と呼びます。MXML には、次の種類の RPC コンポーネントがあります。

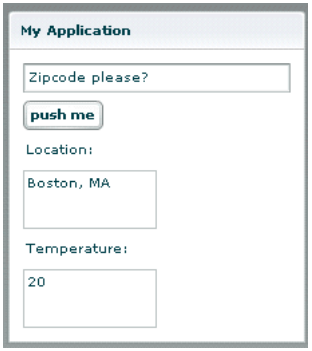
- **WebService** は、SOAP ベースの Web サービスへのアクセスを提供します。
- **HTTPService** は、データを返す HTTP URL へのアクセスを提供します。
- **RemoteObject** は、AMF プロトコルを使用して Java オブジェクトへのアクセスを提供します (Flex データサービスのみ)。

次の図は、気象情報を提供する Web サービスを呼び出して、特定の ZIP コードに対応する地域の現在の気温を表示するアプリケーションです。このアプリケーションは、ユーザーが TextInput コントロールに入力した ZIP コードを Web サービスの入力パラメータにバインドします。さらに、Web サービスの結果に含まれる現在の温度値を TextArea コントロールにバインドします。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <!-- Web サービスへの接続を定義する
    ( 指定されている WSDL URL は架空のもの ) -->
  <mx:WebService id="WeatherService"
    wsdl="http:/example.com/ws/WeatherService?wsdl" useProxy="false">

    <!-- TextInput コントロールに入力された ZIP コードの値を
      GetWeather 処理の ZipCode パラメータにバインドする -->
    <mx:operation name="GetWeather">
      <mx:request>
        <ZipCode>{zip.text}</ZipCode>
      </mx:request>
    </mx:operation>
  </mx:WebService>

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <!-- ZIP コードを TextInput コントロールに入力する -->
    <mx:TextInput id="zip" width="200" text="Zipcode please?"/>

    <!-- Button のクリックにより、Web サービス処理を呼び出す -->
```

```
    <mx:Button width="60" label="Get Weather"
      click="WeatherService.GetWeather.send();"/>

    <!-- 指定された ZIP コードに対応する地域を表示する -->
    <mx:Label text="Location:"/>
    <mx:TextArea text="{WeatherService.GetWeather.lastResult.Location}"/>

    <!-- 指定された ZIP コードに対応する地域の現在の気温を表示する -->
    <mx:Label text="Temperature:"/>
    <mx:TextArea
      text="{WeatherService.GetWeather.lastResult.CurrentTemp}"/>
  </mx:Panel>
</mx:Application>
```

次の図は、Web ブラウザウィンドウに表示されたアプリケーションです。



RPC サービスの使用の詳細については、1279 ページ、第 44 章の「RPC コンポーネントについて」を参照してください。

データモデルへのデータの格納

データモデルを使用すると、アプリケーション固有のデータを格納できます。" データモデル " はデータを格納するためのプロパティを提供する ActionScript オブジェクトで、必要に応じて追加機能を実行するメソッドを含めることもできます。データモデルを使用して、Flex アプリケーションからサーバーへ送信する前のデータを格納したり、サーバーから送られてきたデータをアプリケーションで使用するまで格納したりします。

メソッドを必要としない単純なデータモデルは、<mx:Model> タグ、<mx:XML> タグ、または <mx:XMLList> タグ内に宣言できます。次の例は、連絡先情報を入力するための TextInput コントロールと、<mx:Model> タグで表された、連絡先情報を格納するためのデータモデルを含むアプリケーションです。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <!-- データモデル "contact" に連絡先情報を格納します。
    上記の各 TextInput コントロールの text プロパティが
    データモデルのフィールドに渡される -->
  <mx:Model id="contact">
    <info>
      <homePhone>{homePhoneInput.text}</homePhone>
      <cellPhone>{cellPhoneInput.text}</cellPhone>
      <email>{emailInput.text}</email>
    </info>
  </mx:Model>

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >
    <!-- ユーザーが TextInput コントロールに連絡先情報を入力する -->
    <mx:TextInput id="homePhoneInput"
      text="This isn't a valid phone number."/>
    <mx:TextInput id="cellPhoneInput" text="(999)999-999"/>
    <mx:TextInput id="emailInput" text="me@somewhere.net"/>
  </mx:Panel>

</mx:Application>
```

データ検証

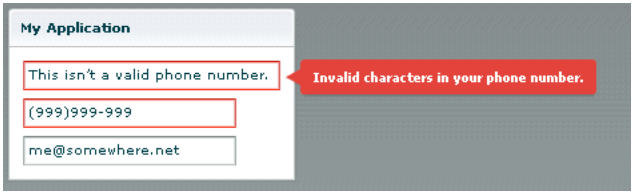
バリデータコンポーネントを使用すると、データモデルまたは Flex ユーザーインターフェイスコンポーネントに格納されているデータを検証できます。Flex には標準バリデータコンポーネントのセットが用意されています。また、自分で作成することもできます。

次の例では、バリデータコンポーネントを使用して、TextInput フィールドに目的のデータ型が入力されたかどうかを検証しています。検証は、ユーザーが TextInput コントロールを編集すると自動的にトリガされます。検証が失敗した場合、ユーザーは即座に視覚的なフィードバックを受け取ります。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <!-- データモデル "contact" に連絡先情報を格納する
    上記の各 TextInput コントロールの text プロパティが
    データモデルのフィールドに渡される -->
  <mx:Model id="contact">
    <info>
      <homePhone>{homePhoneInput.text}</homePhone>
      <cellPhone>{cellPhoneInput.text}</cellPhone>
      <email>{emailInput.text}</email>
    </info>
  </mx:Model>
```

```
  <!-- TextInput コントロールに入力されたデータをバリデータコンポーネントで検証する -->
  <mx:PhoneNumberValidator id="pnV" source="{homePhoneInput}" property="text"/>
  <mx:PhoneNumberValidator id="pnV2" source="{cellPhoneInput}" property="text"/
  >
  <mx:EmailValidator id="emV" source="{emailInput}" property="text" />

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >
    <!-- ユーザーが TextInput コントロールに連絡先情報を入力する -->
    <mx:TextInput id="homePhoneInput"
      text="This isn't a valid phone number."/>
    <mx:TextInput id="cellPhoneInput" text="(999)999-999"/>
    <mx:TextInput id="emailInput" text="me@somewhere.net"/>
  </mx:Panel>

</mx:Application>
```

次の図は、Web ブラウザウィンドウに表示されたアプリケーションです。



データモデルの使用の詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。バリデータの詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

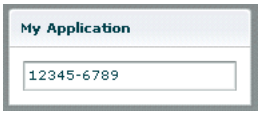
データのフォーマット

フォーマッタコンポーネントは、生のデータを一方向に変換してフォーマットされたストリングを生成する ActionScript コンポーネントです。フォーマッタコンポーネントは、データがテキストフィールドに表示される直前にトリガされます。Flex には標準フォーマッタのセットが用意されています。さらに、自分でフォーマッタを作成することもできます。次の例は、標準の ZipCodeFormatter コンポーネントを使用して変数の値をフォーマットするアプリケーションを示します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <!-- ZipCodeFormatter を宣言し、パラメータを定義する -->
  <mx:ZipCodeFormatter id="ZipCodeDisplay" formatString="#####-####" />

  <mx:Script>
    <![CDATA[
      private var storedZipCode:Number=123456789;
    ]]>
  </mx:Script>
```

```
  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <!-- フォーマッタをトリガし、ストリングにデータを読み込む -->
    <mx:TextInput text="{ZipCodeDisplay.format(storedZipCode)}" />

  </mx:Panel>
</mx:Application>
```

次の図は、Web ブラウザウィンドウに表示されたアプリケーションです。



フォーマッタコンポーネントの詳細については、1207 ページ、第 41 章の「データのフォーマット」を参照してください。

CSS (カスケーディングスタイルシート) の使用

CSS (カスケーディングスタイルシート) 標準に基づくスタイルシートを使用して、Flex コンポーネントにスタイルを宣言できます。MXML の <mx:Style> タグに、インラインスタイル定義か、またはスタイル定義が格納されている外部ファイルへの参照を記述します。

<mx:Style> タグは、MXML ファイルのルートタグの直接の子でなければなりません。スタイルは、クラスセレクタを使用して個々のコンポーネントに適用したり、タイプセレクタを使用して特定のタイプのすべてのコンポーネントに適用することができます。

次の例では、<mx:Style> タグ内でクラスセレクタとタイプセレクタを定義しています。クラスセレクタとタイプセレクタの両方が Button コントロールに適用されます。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
    .myClass { color: Red } /* クラスセレクタ */
    Button { font-size: 18pt} /* タイプセレクタ */
  </mx:Style>

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <mx:Button styleName="myClass" label="This is red 18 point text."/>

  </mx:Panel>
</mx:Application>
```

スタイル定義の中でピリオドに続けて指定された部分は " クラスセレクタ " と呼ばれ、新たに指定するスタイルを定義しています。上の例では myClass がこれに相当します。定義したスタイルは、styleName プロパティを使ってあらゆるコンポーネントに適用できます。上の例では、このスタイルを Button コントロールに適用してフォントの色を赤に設定しています。

" タイプセレクタ " は、特定のコンポーネントタイプのすべてのインスタンスにスタイルを適用します。上の例では、すべての Button コントロールのフォントサイズを 18 ポイントに設定します。

次の図は、Web ブラウザウィンドウに表示されたアプリケーションです。

カスケーディングスタイルシートの使用の詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

スキンの使用

" スキニング " とは、コンポーネントのグラフィックエレメントを修正または置換することで外観を変更する処理です。使用するグラフィックエレメントは、イメージまたは描画 API メソッドの出力で構成され、" シンボル " と呼ばれます。スキンは、Flex コンポーネントの機能に変更を加えることなく交換できます。Flex アプリケーション用の新しいスキンを格納するファイルのことを " テーマ " と呼びます。

Flex のスキンには、グラフィカルスキンとプログラムスキンの 2 種類があります。" グラフィカルスキン " は、Adobe® オーサリング環境から Adobe® Flash® Professional 8 で直接変更できる Adobe Flash シンボルです。" プログラムスキン " は、ActionScript ステートメントを使用して描画するスキンで、クラスファイルの中に定義します。グラフィカルスキンを使用するメリットが大きい場合もあれば、プログラムスキンのほうが合理的といえる場合もあるので、この 2 種類の方法は使い分けることができますが、同じテーマファイルの中でグラフィカルスキンとプログラムスキンを組み合わせることはできません。

スキンの使用の詳細については、741 ページ、第 20 章の「スキンの使用」を参照してください。

エフェクトの使用

" エフェクト " とは、短い時間内にコンポーネントに変化を与えることを指します。エフェクトの例としては、コンポーネントのフェーディング、サイズ変更、移動などがあります。コンポーネント上でマウスがクリックされた、コンポーネントがフォーカスを受け取った、コンポーネントが可視状態になった、などの " トリガ " とエフェクトを組み合わせることで、" ビヘイビア " が発生します。MXML では、コントロールまたはコンテナのプロパティとしてエフェクトを適用します。Flex には、デフォルトのプロパティを持つビルトインエフェクトのセットが用意されています。

次のサンプルアプリケーションに含まれる Button コントロールでは、rollOverEffect プロパティに WipeLeft エフェクトが設定されており、ユーザーがその上にマウスを移動したときに WipeLeft エフェクトが発生します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <!-- エフェクトを定義する -->
  <mx:WipeLeft id="myWL" duration="1000"/>

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <!-- エフェクトをターゲットに割り当てる -->
    <mx:Button id="myButton" rollOverEffect="{myWL}"/>
  </mx:Panel>
</mx:Application>
```

エフェクトの詳細については、603 ページ、第 17 章の「ビヘイビアの使用」を参照してください。

カスタム MXML コンポーネントの定義

カスタム MXML コンポーネントは、他の MXML ファイルの中でカスタム MXML タグとして使用するためにユーザーが作成する MXML ファイルです。MXML コンポーネントは、既存の Flex コンポーネントの機能をカプセル化および拡張します。MXML アプリケーションファイルと同様に、MXML コンポーネントファイルには、MXML タグと ActionScript コードの組み合わせを記述できます。MXML ファイルの名前がクラス名となり、この名前を他の MXML ファイル内で使用してコンポーネントを参照します。

メモ

Web ブラウザからカスタム MXML コンポーネントの URL に直接アクセスすることはできません。

次の例は、リストアイテムが設定されたカスタム ComboBox コントロールです。

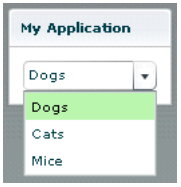
```
<?xml version="1.0"?>
<!-- MyComboBox.mxml -->

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:ComboBox >
      <mx:dataProvider>
        <mx:String>Dogs</mx:String>
        <mx:String>Cats</mx:String>
        <mx:String>Mice</mx:String>
      </mx:dataProvider>
    </mx:ComboBox>

</mx:VBox>
```

次の例は、MyComboBox コンポーネントをカスタムタグとして使用するアプリケーションです。値 * を指定することで、"local" 名前空間が現在のディレクトリに割り当てられます。

```
<?xml version="1.0"?>
<!-- MyApplication.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:MyComps="containers.boxes.*">

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
    paddingLeft="10" paddingRight="10" >

    <MyComps:MyComboBox/>

  </mx:Panel>
</mx:Application>
```

次の図は、Web ブラウザウィンドウに表示されたアプリケーションです。



MXML コンポーネントの詳細については、『Flex 2 コンポーネントの作成と拡張』の第 7 章の「シンプルな MXML コンポーネントの作成」を参照してください。

カスタム Flex コンポーネントを ActionScript で定義することもできます。詳細については、『Flex 2 コンポーネントの作成と拡張』の第 9 章の「ActionScript によるシンプルなビジュアルコンポーネントの作成」を参照してください。

第 3 章

MXML のシンタックス

3

MXML は XML 言語の一種で、Adobe Flex アプリケーションのユーザーインターフェイスコンポーネントをレイアウトするために使用します。このトピックでは、MXML の基本的なシンタックスについて説明します。

目次

MXML の基本的なシンタックス

ほとんどの MXML タグは、ActionScript 3.0 のクラスまたはクラスのプロパティに対応します。Flex は、MXML タグを解析し、対応する ActionScript オブジェクトを含む SWF ファイルをコンパイルします。

ActionScript 3.0 のシンタックスは、ECMAScript Edition 4 言語仕様草案をベースとします。ActionScript 3.0 の特徴を次に示します。

- 公式のクラス定義シンタックス
- 公式のパッケージ構造
- 変数、パラメータ、および戻り値の型指定 (コンパイル時のみ)
- `get` および `set` キーワードを使用する、暗黙的な getter および setter
- 継承
- パブリックメンバーとプライベートメンバー
- 静的メンバー
- キャスト演算子

ActionScript 3.0 の詳細については、91 ページ、第 7 章の「ActionScript の使用」を参照してください。

MXML ファイルの命名

MXML ファイルに名前を付ける場合は、次の命名規則に従う必要があります。

- ファイル名は有効な ActionScript 識別子であることが必要です。つまり、ファイル名は必ず英字または下線文字 (_) で始まり、英数字と下線文字だけで構成される必要があります。
- ファイル名は、ActionScript クラス名、コンポーネント id 値、または "application" という単語と同じにしないでください。また、mx 名前空間に含まれる MXML タグの名前と一致するファイル名を使用しないでください。
- ファイル名は、小文字の拡張子 .mxml で終了する必要があります。

ActionScript クラスを表すタグの使用

ActionScript クラスに対応する MXML タグは、ActionScript クラスと同じ命名規則を使用します。クラス名は大文字で始まり、クラス名内の大文字はそこに含まれる個々の単語の区切りとなります。たとえば、あるタグがある ActionScript クラスに対応する場合、そのプロパティは当該クラスのプロパティおよびイベントに対応します。

コンポーネントプロパティの設定

MXML では、コンポーネントプロパティは対応する ActionScript プロパティと同じ命名規則を使用します。プロパティ名は小文字で始まり、プロパティ名の中の大文字はそこに含まれる個々の単語の区切りとなります。

ほとんどのコンポーネントプロパティは、次のようにタグ属性として設定できます。

```
<mx:Label width="50" height="25" text="Hello World"/>
```

すべてのコンポーネントプロパティは、次のように子タグとして設定できます。

```
<mx:Label>
   <mx:width>50</mx:width>
   <mx:height>25</mx:height>
   <mx:text>Hello World</mx:text>
</mx:Label>
```

子タグは通常、プロパティの値を複雑な Object に設定するときに使用します。複雑な Object はタグ属性の値としては指定できないためです。次の例では、子タグを使用して、ComboBox コントロールのデータプロバイダを ArrayCollection オブジェクトに設定しています。

```
<mx:ComboBox>
   <mx:dataProvider>
      <mx:ArrayCollection>
         <mx:String>AK</mx:String>
         <mx:String>AL</mx:String>
         <mx:String>AR</mx:String>
```

```
    </mx:ArrayCollection>
  <mx:dataProvider>
</mx:ComboBox>
```

子タグを使用してプロパティを設定する場合は 1 つの制約があり、子タグの名前空間の接頭辞 (上の例では mx:) とコンポーネントタグの名前空間の接頭辞が一致している必要があります。

コンポーネントの各プロパティは、次のいずれかのタイプに分類されます。

- スカラープロパティ (たとえば、数値やストリング)
- スカラー値の配列 (たとえば、数値やストリングの配列)
- ActionScript オブジェクト
- ActionScript オブジェクトの配列
- ActionScript プロパティ
- XML データ

スカラー値はタグ属性として割り当て、ActionScript オブジェクトなどの複雑な型は子タグを使用して割り当てることをお勧めします。

スカラープロパティの設定

通常、スカラープロパティの値は、次の例のように、コンポーネントタグのプロパティとして指定します。

```
<mx:Label width="50" height="25" text="Hello World"/>
```

定数を使用したプロパティの設定

多くのコンポーネントプロパティでは、静的定数によって有効値が定義されています。この場合、これらの静的定数の定義は ActionScript クラス内にあります。MXML では、静的定数を使用してプロパティを設定することも、静的定数の値を使用することもできます。次に例を示します。

```
<!-- 静的定数を使用してプロパティを設定する -->
<mx:HBox width="200" horizontalScrollPolicy="{ScrollPolicy.OFF}">
   ...
</mx:HBox>

<!-- 静的定数の値を使用してプロパティを設定する -->
<mx:HBox width="200" horizontalScrollPolicy="off">
   ...
</mx:HBox>
```

HBox コンテナには、コンテナの水平スクロールバーの動作を定義する horizontalScrollPolicy というプロパティがあります。この例では、horizontalScrollPolicy プロパティを明示的に設定して水平スクロールバーを無効にしています。

最初の例では、OFF という静的定数を使用して horizontalScrollPolicy プロパティを設定しています。この静的定数は ScrollPolicy クラスで定義されています。MXML でプロパティの値を静的定数に設定するときは、データバインディングシンタックスを使用する必要があります。静的定数を使用する利点は、プロパティの値が正しくない場合に Flex コンパイラがそれを認識し、コンパイル時にエラーメッセージを生成する点です。

別の方法として、horizontalScrollPolicy プロパティの値を静的定数の値に設定することもできます。静的定数 OFF の値は "off" です。静的定数の値を使用してプロパティの値を設定した場合、Flex コンパイラは、指定された値がサポートされているかどうかを判断できません。そのため、プロパティの設定が誤っていても、実行時エラーが発生するまでわかりません。

ActionScript では、次の例のように、常に静的定数を使用してプロパティの値を設定するようにしてください。

```
var myHBox:HBox = new HBox();
myHBox.horizontalScrollPolicy=ScrollPolicy.OFF;
```

デフォルトプロパティの設定

多くの Flex コンポーネントでは、1つのデフォルトプロパティを定義します。" デフォルトプロパティ " とは、プロパティを明示的に指定しない場合に、MXML タグの内側のコンテンツに対して暗黙的な MXML タグプロパティです。たとえば、次のような MXML タグ定義があるとします。

```
<mx:SomeTag>
  anything here
</mx:SomeTag>
```

このタグで default_property という名前のデフォルトプロパティを定義する場合、前に示したタグ定義は次のコードと同等です。

```
<mx:SomeTag>
  <default_property>
    anything here
  </default_property>
</mx:SomeTag>
```

また次のコードとも同等です。

```
<mx:SomeTag default_property="anything here"/>
```

デフォルトプロパティは、単一のプロパティを設定するための簡単なメカニズムを提供します。ComboBox の場合、デフォルトプロパティは dataProvider プロパティです。したがって、次のコードの 2 つの ComboBox 定義は同等です。

```
<?xml version="1.0"?>
<!-- mxml\DefProp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <!-- Omit the default property. -->
    <mx:ComboBox>
```

```
        <mx:ArrayCollection>
            <mx:String>AK</mx:String>
            <mx:String>AL</mx:String>
            <mx:String>AR</mx:String>
        </mx:ArrayCollection>
    </mx:ComboBox>

    <!-- Explicitly speficy the default property. -->
    <mx:ComboBox>
        <mx:dataProvider>
            <mx:ArrayCollection>
                <mx:String>AK</mx:String>
                <mx:String>AL</mx:String>
                <mx:String>AR</mx:String>
            </mx:ArrayCollection>
        </mx:dataProvider>
    </mx:ComboBox>
</mx:Application>
```

すべての Flex コンポーネントでデフォルトプロパティを定義するとはかぎりません。各コンポーネントのデフォルトプロパティを判別するには、『Adobe Flex 2 リファレンスガイド』を参照してください。

カスタムコンポーネントを作成するときにも、デフォルトプロパティを定義できます。詳細については、『Flex 2 コンポーネントの作成と拡張』の第 5 章の「カスタムコンポーネントでのメタデータタグの使用」を参照してください。

円記号を使用した文字のエスケープ

MXML でプロパティの値を設定する場合は、次に示す例のように、接頭辞として円記号 "\" を付けることにより、予約文字をエスケープすることができます。

```
<?xml version="1.0"?>
<!-- mxml\EscapeChar.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Label text="\{\}"/>

</mx:Application>
```

この例では、テキストストリング内にリテラル中括弧 ({ }) を使用しようとしています。Flex では、中括弧はデータバインディング操作を示すために使用します。そのため、MXML コンパイラでこれらの中括弧がリテラル文字として解釈されるよう、接頭辞として円記号を各中括弧に付ける必要があります。

円記号を使用した String プロパティの設定

MXML コンパイラは、String 型プロパティの値に円記号が含まれている場合、その円記号を自動的にエスケープします。したがって、"\" は常に "\\" に変換されます。

これが必要となる理由は、ActionScript コンパイラが "\\" をリテラル文字 "\" の文字シーケンスと認識し、プロパティ値の初期化時に先頭の円記号を除去するためです。

メモ	アプリケーションアセットへのパスの区切り文字には円記号 (\) を使用せず、常にスラッシュ (/) を使用してください。

String 値に改行文字を含める

String 型のプロパティでは、次の 2 とおりの方法でストリングに改行文字を挿入できます。

- MXML 内で String 値に
 コードを挿入する
- MXML プロパティの初期化に使用する ActionScript の String 変数で "\n" を挿入する

 コードを使用して改行文字を挿入するには、MXML 内のプロパティ値にそのコードを含めます。次に例を示します。

```
<mx:TextArea width="100%" text="Display&#13;Content"/>
```

ActionScript の String 変数を使用して改行文字を挿入するには、ActionScript 変数を作成し、データバインディングを使用して MXML 内でプロパティを設定します。次に例を示します。

```
<mx:Script>
  <![CDATA[
    [Bindable]
    public var myText:String = "Display" + "\n" + "Content";
  ]]>
</mx:Script>

<mx:TextArea width="100%" text="{myText}"/>
```

この例では、TextArea コントロールの text プロパティに改行文字を含む値を設定しています。

この例では、プロパティ定義の前に [Bindable] メタデータタグが付いています。このメタデータタグは、myText プロパティがデータバインディング式のソースとして使用できることを指定します。データバインディングでは、実行時にあるオブジェクトの source プロパティが変更されると、その source プロパティの値が別のオブジェクトの宛先プロパティに自動的にコピーされます。

このメタデータタグを省略すると、コンパイル時に、このプロパティがデータバインディングのソースとして使用できないことを示す警告メッセージが表示されます。詳細については、1133 ページ、第 38 章の「データバインディング」を参照してください。

スカラー値の配列の設定

クラスのプロパティが配列を値として受け取る場合は、そのプロパティを MXML 内で子タグを使用して表すことができます。次の例に示すコンポーネントは、数値の配列を含む dataProvider プロパティを持っています。

```
<mx:List width="150">
  <mx:dataProvider>
    <mx:Array>
      <mx:Number>94062</mx:Number>
      <mx:Number>14850</mx:Number>
      <mx:Number>53402</mx:Number>
    </mx:Array>
  </mx:dataProvider>
</mx:List>
```

配列エレメントを囲む <mx:Array> タグと </mx:Array> タグはオプションです。したがって、この例は次のように記述することもできます。

```
<mx:List width="150">
  <mx:dataProvider>
    <mx:Number>94062</mx:Number>
    <mx:Number>14850</mx:Number>
    <mx:Number>53402</mx:Number>
  </mx:dataProvider>
</mx:List>
```

この例では dataProvider プロパティのデータ型が Array として定義されているため、Flex は 3 つの数値の定義を 3 つのエレメントを持つ配列に自動的に変換します。

コンポーネントを開発するときは、コンポーネント定義の中に配列エレメントのデータ型を定義する情報を追加で指定できます。たとえば、dataProvider プロパティが String 型のエレメントのみをサポートするように指定した場合、この例では配列エレメントに数値が指定されているため、コンパイラエラーが発生します。配列エレメントに必要なデータ型を定義する Array プロパティについては、『Adobe Flex 2 リファレンスガイド』に説明があります。

Object プロパティの設定

コンポーネントのプロパティがオブジェクトを値として受け取る場合は、次の例のように、タグ属性が定義された子タグを使用して、MXML 内のそのプロパティを表すことができます。

```
<mynamespace:MyComponent>
  <mynamespace:nameOfProperty>
    <mynamespace:typeOfObject prop1="val1" prop2="val2"/>
  </mynamespace:nameOfProperty>
</mynamespace:MyComponent>
```

次の例は、Address オブジェクトを定義する ActionScript クラスを示します。このオブジェクトは、その次の例の中で PurchaseOrder コンポーネントのプロパティとして使用されます。

```
class Address
{
  public var name:String;
  public var street:String;
  public var city:String;
  public var state:String;
  public var zip:Number;
}
```

次の例は、Address 型のプロパティを持つ PurchaseOrder コンポーネントを定義する ActionScript クラスです。

```
import example.Address;

class PurchaseOrder {
  public var shippingAddress:Address;
  public var quantity:Number;
  ...
}
```

MXML では、PurchaseOrder コンポーネントを次のように定義します。

```
<mynamespace:PurchaseOrder quantity="3" xmlns:e="example">
  <mynamespace:shippingAddress>
    <mynamespace:Address name="Fred" street="123 Elm St."/>
  </mynamespace:shippingAddress>
</mynamespace:PurchaseOrder>
```

shippingAddress プロパティの値が Address のサブクラス (たとえば DomesticAddress) である場合は、次のようにプロパティ値を宣言できます。

```
<mynamespace:PurchaseOrder quantity="3" xmlns:e="example">
  <mynamespace:shippingAddress>
    <mynamespace:DomesticAddress name="Fred" street="123 Elm St."/>
  </mynamespace:shippingAddress>
</mynamespace:PurchaseOrder>
```

次の例の value プロパティのようにプロパティが Object として明示的に型指定されている場合は、<mx:Object> タグを使用して匿名オブジェクトを指定できます。

```
class ObjectHolder {
  public var value:Object
}
```

次の例は、value プロパティの値として匿名オブジェクトを指定する方法を示しています。

```
<mynamespace:ObjectHolder>
  <mynamespace:value>
    <mx:Object foo='bar' />
  </mynamespace:value>
</mynamespace:ObjectHolder>
```

Object への配列の設定

コンポーネントのプロパティが Object 型で、その Object が配列を値として受け取る場合は、次の例のように、そのプロパティを MXML 内で子タグを使用して表すことができます。

```
<mynamespace:MyComponent>
  <mynamespace:nameOfObjectProperty>
    <mx:Array>
      <mx:Number>94062</mx:Number>
      <mx:Number>14850</mx:Number>
      <mx:Number>53402</mx:Number>
    </mx:Array>
  </mynamespace:nameOfObjectProperty>
</mynamespace:MyComponent>
```

この例では、Object を 3 つのエレメントを持つ数値の配列に初期化しています。

47 ページの「スカラー値の配列の設定」で説明したように、配列エレメントを囲む <mx:Array> タグと </mx:Array> タグはオプションであり、次のように省略できます。

```
<mynamespace:MyComponent>
  <mynamespace:nameOfObjectProperty>
    <mx:Number>94062</mx:Number>
    <mx:Number>14850</mx:Number>
    <mx:Number>53402</mx:Number>
  </mynamespace:nameOfObjectProperty>
</mynamespace:MyComponent>
```

この規則の唯一の例外は、Object プロパティに対して単一の配列エレメントを指定する場合です。この場合、単一エレメントの配列を含む Object は作成されず、代わりにオブジェクトが作成されて、そのオブジェクトに指定した値が設定されます。これらは次のように異なります。

```
object=[element] // 単一エレメントの配列を含む Object
object=element // オブジェクトは値と等しい
```

単一エレメントの配列を作成する場合は、次のように、配列エレメントを <mx:Array> タグと </mx:Array> タグで囲みます。

```
<mynamespace:MyComponent>
  <mynamespace:nameOfObjectProperty>
    <mx:Array>
      <mx:Number>94062</mx:Number>
    </mx:Array>
  </mynamespace:nameOfObjectProperty>
</mynamespace:MyComponent>
```

オブジェクトの配列の設定

コンポーネントのプロパティがオブジェクトの配列を値として受け取る場合は、次の例のように、そのプロパティを MXML 内で子タグを使用して表すことができます。

```
<mynamespace:MyComponent>
  <mynamespace:nameOfProperty>
    <mx:Array>
      <mynamespace:objectType prop1="val1" prop2="val2"/>
      <mynamespace:objectType prop1="val1" prop2="val2"/>
      <mynamespace:objectType prop1="val1" prop2="val2"/>
    </mx:Array>
  </mynamespace:nameOfProperty>
</mynamespace:MyComponent>
```

次の例のコンポーネントには、ListItem オブジェクトの配列が含まれています。それぞれの ListItem オブジェクトには、label および data という名前のプロパティがあります。

```
<mynamespace:MyComponent>
  <mynamespace:dataProvider>
    <mx:Array>
      <mynamespace:ListItem label="One" data="1"/>
      <mynamespace:ListItem label="Two" data="2"/>
    </mx:Array>
  </mynamespace:dataProvider>
</mynamespace:MyComponent>
```

次の例は、dataProvider プロパティの値として匿名オブジェクトを指定する方法を示しています。

```
<mynamespace:MyComponent>
  <mynamespace:dataProvider>
    <mx:Array>
      <mx:Object label="One" data="1"/>
      <mx:Object label="Two" data="2"/>
    </mx:Array>
  </mynamespace:dataProvider>
</mynamespace:MyComponent>
```

47 ページの「スカラー値の配列の設定」で説明したように、配列エレメントを囲む <mx:Array> タグと </mx:Array> タグはオプションであり、次のように省略できます。

```
<mynamespace:MyComponent>
  <mynamespace:dataProvider>
    <mx:Object label="One" data="1"/>
    <mx:Object label="Two" data="2"/>
  </mynamespace:dataProvider>
</mynamespace:MyComponent>
```

XML データを含むプロパティの設定

コンポーネントが XML データを受け取るプロパティを持つ場合、そのプロパティの値は名前空間を適用できる XML フラグメントになります。次の例では、MyComponent オブジェクトの `value` プロパティは XML データです。

```
<mynamespace:MyComponent>
  <mynamespace:value xmlns:a="http://www.example.com/myschema">
    <mx:XML>
      <a:purchaseorder>
        <a:billingaddress>
        ...
        </a:billingaddress>
        ...
    </a:purchaseorder>
  </mx:XML>
  </mynamespace:value>
</mynamespace:MyComponent>
```

MXML でのスタイルプロパティとエフェクトプロパティの設定

MXML タグのスタイルプロパティまたはエフェクトプロパティは、ActionScript クラスのプロパティではなく ActionScript スタイルまたはエフェクトに対応する点で、他のプロパティとは異なります。これらのプロパティを ActionScript で設定する場合は、`object.property=value` の表記法ではなく、`setStyle(`*`stylename`*`, `*`value`*`)` メソッドを使用します。

スタイルプロパティまたはエフェクトプロパティを ActionScript クラスで定義する場合は、ActionScript 変数または setter/getter メソッドとして定義するのではなく、`[Style]` または `[Effect]` メタデータタグを使用します。詳細については、『Flex 2 コンポーネントの作成と拡張』の第 5 章の「カスタムコンポーネントでのメタデータタグの使用」を参照してください。

たとえば、MXML で `fontFamily` スタイルプロパティを設定するには、次のコードを使用します。

```
<mx:TextArea id="myText" text="hello world" fontFamily="Tahoma"/>
```

この MXML コードは、次の ActionScript コードと同じです。

```
myText.setStyle("fontFamily", "Tahoma");
```

MXML でのイベントプロパティの設定

MXML タグのイベントプロパティを使用して、イベントのイベントリスナー関数を指定できます。このプロパティは、ActionScript で addEventListener() メソッドを使用してイベントリスナーを設定するのと同じです。

イベントプロパティを ActionScript クラスで定義する場合は、ActionScript 変数または setter/getter メソッドとして定義するのではなく、[Event] メタデータタグを使用します。詳細については、『Flex 2 コンポーネントの作成と拡張』の第 5 章の「カスタムコンポーネントでのメタデータタグの使用」を参照してください。

たとえば、MXML で creationComplete イベントプロパティを設定するには、次のコードを使用します。

```
<mx:TextArea id="myText" creationComplete="creationCompleteHandler()"/>
```

この MXML コードは、次の ActionScript コードと同じです。

```
myText.addEventListener("creationComplete", creationCompleteHandler);
```

URL 値の指定

<mx:Script> タグなどのいくつかの MXML タグには、外部ファイルの URL を値として受け取るプロパティが用意されています。たとえば、<mx:Script> タグの本体に ActionScript を直接入力する代わりに <mx:Script> タグ内で source プロパティを使用することで、外部 ActionScript ファイルを参照できます。

メモ

<mx:Script> タグの source プロパティにはスクリプトを指定します。source プロパティには ActionScript クラスは指定しないでください。ActionScript クラスの使用の詳細については、『Flex 2 開発ガイド』の 74 ページの「ActionScript コンポーネントの作成」を参照してください。

MXML では、次の種類の URL がサポートされます。

- 絶対 URL。次に例を示します。

  ```
  <mx:Style source="http://www.somesite.com/mystyles.css">
  ```

- Flex アプリケーションが実行されている Java Web アプリケーションのコンテキストルートに対して相対的な、実行時に使用されるパス。次に例を示します。

  ```
  <mx:HTTPService url="@ContextRoot()/directory/myfile.xml"/>
  ```

- Flex アプリケーションが実行されている Java Web アプリケーションのコンテキストルートに対して相対的な、コンパイル時に使用されるパス。次に例を示します。

  ```
  <mx:Script source="/myscript.as"/>
  ```

- 現在のファイルの位置に対する相対 URL。次に例を示します。

  ```
  <mx:Script source="../myscript.as"/>
  ```

RegExp 値の指定

RegExp 型のプロパティの値を MXML で指定するときは、次の形式を使用します。

"/*pattern*/*flags*"

pattern　2 つのスラッシュの間に正規表現を指定します。どちらのスラッシュも省略できません。

flags　(オプション) 正規表現に対するフラグを指定します。

たとえば、MXML コンポーネントの `regExpression` プロパティは RegExp 型です。したがって、次のように値を設定できます。

```
<mynamespace:MyComponent regExpression="/\Wcat/gi"/>
```

あるいは、次のように子タグを使用することも可能です。

```
<mynamespace:MyComponent>
  <mynamespace:regExpression>/\Wcat/gi</mynamespace:regExpression>
</mynamespace:MyComponent>
```

正規表現の *flags* 部分はオプションなので、次のように指定することもできます。

```
<mynamespace:MyComponent regExpression="/\Wcat/"/>
```

コンパイラタグの使用

" コンパイラタグ " は、ActionScript のオブジェクトやプロパティに直接対応しないタグです。次に示すコンパイラタグは、名前の先頭文字のみが大文字で表記されます。

- `<mx:Binding>`
- `<mx:Component>`
- `<mx:Metadata>`
- `<mx:Model>`
- `<mx:Script>`
- `<mx:Style>`
- `<mx:XML>`
- `<mx:XMLList>`

次に示すコンパイラタグは、すべて小文字で表記されます。

- `<mx:operation>`
- `<mx:request>`
- `<mx:method>`
- `<mx:arguments>`

MXML タグの規則

MXML のシンタックスに関する要件を次に示します。

- id プロパティは、どのタグにおいても必須ではありません。
- id プロパティは、ルートタグには使用できません。
- ブール型のプロパティは、true と false の値だけを受け取ります。
- <mx:Binding> タグには source プロパティと destination プロパティが両方とも必要です。
- <mx:Binding> タグに id プロパティを含めることはできません。
- <mx:WebService> タグは wsdl 値と serviceName 値のどちらかを必要としますが、両方とも指定することはできません。
- <mx:RemoteObject> タグは source 値と named 値のどちらかを必要としますが、両方とも指定することはできません。
- <mx:HTTPService> タグは url 値と serviceName 値のどちらかを必要としますが、両方とも指定することはできません。
- <mx:operation> タグは name 値を必要としますが、重複した name エントリを指定することはできません。
- <mx:operation> タグに id プロパティを含めることはできません。
- <mx:method> タグは name 値を必要としますが、重複した name エントリを指定することはできません。
- <mx:method> タグに id プロパティを含めることはできません。

第 4 章

ActionScript の使用

4

Flex 開発者は、ActionScript を使用して Adobe Flex アプリケーションの機能を拡張できます。ActionScript は、MXML には用意されていないフロー制御およびオブジェクト操作機能を提供します。このトピックでは、MXML アプリケーションでの ActionScript の使用方法について説明します。ActionScript の詳細と使用時のリファレンスについては、『ActionScript 3.0 のプログラミング』および『ActionScript 3.0 リファレンスガイド』を参照してください。

目次

Flex アプリケーションでの ActionScript の使用

Flex 開発者は、ActionScript を使用して Flex アプリケーション内にカスタムビヘイビアを実装できます。最初に MXML タグを使用して、アプリケーションに必要なコンテナ、コントロール、エフェクト、フォーマッタ、バリデータ、Web サービスなどを宣言し、そのユーザーインターフェイスをレイアウトします。これらの各コンポーネントは、あらかじめ定められた標準のビヘイビアを提供します。たとえばボタンの上にマウスを置くと、ActionScript を 1 行も記述しなくても、ボタンが自動的にハイライト表示されます。しかし、MXML のような宣言型言語は、ユーザーがボタンをクリックしたときの動作のコーディングには適していません。そのため、実行可能なメソッド、さまざまなタイプの保持変数、条件やループなどのフロー制御を備えた ActionScript のような手続き型言語を使用する必要があります。一般的な意味では、MXML はアプリケーションの静的側面を実装し、ActionScript は動的側面を実装すると言えます。

ActionScript はオブジェクト指向の手続き型プログラミング言語で、ECMAScript (ECMA-262) Edition 4 言語仕様草案をベースとします。ActionScript と MXML は、次のようなさまざまな方法で組み合わせることができます。

- ActionScript を使用して MXML イベント属性内でイベントリスナーを定義する
- <mx:Script> タグを使用してスクリプトブロックを追加する
- 外部 ActionScript ファイルをインクルードする
- ActionScript クラスを読み込む
- ActionScript コンポーネントを作成する

ActionScript のコンパイル

単純な Flex アプリケーションは単一の MXML ファイルまたは ActionScript (AS) ファイルに記述できますが、ほとんどのアプリケーションは複数のファイルに分割されます。たとえば、<mx:Application> の <mx:Script> ブロックと <mx:Style> ブロックを個別の AS および CSS ファイルに移動し、それらのファイルをアプリケーションにインクルードするのが一般的です。また、通常は、カスタムの MXML コンポーネントと ActionScript コンポーネントをアプリケーションに読み込みます。これらのカスタムコンポーネントは他のファイル内で定義する必要があり、MXML コンポーネントではさらに別の AS ファイルをインクルードして、その中に自身の <mx:Script> ブロックを移動できます。ソースコードではなくプリコンパイルされた SWC ファイルからコンポーネントを読み込むこともできます。また、実行可能なコードを含む SWF ファイルをアプリケーションに埋め込むことも可能です。これらすべての入力ファイルは最終的に 1 つの SWF ファイルになります。

ActionScript コードは、MXML ファイル内のいくつかの場所で使用できます。Flex コンパイラは、メイン MXML ファイルとインクルードされた他のファイルを 1 つの ActionScript クラスに変換します。したがって、MXML ファイルおよびインクルードされた ActionScript ファイルの関数の外部で、クラスを定義したりステートメントを使用したりすることはできません。

読み込んだ ActionScript クラスは MXML アプリケーションファイルから参照できます。これらのクラスは最終的な SWF ファイルに追加されます。ActionScript ファイルへの変換が完了すると、Flex はすべての ActionScript コンポーネントをリンクし、これらのクラスを最終的な SWF ファイルにインクルードします。

生成された ActionScript について

MXML ファイルを記述してコンパイルすると、Flex コンパイラによって新しいクラスが作成され、そのクラスで使用する ActionScript が生成されます。次のリストは、生成されたクラスでの MXML タグと ActionScript の使用方法を示しています。Flex を使用する上でこのセクションの内容を理解する必要は必ずしもありませんが、Flex 開発者の見えないところで何が行われているかを知っておくと役に立つことがあります。

MXML アプリケーション (<mx:Application> タグで始まるファイル) は、Application クラスのサブクラスを定義します。同様に、MXML コンポーネント (<mx:Button> など、その他のコンポーネントタグで始まるファイル) は、そのコンポーネントのサブクラスを定義します。

ファイル名がサブクラスの名前になります。基本クラスは最上位タグのクラスです。MXML アプリケーションは実際には次のクラスを定義します。

```
class MyApp extends Application
```

<mx:Button> で始まる "MyButton.mxml" を作成した場合は、実際には次のクラスを定義していることになります。

```
class MyButton extends Button
```

<mx:Script> ブロックの変数と関数の宣言は、サブクラスのプロパティとメソッドを定義します。

クラスの中でコンポーネントインスタンスの id プロパティを設定すると、そのコンポーネントインスタンスへの参照を含むパブリック変数がサブクラスで自動生成されます。たとえば、<mx:Button id="myButton"/> タグが複数のコンテナの中に深くネストされていても、このボタンを myButton として参照することが可能です。

イベント属性は、サブクラスで自動生成されたイベントリスナーメソッドの本体になります。次に例を示します。

```
<mx:Button id="myButton" click="foo = 1; doSomething()">
```

この行は次のようになります。

```
private function __myButton_click(event:MouseEvent):void {
   foo = 1;
   doSomething()
}
```

イベント属性はメソッドの本体になるため、サブクラスの他のプロパティやメソッドにアクセスできます。

MXML ファイル内の ActionScript はすべて、それが <mx:Script> ブロックにあるかタグ内部にあるかにかかわらず、サブクラスのインスタンスを参照する this キーワードで実行されます。

クラスのパブリックプロパティおよびメソッドは、他のコンポーネントの ActionScript コードからアクセスできます。ただしその場合は、myCheckoutAccordion.myAddressForm.firstNameTextInput.text のようなドット表記を使用してその位置を指定するか、parentDocument、parentApplication、または Application.application を使用してプロパティまたはメソッドがどのコンポーネントに存在するかを指定する必要があります。

MXML イベントハンドラでの ActionScript の使用

ActionScript コードを Flex アプリケーションで使用する方法の 1 つとして、MXML タグのイベントハンドラの中に ActionScript コードをインクルードする方法があります。次の例に示します。

```
<?xml version="1.0"?>
<!-- usingas/HelloWorldAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Panel title="My Application" >
     <mx:TextArea id="textarea1"/>
     <mx:Button label="Submit" click="textarea1.text='Hello World';"/>
  </mx:Panel>
</mx:Application>
```

この例では、Button コントロールの click イベントハンドラの本体に ActionScript コードをインクルードします。MXML コンパイラはこの click="..." 属性を受け取り、次のイベントハンドラメソッドを生成します。

```
public function __myButton_click(event:MouseEvent):void {
   textarea1.text='Hello World';
}
```

このコードでは、ユーザーがボタンをクリックすると、TextArea コントロールの text プロパティの値が “Hello World” というストリングに設定されます。ほとんどの場合、生成されたコードを開発者が見る必要はありませんが、イベントハンドラをインラインで記述したときの動作を知っておくと役に立ちます。

生成されたコードを確認するには、keep-generated-actionscript コンパイラオプションの値を true に設定します。こうすると、コンパイル時に *.as ヘルパーファイルが "/generated" ディレクトリに保存されます。このディレクトリは SWF ファイルが存在するディレクトリの下にあります。

イベントの詳細については、81 ページ、第 5 章の「イベントの使用」を参照してください。コマンドラインコンパイラの使用の詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 9 章の「Flex コンパイラの使用」を参照してください。

MXML ファイル内での ActionScript ブロックの使用

<mx:Script> タグは、MXML ファイルに ActionScript ブロックを挿入するために使用します。ActionScript ブロックには、MXML アプリケーションで使用する ActionScript 関数および変数宣言を含めることができます。Code inside <mx:Script> tags can also declare constants (with the const statement) and namespaces (with namespace), include ActionScript files (with include), import declarations (with import), and use namespaces (with use namespace).

<mx:Script> タグは、<mx:Application> またはその他の最上位コンポーネントタグの子である必要があります。

ステートメントおよび式は、関数内にラップされている場合のみ、使用できます。また、<mx:Script> ブロック内では、新しいクラスやインターフェイスを定義することはできません。その代わりに、新しいクラスやインターフェイスを別の AS ファイルに配置して、そのファイルを読み込む必要があります。

ブロック内の ActionScript は、アプリケーションがコンパイルされるときに、それ自体を含むファイルのクラスに追加されます。次の例では、関数内で変数を宣言し、その変数の値を設定します。

```
<?xml version="1.0"?>
<!-- usingas/StatementSyntax.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="doSomething()">
    <mx:Script><![CDATA[
            public var s:Boolean;
            public function doSomething():void {
                // The following statements must be inside a function.
                s = label1.visible;
                label1.text = String(s);
            }
    ]]></mx:Script>

    <mx:Label id="label1"/>

</mx:Application>
```

ほとんどのステートメントは、<mx:Script> ブロック内の関数の内部に記述する必要があります。ただし、次のステートメントは関数の外部に記述できます。

- import
- var
- include
- const
- namespace
- use namespace

<mx:Script> ブロックを使用する場合は、CDATA 構造内のその内容を折り返す必要があります。これにより、コンパイラによってスクリプトブロックの内容が XML として解釈されることなく、ActionScript が正常に生成されるようになります。<mx:Script> の開始タグと終了タグはすべて、次の例のように記述することをお勧めします。

```
<mx:Script>
  <![CDATA[
    ...
  ]]>
</mx:Script>
```

Flex では CDATA 構造内のテキストは解析されないため、山括弧 (< および >) やアンパサンド (&) など、XML で解析対象となる文字を使用できます。たとえば、小なり (<) 記号を含む次のスクリプトは、CDATA 構造内に記述する必要があります。

```
<?xml version="1.0"?>
<!-- usingas/UsingCDATA.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="doSomething()">
    <mx:Script><![CDATA[
        public var m:Number;
        public var n:Number;
        public function doSomething():void {
            n = 42;
            m = 40;
            label1.text = String(n > m);
        }
    ]]></mx:Script>

    <mx:Label id="label1"/>

</mx:Application>
```

ActionScript マニュアルへのアクセス

ActionScript 3.0 プログラミング言語は、Flash Professional や Adobe Flex Builder を始めとするさまざまな開発環境で使用できます。

Flex マニュアルには、ActionScript 言語について解説した『ActionScript 3.0 のプログラミング』があります。ActionScript API リファレンスは、『Adobe Flex 2 リファレンスガイド』の一部に含まれています。

Flex コンポーネントの操作

Flex アプリケーションで ActionScript を使用するときは、ほとんどの場合、アプリケーションでビジュアルコントロールやコンテナを操作することを目的とします。このセクションではそのための手法、たとえば ActionScript 内で Flex コントロールを参照する方法や、コントロールおよびコンテナのインスタンス化時にプロパティを操作する方法などについて説明します。

Flex コンポーネントの参照

ActionScript を使用してコンポーネントを操作するには、通常、MXML タグで目的のコンポーネントの id プロパティを定義します。たとえば次のコードは、Button コントロールの id プロパティを "myButton" というストリングに設定します。

```
<mx:Button id="myButton" label="Click Me"/>
```

ActionScript を使用してコンポーネントにアクセスしない場合は、このプロパティを省略できます。

このコードをコンパイルすると、myButton という名前のパブリック変数が自動生成され、その中に Button インスタンスへの参照が格納されます。この自動生成された変数を ActionScript で使用することにより、コンポーネントのインスタンスにアクセスできます。つまり、任意の ActionScript クラスまたはスクリプトブロックの中で Button コントロールの id インスタンス参照を使用して、Button コントロールのインスタンスを明示的に参照することが可能です。コンポーネントのインスタンスを参照することで、プロパティを変更したり、メソッドを呼び出したりすることができます。

たとえば、次の ActionScript ブロックは、ユーザーがボタンをクリックしたときに Button コントロールの label プロパティの値を変更します。

```
<?xml version="1.0"?>
<!-- usingas/ButtonExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        private function setLabel():void {
            myButton.label = "Clicked";
        }
    ]]></mx:Script>

    <mx:Button id="myButton" label="Click Me" click="setLabel();"/>

</mx:Application>
```

MXML コンポーネント内のすべてのタグの ID は、そのタグがどれだけ深くネストされていても、定義されているコンポーネントのパブリック変数を生成します。したがって、すべての id プロパティがドキュメント内で一意であることが必要となります。これはまた、コンポーネントインスタンスの ID を指定すれば、関数、外部クラスファイル、読み込んだ ActionScript ファイル、インラインスクリプトなど、アプリケーション内のどこからでもそのコンポーネントにアクセスできることを意味します。

Flex コンポーネントに id プロパティが指定されていない場合は、getChildAt() や getChildByName() などのコンポーネントのコンテナのメソッドを使用して参照できます。

それが含まれている現在のドキュメントまたは現在のオブジェクトは、this キーワードを使用して参照できます。

コンポーネントの名前に一致するストリングがある場合は、それを使用してコンポーネントへの参照を取得することもできます。アプリケーション上のオブジェクトにアクセスするには、this キーワードの後に角括弧を続けて、その中にストリングを指定します。これにより、指定したストリングに名前が一致するオブジェクトを参照することができます。

次の例は、複合ストリングを使用してオブジェクトへの参照を取得し、各 Button コントロールのスタイルプロパティを変更します。

```
<?xml version="1.0"?>
<!-- usingas/FlexComponents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  public function changeLabel(s:String):void {
     s = "myButton" + s;
     this[s].setStyle("fontStyle","italic");
     this[s].setStyle("fontSize","18");
  }
  ]]></mx:Script>

  <mx:Button id="myButton1" click="changeLabel('2')" label="Change Other
     Button's Styles"/>
  <mx:Button id="myButton2" click="changeLabel('1')" label="Change Other
     Button's Styles"/>
</mx:Application>
```

この手法は特に、Repeater コントロールを使用する場合や、ActionScript でオブジェクトを作成するときに参照するオブジェクトの名前が実行時までわからない場合に役立ちます。ただし、ActionScript でオブジェクトをインスタンス化する場合、そのオブジェクトをプロパティ配列に追加するには、変数をパブリックとして、関数内ではなくクラスのスコープで宣言する必要があります。

次の例では、ActionScript を使用して 2 つの Label コントロールをアプリケーションスコープで宣言しています。各ラベルは初期化時にインスタンス化されて text プロパティが設定されます。この例ではさらに、ユーザーが Button コントロールをクリックしたときに、渡された変数をストリングに追加することによって Label コントロールへの参照を取得しています。

```
<?xml version="1.0"?>
<!-- usingas/ASLabels.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initLabels()">
  <mx:Script><![CDATA[
  import mx.controls.Label;

  public var label1:Label;
  public var label2:Label;
```

```
  // Objects must be declared in the application scope. Adds the names to
  // the application's properties array.

  public function initLabels():void {
     label1 = new Label();
     label1.text = "Change Me";

     label2 = new Label();
     label2.text = "Change Me";

     addChild(label1);
     addChild(label2);
  }

  public function changeLabel(s:String):void {
     // Create a String that matches the name of the Label control.
     s = "label" + s;
     // Get a reference to the label control using the
     // application's properties array.
     this[s].text = "Changed";
  }
  ]]></mx:Script>

  <mx:Button id="b1" click="changeLabel('2')" label="Change Other Label"/>
  <mx:Button id="b2" click="changeLabel('1')" label="Change Other Label"/>
</mx:Application>
```

コンポーネントのメソッドの呼び出し

コンポーネントインスタンスのパブリックメソッドは、次のドット表記シンタックスを使用して Flex アプリケーション内で呼び出すことができます。

componentInstance.method([*parameters*]);

次の例は、ユーザーがボタンをクリックしたときに adjustThumb() メソッドを呼び出します。このメソッドは HSlider コントロールのパブリックメソッドである setThumbValueAt() を呼び出します。

```
<?xml version="1.0"?>
<!-- usingas/ComponentMethods.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        public function adjustThumb(s:HSlider):void {
            s.setThumbValueAt(0,4);
        }
    ]]></mx:Script>

    <mx:HSlider id="slider1"/>

   <mx:Button id="myButton" label="Set Thumb" click="adjustThumb(slider1);"/>

</mx:Application>
```

子ドキュメント (カスタム MXML コンポーネントなど) からメソッドを呼び出すには、parentApplication、parentDocument、または Application.application プロパティを使用します。詳細については、495 ページ、第 14 章の「Application コンテナの使用」を参照してください。

ActionScript でのビジュアル Flex コンポーネントの作成

ActionScript では、ActionScript クラスのインスタンスを作成する場合と同じように、new 演算子を使用してプログラムからビジュアル Flex コンポーネントを作成できます。作成したコンポーネントはそのプロパティにデフォルト値が設定されますが、まだ親も子も持っていません (すべての種類の内部 DisplayObject を含む)。また、Flash Player の表示リストにも登録されていないので、表示できません。コンポーネントを作成したら、標準の代入ステートメントを使用して、デフォルト値のままでは適切でないプロパティを設定します。

最後に、新しいコンポーネントをコンテナに追加する必要があります。これにはコンテナの addChild() メソッドまたは addChildAt() メソッドを使用します。これで新しいコンポーネントが Flex アプリケーションのビジュアル階層の一部になります。コンポーネントが初めてコンテナに追加されたとき、コンポーネントの子が作成されます。子はコンポーネントのライフサイクルの後の方で作成されます。これにより、子が作成されたときにその子に影響を与えるプロパティを設定できるようになります。

ビジュアルコントロールを作成するときは、適切なパッケージを読み込む必要があります。ほとんどの場合、これは mx.controls パッケージですが、『Adobe Flex 2 リファレンスガイド』で確認してください。

次の例は、HBox 内に Button コントロールを作成します。

```
<?xml version="1.0"?>
<!-- usingas/ASVisualComponent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.controls.Button;
     public var button2:Button;

     public function createObject():void {
        button2 = new Button();
        button2.label = "Click Me";
        hb1.addChild(button2);
     }
  ]]></mx:Script>
  <mx:HBox id="hb1">
     <mx:Button label="Create Object" click="createObject()"/>
  </mx:HBox>
</mx:Application>
```

新しい子はコンテナ内の最後の子として作成されます。新しい子をコンテナ内の最後の子にしたくない場合は、addChildAt() メソッドを使用して順序を変更します。addChild() メソッドの後に setChildIndex() メソッドを呼び出す方法もありますが、これは効率的ではありません。

動的に作成するコンポーネントごとにインスタンス変数を宣言し、新しく作成したコンポーネントへの参照をその中に格納します。これは、コンポーネントインスタンスタグで id プロパティを設定したときに MXML コンパイラが行うことと同じです。こうすると、MXML で宣言的に作成したコンポーネントと同じように、動的に作成したコンポーネントにアクセスできます。

プログラムによってコントロールを削除するには、removeChild() メソッドまたは removeChildAt() メソッドを使用します。removeAllChildren() メソッドを使用して、コンテナからすべての子コントロールを削除することもできます。これらのメソッドを呼び出しても、実際にはオブジェクトは削除されません。子への参照が他になければ、その子は将来のいずれかの時点で Flash Player のガベージコレクションによって回収されます。しかし、いずれかのオブジェクトがその子への参照を保持している場合、その子はメモリから削除されません。

場合によっては、MXML タグでコンポーネントを宣言的に定義します。コンポーネントのコンテナの creationPolicy プロパティを none に設定すると、そのコンテナ内のコントロールの作成を保留できます。タグで宣言したものの、まだインスタンス化されていないコンポーネントを作成するには、createComponentFromDescriptor() メソッドまたは createComponentsFromDescriptors() メソッドを使用します。これらのメソッドを使用すると、宣言ではなくプログラムによってコンポーネントを作成できます。creationPolicy プロパティの使用の詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 6 章の「起動時のパフォーマンスの向上」を参照してください。

addChild() メソッドに渡せるコンポーネントは、UIComponent だけです。つまり、新しく作成したオブジェクトが mx.core.UIComponent のサブクラスでない場合は、そのオブジェクトをコンテナに関連付ける前に UIComponent でラップする必要があります。次の例では、UIComponent のサブクラスではない新しい Sprite オブジェクトを作成し、それを Panel コンテナに追加する前に UIComponent コンテナの子として追加します。

```
<?xml version="1.0"?>
<!-- usingas/AddingChildrenAsUIComponents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import flash.display.Sprite;
        import mx.core.UIComponent;

        private function addChildToPanel():void {

            var circle:Sprite = new Sprite();
            circle.graphics.beginFill(0xFFCC00);
            circle.graphics.drawCircle(0, 0, 20);

            var c:UIComponent = new UIComponent();
            c.addChild(circle);
```

```
            panel1.addChild(c);

        }
    ]]></mx:Script>

    <mx:Panel id="panel1" height="100" width="100"/>

    <mx:Button id="myButton" label="Click Me" click="addChildToPanel();"/>

</mx:Application>
```

スコープについて

ActionScript におけるスコープの設定とは、主に、this キーワードがある特定の時点で何を参照するかを記述することです。アプリケーションのコア MXML ファイルでは、this キーワードを使用して Application オブジェクトにアクセスできます。MXML コンポーネントを定義しているファイルでは、this は、そのコンポーネントの現在のインスタンスへの参照になります。

ActionScript クラスファイルでは、this キーワードはそのクラスのインスタンスを参照します。次の例の this キーワードは、myClass のインスタンスを参照します。this は暗黙的なので実際には必要ありませんが、この例では、その意味を示すために記述しています。

```
class myClass {
  var _x:Number = 3;
  function get x():Number {
    return this._x;
  }
  function set x(y:Number):void {
    if (y > 0) {
      this._x = y;
    } else {
      this._x = 0;
    }
  }
}
```

しかし、カスタム ActionScript および MXML コンポーネント、または外部 ActionScript クラスファイルでは、Flex はそれらのオブジェクトおよびクラスのコンテキスト内で実行され、this キーワードは Application オブジェクトスコープではなく現在のスコープを参照します。

Flex には、ルートアプリケーションへのアクセスに使用できる Application.application プロパティが用意されています。また、parentDocument プロパティを使用して Flex アプリケーションのドキュメントチェーンの 1 つ上のレベルにアクセスできます。さらに、ある Application オブジェクトが SWFLoader コンポーネントを使用して別の Application オブジェクトをロードしている場合には、parentApplication プロパティを使用してアプリケーションチェーンの 1 つ上のレベルにアクセスすることも可能です。

コンポーネントのイベントリスナーで ActionScript を記述する場合、スコープはコンポーネントではなくアプリケーションになります。たとえば次のコードは、Button コントロールがクリックされたときに Button コントロールのラベルを "Clicked" に変更します。

```
<?xml version="1.0"?>
<!-- usingas/ButtonScope.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button id="myButton" label="Click Me" click="myButton.label='Clicked'"/>

</mx:Application>
```

この例を次のコードと比較します。

```
<?xml version="1.0"?>
<!-- usingas/AppScope.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- The following does nothing because the app level scope does
         not have a label to set -->
    <mx:Button id="myButton" label="Click Me" click="label='Clicked'"/>

</mx:Application>
```

イベントリスナーが実行されたとき、this キーワードは Button インスタンスを参照しないため、このコードは動作しません。この場合は Application またはその他の最上位コンポーネントインスタンスを参照します。2 番目の例は、Button の label プロパティではなく、Application オブジェクトの label プロパティを設定しようとします。

関数内で宣言された変数のスコープは、その関数にローカルに設定されます。このような変数は、外側のスコープの変数と同じ名前を共有できますが、外側のスコープの変数には影響を与えません。1 つのメソッドで一時的に変数を使用するだけの場合は、その変数をインスタンス変数ではなく、そのメソッドのローカル変数にします。インスタンス変数はインスタンスが存続している間メモリを占有し続けるため、インスタンスの状態を保存する場合のみ使用してください。外側のスコープの変数は、接頭辞 this. を使用して参照できます。

ActionScript コードのインクルードおよび読み込みの比較

MXML コードの可読性をさらに向上させるため、大きなスクリプトブロックを挿入する代わりに、<mx:Script> タグ内で ActionScript ファイルを参照できます。ActionScript ファイルはインクルードまたは読み込むことができます。

ActionScript では、コードのインクルード操作と読み込み操作には明確な違いがあります。" インクルード " とは、あるファイルから別のファイルにコード行をコピーすることで、あたかも include ステートメントの位置にコード行がペーストされたかのようになります。" 読み込み " とは、クラスファイルまたはパッケージに参照を追加して、外部クラスによって定義されたオブジェクトやプロパティにアクセスできるようにすることです。読み込むファイルは、ソースパス上に存在する必要があります。インクルードするファイルは、import ステートメントを使用しているファイルへの相対パス上に存在するか、または絶対パスを指定する必要があります。

ActionScript コードを Flex アプリケーションに追加するには、include ステートメントまたは <mx:Script source="filename"> タグを使用します。

Flex アプリケーションで使用する ActionScript クラスおよびパッケージの場所を定義するには、<mx:Script> ブロック内で import ステートメントを使用します。

以降のセクションで、ActionScript コードのインクルードと読み込みについて詳しく説明します。

ActionScript ファイルのインクルード

ActionScript コードをインクルードするには、<mx:Script> タグ内で外部 ActionScript ファイルを参照します。コンパイル時に、ファイルのすべての内容が実際にそこに入力されたかのように MXML アプリケーションにコピーされます。インクルードファイル内の ActionScript は、<mx:Script> ブロック内の ActionScript と同様に、関数の外部にある場合には変数宣言のみで構成されている必要があります。インクルードファイルでは、定数および名前空間の宣言、他の ActionScript ファイルのインクルード、宣言の読み込み、および名前空間の使用も行うことができます。インクルードファイル内でクラスを定義することはできません。

インクルードされた ActionScript ファイル内に定義されている変数および関数は、MXML ファイル内の任意のコンポーネントで使用できます。インクルードされた ActionScript ファイルは、読み込まれた ActionScript クラスと同じではありません。Flex はインクルードファイルの変数および関数へのアクセスを提供しますが、新しいクラスを追加しません。これは、MXML ファイルそのものがクラスであるためです。

インクルードする ActionScript ファイルは、MXML ファイルと同じディレクトリに存在する必要はありません。ただし、ActionScript ファイルは論理ディレクトリ構造に体系化することをお勧めします。

Adobe Flex データサービス を使用している場合、ActionScript ファイルの変更はタイムスタンプに基づいて判断されます。前回の要求以降にファイルが変更されている場合、Flex は、クライアントに応答する前にアプリケーションを再生成します。読み込まれる ActionScript ファイルの ActionScript を変更した場合、その変更は、次回アプリケーションが要求されたときに反映されます。

外部 ActionScript ファイルを Flex アプリケーションにインクルードする方法には次の 2 とおりがあります。

- <mx:Script> タグの source 属性。これは、外部 ActionScript クラスファイルをインクルードする場合に推奨される方法です。
- <mx:Script> ブロック内の include ステートメント。

以降のセクションでは、外部 ActionScript ファイルをインクルードするためのこの 2 つの方法について説明します。

source 属性を使用した ActionScript ファイルのインクルード

<mx:Script> タグの source 属性を使用して、外部 ActionScript ファイルを Flex アプリケーションにインクルードします。この方法を使用すると、MXML ファイルが見やすくなり、異なるアプリケーション間でのコードの再利用性が高まります。

スクリプトファイルの名前をアプリケーションファイルと同じにしないでください。同じにすると、コンパイルエラーが発生します。

"IncludedFile.as" ファイルの内容を次に示します。

```
// usingas/includes/IncludedFile.as
public function computeSum(a:Number, b:Number):Number {
  return a + b;
}
```

次の例では、"IncludedFile.as" ファイルの内容を読み込みます。このファイルは "includes" サブディレクトリにあります。.

```
<?xml version="1.0"?>
<!-- usingas/SourceInclude.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script source="includes/IncludedFile.as"/>
  <mx:TextInput id="ta1st" text="3"/>
  <mx:TextInput id="ta2nd" text="3"/>
  <mx:TextArea id="taMain"/>
  <mx:Button id="b1" label="Compute Sum"
     click="taMain.text=String(computeSum(Number(ta1st.text),
     Number(ta2nd.text)));"/>
</mx:Application>
```

<mx:Script> タグの source 属性では、相対パスと絶対パスの両方がサポートされます。詳細については、71 ページの「インクルードされた外部ファイルの参照」を参照してください。

<mx:Script> タグの source 属性を使用して、その <mx:Script> タグの中に、ActionScript コードをラップすることはできません。1 つの MXML ファイル内でファイルをインクルードし、さらに ActionScript を記述するには、2 つの <mx:Script> タグを使用します。

include ディレクティブの使用

include ディレクティブは、指定されたファイルの内容を MXML ファイルにコピーする ActionScript ステートメントです。include ディレクティブのシンタックスは次のとおりです。

```
include "file_name";
```

次の例は、"myfunctions.as" ファイルをインクルードします。

```
<?xml version="1.0"?>
<!-- usingas/AppScope.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script><![CDATA[

        include "includes/myfunctions.as";

    ]]></mx:Script>

    <mx:Button id="myButton" label="Click Me"
click="doSomething();doSomethingElse()"/>

</mx:Application>
```

1つの include ディレクティブで指定できるのは1つのファイルだけですが、include ディレクティブはいくつでも使用できます。include ディレクティブはネストできます。つまり、include ディレクティブを指定したファイルに、include ディレクティブを含む他のファイルをインクルードできます。

include ディレクティブでは、相対パスのみがサポートされます。詳細については、71 ページの「インクルードされた外部ファイルの参照」を参照してください。

include を記述する場所は、複数のステートメントを使用できる場所に限られます。たとえば、次のようなコードは許可されません。

```
if (expr)
  include "foo.as"; // 最初のステートメントは IF によって保護されますが、それ以外は保護されません。
...
```

次のようなコードは許可されます。

```
if (expr) {
  include "foo.as"; // { } 内のすべてのステートメントが IF によって保護されます。
}
```

中括弧 ({ }) 内に複数のステートメントを追加することで、複数のステートメントを使用できます。

大量の ActionScript ファイルをインクルードする場合は include ディレクティブを使用しないことをお勧めします。適切であればコードを別個のクラスファイルに分割し、それらを論理パッケージ構造に格納するようにしてください。

インクルードされた外部ファイルの参照

<mx:Script> タグの source 属性と include ディレクティブは、異なる方法でファイルを参照します。

<mx:Script> タグの source 属性を使用して外部ファイルを参照する場合は、次のパスが有効です。

- Flex データサービス のみ：/scripts/myscript.as などのサイト相対 URL。スラッシュで始まる URL は、アプリケーションのコンテキストルートからの相対 URL として解決されます。デフォルトのアプリケーションルートは "/Flex アプリケーションのルート " です。
- ../myscript.as などの相対 URL。スラッシュで始まらない URL は、それを使用するファイルからの相対 URL として解決されます。<mx:Script source="../IncludedFile.as"> タグが "mysite/myfiles/myapp.mxml" に含まれている場合は、"mysite/IncludedFile.as" が検索されます。

ActionScript include ディレクティブの場合は、相対 URL しか参照できません。

インポートされるクラスおよびパッケージはソースパス上で検索されます。include ディレクティブまたは <mx:Script> タグの source 属性を使用してインクルードされるファイルについては、ソースパスは検索されません。

クラスおよびパッケージの読み込み

多くのユーティリティクラスを作成する場合や、複数の ActionScript ファイルをインクルードしてよく使用する関数を利用できるようにする場合は、これらのファイルをクラスのセットとして専用のパッケージに格納できます。ActionScript クラスおよびパッケージは、import ステートメントを使用して読み込むことができます。この場合、ActionScript 内でクラスにアクセスするときに完全修飾クラス名を明示的に入力する必要はありません。

次の例では、MyPackage.Util パッケージ内の MyClass クラスを読み込みます。

```
<?xml version="1.0"?>
<!-- usingas/AccessingPackagedClasses.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script><![CDATA[
        import MyPackage.Util.MyClass;

        private var mc:MyClass = new MyClass;

    ]]></mx:Script>

    <mx:Button id="myButton" label="Click Me"
click="myButton.label=mc.returnAString()"/>

</mx:Application>
```

ActionScript コードでは、完全修飾パッケージ名 (MyPackage.Util.MyClass) を使用してクラスを参照する代わりに、MyClass として参照します。

また、ワイルドカード文字 (*) を使用してパッケージ全体を読み込むこともできます。たとえば、次のステートメントを使用すると、MyPackage.Util パッケージ全体が読み込まれます。

```
import MyPackage.Util.*;
```

読み込まれるファイルおよびパッケージはソースパス上で検索され、最終的な SWF ファイルで使用されるもののみが組み込まれます。

単に完全修飾クラス名を指定するだけでは十分ではありません。完全修飾クラス名は、異なるパッケージに含まれる同じ名前のクラスを区別する場合のみ使用します。

クラスを読み込んだだけでアプリケーション内で使用していない場合、そのクラスは結果の SWF ファイルのバイトコードには含められません。結果として、ワイルドカードを使用してパッケージ全体を読み込んだ場合でも、必要以上に大きな SWF ファイルは作成されません。

ActionScript を MXML から分離するテクニック

このセクションでは、1 つのサンプルアプリケーションを使用して、ActionScript を MXML から分離する方法について説明します。Temperature アプリケーションは、1 つの入力フィールドから入力を受け取り、関数を使用して入力値を華氏から摂氏に変換します。続いて、変換結果を Label コントロールに表示します。

次の図に Temperature サンプルアプリケーションを示します。

1 つの関数を呼び出すこの単純なアプリケーションにおいて、MXML と ActionScript を分離する方法はいくつかあります。

- 73 ページの「1 つの MXML ドキュメント (イベント属性でのイベント処理ロジック)」
- 73 ページの「1 つの MXML ドキュメント (<mx:Script> ブロックでのイベント処理ロジック)」
- 74 ページの「1 つの MXML ドキュメントと 1 つの ActionScript ファイル (個別のスクリプトファイルでのイベント処理ロジック)」

以降のセクションでは、それぞれの方法について説明します。

1つの MXML ドキュメント (イベント属性でのイベント処理ロジック)

次のコードでは、MXML タグの click イベントに ActionScript イベント処理ロジックを記述しています。

```
<?xml version="1.0"?>
<!-- usingas/ASOneFile.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
     paddingLeft="10" paddingRight="10">
     <mx:HBox>
        <mx:Label text="Temperature in Farenheit:"/>
        <mx:TextInput id="farenheit" width="120"/>
        <mx:Button label="Convert"
           click="celsius.text=String((Number(farenheit.text)-32)/1.8);"/>
        <mx:Label text="Temperature in Celsius:"/>
     <mx:Label id="celsius" width="120" fontSize="24"/>
     </mx:HBox>
  </mx:Panel>
</mx:Application>
```

1つの MXML ドキュメント (<mx:Script> ブロックでのイベント処理ロジック)

この例では、MXML ドキュメントの <mx:Script> ブロック内に関数のロジックを記述し、MXML タグの click イベントから呼び出しています。

```
<?xml version="1.0"?>
<!-- usingas/ASScriptBlock.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     public function calculate():void {
        var n:Number = Number(farenheit.text);
        celsius.text=String((n-32)/1.8);
     }
  ]]></mx:Script>

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
     paddingLeft="10" paddingRight="10">
     <mx:HBox>
        <mx:Label text="Temperature in Farenheit:"/>
        <mx:TextInput id="farenheit" width="120"/>
        <mx:Button label="Convert" click="calculate();" />
        <mx:Label text="Temperature in Celsius:"/>
        <mx:Label id="celsius" width="120" fontSize="24"/>
     </mx:HBox>
  </mx:Panel>
</mx:Application>
```

1つの MXML ドキュメントと1つの ActionScript ファイル (個別のスクリプトファイルでのイベント処理ロジック)

次の例では、MXML イベント属性に関数呼び出しを記述し、関数を別の ActionScript ファイルで定義しています。

```
<?xml version="1.0"?>
<!-- usingas/ASSourceFile.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <!-- Specify the ActionScript file that contains the function. -->
  <mx:Script source="includes/Sample3Script.as"/>

  <mx:Panel title="My Application" paddingTop="10" paddingBottom="10"
     paddingLeft="10" paddingRight="10">
     <mx:HBox>
        <mx:Label text="Temperature in Farenheit:"/>
        <mx:TextInput id="farenheit" width="120"/>
        <mx:Button label="Convert" click="calculate();"/>
        <mx:Label text="Temperature in Celsius:"/>
        <mx:Label id="celsius" width="120" fontSize="24"/>
     </mx:HBox>
  </mx:Panel>
</mx:Application>
```

"Sample3Script.as" ActionScript ファイルの内容は次のとおりです。

```
// usingas/includes/Sample3Script.as
public function calculate():void {
  celsius.text=String((Number(farenheit.text)-32)/1.8);
}
```

ActionScript コンポーネントの作成

ActionScript を使用する再利用可能なコンポーネントを作成し、Flex アプリケーション内でこれらのコンポーネントを MXML タグとして参照できます。ActionScript を使用して作成したコンポーネントでは、グラフィカルエレメントの格納、カスタムビジネスロジックの定義、または既存の Flex コンポーネントの拡張が可能です。コンポーネントは、Flex で使用できる任意のコンポーネントを継承できます。

ActionScript を使用して独自にコンポーネントを定義することにはいくつかのメリットがあります。コンポーネントを使用すると、個別に開発および保守できる複数のモジュールにアプリケーションを分割できます。よく使用するロジックをカスタムコンポーネントに実装することで、複数の Flex アプリケーションで共有できる、再利用可能なコンポーネントのセットを構築できます。

また、Flex クラス階層を拡張することで、Flex コンポーネントのセットに基づくカスタムコンポーネントを作成できます。Flex ビジュアルコントロールのカスタムバージョンだけでなく、データバリデータ、フォーマッタ、エフェクトなどの非ビジュアルコンポーネントのカスタムバージョンも作成可能です。

たとえば、次の例に示すように、Button コントロールから派生したカスタムボタンを myControls パッケージに定義できます。

```
package myControls {
   import mx.controls.Button;
   public class MyButton extends Button {
      public function MyButton() {
         ...
      }
      ...
   }
}
```

この例では、"MyButton.as" ファイルに MyButton コントロールを記述し、このファイルを Flex アプリケーションのルートディレクトリの "myControls" サブディレクトリに格納します。コンポーネントの完全修飾クラス名は、その位置を示します。この例のコンポーネントの完全修飾クラス名は、myControls.MyButton です。

カスタム Button コントロールは、"MyApp.mxml" などの Flex アプリケーションファイルから参照できます。次に例を示します。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   xmlns:cmp="myControls.*">
   <cmp:MyButton label="Jack"/>
</mx:Application>
```

この例では、アプリケーションのディレクトリ構造内のカスタムコンポーネントの位置を定義する cmp 名前空間を定義します。次に、名前空間の接頭辞を使用して、コンポーネントを MXML タグとして参照しています。

通常、カスタム ActionScript コンポーネントは、ソースパスに含まれているディレクトリに配置します。これには、アプリケーションのルートディレクトリ、"Flex アプリケーションのルート /WEB-INF/flex/user_classes" ディレクトリ (Flex データサービスのみ)、"flex-config.xml" ファイルの <source-path> タグに指定した任意のディレクトリが含まれます。

また、MXML を使用してカスタムコンポーネントを作成することもできます。詳細については、『Flex 2 コンポーネントの作成と拡張』を参照してください。

カスタムコンポーネントのタイプ

ActionScript を使用すると、次のタイプのコンポーネントを作成できます。

ユーザーインターフェイスコンポーネント " ユーザーインターフェイスコンポーネント " には、処理ロジックとビジュアルエレメントの両方が含まれます。通常、これらのコンポーネントは、Flex コンポーネント階層を拡張します。UIComponent クラスや、Button、ComboBox、DataGrid などの任意の Flex コンポーネントを拡張できます。ActionScript カスタムコンポーネントは、基本クラスのパブリックメソッド、パブリックプロパティ、プロテクトプロパティのすべてを継承します。

非ビジュアルコンポーネント " 非ビジュアルコンポーネント " は、ビジュアルエレメントを定義しません。非ビジュアルコンポーネントは、UIComponent クラスを拡張しない ActionScript クラスです。非ビジュアルコンポーネントを使用すると、実行時の効率が向上します。

オブジェクトイントロスペクションの実行

" オブジェクトイントロスペクション " は、クラスのエレメント (そのプロパティやメソッドなど) を実行時に調べる手法です。ActionScript でイントロスペクションを実行するには、次の 2 とおりの方法があります。

- for..in ループの使用
- イントロスペクション API の使用

このセクションでは、これらの方法について説明します。

オブジェクトイントロスペクションが便利だと感じるのは、アプリケーションをデバッグするときです。たとえば、Object 型の汎用オブジェクトをパラメータとして受け取るメソッドを記述するとします。この場合は、イントロスペクションを使用して Object のプロパティとメソッドをすべて出力し、アプリケーションから何が渡されたかを正確に把握できます。

for..in ループの使用

for..in ループでは、動的に追加されたプロパティのみが列挙されます。クラスの宣言された変数とメソッドは、for..in ループでは列挙されません。これは、ActionScript API のクラスで for..in ループを使用しても、ほとんどの場合、どのプロパティも表示されないことを意味します。それでも汎用型の Object は動的オブジェクトなので、for..in ループによってプロパティが表示されます。

次の例では、Object 型の汎用オブジェクトを作成し、そのオブジェクトにプロパティを追加してから、ボタンをクリックするときにオブジェクトを反復処理してプロパティを確認します。

```
<?xml version="1.0"?>
<!-- usingas/IntrospectionForIn.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
```

```
    private var obj:Object = new Object();

    private function initApp():void {
       // Create the object.
       obj.a = "Schotten Totten";
       obj.b = "Taj Majal";
       obj.c = "Durche die Wuste";
    }

    public function dumpObj():void {
       for (var p:String in obj) {
          ta1.text += p + ":" + obj[p] + "\n";
       }
    }
  ]]></mx:Script>
  <mx:TextArea id="ta1" width="400" height="500"/>
  <mx:Button label="Dump Object" click="dumpObj()"/>
</mx:Application>
```

また、mx.utils.ObjectUtil.toString() メソッドを使用して、オブジェクトの動的に追加されたプロパティをすべて出力することもできます。次に例を示します。

```
<?xml version="1.0"?>
<!-- usingas/IntrospectionForIn.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.utils.ObjectUtil;

     private var obj:Object = new Object();

     private function initApp():void {
        // Create the object.
        obj.a = "Schotten Totten";
        obj.b = "Taj Majal";
        obj.c = "Durche die Wuste";
     }

     public function dumpObj():void {
        ta1.text = ObjectUtil.toString(obj);
     }
  ]]></mx:Script>
  <mx:TextArea id="ta1" width="400" height="500"/>
  <mx:Button label="Dump Object" click="dumpObj()"/>
</mx:Application>
```

mx.utils.ObjectUtil クラスにはその他にも、compare()、copy()、isSimple() などの便利なメソッドがあります。詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

イントロスペクション API の使用

非動的クラス (sealed クラス) またはクラスインスタンスのパブリックプロパティとメソッドをすべて列挙する場合は、describeType() メソッドを使用し、その結果を E4X API を使用して解析します。describeType() メソッドは flash.utils パッケージに含まれています。このメソッドは、イントロスペクトするターゲットオブジェクトのみをパラメータとして受け取ります。このパラメータには任意の ActionScript 値を渡すことができます。これには、使用可能なすべての ActionScript 型 (オブジェクトインスタンスなど)、uint などのプリミティブ型、クラスオブジェクトなどが含まれます。describeType() メソッドの戻り値は、オブジェクトの型の XML 記述を含む E4X XML オブジェクトです。

describeType() メソッドはパブリックメンバーのみを返します。呼び出し元のスーパークラス、または呼び出し元がインスタンスでないその他のクラスのプライベートメンバーは返しません。describeType(this) を呼び出すと、クラスの非静的メンバーについてのみ情報が返されます。describeType(getDefinitionByName("MyClass")) を呼び出すと、ターゲットの静的メンバーについてのみ情報が返されます。

次の例は、Button コントロールをイントロスペクトして詳細を TextArea コントロールに出力します。

```
<?xml version="1.0"?>
<!-- usingas/IntrospectionAPI.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="getDetails()">
    <mx:Script><![CDATA[
        import flash.utils.*;

        public function getDetails():void {
            // Get the Button control's E4X XML object description.
            var classInfo:XML = describeType(button1);

            // Dump the entire E4X XML object into ta2.
            ta2.text = classInfo.toString();

            // List the class name.
            ta1.text = "Class " + classInfo.@name.toString() + "\n";

            // List the object's variables, their values, and their types.
            for each (var v:XML in classInfo..variable) {
                ta1.text += "Variable " + v.@name + "=" + button1[v.@name] +
                    " (" + v.@type + ")\n";
            }

            // List accessors as properties.
            for each (var a:XML in classInfo..accessor) {
                // Do not get the property value if it is write only.
                if (a.@access == 'writeonly') {
                    ta1.text += "Property " + a.@name + " (" + a.@type +")\n";
                }
```

```
                else {
                    ta1.text += "Property " + a.@name + "=" +
                        button1[a.@name] +  " (" + a.@type +")\n";
                }
            }

            // List the object's methods.
            for each (var m:XML in classInfo..method) {
                ta1.text += "Method " + m.@name + "():" + m.@returnType + "\n";
            }
        }
    ]]></mx:Script>

    <mx:Button label="Submit" id="button1"/>
    <mx:TextArea id="ta1" width="400" height="200"/>
    <mx:TextArea id="ta2" width="400" height="200"/>
</mx:Application>
```

出力結果には Button コントロールのアクセサ、変数、およびメソッドが表示され、次のようになります。

```
Class mx.controls::Button
...
Variable id=button1 (String)
Variable __width=66 (Number)
Variable layoutWidth=66 (Number)
Variable __height=22 (Number)
Variable layoutHeight=22 (Number)
...
Property label=Submit (String)
Property enabled=true (Boolean)
Property numChildren=2 (uint)
Property enabled=true (Boolean)
Property visible=true (Boolean)
Property toolTip=null (String)
...
Method dispatchEvent():Boolean
Method hasEventListener():Boolean
Method layoutContents():void
Method getInheritingStyle():Object
Method getNonInheritingStyle():Object
```

もう1つの便利なメソッドとして、ObjectUtil の `getClassInfo()` メソッドがあります。このメソッドは、ターゲットオブジェクトの名前とプロパティを格納した Object を返します。次の例は、`getClassInfo()` メソッドと `toString()` メソッドを使用して、Button コントロールのプロパティを表示しています。

```
<?xml version="1.0"?>
<!-- usingas/IntrospectionObjectUtil.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
```

```
      import mx.controls.Alert;
      import mx.utils.ObjectUtil;
      private function showProps(b:Button):void {
         var o:Object = ObjectUtil.getClassInfo(b);
         ta1.text = ObjectUtil.toString(o);
      }
   ]]></mx:Script>
   <mx:Button id="b1" label="Show Properties" click="showProps(b1)"/>
   <mx:TextArea id="ta1" width="300" height="500"/>
</mx:Application>
```

E4X の使用の詳細については、『ActionScript 3.0 のプログラミング』を参照してください。

第 5 章

イベントの使用

5

Adobe Flex アプリケーションの最も重要な部分の 1 つとして、イベント処理があります。このトピックでは、イベントフローについて説明し、Flex アプリケーションでコントロールと ActionScript を使用してイベントを処理する方法を示します。

目次

イベントについて

このセクションでは、Flex 2 のイベントモデルの概要を示します。また、Event オブジェクトとそのサブクラス、およびイベント送出モデルについても説明します。Flex でイベントを使用する方法をすぐに知りたい場合は、このセクションをスキップして、86 ページの「イベントの使用」のサンプルコードを参照してください。

イベントを使用すると、Flex アプリケーションで何かが起こったときにその内容を知ることができます。イベントは、マウスやキーボードなどのユーザー入力デバイスや、Web サービス呼び出しからの戻りなどの外部入力によって生成することができます。イベントは、コンポーネントの作成や廃棄、サイズの変更など、コンポーネントの外観やライフサイクルで変更が発生したときにもトリガされます。

アプリケーションでユーザー操作が行われると、イベントが生成されます。また、直接的なユーザー操作が行われていなくてもイベントが発生することがあります。たとえば、サーバーからのデータのロードが終了した場合や、接続されたカメラがアクティブになった場合などです。コードでこれらのイベントを " 処理 " するには、イベントハンドラを追加します。" イベントハンドラ " とは、特定のイベントに応答するために記述する関数やメソッドです。これらは、" イベントリスナー " と呼ばれることもあります。

Flex のイベントモデルは、ドキュメントオブジェクトモデル (DOM) Level 3 Events に基づいています。Flex は DOM 標準に厳密には準拠していませんが、両者の実装は非常に似ています。

コンポーネントは、イベントを生成および送出し、他のイベントを受け取ります。他のオブジェクトのイベントに関する情報を必要とするオブジェクトは、リスナーをそのオブジェクトに登録します。イベントが発生すると、対象のオブジェクトでは登録時に指定された関数を呼び出すことで、登録されたリスナーすべてにイベントを送出します。同一のオブジェクトから複数のイベントを受け取るには、イベントごとにリスナーを登録する必要があります。

コンポーネントにはビルトインイベントがあります。このようなイベントは、MXML アプリケーションの ActionScript ブロック内で処理します。Flex イベントシステムのディスパッチャー / リスナーモデルを使用して、アプリケーションの外部で独自にイベントリスナーを定義したり、特定のイベントを受け取るカスタムリスナーのメソッドを定義することもできます。ターゲットオブジェクトがイベントを送出したときにリスナーが呼び出されるよう、リスナーをターゲットオブジェクトに登録することができます。

Flex コントロールやコンテナなどのビジュアルオブジェクトはすべて、DisplayObject クラスのサブクラスです。これらは、アプリケーションを構成するビジュアルオブジェクトのツリーにあります。ツリーのルートは Stage です。その下に SystemManager オブジェクトがあり、さらにその下には Application オブジェクトが続きます。子コンテナおよびコンポーネントはツリーのリーフノードに当たります。このツリーのことを " 表示リスト " と呼びます。表示リスト内のオブジェクトは、DOM 階層構造のノードに似ています。このトピックでは、" 表示リストオブジェクト " と " ノード " という用語は同じ意味で使用されています。

各コンポーネントのイベントの詳細については、241 ページ、第 9 章の「コントロールの使用」にあるコンポーネントの説明部分、または『Adobe Flex 2 リファレンスガイド』のコントロールの項目を参照してください。

コンポーネントの起動ライフサイクルの詳細 (起動ライフサイクルの主な事象など) については、『Flex 2 コンポーネントの作成と拡張』の第 10 章の「ActionScript による高度なビジュアルコンポーネントの作成」を参照してください。

イベントフローについて

他のコンテナやコントロールによって送出されたイベントを受け取るコンテナまたはコントロールを指定することができます。Adobe Flash Player によって Event オブジェクトが送出されると、その Event オブジェクトは表示リストのルートからターゲットノードまでの間を往復し、各ノードをチェックして登録されたリスナーがあるかどうかを調べます。ターゲットノードは、表示リスト内の、イベントが発生したノードです。たとえば、ユーザーが Child1 という名前の Button コントロールをクリックすると、ターゲットノードとして定義された Child1 を含む Event オブジェクトが送出されます。

概念上、イベントフローは 3 つの部分に分けられます。次のセクションでは、これらの部分について説明します。イベントフローの詳細については、107 ページの「イベントの伝播」を参照してください。

キャプチャ段階について

イベントフローの最初の部分を " キャプチャ段階 " と言います。ここでは、ルートノードからターゲットノードの親までのノードがすべて対象となります。この段階では、Flash Player によって各ノードがルートから順に確認され、イベントを処理するリスナーが登録されているかどうかがチェックされます。リスナーが登録されている場合は、Event オブジェクトの値が適切に設定され、そのリスナーが呼び出されます。このフローはターゲットノードの親に達するまで続き、親に登録されているリスナーが呼び出されると終わります。キャプチャ段階の詳細については、109 ページの「キャプチャ段階」を参照してください。

ターゲット段階について

イベントフローの 2 番目の部分を " ターゲット段階 " と言います。ここでは、ターゲットノードのみが対象となります。Flash Player は Event オブジェクトに適切な値を設定し、ターゲットノードにイベントリスナーが登録されているかどうかを調べて、それらのリスナーを呼び出します。ターゲット段階の詳細については、110 ページの「ターゲット段階」を参照してください。

バブリング段階について

イベントフローの 3 番目の部分を " バブリング段階 " と言います。ここでは、ターゲットノードの親からルートノードまでのすべてのノードが対象となります。Flash Player は、ターゲットノードの親から順に、適切な値を Event オブジェクトに設定して各ノードのイベントリスナーを呼び出します。最後にルートノードのリスナーが呼び出されて、このフローは終わります。バブリング段階の詳細については、110 ページの「バブリング段階」を参照してください。

Event クラスについて

flash.events.Event クラスは、発生したイベントに関する情報を格納するためのプロパティを含む ActionScript クラスです。Event オブジェクトは、JSP (JavaServer Page) の要求および応答オブジェクトがアプリケーションサーバーによって暗黙的に作成されるのと同様に、暗黙的に作成されるオブジェクトです。

Event オブジェクトは、イベントが送出されるたびに作成されます。イベントリスナー内で Event オブジェクトを使用すると、送出されたイベントの詳細や、イベントを送出したコンポーネントの詳細にアクセスできます。イベントリスナーに Event オブジェクトを渡して使用するかどうかは任意です。ただし、イベントリスナー内で Event オブジェクトのプロパティにアクセスする場合は、Event オブジェクトをリスナーに渡す必要があります。

Event オブジェクトは、イベントが送出されるときに 1 つだけ作成されます。バブリング段階およびキャプチャ段階で表示リストを上下に移動するときは、ノードごとに新しい Event オブジェクトが作成されるのではなく、特定の Event オブジェクトの値が変更されます。

イベントのサブクラスについて

flash.events.Event クラスを拡張するクラスが多数用意されています。ほとんどの場合、これらのクラスは次の 2 つのパッケージで定義されます。

- mx.events.*
- flash.events.*

mx.events パッケージでは、DataGridEvent、DragEvent、ColorPickerEvent などの Flex コントロール固有のイベントクラスが定義されています。flash.events パッケージでは、Flex に固有のイベントではなく、Flash Player によって定義されるイベントが定義されています。これらのイベントクラスには、MouseEvent、DataEvent、TextEvent などがあります。これらのイベントはどれも、Flex アプリケーションでよく使用されます。

これらのパッケージに加えて、独自のイベントオブジェクトを定義するパッケージもあります。たとえば、mx.messaging.events.ChannelEvent や mx.logging.LogEvent などです。

Event クラスの子クラスには、その子クラスに固有の他のプロパティとメソッドが含まれています。汎用 Event オブジェクトではなく、特定のイベントタイプを使用して、これらの固有プロパティやメソッドにアクセスする必要がある場合があります。たとえば LogEvent クラスには、Event クラスには含まれていない `getLevelString()` メソッドがあります。

Event サブクラスの使用の詳細については、115 ページの「イベントのサブクラスの使用」を参照してください。

EventDispatcher クラスについて

表示リストの各オブジェクトでは、そのクラスの継承関係を DisplayObject クラスまでさかのぼって確認することができます。さらに DisplayObject クラスは EventDispatcher クラスを継承しています。EventDispatcher クラスは、表示リストの各オブジェクトに重要なイベントモデル機能を提供する基本クラスです。DisplayObject クラスは EventDispatcher クラスを継承しているため、表示リストの任意のオブジェクトから EventDispatcher クラスのメソッドにアクセスすることができます。

表示リストのどの項目でもイベントモデルに含めることができるため、この機能は重要です。表示リストの各オブジェクトは、EventDispatcher クラスから継承された独自の `addEventListener()` メソッドを使用して、特定のイベントを受け取ることができます。ただし、これは、リスナーオブジェクトがそのイベントのイベントフローの一部である場合のみです。

EventDispatcher という名前は、このクラスの主な目的が Event オブジェクトを送信 (送出) することを意味するように見えますが、このクラスのメソッドは、イベントリスナーを登録、確認、および削除するために頻繁に使用されます。

EventDispatcher クラスは IEventDispatcher インターフェイスを実装しています。これにより、開発者は、EventDispatcher やそのサブクラスを継承できないカスタムクラスを作成した場合でも、IEventDispatcher インターフェイスを実装して、そのメソッドにアクセスすることができます。

`addEventListener()` メソッドは、このクラスで最も頻繁に使用されるメソッドです。このメソッドを使用して、イベントリスナーを登録します。`addEventListener()` メソッドの使用の詳細については、93 ページの「addEventListener() メソッドの使用」を参照してください。

熟練したプログラマは、`dispatchEvent()` メソッドを使用して、イベントを手動で送出したり、カスタム Event オブジェクトをイベントフローに送信したりします。詳細については、104 ページの「手動によるイベントの送出」を参照してください。

EventDispatcher クラスの他のメソッドから、イベントリスナーの存在に関する有益な情報を得ることができます。`hasEventListener()` メソッドは、特定の表示リストオブジェクトで特定のイベントタイプに対するイベントリスナーが見つかった場合に `true` を返します。`willTrigger()` メソッドは、特定の表示リストオブジェクトにイベントリスナーが存在するかどうかを確認します。また、イベントフローの全段階で、その表示リストオブジェクトのすべての祖先についてリスナーが存在するかどうかも確認します。リスナーが見つかった場合、このメソッドは `true` を返します。

イベントの使用

Flex でイベントを使用するには、2 つの手順を実行します。最初に、イベントに応答する関数またはクラスメソッド (" イベントリスナー " または " イベントハンドラ " と呼びます) を記述します。この関数は一般に、Event オブジェクトのプロパティやアプリケーションの状態に関する設定にアクセスします。この関数のシグネチャには通常、渡されるイベントタイプを指定するパラメータを含めます。

次の例は、コントロールが受け取る対象のイベントをトリガした際に通知する単純なイベントリスナー関数を示しています。

```
<?xml version="1.0"?>
<!-- events/SimpleEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.controls.Alert;

     private function initApp():void {
        b1.addEventListener(MouseEvent.CLICK, myEventHandler);
     }

     private function myEventHandler(event:Event):void {
        Alert.show("An event occurred");
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

この例からわかるように、addEventListener() メソッドを使用して、その関数またはクラスメソッドを表示リストオブジェクトにも登録します。

ほとんどの Flex コントロールでは MXML タグ内でリスナーを指定できるため、リスナーを簡単に登録できます。たとえば、addEventListener() メソッドを使用して Button コントロールの click イベントのリスナー関数を指定する代わりに、<mx:Button> タグの click 属性で指定することができます。

```
<?xml version="1.0"?>
<!-- events/SimplerEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.controls.Alert;

     private function myEventHandler(event:Event):void {
        Alert.show("An event occurred");
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

これは、前述のコード例の addEventListener() メソッドを使用した場合と同じですが、addEventListener() メソッドを使用することをお勧めします。このメソッドを使用すると、優先度やキャプチャ設定を指定したり、イベント定数を使用したりできるため、イベントをより詳細に制御することが可能です。また、addEventListener() を使用してイベントハンドラを追加した場合は、removeEventListener() を使用して不要になったハンドラを削除できます。イベントハンドラをインラインで追加した場合は、そのハンドラに対して removeEventListener() を呼び出すことはできません。

コントロールがイベントを生成するたびに、そのイベントに関する情報を含む Event オブジェクトが生成されます。この情報には、イベントのタイプや、送出コントロールへの参照などが含まれます。生成された Event オブジェクトを使用するには、イベントハンドラ関数でパラメータとして指定します。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/EventTypeHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.controls.Alert;

     private function myEventHandler(e:Event):void {
        Alert.show("An event of type " + e.type + " occurred.");
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

インラインイベントによってトリガされたイベントハンドラで Event オブジェクトにアクセスする場合は、MXML タグ内に event キーワードを追加して、Event オブジェクトが明示的にハンドラに渡されるようにする必要があります。

```
<mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>
```

ハンドラ関数では Event オブジェクトを使用する必要はありません。次の例では、2 つのイベントハンドラ関数を作成して ComboBox コントロールのイベントに登録します。1 つ目のイベントハンドラ、openEvt() はパラメータを受け取りません。2 つ目の changeEvt() は、Event オブジェクトをパラメータとして受け取り、このオブジェクトを使用して、イベントをトリガした ComboBox コントロールの value と selectedIndex にアクセスします。

```
<?xml version="1.0"?>
<!-- events/MultipleEventHandlers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     private function openEvt():void {
        forChange.text="";
     }
     private function changeEvt(e:Event):void {
```

```
            forChange.text=e.currentTarget.value + " " +
               e.currentTarget.selectedIndex;
        }
    ]]></mx:Script>
    <mx:ComboBox open="openEvt()" change="changeEvt(event)">
        <mx:dataProvider>
            <mx:Array>
                <mx:String>AK</mx:String>
                <mx:String>AL</mx:String>
                <mx:String>AR</mx:String>
            </mx:Array>
        </mx:dataProvider>
    </mx:ComboBox>
    <mx:TextArea id="forChange" width="150"/>
</mx:Application>
```

この例では、Event オブジェクトの target プロパティにアクセスしています。詳細については、89 ページの「target プロパティへのアクセス」を参照してください。

Event オブジェクトの指定

次の例に示すように、リスナー関数のシグネチャで、オブジェクトを Event 型として指定します。

```
function myEventListener(e:Event):void { ... }
```

ただし、送出されたイベントのタイプに固有のプロパティにアクセスする場合は、ToolTipEvent や KeyboardEvent など、より詳しいイベントタイプを指定する必要があります。次に例を示します。

```
import mx.events.ToolTip
function myEventListener(e:ToolTipEvent):void { ... }
```

場合によっては、ActionScript ブロック内でイベントのクラスを読み込まなければならないことがあります。

ほとんどのオブジェクトには特定の関連イベントがあります。また、複数のタイプのイベントを送出できます。

Event 型のイベントを宣言した場合、そのイベントをより詳細なタイプにキャストして、イベント固有のプロパティにアクセスすることができます。詳細については、115 ページの「イベントのサブクラスの使用」を参照してください。

target プロパティへのアクセス

Event オブジェクトには、送出コンポーネント (つまり " ターゲット ") のインスタンスへの参照が含まれています。したがって、イベントリスナーで、そのインスタンスのすべてのプロパティおよびメソッドにアクセスすることができます。次の例では、イベントをトリガした Button コントロールの id にアクセスしています。

```
<?xml version="1.0"?>
<!-- events/AccessingCurrentTarget.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.controls.Alert;
     private function myEventHandler(e:Event):void {
        Alert.show("The button " + e.currentTarget.id + " was clicked");
     }
]]></mx:Script>
  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>
</mx:Application>
```

現在のターゲットにおけるメソッド呼び出しとプロパティへのアクセスは、混乱しがちです。currentTarget プロパティの型は DisplayObject です。ActionScript は厳密に型指定されているため、event.currentTarget.*methodName*( ) を呼び出せるのは、*methodName* が DisplayObject で定義されているメソッドの場合のみです。プロパティの場合も同様です。event.currentTarget.*property* にアクセスできるのは、*property* が DisplayObject で定義されている場合のみです。currentTarget で他のメソッド (setStyle( ) など) を呼び出そうとすると、エラーが返されます。setStyle( ) メソッドは、DisplayObject のサブクラスである UIComponent で定義されています。そのため、setStyle( ) メソッドを呼び出す前に、currentTarget を UIComponent にキャストする必要があります。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/InvokingOnCurrentTarget.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.core.UIComponent;

     private function myEventHandler(e:Event):void {
        UIComponent(e.currentTarget).setStyle("color", "red");
     }
  ]]></mx:Script>
  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>
</mx:Application>
```

また、表示リスト内の現在のノードへの参照が含まれる target のメソッドとプロパティにアクセスすることもできます。詳細については、108 ページの「target プロパティと currentTarget プロパティについて」を参照してください。

イベントハンドラの登録

イベントハンドラを Flex コントロールに登録するには、いくつかの方法があります。

- イベントハンドラをインラインで定義します。この場合、ハンドラ関数の呼び出しは、イベントをトリガしたコントロールにバインドされます。

```
<?xml version="1.0"?>
<!-- events/SimplerEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.controls.Alert;

     private function myEventHandler(event:Event):void {
        Alert.show("An event occurred");
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

  この例では、ユーザーが Button コントロールをクリックするたびに、myClickHandler() 関数が呼び出されます。

  イベントハンドラをインラインで定義する方法については、91 ページの「インラインでのイベントリスナーの定義」を参照してください。

- addEventListener() メソッドを使用します。

```
<?xml version="1.0"?>
<!-- events/SimpleEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.controls.Alert;

     private function initApp():void {
        b1.addEventListener(MouseEvent.CLICK, myEventHandler);
     }

     private function myEventHandler(event:Event):void {
        Alert.show("An event occurred");
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

  前の例と同様に、ユーザーが Button コントロールをクリックするたびに、myClickHandler() ハンドラ関数が呼び出されます。このメソッドを使用すると、イベントハンドラをより柔軟に登録することができます。たとえば、複数のコンポーネントをこのイベントハンドラに登録したり、1 つのコンポーネントに対して複数のハンドラの追加と削除を行うことができます。詳細については、93 ページの「addEventListener() メソッドの使用」を参照してください。

- イベントハンドラクラスを作成し、イベント処理のためにそのクラスを使用するコンポーネントを登録します。このイベント処理方式を使用すると、コードの再利用性が高まり、MXML ファイルの外部でイベント処理を一元化できます。カスタムイベントハンドラクラスの作成の詳細については、97 ページの「イベントハンドラクラスの作成」を参照してください。

以降のセクションでは、これらのイベント処理方法について説明します。

インラインでのイベントリスナーの定義

Flex アプリケーションでイベントハンドラを定義する最も簡単な方法は、コンポーネントの MXML タグでハンドラ関数を示すことです。そのためには、コンポーネントのイベントをタグ属性として追加し、その後に ActionScript ステートメントまたは関数呼び出しを記述します。

インラインでイベントハンドラを追加するには、次のシンタックスを使用します。

```
<mx:tag_name event_name="handler_function"/>
```

たとえば、Button コントロールの `click` イベントを受け取るには、`<mx:Button>` タグの `click` 属性にステートメントを追加します。関数を追加する場合は、関数を ActionScript ブロック内で定義します。次の例では、Button コントロールの `click` イベントのハンドラとして、`submitForm()` 関数を定義しています。

```
<mx:Script><![CDATA[
   function submitForm():void {
      // 何らかの処理をする
   }
]]></mx:Script>
<mx:Button label="Submit" click="submitForm();" />
```

イベントハンドラには、有効であればどのような ActionScript コードを記述してもかまいません。たとえば、グローバル関数を呼び出したり、コンポーネントプロパティを戻り値に設定したりすることができます。次の例では、グローバル関数 `trace()` を呼び出しています。

```
<mx:Button label="Get Ver" click="trace('The button was clicked');"/>
```

インラインイベントハンドラ定義に渡すことができる特殊なパラメータとして、`event` パラメータがあります。`event` キーワードをパラメータとして追加した場合、Event オブジェクトが渡されます。これにより、ハンドラ関数内で Event オブジェクトのすべてのプロパティにアクセスできます。

次の例では、Event オブジェクトを `submitForm()` ハンドラ関数に渡し、そのタイプを MouseEvent として指定しています。

```
<?xml version="1.0"?>
<!-- events/MouseEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     private function myEventHandler(event:MouseEvent):void {
        // Do something with the MouseEvent object;
     }
  ]]></mx:Script>
```

```
  <mx:Button id="b1" label="Click Me" click="myEventHandler(event)"/>

</mx:Application>
```

すべてのインラインイベントリスナーを定義する場合は、event キーワードを追加し、完成したリスナー関数に最も厳密な Event オブジェクトタイプを指定することをお勧めします。たとえば、Event ではなく MouseEvent を指定します。

Event オブジェクトを使用して、ターゲットオブジェクト (イベントを送出したオブジェクト) への参照、イベントのタイプ (click など)、または他の関連プロパティ (リストコントロール内の行番号や値など) にアクセスすることができます。また、Event オブジェクトを使用して、ターゲットコンポーネント、つまりイベントを送出したコンポーネントのメソッドとプロパティにアクセスすることもできます。

ほとんどの場合は、Event オブジェクト全体をイベントリスナーに渡しますが、プロパティを個別に渡すこともできます。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/PropertyHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     private function myEventHandler(s:String):void {
        trace(s);
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me"
click="myEventHandler(event.currentTarget.id)"/>

</mx:Application>
```

インラインでイベントリスナーを登録する方法は、addEventListener() メソッドを使用してイベントリスナーを登録する方法と比べて柔軟性に劣ります。たとえば、Event オブジェクトの useCapture プロパティや priority プロパティを設定できない、追加したリスナーを削除できないなどが欠点として挙げられます。

addEventListener() メソッドの使用

addEventListener() メソッドを使用して、イベントリスナー関数を特定のコントロールまたはオブジェクトに登録できます。次の例では、myClickListener() 関数を Button コントロールの b1 インスタンスに追加しています。ユーザーが b1 をクリックすると、myClickListener() メソッドが呼び出されます。

```
b1.addEventListener(MouseEvent.CLICK, myClickListener);
```

addEventListener() メソッドのシグネチャを次に示します。

```
componentInstance.addEventListener(event_type:String,
  event_listener:Function, use_capture:Boolean, priority:int,
  weakRef:Boolean)
```

event_type パラメータは、このコンポーネントが送出するイベントの種類です。このパラメータには、click や mouseOut などのイベントタイプストリング、または MouseEvent.CLICK や MouseEvent.MOUSE_OUT などのイベントタイプ静的定数を指定できます。このパラメータは必須です。

定数を使用すると、特定のイベントタイプを簡単に参照することができます。定数が表すストリングではなく、定数そのものを使用する必要があります。コードで入力した定数名にスペルミスがある場合、コンパイラによってその誤りが検出されます。ストリングを使用すると、表記に間違いがあった場合に、その間違いをデバッグすることが難しいため、予期しない結果が生じることがあります。

可能な場合は、常に定数を使用します。たとえば、Event オブジェクトが特定のタイプであるかどうかを確認する場合には、次のコードを使用します。

```
if (myEventObject.type == MouseEvent.CLICK) {// ここにコードを記述
```

次のコードは使用しません。

```
if (myEventObject.type == "click") {// ここにコードを記述
```

event_listener パラメータは、イベントを処理する関数です。このパラメータは必須です。

addEventListener() メソッドの *use_capture* パラメータを使用すると、登録したリスナーをイベントフローのどの段階でアクティブにするかを制御できます。このパラメータにより、Event オブジェクトの useCapture プロパティの値が設定されます。useCapture を true に設定すると、リスナーはイベントフローのキャプチャ段階でアクティブになります。useCapture を false に設定すると、リスナーはイベントフローのターゲット段階とバブリング段階ではアクティブになりますが、キャプチャ段階では無効になります。デフォルト値はイベントのタイプによって決定されますが、ほとんどの場合 false です。

イベントフローのすべての段階でイベントを受け取るには、addEventListener() を 2 回、つまり *use_capture* パラメータを true に設定して 1 回、false に設定して 1 回呼び出す必要があります。このパラメータは省略可能です。詳細については、109 ページの「キャプチャ段階」を参照してください。

priority パラメータは、そのイベントリスナーの優先度を設定します。この値が大きいイベントリスナーほど、同じイベントの他のイベントリスナーに比べて先に実行されます。優先度が同じイベントリスナーは、追加された順に実行されます。このパラメータによって、Event オブジェクトの priority プロパティが設定されます。デフォルト値は 0 ですが、負の整数値または正の整数値を指定することができます。優先度が指定されていないイベントリスナーが複数ある場合は、追加順の早いリスナーが先に実行されます。優先度の設定の詳細については、114 ページの「イベントの優先度」を参照してください。

weakRef パラメータを使用すると、リスナーのメモリリソースを制御できます。強参照 (weakRef が false) の場合、そのリスナーはガベージコレクションによって回収されません。弱参照 (weakRef が true) の場合は回収されます。デフォルト値は false です。

追加したリスナー関数が呼び出されると、Event オブジェクトが暗黙的に作成され、リスナー関数に渡されます。そのため、Event オブジェクトをリスナー関数のシグネチャで宣言する必要があります。

addEventListener() メソッドを使用してイベントリスナーを追加する場合は、Event オブジェクトを *listener_function* のパラメータとして宣言する必要があります。次に例を示します。

```
b1.addEventListener(MouseEvent.CLICK, performAction);
```

リスナー関数では、次のように Event オブジェクトをパラメータとして宣言します。

```
public function performAction(e:MouseEvent):void {
    ...
}
```

次の例では、myClickListener() という新しいハンドラ関数を定義し、Button コントロールの click イベントをそのハンドラに登録しています。ユーザーがボタンをクリックすると、myClickHandler() 関数が呼び出されます。

```
<?xml version="1.0"?>
<!-- events/AddEventListenerExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="createListener()">
  <mx:Script><![CDATA[
     import mx.controls.Alert;
     private function createListener():void {
        b1.addEventListener(MouseEvent.CLICK, myClickHandler, false, 0);
     }
     private function myClickHandler(e:MouseEvent):void {
        Alert.show("The button was clicked");
     }
  ]]></mx:Script>
  <mx:Button label="Click Me" id="b1"/>
</mx:Application>
```

MXML タグ内での addEventListener() の使用

コンポーネント定義にインラインで addEventListener() メソッドを記述して、イベントリスナーを追加できます。次の Button コントロール定義では、Button コントロールの initialize プロパティにインラインで addEventListener() メソッド呼び出しを追加しています。

```
<?xml version="1.0"?>
<!-- events/CallingAddEventListenerInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.controls.Alert;

  private function myClickHandler(event:Event):void {
     Alert.show("This is a log message");
  }
  ]]></mx:Script>

  <mx:Button id='b1'
     label="Click Me"
     initialize='b1.addEventListener(MouseEvent.CLICK, myClickHandler, false,
1);'
  />

</mx:Application>
```

これはイベントハンドラをインラインで定義するのと同じです。ただし、click="*handler_function*" ではなく、addEventListener() メソッドを使用してハンドラを定義する場合は、Event オブジェクトの useCapture プロパティと priority プロパティの値を設定できます。また、インラインで追加したハンドラは削除できませんが、addEventListener() メソッドを使用してハンドラを追加した場合は、removeEventListener() メソッドを呼び出してそのハンドラを削除できます。

イベントハンドラの削除

不要になったハンドラは削除することをお勧めします。そうすることでオブジェクトへの参照が削除されるため、それらをメモリから消去できます。不要になったイベントハンドラを削除するには、removeEventListener() メソッドを使用します。addEventListener() を呼び出せるコンポーネントでは、removeEventListener() メソッドも呼び出せます。removeEventListener() メソッドのシンタックスは次のとおりです。

```
componentInstance.removeEventListener(event_type:String,
   listener_function:Function, use_capture:Boolean)
```

たとえば、次のようなコードがあるとします。

```
myButton.removeEventListener(MouseEvent.CLICK, myClickHandler);
```

event_type パラメータと *listener_function* パラメータは必須です。これらは、addEventListener() メソッドで必要なパラメータと同じです。

また、*use_capture* パラメータも、addEventListener() メソッドで使用されるパラメータと同じです。前述のように、addEventListener() を 2 回、つまり *use_capture* を true に設定して 1 回と false に設定して 1 回呼び出すことにより、イベントのすべての段階でイベントを受け取ることができます。両方のイベントリスナーを削除するには、removeEventListener() を 2 回、つまり *use_capture* を true に設定して 1 回と false に設定して 1 回呼び出す必要があります。

削除できるのは、ActionScript ブロック内で addEventListener() メソッドを使用して追加したイベントリスナーのみです。MXML タグ内で定義したイベントリスナーは、タグ属性内で addEventListener() メソッドを呼び出して登録したものであっても、削除できません。

次のサンプルアプリケーションは、削除できるハンドラと削除できないハンドラのタイプを示しています。

```
<?xml version="1.0"?>
<!-- events/RemoveEventListenerExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="createHandler(event)">
  <mx:Script><![CDATA[
  import mx.controls.Alert;
     private function createHandler(e:Event):void {
        b1.addEventListener(MouseEvent.CLICK, myClickHandler);
     }
     private function removeMyHandlers(e:Event):void {
        // Remove listener for b1's click event.
        b1.removeEventListener(MouseEvent.CLICK,myClickHandler);

        // Does NOT remove the listener for b2's click event.
        b2.removeEventListener(MouseEvent.CLICK,myClickHandler);
     }
     private function myClickHandler(e:Event):void {
        Alert.show("This is a log message");
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me"/>
  <mx:Button label="Click Me Too" id="b2" click="myClickHandler(event)"/>
  <mx:Button label="Axe Listeners" id="b3" click="removeMyHandlers(event)"/>
</mx:Application>
```

イベントハンドラクラスの作成

外部クラスファイルを作成して、そのクラスのメソッドをイベントハンドラとして使用することができます。オブジェクト自体をイベントハンドラにすることはできませんが、オブジェクトのメソッドをハンドラにすることは可能です。すべてのイベントハンドラを処理するクラスを1つ定義することにより、アプリケーション間で同じイベント処理ロジックを使用できます。これにより、MXML アプリケーションの可読性が高まり、保守しやすくなります。

イベントを処理するクラスを作成するには、通常、flash.events.Event クラスを読み込みます。また、通常は空のコンストラクタを記述します。次の ActionScript クラスファイルでは、`handleAllEvents()` メソッドでイベントが処理されるたびに、`trace()` 関数が呼び出されます。

```
// events/MyEventHandler.as

package { // Empty package

  import flash.events.Event;

  public class MyEventHandler {
     public function MyEventHandler() {
        // Empty constructor
     }

     public function handleAllEvents(event:Event):void {
        trace("some event happened");
     }
  }
}
```

MXML ファイルでは、MyEventHandler の新しいインスタンスを宣言し、`addEventListener()` メソッドを使用して、その `handleAllEvents()` メソッドを Button コントロールの `click` イベントにハンドラとして登録します。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/CustomHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="createHandler()">
  <mx:Script><![CDATA[
     private var myListener:MyEventHandler = new MyEventHandler();

     private function createHandler():void {
        b1.addEventListener(MouseEvent.CLICK, myListener.handleAllEvents);
  }
  ]]></mx:Script>

  <mx:Button label="Submit" id="b1"/>

</mx:Application>
```

イベントハンドラのメソッドは静的メソッドとして定義することをお勧めします。これにより、MXML アプリケーション内でクラスをインスタンス化する必要がなくなります。次の createHandler() 関数では、MyStaticEventHandler クラスをインスタンス化せずに、handleAllEvents() メソッドをイベントハンドラとして登録しています。

```
<?xml version="1.0"?>
<!-- events/CustomHandlerStatic.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="createHandler()">
  <mx:Script><![CDATA[
     private function createHandler():void {
        b1.addEventListener(MouseEvent.CLICK,
MyStaticEventHandler.handleAllEvents);
  }
  ]]></mx:Script>

  <mx:Button label="Submit" id="b1"/>

</mx:Application>
```

イベントリスナーメソッドを静的メソッドとして追加するには、クラスファイルでメソッドシグネチャに static キーワードを追加するだけです。

```
// events/MyStaticEventHandler.as

package { // Empty package

  import flash.events.Event;

  public class MyStaticEventHandler {
     public function MyStaticEventHandler() {
        // Empty constructor
     }

     public static function handleAllEvents(event:Event):void {
        trace("some event happened");
     }
  }
}
```

イベントリスナークラスはソースパス上のディレクトリに配置します。MXML ファイルと同じディレクトリに ActionScript クラスを配置することもできますが、この方法は推奨されていません。

単一のイベントに対する複数のリスナーの定義

単一のイベントに対して複数のイベントハンドラ関数を定義するには、2 つの方法があります。MXML タグ内でイベントを定義する際には、新しいハンドラ関数をセミコロンで区切ります。次の例では、submitForm( ) 関数と debugMessage( ) 関数を click イベントにハンドラとして追加しています。

```
<?xml version="1.0"?>
<!-- events/MultipleEventHandlersInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     private function submitForm(e:Event):void {
        // Handle event here.
     }
     private function debugMessage(e:Event):void {
        // Handle event here.
     }
  ]]></mx:Script>

  <mx:Button id="b1"
     label="Do Both Actions"
     click='submitForm(event); debugMessage(event);'
  />

</mx:Application>
```

addEventListener( ) メソッドを使用して既にハンドラを追加したイベントに対して addEventListener( ) メソッドをさらに呼び出すことにより、任意の数のハンドラを追加できます。このメソッドを呼び出すたびに、登録するハンドラ関数が指定したオブジェクトに追加されます。次の例では、ハンドラ関数 submitForm( ) および debugMessage( ) を b1 の click イベントに登録しています。

```
<?xml version="1.0"?>
<!-- events/MultipleEventHandlersAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="createHandlers(event)">
  <mx:Script><![CDATA[

     public function createHandlers(e:Event):void {
        b1.addEventListener(MouseEvent.CLICK, submitForm);
        b1.addEventListener(MouseEvent.CLICK, debugMessage);
     }

     private function submitForm(e:Event):void {
        // Handle event here.
     }
     private function debugMessage(e:Event):void {
        // Handle event here.
     }
```

```
  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me"/>

</mx:Application>
```

イベントハンドラをコンポーネントに追加する方法は混在させることが可能です。つまり、インラインで追加する方法と addEventListener() メソッドを使用する方法を選択的に使用できます。次の例では、performAction() メソッドを呼び出す Button コントロールの click イベントハンドラをインラインで追加し、さらに、CheckBox コントロールの状態に応じて logAction() メソッドを呼び出す 2 番目の click ハンドラを追加しています。

```
<?xml version="1.0"?>
<!-- events/ConditionalHandlers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initApp(event)">
  <mx:Script><![CDATA[
     import mx.controls.Alert;

     private function initApp(e:Event):void {
        cb1.addEventListener(MouseEvent.CLICK, handleCheckBoxChange);
        b1.addEventListener(MouseEvent.CLICK, logAction);
     }

     private function handleCheckBoxChange(e:Event):void {
        if (cb1.selected) {
           b1.addEventListener(MouseEvent.CLICK, logAction);
           ta1.text += "added log listener" + "\n";
        } else {
           b1.removeEventListener(MouseEvent.CLICK, logAction);
           ta1.text += "removed log listener" + "\n";
        }
     }

     private function performAction(e:Event):void {
        Alert.show("You performed the action");
     }

     private function logAction(e:Event):void {
        ta1.text += "Action performed: " + e.type + "\n";
     }
  ]]></mx:Script>

  <mx:Button label="Perform Action" id="b1" click="performAction(event)"/>
  <mx:CheckBox id="cb1" label="Log?" selected="true"/>
  <mx:TextArea id="ta1" height="200" width="300"/>

</mx:Application>
```

イベントリスナーが呼び出される順序を設定するには、addEventListener() メソッドの priority パラメータを使用します。MXML を使用してイベントリスナーをインラインで追加した場合、リスナー関数の優先度を設定することはできません。優先度の設定の詳細については、114 ページの「イベントの優先度」を参照してください。

複数のコンポーネントに対する単一のリスナーの登録

同一コンポーネントの複数のイベント、または異なるコンポーネントの複数のイベントに対して、同一リスナー関数を登録することができます。次の例では、submitForm という 1 つのリスナー関数を 2 つのボタンに登録しています。

```
<?xml version="1.0"?>
<!-- events/OneHandlerTwoComponentsInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[

     private function submitForm(e:Event):void {
        // Handle event here.
        trace(e.currentTarget.id);
     }

  ]]></mx:Script>

  <mx:Button id="b1"
     label="Click Me"
     click="submitForm(event)"
  />

  <mx:Button id="b2"
     label="Click Me"
     click="submitForm(event)"
  />

</mx:Application>
```

addEventListener() メソッドを使用して、複数のコンポーネントのイベントを処理する 1 つのリスナーを登録する際には、各インスタンスに対して addEventListener() メソッドを個別に呼び出す必要があります。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/OneHandlerTwoComponentsAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="createHandlers(event)">
  <mx:Script><![CDATA[

     public function createHandlers(e:Event):void {
        b1.addEventListener(MouseEvent.CLICK, submitForm);
        b2.addEventListener(MouseEvent.CLICK, submitForm);
     }
```

```
    private function submitForm(e:Event):void {
       // Handle event here.
       trace(e.currentTarget.id);
    }

  ]]></mx:Script>

  <mx:Button id="b1" label="Click Me"/>

  <mx:Button id="b2" label="Click Me Too"/>

</mx:Application>
```

このとき、イベントのタイプを処理するロジックをイベントリスナーに追加する必要があります。イベントターゲット (イベントを送出したオブジェクト) が Event オブジェクトに自動的に追加されます。イベントをトリガするオブジェクトに関係なく、Event オブジェクトの target プロパティまたは type プロパティに基づき、条件別にイベントの処理を定義できます。これら 2 つのプロパティはすべての Event オブジェクトに追加されます。

次の例では、Button コントロールの click イベントと CheckBox コントロールの click イベントに対して、1つのリスナー関数 (myEventHandler()) を登録します。リスナー関数では、イベントリスナーを呼び出したオブジェクトのタイプを調べるため、case ステートメントを使用して Event オブジェクトのターゲットの className プロパティをチェックしています。

```
<?xml version="1.0"?>
<!-- events/ConditionalTargetHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     public function initApp():void {
        button1.addEventListener(MouseEvent.CLICK, myEventHandler);
        cb1.addEventListener(MouseEvent.MOUSE_DOWN, myEventHandler);
     }

     public function myEventHandler(event:Event):void {
        switch (event.currentTarget.className) {
           case "Button":
            // Process Button click.
            break;
           case "CheckBox":
            // Process CheckBox click.
            break;
        }
     }
  ]]></mx:Script>

  <mx:Button label="Submit" id="button1"/>
  <mx:CheckBox label="All Words" id="cb1"/>
  <mx:TextArea id="ta1" text="Please enter a search term" width="200"/>

</mx:Application>
```

リスナー関数に追加パラメータを渡す方法

リスナーの追加方法によっては、追加パラメータをリスナー関数に渡すことができます。addEventListener() メソッドを使用してリスナーを追加した場合は、リスナー関数に追加パラメータを渡すことはできません。またこのリスナー関数は1つのパラメータ、つまり Event オブジェクト (またはそのサブクラスの1つ) しか宣言できません。

たとえば、clickListener() メソッドには2つのパラメータが必要なため、次のコードはエラーをスローします。

```
<mx:Script>
   public function addListeners():void {
      b1.addEventListener(MouseEvent.CLICK,clickListener);
   }
   public function clickListener(e:MouseEvent, a:String):void { ... }
</mx:Script>
<mx:Button id="b1"/>
```

addEventListener() の2番目のパラメータは関数であるため、addEventListener() 呼び出しでその関数のパラメータを指定することはできません。したがって、追加パラメータをリスナー関数に渡すには、それらをリスナー関数内で定義してから、それらのパラメータを使用して最終メソッドを呼び出す必要があります。MXML タグ内にインラインでイベントリスナーを定義した場合は、リスナー関数のシグネチャで許可されている数のパラメータを追加できます。次の例では、ストリングと Event オブジェクトを runMove() メソッドに渡しています。

```
<?xml version="1.0"?>
<!-- events/MultipleHandlerParametersInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     public function runMove(dir:String, e:Event):void {
        if (dir == "up") {
           moveableButton.y = moveableButton.y - 5;
        } else if (dir == "down") {
           moveableButton.y = moveableButton.y + 5;
        } else if (dir == "left") {
           moveableButton.x = moveableButton.x - 5;
        } else if (dir == "right") {
           moveableButton.x = moveableButton.x + 5;
        }
     }
  ]]></mx:Script>

  <mx:Canvas height="100%" width="100%">
     <mx:Button id="moveableButton"
        label="{moveableButton.x.toString()},{moveableButton.y.toString()}"
        x="200"
        y="200"
        width="80"
     />
```

```
  </mx:Canvas>

  <mx:VBox horizontalAlign="center">
     <mx:Button id="b1"
        label="Up"
        click='runMove("up",event);'
        width="50"
     />
        <mx:HBox horizontalAlign="center">
           <mx:Button id="b2"
            label="Left"
            click='runMove("left",event);'
            width="50"
           />
           <mx:Button id="b3"
            label="Right"
            click='runMove("right",event);'
            width="50"
           />
        </mx:HBox>
     <mx:Button id="b4"
        label="Down"
        click='runMove("down",event);'
        width="50"
     />
  </mx:VBox>

</mx:Application>
```

手動によるイベントの送出

コンポーネントインスタンスの `dispatchEvent()` メソッドを使用すると、イベントを手動で送出できます。このメソッドは、UIComponent を拡張するすべてのコンポーネントに含まれます。このメソッドは、UIComponent の拡張元の EventDispatcher クラスから継承されています。

`dispatchEvent()` メソッドのシンタックスは次のとおりです。

`objectInstance``.dispatchEvent(`*`event`*`:Event):Boolean`

イベントの送出時には、新しい Event オブジェクトを作成する必要があります。Event オブジェクトコンストラクタのシンタックスは次のとおりです。

`Event(`*`event_type`*`:String, `*`bubbles`*`:Boolean, `*`cancelable`*`:Boolean)`

`event_type` パラメータは、Event オブジェクトの `type` プロパティです。*`bubbles`* および *`cancelable`* の各パラメータは省略可能で、これらのデフォルト値はどちらも `false` です。バブリングとキャプチャの詳細については、107 ページの「イベントの伝播」を参照してください。

dispatchEvent() メソッドでは、カスタムイベントだけでなく任意のイベントを送出できます。ユーザーが Button コントロールをクリックしなくても、Button コントロールの click イベントを送出することが可能です。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/DispatchEventExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="createListener(event)">
  <mx:Script><![CDATA[
     import mx.controls.Alert;
     private function createListener(e:Event):void {
        b1.addEventListener(MouseEvent.MOUSE_OVER, myEventHandler);
        b1.addEventListener(MouseEvent.CLICK, myClickHandler);
     }

     private function myEventHandler(e:Event):void {
        var result:Boolean = b1.dispatchEvent(new MouseEvent(MouseEvent.CLICK,
true, false));
     }

     private function myClickHandler(e:Event):void {
        Alert.show("Triggered by the " + e.type + " event");
     }
  ]]></mx:Script>
  <mx:Button id="b1" label="Click Me"/>
</mx:Application>
```

また、MXML タグ内でイベントを手動送出することもできます。次の例では、マウスポインタをボタンの上に移動すると、ボタンの click イベントがトリガされます。

```
<?xml version="1.0"?>
<!-- events/DispatchEventExampleInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="createListener(event)">
  <mx:Script><![CDATA[
     import mx.controls.Alert;
     private function createListener(e:Event):void {
        b1.addEventListener(MouseEvent.CLICK, myClickHandler);
     }

     private function myClickHandler(e:Event):void {
        Alert.show("Triggered by the " + e.type + " event");
     }
  ]]></mx:Script>
  <mx:Button id="b1" label="Click Me" mouseOver="b1.dispatchEvent(new
MouseEvent(MouseEvent.CLICK, true, false))"/>
</mx:Application>
```

Flex アプリケーションでは、新しく送出されたイベントを処理する必要はありません。リスナーがないイベントをトリガした場合、そのイベントは無視されます。

ActionScript で Event オブジェクトのプロパティを設定することは可能ですが、Event オブジェクトは動的でないため、新しいプロパティを追加することはできません。次の例では、click イベントを受け取り、新しい MouseEvent オブジェクトを作成および送出します。また、MouseEvent オブジェクトの shiftKey プロパティの値を true に設定し、Shift キーを押しながらクリックしたときと同じ動作がシミュレートされます。

```
<?xml version="1.0"?>
<!-- events/DispatchCustomizedEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="addListeners()">
  <mx:Script><![CDATA[
     private function customLogEvent(e:MouseEvent):void {
        ta1.text = e.currentTarget.id + ":" +  e.type + ":" + e.shiftKey;

        // Remove current listener to avoid recursion.
        e.currentTarget.removeEventListener("click",customLogEvent);
     }

     private function handleEvent(e:MouseEvent):void {
        // Add new handler for custom event about to be dispatched.
        e.currentTarget.addEventListener("click",customLogEvent);

        // Create new event object.
        var mev:MouseEvent = new MouseEvent("click",false,false);

        // Customize event object.
        mev.shiftKey = true;

        // Dispatch custom event.
        e.currentTarget.dispatchEvent(mev);
     }

     private function addListeners():void {
        b1.addEventListener("click",handleEvent);
        b2.addEventListener("click",handleEvent);
     }

  ]]></mx:Script>

  <mx:VBox id="vb1">
     <mx:Button id="b1" label="B1"/>
     <mx:Button id="b2" label="B2"/>
     <mx:TextArea id="ta1"/>
  </mx:VBox>

</mx:Application>
```

カスタムプロパティを Event オブジェクトに追加する場合には、Event オブジェクトを拡張して、独自のカスタムクラスで新しいプロパティを定義する必要があります。これにより、他のイベントと同様に、dispatchEvent() メソッドを使用してカスタムイベントを手動で送出できるようになります。

UIComponent を拡張しないカスタム ActionScript クラスで独自のイベントを送出する場合は、flash.events.EventDispatcher クラスを拡張して、addEventListener()、removeEventListener()、および dispatchEvent() の各メソッドにアクセスできるようにします。

カスタムクラスの作成の詳細については、『Flex 2 コンポーネントの作成と拡張』を参照してください。

イベントの伝播

イベントがトリガされると、Flex は 3 つの段階でイベントリスナーが存在するかどうかを確認します。これらの段階の順序は次のようになっています。

- 最初にキャプチャ段階
- 次にターゲット段階
- 最後にバブリング段階

各段階において、ノードはイベントに応答する機会を得ます。たとえば、ユーザーが、VBox コンテナ内の Button コントロールをクリックしたとします。キャプチャ段階では、ルートノードである Application オブジェクトと VBox で、イベントを処理するリスナーがあるかどうかが確認されます。次のターゲット段階では、Button のリスナーがトリガされます。バブリング段階では、キャプチャ段階とは逆の VBox が先、Application オブジェクトが後という順序で、イベントを処理する機会が再び与えられます。

ActionScript 3.0 では、ターゲットノードまたはイベントフローの任意のノードで、イベントリスナーを登録することができます。ただし、すべてのイベントが、イベントフローの 3 つの段階すべてで処理されるわけではありません。一部のタイプのイベントはターゲットノードに直接送出されます。この場合、キャプチャ段階とバブリング段階はスキップされます。それが最上位ノードから送出されたものでない限り、どのイベントもキャプチャできます。

他のイベントでは、表示リストにないオブジェクトが対象になることがあります。たとえば、Socket クラスのインスタンスに送出されたイベントなどです。これらのイベントオブジェクトは、キャプチャ段階またはバブリング段階を経ずに、ターゲットノードに直接フローします。また、イベントがイベントモデルにフローする際に、イベントをキャンセルすることもできるので、他の段階に進む予定のイベントを停止することができます。これは cancelable プロパティが true に設定されている場合のみ可能です。

キャプチャおよびバブリングは、Event オブジェクトが表示リストのノード間を移動する際に実行されます。親から子に移動する場合にはキャプチャが、子から親に移動する場合にはバブリングが実行されます。この処理は継承階層とは関係ありません。ターゲット段階に加えて、キャプチャ段階とバブリング段階も経ることができるのは、DisplayObject オブジェクト (コンテナやコントロールなどのビジュアルオブジェクト) のみです。

マウスイベントおよびキーボードイベントは、バブリング段階でのみ処理されます。キャプチャはどのイベントでも可能ですが、明示的に指定しない限り、DisplayObject オブジェクトはキャプチャ段階ではイベントをリッスンしません。つまり、デフォルトではキャプチャは無効になっています。

Validator のようなフェイスレスイベントディスパッチャによってイベントが送出されると、ターゲット段階の処理のみが実行されます。これは、Event オブジェクトにキャプチャおよびバブリング対象のビジュアル表示リストがないためです。

target プロパティと currentTarget プロパティについて

どの Event オブジェクトにも、Event オブジェクトが伝播プロセスのどの段階にあるかを追跡するための target プロパティと currentTarget プロパティがあります。target プロパティは、イベントのディスパッチャを参照します。currentTarget プロパティは、イベントリスナーが存在しているかどうかが現在検査されているノードを参照します。

あるコンポーネントでリスナーを記述し、MouseEvent.CLICK などのマウスイベントを処理する場合、event.target プロパティは必ずしもそのコンポーネントを参照しません。ほとんどの場合は、ラベルを定義するサブコンポーネント (Button コントロールの UITextField など) を参照します。

Flash Player は、マウスポインタの下にあるオブジェクトのうち、一番上にあるものからイベントを送出します。子は親の前面にあるため、Button の UITextField など、内部サブコンポーネントからイベントを送出することもあります。

event.target プロパティは、監視されているオブジェクト (ほとんどのアプリケーションでは Button コントロールの click イベントを監視しています) ではなく、イベントを送出したオブジェクト (この場合は UITextField) に設定されます。

MouseEvent イベントは親チェーンをバブルアップし、任意の祖先で処理できます。イベントがバブルしても、event.target プロパティの値は変わりません (UITextField のまま)。ただし、event.currentTarget プロパティの値は、各レベルで、イベントを処理する祖先に設定されます。最終的には、currentTarget が Button になり、Button コントロールのイベントリスナーがイベントを処理します。このため、event.target プロパティではなく、次のように event.currentTarget プロパティを使用する必要があります。

```
<mx:Button label="OK" click="trace(event.currentTarget.label)"/>
```

この場合、Button イベントの click イベントリスナーでは、event.currentTarget プロパティは常に Button を参照します。それに対して event.target は、ユーザーが Button コントロールのどこをクリックしたかによって、Button またはその UITextField になります。

キャプチャ段階

キャプチャ段階では、表示リスト内のイベントの祖先を検査し、イベントのリスナーとして登録されているかどうかを確認します。検査は表示リスト内のルート祖先から始まり、ターゲットの直接の祖先まで続きます。ほとんどの場合、ルート祖先はステージで、その下が SystemManager、さらにその下が Application オブジェクトです。

たとえば、Panel コンテナを持つアプリケーションがあり、Panel コンテナの中に TitleWindow コンテナ、さらに TitleWindow コンテナの中に Button コントロールが含まれている場合、その構造は次のようになります。

```
Application
  Panel
    TitleWindow
      Button
```

Button コントロールの `click` イベントに対するリスナーが存在し、キャプチャが有効な場合、キャプチャ段階では次の手順が実行されます。

1. Application コンテナをチェックし、click イベントリスナーが存在するかどうかを確認する
2. Panel コンテナをチェックし、click イベントリスナーが存在するかどうかを確認する
3. TitleWindow コンテナをチェックし、click イベントリスナーが存在するかどうかを確認する

キャプチャ段階では、現在リスナーが呼び出されているノードに合わせて Event オブジェクトの `currentTarget` プロパティの値が変更されます。`target` プロパティは、引き続きイベントのディスパッチャを参照します。

デフォルトでは、コンテナはキャプチャ段階で受け取りません。*use_capture* パラメータのデフォルト値は `false` です。キャプチャ段階でリスナーを追加するには、`addEventListener()` メソッドを呼び出す際に、*use_capture* パラメータを `true` に設定します。次に例を示します。

```
myPanel.addEventListener(MouseEvent.MOUSE_DOWN, clickHandler, true);
```

MXML タグを使用してインラインでイベントリスナーを追加すると、このパラメータは `false` に設定されます。この値はオーバーライドできません。

use_capture パラメータを `true` に設定した場合、つまりイベントがキャプチャ段階で伝播する場合でも、そのイベントはバブル可能ですが、キャプチャ段階のリスナーはそれに応答しません。キャプチャ段階とバブリング段階の両方でイベントを処理する場合は、`addEventListener()` を 2 回、つまり *use_capture* を `true` に設定して 1 回と、`false` に設定して 1 回呼び出す必要があります。

キャプチャ段階が使用されることはほとんどありません。またキャプチャ段階を使用すると、計算処理の負荷が大きくなります。キャプチャとは逆に、バブリングは頻繁に実行されます。

ターゲット段階

ターゲット段階では、イベントディスパッチャのリスナーが呼び出されます。表示リスト上の他のノードでは、イベントリスナーの有無が確認されません。ターゲット段階では、Event オブジェクトの currentTarget プロパティと target プロパティの値は同じです。

バブリング段階

バブリング段階では、イベントの祖先を検査し、イベントリスナーが存在するかどうかを確認します。検査はディスパッチャの直接の祖先から始まり、表示リストをルート祖先までさかのぼります。これはキャプチャ段階の逆です。

たとえば、Panel コンテナを持つアプリケーションがあり、Panel コンテナの中に TitleWindow コンテナ、さらに TitleWindow コンテナの中に Button コントロールが含まれている場合、その構造は次のようになります。

```
Application
  Panel
    TitleWindow
      Button
```

Button コントロールの click イベントに対するリスナーが存在し、バブリングが有効な場合、バブリング段階では次の手順が実行されます。

1. TitleWindow コンテナをチェックし、click イベントリスナーが存在するかどうかを確認する
2. Panel コンテナをチェックし、click イベントリスナーが存在するかどうかを確認する
3. Application コンテナをチェックし、click イベントリスナーが存在するかどうかを確認する

イベントがバブルするのは、イベントの bubbles プロパティが true に設定されている場合のみです。バブルするイベントとしては、マウスイベントとキーボードイベントがあります。一般に、Flex によって送出される高レベルのイベントはバブルしません。バブルできるイベントには change、click、doubleClick、keyDown、keyUp、mouseDown、mouseUp などがあります。特定のイベントがバブルするかどうかを確認するには、『Adobe Flex 2 リファレンスガイド』のイベントの項目を参照してください。

バブリング段階では、現在リスナーが呼び出されているノードに合わせて Event オブジェクトの currentTarget プロパティの値が変更されます。target プロパティは、引き続きイベントのディスパッチャを参照します。

イベントリスナーが呼び出されるとき、実際には表示リストの下位のオブジェクトによって Event オブジェクトが送出される場合があります。イベントを送出したオブジェクトそのものが target となります。イベントが現在バブルしているオブジェクトが currentTarget になります。そのため、イベントリスナーで現在のオブジェクトを参照するときは、一般に、target プロパティではなく currentTarget プロパティを使用します。

イベントリスナーは、イベントを送出するオブジェクトにのみ登録できます。たとえば、Button コントロールがその中に含まれていたとしても、click イベントをリッスンするために Form コンテナを登録することはできません。Form コンテナは click イベントを送出しないためです。ただし、Form コンテナは mouseDown イベントを送出するため、Form コンテナタグで mouseDown イベントリスナーを追加することは可能です。こうすると、Button コントロールまたは Form コンテナが mouseDown イベントを受け取るたびにイベントリスナーがトリガされます。

useCapture プロパティを true に設定した場合、つまりイベントがキャプチャ段階で伝播する場合は、バブリングのデフォルト設定に関係なく、そのイベントはバブルしません。キャプチャ段階とバブリング段階の両方でイベントを処理する場合は、addEventListener() を 2 回、つまり useCapture を true に設定して 1 回と、false に設定して 1 回呼び出す必要があります。

イベントは、表示リストの祖先の親チェーンのみをバブルアップします。同じコンテナ内に含まれる 2 つの Button コントロールなどの兄弟は、互いのイベントに影響しません。

イベントの段階の検出

Event オブジェクトの eventPhase プロパティを使用して、イベントがどの段階にあるかを確認できます。このプロパティには、次の定数のいずれかを表す整数が格納されています。

- 1 — キャプチャ段階 (CAPTURING_PHASE)
- 2 — ターゲット段階 (AT_TARGET)
- 3 — バブリング段階 (BUBBLING_PHASE)

次の例では、現在の段階と現在のターゲット ID が表示されます。

```
<?xml version="1.0"?>
<!-- events/DisplayCurrentTargetInfo.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     private function showInfo(e:MouseEvent):void {
        trace(e.eventPhase + ":" + e.currentTarget.id);
     }
  ]]></mx:Script>

     <mx:Button id="b1" label="Click Me" click="showInfo(event)"/>

</mx:Application>
```

伝播の停止

それぞれの段階で表示リスト内の移動を停止するには、Event オブジェクトで次のいずれかのメソッドを呼び出します。

- stopPropagation()
- stopImmediatePropagation()

Event オブジェクトのイベントフロー処理を停止するには、stopPropagation() メソッドまたは stopImmediatePropagation() メソッドを呼び出します。これら 2 つのメソッドはほとんど同じですが、現在のノードの残りのイベントリスナーを実行できるかどうかという点のみが異なります。stopPropagation() メソッドを呼び出すと、Event オブジェクトは次のノードに移動しなくなりますが、現在のノードの他のイベントリスナーはすべて実行されます。

stopImmediatePropagation() メソッドを呼び出しても Event オブジェクトは次のノードに移動しなくなりますが、この場合は現在のノードの他のイベントリスナーは実行されません。

次の例では、Panel コンテナの中に TitleWindow コンテナを作成し、両方のコンテナを mouseDown イベントのリスナーとして登録しています。その結果、TitleWindow コンテナをクリックした場合、stopImmediatePropagation() メソッドの呼び出しを追加しない限り、showAlert() メソッドが 2 回呼び出されます。

```
<?xml version="1.0"?>
<!-- events/StoppingPropagation.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="init(event)">
  <mx:Script><![CDATA[
     import mx.controls.Alert;
     import flash.events.MouseEvent;
     import flash.events.Event;

     public function init(e:Event):void {
        p1.addEventListener(MouseEvent.MOUSE_DOWN,showAlert);
        tw1.addEventListener(MouseEvent.MOUSE_DOWN,showAlert);
        tw1.addEventListener(Event.CLOSE,closeWindow);
     }
     public function showAlert(e:Event):void {
        Alert.show("Alert!\n" + e.currentTarget + "\n" + e.eventPhase);
        e.stopImmediatePropagation();
     }
     public function closeWindow(e:Event):void {
        p1.removeChild(tw1);
     }
  ]]></mx:Script>

  <mx:Panel id="p1" title="Panel 1">
     <mx:TitleWindow id="tw1" width="300" height="300" showCloseButton="true"
title="Title Window 1">
        <mx:Button label="Enter name"/>
        <mx:TextArea id="ta1"/>
     </mx:TitleWindow>
  </mx:Panel>
</mx:Application>
```

例

次の例では、ターゲットがイベントを処理した後、親コンテナの click ハンドラによって、ターゲットコントロールが無効に設定されます。この例は、複数のイベント (すべてのクリック) で 1 つのリスナー (HBox コンテナのクリック) のロジックを再利用できることを示しています。

```
<?xml version="1.0"?>
<!-- events/NestedHandlers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     public function disableControl(event:MouseEvent):void {
        event.currentTarget.enabled = false;
     }
     public function doSomething(event:MouseEvent):void {
        b1.label = "clicked";
        ta1.text += "something wonderful happened";
     }
     public function doSomethingElse(event:MouseEvent):void {
        b2.label = "clicked";
        ta1.text += "something wonderful happened again";
     }
  ]]></mx:Script>
  <mx:HBox height="50" click="disableControl(event)">
     <mx:Button id='b1' label="Click Me" click="doSomething(event)"/>
     <mx:Button id='b2' label="Click Me" click="doSomethingElse(event)"/>
     <mx:TextArea id="ta1"/>
  </mx:HBox>
</mx:Application>
```

複数のリスナー (子コントロールごとに 1 つずつ) を指定する代わりに親コントロールで 1 つのリスナーを指定すると、コードの行数が減り、アプリケーションの効率が向上します。addEventListener() メソッドの呼び出し回数が減ると、アプリケーションの起動時間が短くなり、メモリの消費量も少なくなります。

次の例では、リンクごとにリスナーを登録する代わりに、Panel コンテナに対して 1 つのイベントハンドラを登録しています。Panel コンテナの子はすべてこのイベントハンドラを継承します。Flex はバブルイベントのハンドラを呼び出すため、currentTarget プロパティではなく target プロパティを使用します。このハンドラでは、currentTarget プロパティは Panel コントロールを参照し、target プロパティは必要なラベルを持つ LinkButton コントロールを参照します。

```
<?xml version="1.0"?>
<!-- events/SingleRegisterHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="createLinkHandler()">
  <mx:Script><![CDATA[
  private function linkHandler(event:MouseEvent):void {
     var url:URLRequest = new URLRequest("http://finance.google.com/finance?q=" +
event.target.label);
     navigateToURL(url);
```

```
  }
  private function createLinkHandler():void {
     p1.addEventListener(MouseEvent.CLICK,linkHandler);
  }
  ]]></mx:Script>
  <mx:Panel id="p1" title="Click on a stock ticker symbol">
     <mx:LinkButton label="ADBE"/>
     <mx:LinkButton label="GE"/>
     <mx:LinkButton label="IBM"/>
     <mx:LinkButton label="INTC"/>
  </mx:Panel>
</mx:Application>
```

イベントの優先度

イベントリスナーは、1つのイベントに対していくつでも登録できます。登録は addEventListener() メソッドの呼び出し順に行われ、イベント発生時には登録順にリスナー関数が呼び出されます。ただし、一部のイベントリスナーをインラインで登録し、その他のイベントリスナーを addEventListener() メソッドで登録した場合、1つのイベントに対してリスナーが呼び出される順序は予測できません。

イベントリスナーが呼び出される順序を変更するには、addEventListener() メソッドの *priority* パラメータを使用します。これは、addEventListener() メソッドの 4 番目のパラメータです。

イベントリスナーは優先度の高い順に呼び出されます。つまり、最も優先度の高いイベントが最初に呼び出されます。次の例では、saveInputData() 関数の前に verifyInputData() メソッドが呼び出されます。これは、verifyInputData() メソッドの優先度が最も高いためです。そして、priority パラメータの値が最も小さい returnResult() が最後に呼び出されます。

```
<?xml version="1.0"?>
<!-- events/ShowEventPriorities.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[

  private function returnResult(e:Event):void {
     trace("returnResult");
  }

  private function verifyInputData(e:Event):void {
     trace("verifyInputData");
  }

  private function saveInputData(e:Event):void {
     trace("saveInputData");
  }
```

```
    private function initApp():void {
        b1.addEventListener(MouseEvent.CLICK, returnResult, false, 1);
        b1.addEventListener(MouseEvent.CLICK, saveInputData, false, 2);
        b1.addEventListener(MouseEvent.CLICK, verifyInputData, false, 3);
    }
    ]]></mx:Script>

    <mx:Button id="b1" label="Click Me"/>

</mx:Application>
```

イベントの優先度には有効な整数 (負または正) を指定します。デフォルト値は 0 です。複数のリスナーに対して同じ優先度が設定されている場合は、登録された順に呼び出されます。

イベントリスナーを定義した後でイベントリスナーの優先度を変更する場合は、`removeEventListener()` メソッドを呼び出してリスナーを削除する必要があります。その後、同じイベントを新しい優先度で追加します。

`addEventListener()` メソッドの *`priority`* パラメータは、DOM Level 3 イベントモデルの正式なパラメータではありません。ActionScript 3.0 では、プログラマがイベントリスナーをより柔軟に整理できるようにするため、このパラメータが提供されています。

あるリスナーに高い優先度を与えた場合でも、そのリスナーの実行が完了してから次のリスナーが呼び出されることを保証する手段はありません。前のリスナーの実行が完了しなければ次のリスナーが呼び出せないような、リスナー間の依存関係を作らないでください。Flash Player では、必ずしも前のイベントリスナーの処理が完了してから次のリスナーの処理が始まるとは限らないことを理解しておくことが必要です。

リスナーが互いに依存関係にある場合、リスナー関数を別のリスナー関数から呼び出すか、前のイベントリスナーから新しいイベントを送出できます。イベントを手動で送出する方法については、104 ページの「手動によるイベントの送出」を参照してください。

イベントのサブクラスの使用

イベントのタイプによっては、Event オブジェクトにさまざまなプロパティを含めることができます。これらのプロパティは、W3C の仕様 (www.w3.org/TR/DOM-Level-3-Events/events.html) に規定されているプロパティに基づいていますが、仕様に規定されているすべてのプロパティが Flex で実装されているわけではありません。

リスナー関数で Event オブジェクトを宣言する際には、そのオブジェクトを Event 型として宣言するか、または Event オブジェクトのサブクラスを指定します。次の例では、Event オブジェクトのタイプを MouseEvent に指定しています。

```
public function performAction(e:MouseEvent):void {
    ...
}
```

ほとんどのコントロールでは、特定のイベントタイプのオブジェクトが生成されます。たとえば、マウスクリックでは、MouseEvent タイプのオブジェクトが生成されます。より詳細なイベントタイプを指定することにより、Event オブジェクトを他のオブジェクトにキャストしなくても、特定のプロパティにアクセスすることができます。また、Event オブジェクトの一部のサブクラスには、固有のメソッドを持つものがあります。たとえば LogEvent には、ログレベルをストリングとして返す `getLevelString()` メソッドがあります。汎用的な Event オブジェクトには、このメソッドは含まれていません。

実行時に定義するイベントオブジェクトには、コンパイル時の型のサブクラスを指定できます。特定のイベントタイプを宣言していない場合でも、Event オブジェクトを特定のタイプにキャストすれば、イベントリスナー内のイベント固有のプロパティにアクセスすることができます。次の関数の例では、オブジェクトタイプ Event を定義します。ただし、関数内で MouseEvent クラスに固有の `localX` プロパティおよび `localY` プロパティにアクセスするには、Event オブジェクトを MouseEvent タイプにキャストする必要があります。

```
<?xml version="1.0"?>
<!-- events/AccessEventSpecificProperties.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="addListeners()">
  <mx:Script><![CDATA[
     private function customLogEvent(e:Event):void {
        var a:MouseEvent = MouseEvent(e);
        trace(a.localY + ":" + a.localX);
     }

     private function addListeners():void {
        b1.addEventListener(MouseEvent.CLICK, customLogEvent);
     }
  ]]></mx:Script>
  <mx:VBox id="vb1">
     <mx:Button id="b1" label="Click Me"/>
  </mx:VBox>
</mx:Application>
```

Event オブジェクトを特定のタイプとして宣言した場合は、ハンドラでそのオブジェクトをキャストする必要はありません。次に例を示します。

```
private function customLogEvent(e:MouseEvent):void { ... }
```

前の例では、プロパティにアクセスするためだけに Event オブジェクトをキャストすることもできます。その場合は、次の例に示すシンタックスを使用します。

```
<?xml version="1.0"?>
<!-- events/SinglePropertyAccess.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="addListeners()">
  <mx:Script><![CDATA[
     private function customLogEvent(e:Event):void {
        trace(MouseEvent(e).localY + ":" + MouseEvent(e).localX);
     }
```

```
    private function addListeners():void {
       b1.addEventListener(MouseEvent.CLICK, customLogEvent);
    }
  ]]></mx:Script>
  <mx:VBox id="vb1">
     <mx:Button id="b1" label="Click Me"/>
  </mx:VBox>
</mx:Application>
```

こうすると、メモリとシステムリソースの消費量を抑えることができます。ただし、イベントのタイプはできる限り詳細に宣言することをお勧めします。

Event オブジェクトの各サブクラスには、そのイベントのカテゴリに固有のプロパティおよびイベントタイプが追加されています。MouseEvent クラスでは、その入力デバイスに関連したイベントタイプが複数定義されています。これには、CLICK、DOUBLE_CLICK、MOUSE_DOWN、MOUSE_UP などのイベントタイプがあります。

各 Event サブクラスのタイプの一覧については、『Adobe Flex 2 リファレンスガイド』のサブクラスの項目を参照してください。

キーボードイベントについて

アプリケーションは通常、1 つのキーまたは一連のキーに応答して何らかのアクションを実行します。Ctrl + q を押すとアプリケーションが終了するのはその一例です。Flash Player はベースとなるオペレーティングシステムの基本的なキー組み合わせ機能をすべてサポートしていますが、開発者が任意のキーまたはキーの組み合わせをオーバーライドまたはトラップしてカスタムアクションを実行することも可能です。

キーボードイベントの処理

場合によっては、キーをグローバルにトラップしなければならない、つまりユーザーがアプリケーションのどこを操作しているかにかかわらず、そのキーストロークをアプリケーションで認識してアクションを実行しなければならないことがあります。Flex は、ユーザーがボタンの上にマウスを移動している場面でも、あるいはフォーカスが TextInput コントロール内にある場面でも、グローバルにキーボードイベントを認識します。

グローバルなキープレスの一般的な処理方法は、KeyboardEvent.KEY_DOWN または KeyboardEvent.KEY_UP イベントのリスナーをアプリケーション上に作成することです。アプリケーションコンテナ上のリスナーは、フォーカスの有無にかかわらずキーが押されるたびにトリガされます。ハンドラ内では、KeyboardEvent クラスの charCode プロパティや keyCode プロパティを使用してキーコードまたは文字コードを検査します。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/TrapAllKeys.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     private function initApp():void {
        application.addEventListener(KeyboardEvent.KEY_UP,keyHandler);

        // Set the focus somewhere inside the application.
        myCanvas.setFocus();
     }

     // Quit the application by closing the browser using JavaScript
     // if the user presses Shift+Q. This may not work in all browsers.
     private function keyHandler(event:KeyboardEvent):void {
           trace(event.keyCode + "/" + event.charCode);
           var url:URLRequest = new URLRequest("javascript:window.close()");
           navigateToURL(url,"_self");
     }
  ]]></mx:Script>

  <mx:Canvas id="myCanvas"/>

</mx:Application>
```

UIComponent を拡張するクラスはすべて keyUp イベントと keyDown イベントを送出するため、フォーカスが各コンポーネントにあるときに押されたキーをトラップすることもできます。次に例を示します。

```
<?xml version="1.0"?>
<!-- events/TrapKeysOnTextArea.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     private function keyHandler(event:KeyboardEvent):void {
           trace(event.keyCode + "/" + event.charCode);
     }
  ]]></mx:Script>

  <mx:TextInput id="my_input" keyUp="keyHandler(event)"/>

</mx:Application>
```

keyCode プロパティと charCode プロパティについて

keyCode プロパティと charCode プロパティを使用することで、どのキーが押されたかを判断し、その結果として他のアクションをトリガできます。keyCode プロパティは、キーボードのキーの値に対応する数値です。charCode プロパティは、現在の文字セットにおけるそのキーの数値です。デフォルトの文字セットは UTF-8 で、ASCII をサポートしています。キーコード値と文字値の主な違いは、キーコード値がキーボードの特定のキーを表すのに対し、文字値は特定の文字を表すことです。キーコード値では、キーパッドの 1 はキーボードの一番上の行の 1 とは異なりますが、キーボードの 1 と ! を生成するキーは同じです。文字値では、R と r は異なります。

キーとキーコードのマッピングはデバイスやオペレーティングシステムによって異なります。ASCII 値については、ActionScript マニュアルに掲載されています。

次の例では、押しているキーの文字値とキーコード値を示します。この例を実行するときは、始める前にアプリケーションのフォーカスを置く必要があります。

```
<?xml version="1.0"?>
<!-- charts/ShowCharAndKeyCodes.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
     import flash.events.KeyboardEvent;

     private function init():void {
        ti1.setFocus();
        this.addEventListener(KeyboardEvent.KEY_DOWN, trapKeys);
     }

     private function trapKeys(e:KeyboardEvent):void {
        l0.text = String(e.toString());

        l1.text = numToChar(e.charCode) + " (" + String(e.charCode) + ")";
        l2.text = numToChar(e.keyCode) + " (" + String(e.keyCode) + ")";
     }

    private function numToChar(num:int):String {
        if (num > 47 && num < 58) {
            var strNums:String = "0123456789";
            return strNums.charAt(num - 48);
        } else if (num > 64 && num < 91) {
            var strCaps:String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            return strCaps.charAt(num - 65);
        } else if (num > 96 && num < 123) {
            var strLow:String = "abcdefghijklmnopqrstuvwxyz";
            return strLow.charAt(num - 97);
        } else {
            return num.toString();
        }
    }
```

```
  ]]></mx:Script>

  <mx:TextInput width="50%" id="ti1"/>

  <mx:Canvas id="mainCanvas" width="100%" height="100%">
     <mx:Form>
        <mx:FormItem label="Char (Code)">
           <mx:Label id="l1"/>
        </mx:FormItem>
        <mx:FormItem label="Key (Code)">
           <mx:Label id="l2"/>
        </mx:FormItem>
        <mx:FormItem label="Key Event">
           <mx:Label id="l0"/>
        </mx:FormItem>
     </mx:Form>
  </mx:Canvas>

</mx:Application>
```

KeyboardEvent ハンドラの条件演算子を使用して、特定のキーまたはキーの組み合わせを受け取ることができます。次の例では、Shift キーと q キーの組み合わせをリッスンし、両方のキーが同時に押されたときにブラウザウィンドウを閉じます。

```
<?xml version="1.0"?>
<!-- events/TrapQKey.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     private function initApp():void {
        application.addEventListener(KeyboardEvent.KEY_UP,keyHandler);
        // Set the focus somewhere inside the application.
        myCanvas.setFocus();
     }

     //This function quits the application if the user presses Shift+Q.
     private function keyHandler(event:KeyboardEvent):void {
        var bShiftPressed:Boolean = event.shiftKey;
        if (bShiftPressed) {
           var curKeyCode:int = event.keyCode;
           if (curKeyCode == 81) { // 81 is the keycode value for the Q key
            // Quit the application by closing the browser using JavaScript.
            // This may not work in all browsers.
            var url:URLRequest = new
            URLRequest("javascript:window.close()");
            navigateToURL(url,"_self");
           }
        }
     }
  ]]></mx:Script>
```

```
  <mx:Canvas id="myCanvas"/>

</mx:Application>
```

このアプリケーションは、ブラウザで実行するときにフォーカスを持つ必要があります。そうしないと、アプリケーションがキーボードイベントをキャプチャできません。

KeyboardEvent の優先度について

コントロールとその親の両方に対して keyUp または keyDown イベントリスナーを定義した場合は、イベントバブリングにより、コンポーネントごとにキーボードイベントが送出されます。両者の違いは、KeyboardEvent オブジェクトの currentTarget プロパティが異なる点だけです。

次の例では、アプリケーション、my_vbox コンテナ、my_textinput コントロールが 3 者とも keyUp イベントを keyHandler() イベントリスナー関数に送出します。

```
<?xml version="1.0"?>
<!-- events/KeyboardEventPrecedence.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     private function initApp():void {
        application.addEventListener(KeyboardEvent.KEY_UP, keyHandler);
        my_vbox.addEventListener(KeyboardEvent.KEY_UP, keyHandler);
        my_textinput.addEventListener(KeyboardEvent.KEY_UP, keyHandler);

        // Set the focus somewhere inside the application.
        my_textinput.setFocus();
     }

     private function keyHandler(event:KeyboardEvent):void {
          trace(event.target + "(" + event.currentTarget + "): " + event.keyCode
+ "/" + event.charCode);
     }
  ]]></mx:Script>

  <mx:VBox id="my_vbox">
     <mx:TextInput id="my_textinput"/>
  </mx:VBox>

</mx:Application>
```

trace() メソッドの出力を調べると、KeyboardEvent オブジェクトの target プロパティが同じ値のままであることがわかります。このプロパティは、イベントの元々のディスパッチャ (この場合は my_textinput) を参照するからです。しかし、currentTarget プロパティは現在のノードがバブルしているものによって変わります (この場合は、my_textinput から my_vbox に変わり、アプリケーション自体に変わります)。

イベントリスナーの呼び出し順序はオブジェクト階層によって決まります。addEventListener() メソッドが呼び出された順序ではありません。子コントロールの方がその親より先にイベントを送出します。この例では、キーが押されるたびに、TextInput コントロールが最初にイベントを送出し、その次に VBox コンテナ、最後にアプリケーションがイベントを送出します。

基本オペレーティングシステムまたはブラウザによって認識されるキーまたはキーの組み合わせを処理するとき、通常はオペレーティングシステムまたはブラウザが先にイベントを処理します。たとえば Microsoft Internet Explorer では、Ctrl + w を押すとブラウザウィンドウが閉じます。Flex アプリケーションで Ctrl + w をトラップしても、ActiveX Flash Player がイベントに応答する機会を得る前にブラウザが閉じるため、Internet Explorer ユーザーはそれに気付きません。

キーボードに関連した MouseEvent の処理

MouseEvent クラスとすべての MouseEvent サブクラス (ChartItemEvent、DragEvent、ItemClickEvent、LegendMouseEvent など) は、次のプロパティを備えています。開発者はこれらを使用して、イベント発生時に特定のキーが押されていたかどうかを判断できます。

プロパティ	説明
altKey	ユーザーがマウスボタンを押したときに Alt キーが押されていた場合、true に設定されます。それ以外の場合は false に設定されます。
ctrlKey	ユーザーがマウスボタンを押したときに Ctrl キーが押されていた場合、true に設定されます。それ以外の場合は false に設定されます。
shiftKey	ユーザーがマウスボタンを押したときに Shift キーが押されていた場合、true に設定されます。それ以外の場合は false に設定されます。

次の例では、ユーザーが Shift キーを押しながらマウスボタンを押したかどうかに基づいて、Button コントロールを削除します。

```
<?xml version="1.0"?>
<!-- events/DetectingShiftClicks.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.controls.Button;

     private function initApp():void {
        var b1:Button = new Button();
        var b2:Button = new Button();

        b1.addEventListener(MouseEvent.CLICK, removeButtons);
        b2.addEventListener(MouseEvent.CLICK, removeButtons);

        addChild(b1);
        addChild(b2);
```

```
      }

      private function removeButtons(event:MouseEvent):void {
         if (event.shiftKey) {
            removeChild(Button(event.currentTarget));
         } else {
            event.currentTarget.toolTip = "You must hold the shiftkey down to
delete this button.";
         }
      }
   ]]></mx:Script>

</mx:Application>
```

第 2 部

Flex アプリケーションのユーザーインターフェイスの作成

2

第 II 部では、Adobe Flex 2 コンポーネントを使用してアプリケーションのユーザーインターフェイスを作成する方法について説明します。

次のトピックが含まれます。

第 6 章

Flex ビジュアルコンポーネントの使用

6

Adobe Flex アプリケーションの構築にはビジュアルコンポーネントを使用します。ビジュアルコンポーネント (通常は " コンポーネント " と略されます) は、アプリケーションの要件に合わせて制御および設定できる柔軟な属性を備えています。このトピックでは、コンポーネントの概要、コンポーネントのシンタックス、およびコンポーネントの設定について説明します。

目次

ビジュアルコンポーネントについて

Flex は、アプリケーションやそのユーザーインターフェイスを開発するためのコンポーネントベースの開発モデルをサポートしています。Flex には作成済みのビジュアルコンポーネントが付属しています。これらのコンポーネントを拡張して新しいプロパティやメソッドを追加できるのはもちろん、アプリケーションに必要なコンポーネントを新たに作成することも可能です。

ビジュアルコンポーネントは非常に柔軟性が高いため、コンポーネントの外観、ユーザーの操作に対する応答、表示するテキストに使用するフォントの種類やサイズ、アプリケーション内でのコンポーネントのサイズなど、各種の属性を自在に制御できます。

このトピックでは、ビジュアルコンポーネントの次のような属性について概説します。

サイズ　コンポーネントの高さと幅。ビジュアルコンポーネントにはすべてデフォルトのサイズがあります。デフォルトサイズを使用したり、特定サイズを指定できる他、アプリケーションのレイアウト時に自動的にコンポーネントのサイズを変更することもできます。

イベント　コンポーネントの応答を要求するアプリケーションまたはユーザーの操作。イベントには、コンポーネントの作成、マウスの操作 (目的のコンポーネントにマウスポインタを合わせるなど)、ボタンのクリックなどがあります。

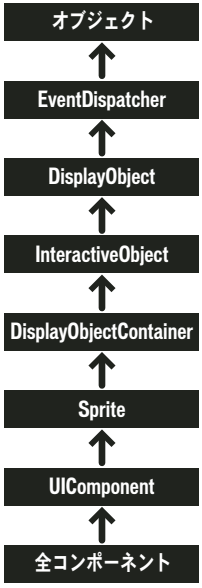
スタイル　フォント、フォントサイズ、テキストの整列などの属性。これらは、カスケーディングスタイルシート (CSS) で定義、使用するスタイルと同じです。

ビヘイビア　アプリケーションやユーザーの操作によってトリガされるコンポーネントの外観や音声の変更。ビヘイビアの例としては、マウスのクリックによるコンポーネントの移動やサイズ変更があります。

スキン　ビジュアルコンポーネントの外観を制御するクラス。

ビジュアルコンポーネントのクラス階層

Flex ビジュアルコンポーネントは ActionScript のクラス階層として実装されています。したがって、アプリケーションの各ビジュアルコンポーネントは ActionScript クラスのインスタンスになります。次の図は、Flash Sprite コンポーネントレベルまでの詳細なクラス階層を示しています。



すべてのコンポーネントは、UIComponent クラスとそのスーパークラス (Flash Sprite クラスから Object クラス) から派生し、それらのスーパークラスのプロパティとメソッドを継承します。また、イベント、スタイル、ビヘイビアの定義などの属性もスーパークラスから継承します。

UIComponent クラスの使用

UIComponent クラスは、すべての Flex ビジュアルコンポーネントの基本クラスです。詳細については、『Adobe Flex 2 リファレンスガイド』の UIComponent を参照してください。

よく使用される UIComponent プロパティ

次の表は、UIComponent クラスを拡張したコンポーネントの最もよく使用されるプロパティのみをまとめたものです。

プロパティ	データ型	説明
doubleClickEnabled	Boolean	true に設定すると、ユーザーがコンポーネントの上にマウスを置き、マウスボタンを 2 回連続してすばやく押して離したときに、コンポーネントが doubleClickEvent イベントを送出します。
enabled	Boolean	true に設定すると、コンポーネントがキーボードフォーカスとマウス入力を受け取ります。デフォルト値は true です。 あるコンテナに対して enabled を false に設定すると、そのコンテナとそのすべての子がグレー表示になり、それらに入力できなくなります。
height	Number	コンポーネントの高さ (ピクセル単位)。 MXML タグではこのプロパティに、使用可能な領域に対するパーセント値 (たとえば 70% など) を設定できます。ただし、ActionScript ではこのような指定はできず、percentHeight プロパティを使用する必要があります。 このプロパティは常にピクセル数を返します。ActionScript では、perCent を使用します。
id	String	コンポーネントの識別子を指定します。この値はオブジェクトの特定のインスタンスを識別します。値に空白または特殊文字を含めないでください。Flex ドキュメント内の各コンポーネントが一意の id 値を持つ必要があります。たとえば、カスタムコンポーネントが 2 つある場合、各コンポーネントに "okButton" という id を持つ Button コントロールを 1 つだけ含めることができます。

プロパティ	データ型	説明
percentHeight	Number	コンポーネントの高さを、その親コンテナに対するパーセント値で設定します。<mx:Application> タグでは、ブラウザの高さ全体に対するパーセント値になります。幅がパーセント値で設定されていない場合、または percentWidth が設定された後に width プロパティが設定された場合は、NaN を返します。
percentWidth	Number	コンポーネントの幅を、その親コンテナに対するパーセント値で設定します。<mx:Application> タグでは、ブラウザの幅全体に対するパーセント値になります。幅がパーセント値で設定されていない場合、または percentWidth が設定された後に width プロパティが設定された場合は、NaN を返します。
styleName	String	コンポーネントに適用するスタイルクラスセレクタを指定します。
toolTip	String	コンポーネントにマウスポインタを合わせたときに表示されるテキストを指定します。
visible	Boolean	コンテナを表示するか非表示にするかを指定します。デフォルト値は true で、コンテナが表示されることを示します。
width	Number	コンポーネントの幅 (ピクセル単位)。 MXML タグではこのプロパティに、使用可能な領域に対するパーセント値 (たとえば 70% など) を設定できます。ただし、ActionScript ではこのような指定はできず、percentWidth プロパティを使用する必要があります。 このプロパティは常にピクセル数を返します。
x	Number	親コンテナ内でのコンポーネントの *x* 座標。このプロパティを直接設定することは、親コンテナが絶対配置を使用している場合のみ効果があります。
y	Number	親コンテナ内でのコンポーネントの *y* 座標。このプロパティを直接設定することは、親コンテナが絶対配置を使用している場合のみ効果があります。

MXML および ActionScript でのコンポーネントの使用

すべての Flex コンポーネントには、MXML API および ActionScript API が実装されています。コンポーネントの MXML タグの属性は、ActionScript のプロパティ、スタイル、ビヘイビア、およびイベントに相当します。コンポーネントに対する操作は、MXML と ActionScript のどちらを使用しても可能です。

コンポーネントを設定するには :

1. コンポーネントのプロパティ、イベント、スタイル、ビヘイビアなどの値を MXML タグを使って宣言形式で設定するか、ActionScript コードから実行時に設定します。
2. コンポーネントのメソッドは ActionScript コードから実行時に呼び出します。MXML API では ActionScript クラスのメソッドは公開されていません。

次のコードは、MXML で Button コントロールを作成する例です。Button コントロールをクリックすると、ActionScript の関数により TextArea コントロールのテキストが更新されます。

```
<?xml version="1.0"?>
<!-- components\ButtonApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function handleClick():void {
                text1.text="Thanks for the click!";
            }
        ]]>
    </mx:Script>

    <mx:Button id="button1"
        label="Click here!"
        width="100"
        fontSize="12"
        click="handleClick();"/>
    <mx:TextArea id="text1"/>
</mx:Application>
```

この例には次のエレメントがあります。

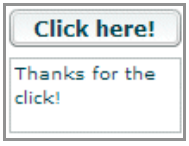
- `id` プロパティは、Button コントロールによって UIComponent クラスから継承されます。これはコンポーネントの識別子を指定するために使用します。このプロパティは省略可能ですが、ActionScript でコンポーネントにアクセスする場合は指定する必要があります。
- `label` プロパティは、Button コントロールによって定義されています。ボタンに表示するテキストを指定します。
- `width` プロパティは、UIComponent クラスから継承されます。省略可能で、ボタンの幅をピクセル単位で指定します。

- Button コントロールは、ユーザーがマウスのメインボタンを押して離したときに `click` イベントを送出します。MXML の `click` 属性は、click イベントに応答して ActionScript コードを実行することを指定します。
- `fontSize` スタイルは、UIComponent クラスから継承されます。ラベルテキストのフォントサイズをピクセル単位で指定します。

`click` イベント属性では、パラメータに関数を渡す代わりに、ActionScript のコードを直接指定することもできます。したがって、前述のコードは次のように書き換えることができます。

```
<?xml version="1.0"?>
<!-- components\ButtonAppAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button id="button1"
        label="Click here!"
        width="100"
        fontSize="12"
        click="text1.text='Thanks for the click!';"/>

    <mx:TextArea id="text1"/>
</mx:Application>
```

これらの例はどちらも、ボタンをクリックすると次のイメージが表示されます。



メモ

`click` イベント属性の値には、複数行の ActionScript コードをセミコロンで区切って指定できますが、読みやすくするために `click` イベントを 1 ～ 2 行以内に収めることをお勧めします。

実行時のコンポーネントの初期化

コンポーネントは MXML プロパティ属性を使用して初期化されます。ただし、実行時に初期値が決まるようなロジックが必要になる場合もあります。たとえば、現在の日時に基づいてコンポーネントを初期化するような場合がこれに該当します。このような情報は、アプリケーションが実行された時点で計算する必要があります。

コンポーネントは、ライフサイクルの間にいくつかのイベントを送出します。特に、コンポーネントは次のイベントを必ず送出します。そのため、これらのイベントを使用して、コンポーネントを初期化する ActionScript を指定できます。

preInitialize　コンポーネントが未加工の状態で作成されたときに送出されます。このとき、子はまだ作成されていません。

initialize　コンポーネントとそのすべての子が作成された後、コンポーネントのサイズが決定される前に送出されます。

creationComplete　コンポーネントのレイアウトが完了し、必要に応じてコンポーネントが表示されたときに送出されます。

メモ

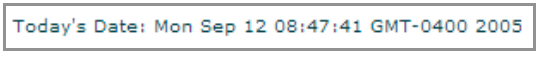
コンポーネントの作成方法の詳細については、141 ページの「コンポーネントのインスタンス化ライフサイクルについて」を参照してください。

`initialize` イベントではほとんどのコンポーネント属性を設定できますが、その中でも特にコンポーネントのサイズに影響を与える値を設定します。`creationComplete` イベントは、初期化コードでコンポーネントのレイアウトに関する情報が必要となる場合に使用します。

次の例では、Label コントロールの初期化時に `initDate()` 関数を呼び出すように設定しています。Label コントロールのインスタンス化が完了した後、アプリケーションが表示される前に、`initDate()` 関数が呼び出されます。

```
<?xml version="1.0"?>
<!-- components\LabelInit.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function initDate():void {
                label1.text += new Date();
            }
        ]]>
    </mx:Script>

    <mx:Box borderStyle="solid">
        <mx:Label id="label1"
            text="Today's Date: "
            initialize="initDate();"/>
    </mx:Box>
</mx:Application>
```

この例では、次のイメージが作成されます。



また、コンポーネントの定義に直接 ActionScript コードを記述することで、関数呼び出しを使わずに上のコード例を表現することもできます。明示的な関数呼び出しを使わずに記述すると、次のようになります。

```
<?xml version="1.0"?>
<!-- components\LabelInitAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Box borderStyle="solid">
        <mx:Label id="label1"
```

```
            text="Today's Date:"
            initialize="label1.text += new Date();"/>
    </mx:Box>
</mx:Application>
```

コンポーネント定義に埋め込まれた他の呼び出しと同様に、それぞれの関数またはメソッド呼び出しをセミコロンで区切ることにより、複数の ActionScript ステートメントを `initialize` MXML 属性に追加できます。次の例は、`label1` コンポーネントがインスタンス化されるときに `initDate()` 関数を呼び出し、メッセージを "flexlog.txt" ファイルに書き込みます。

```
<?xml version="1.0"?>
<!-- components\LabelInitASAndEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function initDate():void {
                label1.text += new Date();
            }
        ]]>
    </mx:Script>

    <mx:Box borderStyle="solid">
        <mx:Label id="label1"
            text="Today's Date:"
            initialize="initDate(); trace('The label is initialized!');"/>
    </mx:Box>
</mx:Application>
```

コンポーネントの設定 : シンタックス一覧

コンポーネントの設定に使用できる MXML と ActionScript コンポーネントの API を次の表に示します。

	MXML の例	ActionScript の例
読み取り / 書き込み可能プロパティ	`<mx:Tile id="tile1" label="My Tile" visible="true"/>`	`tile1.label="My Tile";` `tile1.visible=true;`
読み取り専用プロパティ	MXML の属性として読み取り専用プロパティを使用することはできません。	読み取り専用プロパティから値を取得する `var theClass:String=mp1.className;`
メソッド	MXML ではメソッドは使用できません。	`myList.sortItemsBy("data", "DESC");`

	MXML の例	ActionScript の例

イベント	`<mx:Accordion id="myAcc"change="changeHandler(event);"/>`
	
	ActionScript の例のように、`changeHandler()` 関数も定義する必要があります。
		`private functionchangeHandler(event:MouseEvent): void {`
		`...`
		`}`
		
		`myButton.addEventListener("click", changeHandler);`

スタイル	`<mx:Tile id="tile1"paddingTop="12" paddingBottom="12"/>`	スタイルを設定する
		`tile1.setStyle("paddingTop", 12);`
		`tile1.setStyle("paddingBottom", 12);`
		スタイルを取得する
		`var currentPaddingTop:Number = tile1.getStyle("paddingTop");.`

ビヘイビア	`<mx:Tile id="tile1"showEffect="{AWipeEffect}"/>`	ビヘイビアを設定する
		`myButton.setStyle('showEffect', AWipeEffect);`
		ビヘイビアを取得する
		`var currentShowEffect:String = tile1.getStyle("showEffect");`

ビジュアルコンポーネントのサイズ設定

Flex のサイズ設定およびレイアウトメカニズムでは、次の方法でビジュアルコンポーネントのサイズを制御できます。

デフォルトサイズ設定　コントロールとコンテナのサイズを自動的に決定します。デフォルトサイズ設定を使用するには、コンポーネントのサイズやレイアウト制約を指定しないようにします。

明示的サイズ設定　`height` プロパティと `width` プロパティにピクセル値を設定します。こうすると、コンポーネントのサイズが絶対サイズに設定されます。これらの値が Flex によってオーバーライドされることはありません。たとえば次の `<mx:Application>` タグは、Application のサイズを明示的に設定します。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  height="300"
  width="600">
```

パーセント値ベースのサイズ設定　コンポーネントのサイズをコンテナサイズのパーセント値で指定します。そのためには、`percentHeight` プロパティと `percentWidth` プロパティを指定するか、または MXML タグ内で `height` プロパティと `width` プロパティを 100% などのパーセント値に設定します。たとえば次のコードは、HBox コンテナに対してパーセント値ベースのサイズを設定します。

```
<mx:HBox id="hBox1" xmlns:mx="http://www.adobe.com/2006/mxml"
  height="30%" width="90%"/>
```

次の ActionScript 行は、HBox の幅を異なるパーセント値に再設定します。

```
hBox1.percentWidth=40;
```

制約ベースのレイアウト　コンポーネントの端または中心をコンテナ内の特定の位置に固定することで、サイズと位置を同時に制御できます。指定可能なスタイルには、top、bottom、left、right、horizontalCenter、および verticalCenter があります。制約ベースのレイアウトを使用できるのは、絶対レイアウトを使用するコンテナの子のみです。Application コンテナと Panel コンテナはオプションでこのレイアウト方式を使用できます。Canvas コンテナは常にこのレイアウト方式を使用します。次の例では、制約ベースのレイアウトを使用して HBox の水平方向の位置を指定し、垂直方向の高さと位置は明示的に設定しています。

```
<mx:HBox id="hBox2" xmlns:mx="http://www.adobe.com/2006/mxml"
  left="30"right="30"
  y="150"
  height="100"/>
```

サイズ設定方法は組み合わせることができますが、適切な組み合わせにする必要があります。コンポーネントのサイズに対して複数のサイズ設定を指定しないでください。たとえば、1 つのコンポーネントに対して height と percentHeight を同時に指定することは避けてください。また、結果的なサイズが適切であることも確認する必要があります。たとえば、スクロールバーを使わない場合やコンポーネントがクリッピングされないようにする場合には、コンテナの子のサイズがコンテナのサイズを超えないようにしてください。

Flex でコンポーネントのサイズがどのように設定されるか、およびサイズ設定の指定方法の詳細については、207 ページ、第 8 章の「コンポーネントのサイズと位置の制御」を参照してください。

コンポーネントのサイズ設定の例

次の例は、明示的にサイズ設定されたコンテナの中に幅が明示的に指定された子コントロールとパーセント値で指定された子コントロールが混在している場合を示します。これは、コンポーネントのサイズを決定する際に伴う柔軟性と複雑さを表しています。後でサイズ設定動作を確認できるように、コンポーネントのサイズを "flashlog.txt" にログ出力しています。

```
<?xml version="1.0"?>
<!-- components\CompSizing.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="logSizes();">

    <mx:Script>
        <![CDATA[
            private function logSizes():void {
                trace("HBox: "+ hb1.width);
                trace("Label: "+ lb1.width);
                trace("Image: "+ img1.width);
                trace("Button: "+ b1.width);
            }
        ]]>
    </mx:Script>
```

```
    <mx:HBox id="hb1" width="250">
        <mx:Label id="lb1"
            text="Hello"
            width="50"/>
        <mx:Image id="img1"
            source="@Embed(source='assets/flexlogo.jpg')"
            width="75%"/>
        <mx:Button id="b1"
            label="Button"
            width="25%"/>
    </mx:HBox>
</mx:Application>
```

このアプリケーションは、幅 250 ピクセルの HBox コンテナと、その中に含まれる幅 50 ピクセルのラベル、コンテナの幅の 75% を要求するイメージ、およびコンテナの幅の 25% を要求するボタンから成ります。コンポーネントのサイズは次のように決定されます。

1. 明示的にサイズ設定された Label コントロールのために 50 ピクセルが確保されます。
2. コンポーネントの間隔はデフォルトで 8 ピクセルなので、この間隔のために 16 ピクセルが確保されます。これで、残り 2 つのパーセント値ベースのコンポーネントで使用できる幅は 184 ピクセルになります。
3. 明示的に幅を指定していない場合、Button コンポーネントの最小の幅は、ラベルテキストとその周囲のパディングの合計値になります。この例では、最小サイズは 65 ピクセルです。この値はコンポーネントの 25% より大きいため、要求されたパーセント値は使用されず、Button コントロールに 65 ピクセルが確保されます。
4. パーセント値ベースのイメージは 250 ピクセルの 75%、つまり 187 ピクセルを要求していますが、残りの領域が 119 ピクセルなので、このピクセル数が割り当てられます。

ボタンとイメージの size プロパティを 50% に変更すると、ボタンの最小サイズが要求サイズより小さくなるため、ボタンとイメージにそれぞれ残りの領域の 50% (92 ピクセル) が割り当てられます。

次の例では、Image コントロールのサイズを明示的に設定し、Button コントロールにはデフォルトサイズを使用しています。このコードでも、上の例と同じ結果になります。

```
<?xml version="1.0"?>
<!-- components\CompSizingExplicit.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="logSizes();">

    <mx:Script>
        <![CDATA[
            private function logSizes():void {
                trace("HBox: "+ hb1.width);
                trace("Label: "+ lb1.width);
                trace("Image: "+ img1.width);
                trace("Button: "+ b1.width);
            }
```

```
        ]]>
    </mx:Script>

    <mx:HBox id="hb1" width="250">
        <mx:Label id="lb1"
            text="Hello"
            width="50"/>
        <mx:Image id="img1"
            source="@Embed(source='assets/flexlogo.jpg')"
            width="119" />
        <mx:Button id="b1"
            label="Button"/>
    </mx:HBox>
</mx:Application>
```

パーセント値ベースのサイズ設定を使用すると、サイズ計算時にコンテナの間隔と余白を考慮する必要がなくなりますが、その最大の利点はコンテナのサイズを変更したときに発揮されます。コンテナのサイズを変更すると、パーセント値ベースの子のサイズも、使用可能なコンテナのサイズに基づいて自動的に変更されます。次の例では、ブラウザのサイズを変更すると、左側に配置された一連のコントロールのサイズも変更されます。ただし、右側に配置された同じコントロールは、コンテナのサイズが固定されているため、同じサイズのまま維持されます。最初の Button コントロールをクリックすると、コンポーネントのサイズが "flashlog.txt" にログ出力されます。

```
<?xml version="1.0"?>
<!-- components\CompSizingPercent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function logSizes():void {
                trace("HBox: "+ hb1.width);
                trace("Label: "+ lb1.width);
                trace("Image: "+ img1.width);
                trace("Button: "+ b1.width);
            }
        ]]>
    </mx:Script>

    <mx:HBox width="100%">
        <mx:HBox id="hb1" width="40%" borderStyle="solid">
            <mx:Label id="lb1"
                text="Hello"
                width="50"/>
            <mx:Image id="img1"
                source="@Embed(source='assets/flexlogo.jpg')"
                width="60%"/>
            <mx:Button id="b1"
                label="Button"
                width="40%"
```

```
                click="logSizes();"/>
        </mx:HBox>

        <mx:HBox width="260" borderStyle="solid">
            <mx:Label
                text="Hello"
                width="50"/>
            <mx:Image
                source="@Embed(source='assets/flexlogo.jpg')"
                width="119" />
            <mx:Button
                label="Button"/>
        </mx:HBox>
    </mx:HBox>
</mx:Application>
```

サイズ設定に関する注意事項については、214 ページの「コンポーネントのサイズ設定」を参照してください。

イベントの処理

Flex アプリケーションはイベント駆動型です。" イベント " は、ユーザーがインターフェイスコンポーネントを操作したことをプログラマに通知します。また、コンポーネントの作成や廃棄、サイズの変更など、コンポーネントの外観やライフサイクルで重要な変更が発生したことも通知します。

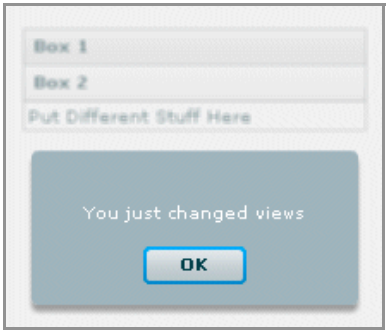
コンポーネントのインスタンスがイベントを送出すると、そのイベントの " リスナー " として登録されているオブジェクトが通知を受けます。イベントリスナー (" イベントハンドラ " と呼ぶこともあります) は、イベントを処理する ActionScript の形で定義します。イベントリスナーは、コンポーネントの MXML 宣言または ActionScript の中で登録します。イベント処理の詳細な例については、132 ページの「実行時のコンポーネントの初期化」を参照してください。

次の例は、Accordion コンテナでビューを変更したときに処理されるイベントリスナーを MXML で登録しています。

```
<?xml version="1.0"?>
<!-- components\CompIntroEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="300"
    height="280">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            private function handleAccChange():void {
                Alert.show("You just changed views");
            }
        ]]>
    </mx:Script>
```

```
    <!-- The Accordion control dispatches a change event when the
            selected child container changes. -->
    <mx:Accordion id="myAcc"
        height="60"
        width="200"
        change="handleAccChange();">
        <mx:HBox label="Box 1">
            <mx:Label text="Put Some Stuff Here"/>
        </mx:HBox>

        <mx:HBox label="Box 2">
            <mx:Label text="Put Different Stuff Here"/>
        </mx:HBox>
    </mx:Accordion>
</mx:Application>
```

この例では、次のイメージが作成されます。



コンポーネントからイベントリスナーに対して、イベントに関する情報を含むイベントオブジェクトを渡すことができます。

Accordion コンテナの場合、change イベントのイベントリスナーに渡されるイベントオブジェクトは IndexChangedEvent クラスです。次の例のように、イベントオブジェクトにアクセスするイベントリスナーを記述できます。

```
<?xml version="1.0"?>
<!-- components\CompIntroEventAcc.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="300"
    height="280">

    <mx:Script>
        <![CDATA[
            // Import the class that defines the event object.
            import mx.events.IndexChangedEvent;
            import mx.controls.Alert;
```

```
            private function handleChange(event:IndexChangedEvent):void {
                var currentIndex:int=event.newIndex;
                Alert.show("You just changed views \nThe new index is "
                    + event.newIndex);
            }
        ]]>
    </mx:Script>

    <!-- The Accordion control dispatches a change event when the
            selected child container changes. -->
    <mx:Accordion id="myAcc"
        height="60"
        width="200"
        change="handleChange(event);">
        <mx:HBox label="Box 1">
            <mx:Label text="Put Some Stuff Here"/>
        </mx:HBox>

        <mx:HBox label="Box 2">
            <mx:Label text="Put Different Stuff Here"/>
        </mx:HBox>
    </mx:Accordion>
</mx:Application>
```

この例では、Accordion コンテナの新しい子のインデックスを特定するために、IndexChangedEvent オブジェクトの newIndex プロパティにアクセスしています。イベントの詳細については、81 ページ、第 5 章の「イベントの使用」を参照してください。

コンポーネントのインスタンス化ライフサイクルについて

コンポーネントインスタンス化ライフサイクルは、コンポーネントクラスからコンポーネントオブジェクトを作成するときに生じる一連の手順を記述したものです。Flex は、このライフサイクルの一環として、コンポーネントメソッドの呼び出し、イベントの送出、およびコンポーネントの可視化を自動的に行います。

次の例では、Button コントロールを作成してコンテナに追加しています。

```
<?xml version="1.0"?>
<!-- components\AddButtonToContainer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Box id="box1" width="200">
        <mx:Button id="button1" label="Submit"/>
    </mx:Box>
</mx:Application>
```

次の ActionScript は、MXML コードの一部に対応します。Flex はどちらの例でも同じ一連の手順を実行します。

```
// Box コンテナを作成します。
var box1:Box = new Box();
// Box コンテナを設定します。
box1.width=200;

// Button コントロールを作成します。
var button1:Button = new Button()
// Button コントロールを設定します。
button1.label = "Submit";

// Button コントロールを Box コンテナに追加します。
box1.addChild(button1);
```

次の手順は、ActionScript コードを実行して Button コントロールを作成し、それを Box コンテナに追加したときに何が行われるかを示したものです。MXML でコンポーネントを作成すると、Flex 2 SDK によってそれに対応するコードが生成されます。

1. 次のコードに示すように、コンポーネントのコンストラクタを呼び出します。

   ```
   // Button コントロールを作成します。
   var button1:Button = new Button()
   ```

2. 次のコードに示すように、コンポーネントのプロパティを設定します。

   ```
   // Button コントロールを設定します。
   button1.label = "Submit";
   ```

3. 次のコードに示すように、addChild() メソッドを呼び出してコンポーネントをその親に追加します。

   ```
   // Button コントロールを Box コンテナに追加する。
   box1.addChild(button1);
   ```

   Flex はこの行を処理するため、次のアクションを実行します。

   a. コンポーネントの parent プロパティがその親コンテナを参照するように設定します。

   b. コンポーネントのスタイル設定を計算します。

   c. Button コントロールから add イベントを送出します。

   d. 親コンテナから childAdd イベントを送出します。

   e. コンポーネントの preinitialize イベントを送出します。このイベントが送出されるとき、コンポーネントは未加工の状態にあります。Button コントロールなどの多くのコンポーネントは、内部子コンポーネントを作成して機能を実装します。たとえば、Button コントロールは内部 UITextField コンポーネントを作成してラベルテキストを表現します。Flex が preinitialize イベントを送出したとき、コンポーネントの子 (内部の子を含む) はまだ作成されていません。

   f. コンポーネントの子 (内部の子を含む) を作成して初期化します。

   g. コンポーネントの initialize イベントを送出します。この時点で、コンポーネントの子はすべて初期化されていますが、コンポーネントの処理は完全には終わっていません。特に、レイアウトのためのサイズ設定がまだ行われていません。

4. その後、アプリケーションを表示するために render イベントがトリガされます。このとき Flex は次のことを行います。
   a. コンポーネントのレイアウトなど、コンポーネントの表示に必要な処理をすべて完了します。
   b. visible プロパティを true に設定して、コンポーネントを可視化します。
   c. コンポーネントの creationComplete イベントを送出します。コンポーネントのサイズ設定とレイアウトに関する処理が完了し、すべてのプロパティが設定されます。このイベントはコンポーネントが作成されたときに1回だけ送出されます。
   d. コンポーネントの updateComplete イベントを送出します。コンポーネントの位置やサイズなどの視覚的特徴が変更され、表示が更新されるたびに、追加の updateComplete イベントが送出されます。

後で removeChild() メソッドを使用して、コンテナからコンポーネントを削除できます。削除した子の parent プロパティは null に設定されます。削除した子を別のコンテナに追加した場合、その子は最後の既知の状態を保持します。コンポーネントへの参照がなくなると、Adobe Flash Player のガベージコレクションメカニズムによって最終的にメモリから削除されます。

この一連のアクションから考えて、イベントは次のように使用するのが適切です。

- preinitialize イベントはコンポーネントライフサイクルの早い段階で発生し、ほとんどの初期化アクティビティに対応します。ただしこのイベントが役立つのは、子が作成される前に親のプロパティを設定しなければならないというまれな状況に限られます。
- 視覚的外観が Flex によって決定される前にコンポーネントを設定するには、initialize イベントを使用します。たとえば、このイベントを使用して、外観、高さ、または幅に影響を与えるプロパティを設定します。
- creationComplete イベントは、コンポーネント作成時のサイズや位置の正確な値を必要とする処理に対して使用します。このイベントを使用してコンポーネントの視覚的外観を変更する処理を行うと、レイアウトの再計算が必要となり、アプリケーションに不要な処理オーバーヘッドが追加されます。
- updateComplete イベントは、コンポーネントの作成時ではなく、コンポーネントの属性が変更されるたびに実行する処理に対して使用します。

スタイルの使用

Flex ではスタイルを定義することにより、フォント、パディング、整列方法など、コンポーネントの一部の属性を設定できます。これらは、カスケーディングスタイルシート (CSS) で定義、使用するスタイルと同じです。各ビジュアルコンポーネントはスーパークラスのスタイルの多くを継承しながら、独自のスタイルを定義できます。スーパークラスのスタイルの中にはサブクラスで使用されないものもあります。ビジュアルコンポーネントでサポートされているスタイルを確認するには、『Adobe Flex 2 リファレンスガイド』のコンポーネントのページにあるスタイルのセクションを参照してください。

MXML では、すべてのスタイルをタグ属性として設定できます。したがって、次の例のように、paddingTop プロパティと paddingBottom プロパティを使用して Box コンテナの境界とその内容との間隔を設定できます。

```
<?xml version="1.0"?>
<!-- components\MXMLStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox id="myVBox1" borderStyle="solid">
        <mx:Button label="Submit"/>
    </mx:VBox>

    <mx:VBox id="myVBox2"
        borderStyle="solid"
        paddingTop="12"
        paddingBottom="12" >

        <mx:Button label="Submit"/>
    </mx:VBox>
</mx:Application>
```

また、ActionScript で setStyle() メソッドを使用するか、MXML で <mx:Style> タグを使用して、スタイルを設定することもできます。setStyle() メソッドは、スタイル名と値の 2 つのパラメータを受け取ります。次の例は、前の例と同じ機能を持ちます。

```
<?xml version="1.0"?>
<!-- components\ASStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function initVBox():void {
                myVBox2.setStyle("paddingTop", 12);
                myVBox2.setStyle("paddingBottom", 12);
            }
        ]]>
    </mx:Script>

    <mx:VBox id="myVBox1" borderStyle="solid">
        <mx:Button label="Submit"/>
    </mx:VBox>

    <mx:VBox id="myVBox2"
        borderStyle="solid"
        initialize="initVBox();">

        <mx:Button label="Submit"/>
    </mx:VBox>
</mx:Application>
```

<mx:Style> タグを使用するときは、CSS シンタックス、またはスタイル宣言を含む外部ファイルへの参照を使用してスタイルを設定します。次に例を示します。

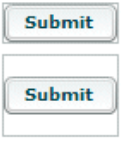
```
<?xml version="1.0"?>
<!-- components\TagStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        .myStyle {paddingTop: 12; paddingBottom: 12;}
    </mx:Style>

    <mx:VBox id="myVBox1" borderStyle="solid">
        <mx:Button label="Submit"/>
    </mx:VBox>

    <mx:VBox id="myVBox2"
        styleName="myStyle"
        borderStyle="solid">

        <mx:Button label="Submit"/>
    </mx:VBox>
</mx:Application>
```

スタイル定義の中でピリオドに続けて指定された部分は " クラスセレクタ " と呼ばれ、新たに指定するスタイルを定義しています。上の例では myClass がこれに相当します。定義したスタイルは、styleName プロパティを使用してコンポーネントに適用できます。上の例では、このスタイルを 2 番目の VBox コンテナに適用しています。

" タイプセレクタ " は、特定のコンポーネントタイプのすべてのインスタンスにスタイルを適用します。次の例では、すべての Box コンテナの上下の余白を定義しています。

```
<?xml version="1.0"?>
<!-- components\TypeSelStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        Box {paddingTop: 12; paddingBottom: 12;}
    </mx:Style>

    <mx:VBox id="myVBox" borderStyle="solid">
        <mx:Button label="Submit"/>
    </mx:VBox>

    <mx:Box id="myBox" borderStyle="solid">
        <mx:Button label="Submit"/>
    </mx:Box>
</mx:Application>
```

Flex の一部のスタイル (すべてのスタイルではない) は親コンテナから子コンテナに継承され、さらにスタイルタイプ全体やクラス全体に継承されます。borderStyle スタイルは VBox コンテナによって継承されないため、この例は前の例と同じ結果になります。上記のどの例も、次のようなアプリケーションになります。



スタイルの詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

ビヘイビアの使用

" ビヘイビア " とは、トリガとエフェクトを組み合わせたものです。" トリガ " はイベントとよく似ていますが、マウスのクリック、フォーカスの移動、不可視状態から可視状態への変更など、コンポーネントに対して行われる操作を特に指します。" エフェクト " はコンポーネント上で一定時間 (ミリ秒単位) に発生する視覚的または聴覚的変化です。エフェクトには、コンポーネントのフェード、サイズ変更、移動などがあります。1 つのトリガに対して複数のエフェクトを定義することもできます。

Flex のトリガプロパティは CSS スタイルで実装されています。『Adobe Flex 2 リファレンスガイド』では、トリガは " エフェクト " の見出しの下に一覧で示されています。Flex のエフェクトは、mx.effects.Fade クラスなどのクラスです。

ビヘイビアを使用すると、アニメーション、モーション、サウンドなどを、ユーザーまたはプログラムによって実行される特定の操作に応答する形で、アプリケーションに追加できます。たとえば、ビヘイビアを使用すると、ダイアログボックスにフォーカスが移ったときにわずかに飛び跳ねるような効果を付けたり、不適切な値が入力された場合に音が鳴るようにすることができます。

エフェクトトリガは CSS スタイルで実装されているため、トリガプロパティは MXML のタグ属性、<mx:Style> タグ、または ActionScript の setStyle 関数を使用して設定できます。

ビヘイビアを作成するには、特定のエフェクトを一意の ID で定義し、それをトリガにバインドします。たとえば次のコードでは、myFade という名前のフェードエフェクトを作成し、MXML 属性を使用して、そのエフェクトを Image コントロールの creationCompleteEffect トリガにバインドしています。

```
<?xml version="1.0"?>
<!-- components\CompIntroBehaviors.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="110"
    height="100"
    backgroundImage="">

    <mx:Fade id="myFade" duration="5000"/>
    <mx:Image
```

```
        creationCompleteEffect="{myFade}"
        source="@Embed(source='assets/flexlogo.jpg')"/>
</mx:Application>
```

この例では、Image コントロールが 5000 ミリ秒かけてフェードインします。イメージは時間の経過に伴って次のようにフェードインします。

ビヘイビアの使用の詳細については、603 ページ、第 17 章の「ビヘイビアの使用」を参照してください。

スキンの適用

" スキン " は、コンポーネントで外観の制御に使用されるグラフィカルなスタイルエレメントです。Flex コンポーネントは複数のスキンから構成できます。たとえば Button コンポーネントには 8 種類のスキンがあり、それぞれがアップ、ダウン、無効、選択などのボタンの状態に対応しています。また、コントロールのサブコンポーネントごとに異なるスキンを持つことができます。たとえば、スクロールバーには下矢印、上矢印、およびサムに対してそれぞれ異なるスキンがあります。

スキンの詳細については、741 ページ、第 20 章の「スキンの使用」を参照してください。

実行時におけるコンポーネントの外観の変更

次のようなコンポーネントプロパティ、スタイル、または ActionScript メソッドを使用して、実行時にコンポーネントの外観、サイズ、または位置を変更できます。

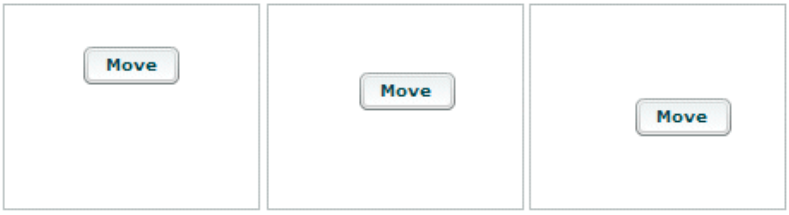
- `x` および `y`
- `width` および `height`
- `setStyle(stylename, value)` を使用したスタイル

コンポーネントの x および y プロパティは、コンポーネントが絶対配置を使用するコンテナに含まれている場合にのみ設定できます。絶対配置を使用するコンテナとは、Canvas コンテナと、`layout` プロパティが `absolute` に設定されている Application コンテナまたは Panel コンテナを指します。それ以外のコンテナでは、レイアウト規則を使用して自動レイアウトが実行され、子の x および y プロパティが設定されます。

たとえば次の例のように、Button コントロールの click イベントに応答し、x および y プロパティを使用して Button コントロールの位置を下方向および右方向にそれぞれ 10 ピクセルずつ移動することができます。

```
<?xml version="1.0"?>
<!-- components\ButtonMove.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="150"
    height="120"
    layout="absolute">

    <mx:Script>
        <![CDATA[
            public function moveButton():void {
                myButton.x += 15;
                myButton.y += 15;
            }
        ]]>
    </mx:Script>

    <mx:Button id="myButton"
        x="15"
        y="15"
        label="Move"
        click="moveButton();"/>
</mx:Application>
```

このアプリケーションの初期イメージと、ユーザーがボタンを 1 回および 2 回クリックしたときの結果は次のようになります。



このアプリケーションでは、他のコンポーネントを考慮することなく Button コントロールを移動できます。しかし、アプリケーションに含まれている複数のコンポーネントのうちの 1 つのコンポーネントを移動したり、コンテナに含まれている複数の子のうちの 1 つの子を移動すると、コンポーネントが重なり合ったり、アプリケーションのレイアウトが影響を受けたりする可能性があります。したがって、コンテナのレイアウトを実行時に変更する場合は注意が必要です。

コンポーネントの `width` および `height` プロパティは、コンテナの種類に関係なく設定できます。次の例では、ユーザーが Button コントロールをクリックするたびに、Button コントロールの `width` と `height` の値を 15 ピクセルずつ増やしています。

```
<?xml version="1.0"?>
<!-- components\ButtonSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="150"
    height="150">
```

```
    <mx:Script>
        <![CDATA[
            public function resizeButton():void {
                myButton.height = myButton.height + 15;
                myButton.width = myButton.width + 15;
            }
        ]]>
    </mx:Script>

    <mx:VBox
        borderStyle="solid"
        height="80"
        width="100" >

        <mx:Button id="myButton"
            label="Resize"
            click="resizeButton();"/>
    </mx:VBox>
</mx:Application>
```

ユーザーがボタンをクリックすると、次のような結果になります。

Button コントロールを格納しているコンテナが絶対配置を使用していない場合は、Button コントロールの新しいサイズに基づいて、コンテナの子の位置が変更されます。Canvas コンテナと、`layout="absolute"` が設定された Panel および Application コンテナでは自動レイアウトは行われないので、1つの子のサイズを変更しても、他の子の位置やサイズは変更されません。

メモ

`width` と `height` に格納される値は、その値が元々固定値で設定されたものか、パーセント値で設定されたものか、あるいは値が設定されていなかったかには関係なく、常にピクセル単位になります。

コンポーネントの拡張

Flex 2 SDK で既存のコンポーネントを拡張してコンポーネントを作成するには、いくつかの方法があります。コンポーネントを拡張することで、コンポーネントに新しいプロパティやメソッドを追加できます。

たとえば、次の MXML コンポーネントは "MyComboBox.mxml" ファイルに定義されているコンポーネントで、標準の ComboBox コントロールを拡張し、ニューイングランド地方の州の略称で初期化されます。

```
<?xml version="1.0"?>
<!-- components\myComponents\MyComboBox.mxml -->
<mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:dataProvider>
        <mx:String>CT</mx:String>
        <mx:String>MA</mx:String>
        <mx:String>ME</mx:String>
        <mx:String>NH</mx:String>
        <mx:String>RI</mx:String>
        <mx:String>VT</mx:String>
    </mx:dataProvider>
</mx:ComboBox>
```

この例はまた、省略可能なコーディングの一例も示しています。Flex では dataProvider は配列と想定されているため、<mx:Array> タグを指定する必要はありません。

新しいコンポーネントを作成したら、そのファイル名を MXML タグ名として指定することで、アプリケーションの任意の場所で新しいコンポーネントを使用できます。次に例を示します。

```
<?xml version="1.0"?>
<!-- components\MainMyComboBox.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:MyComps="myComponents.*"
    width="150"
    height="150">

    <MyComps:MyComboBox id="stateNames"/>
</mx:Application>
```

この例では、新しいコンポーネントがアプリケーションファイルと同じディレクトリに存在するため、アスタリスク (*) で示されたローカル名前空間を MyComps 識別子にマップしています。

Flex では、次の方法のいずれかを使用してカスタムコンポーネントを作成できます。どの方法を選ぶかは、アプリケーションとコンポーネントの要件によって異なります。

- コンポーネントを MXML ファイルとして作成し、それをカスタムタグとして他の MXML ファイル内で使用する。MXML コンポーネントは、既存のコンポーネントの拡張、特に既存コンポーネントのビヘイビアの変更や既存コンポーネントへの基本機能の追加が簡単に行えるようになっています。
- UIComponent クラスまたはそのいずれかのサブクラスをサブクラス化することによってコンポーネントを ActionScript ファイルとして作成し、そのクラスをカスタムタグとして使用する。ActionScript コンポーネントは、ビジュアルまたは非ビジュアルコンポーネントを新たに作成するための強力なツールを備えています。

カスタムコンポーネントの作成の詳細については、『Flex 2 コンポーネントの作成と拡張』を参照してください。

第 7 章

データプロバイダおよびコレクションの使用

7

Adobe Flex コントロールの中には、配列や XML オブジェクトなどのデータプロバイダから入力を受け取るものがあります。たとえば、Tree コントロールはデータプロバイダからデータを読み取ることによって、ツリー構造と各ツリーノードに割り当てられたデータを定義します。Flex コントロールでは、データプロバイダを表すためにコレクションクラスを使用します。コレクションにはオブジェクトのグループが含まれており、コレクション内のアイテムのアクセス、ソート、フィルタ、および変更を行うための一連のメソッドを提供しています。

このトピックでは、データプロバイダとコレクションクラスについて説明します。また、コントロールでのデータプロバイダの使用について概説するとともに、コレクションクラスを使用してデータにアクセスし、操作する方法について説明します。データプロバイダを使用する Flex コントロールについては、411 ページ、第 12 章の「データ駆動型コントロールの使用」や 381 ページ、第 11 章の「メニューベースのコントロールの使用」などに説明があります。

目次

データプロバイダおよびコレクションについて

多くのコントロールやコンテナがメニューなどのコントロールの表示、ユーザー操作、または構成のためにデータを必要とします。このデータを提供するには、データプロバイダおよびコレクションの概念と、コレクションのしくみを理解する必要があります。

データプロバイダについて

リストコントロールなどの一部の Flex フレームワーク コンポーネントは、" データプロバイダ " からのデータを表します。データプロバイダとは、コントロールによって必要とされるデータを含むオブジェクトのことです。たとえば、Tree コントロールのデータプロバイダは、ツリー構造と各ツリーノードに割り当てられるデータを決定し、ComboBox コントロールのデータプロバイダは、コントロールのドロップダウンリストの項目を決定します。標準コントロールの中にも、ColorPicker や MenuBar のようにデータプロバイダからデータを取得するものが数多くあります。アプリケーションデータを表示するコントロールのことを、"データプロバイダコントロール" と呼ぶことがあります。

データプロバイダを使用するコンポーネントは次のとおりです。

- Repeater コンポーネント
- LineChart コントロールなどのチャートコントロール
- ColorPicker コントロール
- ComboBox コントロール
- DataGrid コントロール
- DateField コントロール
- HorizontalList コントロール
- List コントロール
- Menu コントロール
- MenuBar コントロール
- PopUpMenuButton コントロール
- TileList コントロール
- Tree コントロール
- ButtonBar コントロール
- LinkBar コントロール
- TabBar コントロール
- ToggleButtonBar コントロール

最も単純なデータプロバイダとしては、ストリングまたはオブジェクトの配列が考えられます。たとえば、静的な ComboBox コントロールを定義するために次のコードを使用できます。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
        [Bindable]
        public var myArray:Array = ["AL", "AK", "AR"];
    ]]>
  </mx:Script>
  <mx:ComboBox id="myCBO" dataProvider="{myArray}"/>
</mx:Application>
```

ただし、Array や Object などの生データオブジェクトをデータプロバイダとして使用する場合は、次のような制限があります。

- 変化するデータを使用する場合、生オブジェクトでは十分でないことが少なくありません。その理由は、コントロールが基本オブジェクトの変更通知を受信しないためです。そのため、コントロールはアプリケーションの他の部分の変更によって再描画されるか、またはデータプロバイダが再割り当てされない限り、更新されません。再描画やデータプロバイダの再割り当てが起こると、更新された Array から再びデータが取得されます。前の例では、プログラムによって Array に州を追加しても、コントロールの dataProvider プロパティを再割り当てしない限り、追加した州は ComboBox コントロールに表示されません。
- 生オブジェクトは、データのアクセス、ソート、またはフィルタのための高度なツールを提供していません。たとえば、データプロバイダとして Array を使用する場合、データを操作するためにネイティブの Adobe Flash Array メソッドを使用する必要があります。

Flex は、データの同期を保証するとともに、より簡単で高度なデータアクセスおよび操作ツールを提供するため、" コレクション " メカニズムを備えています。また、コレクションを使用すると、異なるタイプのデータへのアクセスとその管理を一貫したインターフェイスを通じて行うことができます。コレクションの詳細については、157 ページの「コレクションについて」を参照してください。

個々のコントロールの詳細については、『Adobe Flex 2 リファレンスガイド』のコントロールのページを参照してください。データプロバイダベースのコントロールを使用したプログラミングについては、411 ページ、第 12 章の「データ駆動型コントロールの使用」を参照してください。

データプロバイダの種類

Flex フレームワークでサポートされている基本的なデータプロバイダの種類は次の 2 つです。

リニアまたはリストデータプロバイダは、それぞれが同じ構造を持ついくつかのオブジェクトから構成されるフラットなデータです。これは通常、1 次元の配列か、またはそのような配列から派生したオブジェクト (ActionScript オブジェクトグラフなど) です。また、XMLListCollection オブジェクトをリニアデータプロバイダとして使用することもできます。

リストデータプロバイダはすべてのデータプロバイダコントロールで使用できますが、通常は Tree コントロールと大部分のメニューコントロールのインスタンスを除くデータプロバイダコントロールで使用します。データプロバイダの内容が動的に変更される場合は一般に、ArrayCollection クラスと IList または ICollectionView インターフェイスのメソッドとプロパティを使用して、これらのデータプロバイダの表現と操作を行います。

階層データプロバイダは、カスケード構造のデータ (非対称データの場合が多い) から成ります。データのソースには XML オブジェクトがよく使われますが、汎用 Object または具象クラスをソースとすることもできます。階層データプロバイダは通常、階層データを表示するために設計された次のような Flex コントロールで使用します。

- Tree
- Menu、MenuBar、PopUpMenuButton

また、階層データプロバイダからデータを抽出し、リニアデータを受け取る List や DataGrid などのコントロールで使用することもできます。

階層データプロバイダを使用すると、ツリーやカスケードメニューのレイアウトに合わせてデータ構造を定義できます。たとえば、ツリーには通常、ルートノードの下にブランチノードまたはリーフ子ノードがあります。ブランチノードの下には他の子ブランチノードまたはリーフノードがありますが、リーフノードはツリーの終点です。

Flex 階層データコントロールは、階層データプロバイダにアクセスして操作するために " データ記述子 " インターフェイスを使用します。Flex フレームワークには、必要なインターフェイスを実装した DefaultDataDescriptor クラスが用意されています。データプロバイダが、デフォルトデータ記述子によって必要とされる構造に適合しない場合は、独自のデータ記述子クラスを作成して必要なインターフェイスを実装できます。

動的な階層データへのアクセスと操作には、ArrayCollection クラス、XMLListCollection クラス、ICollectionView インターフェイス、または IList インターフェイスを使用します。

階層データプロバイダの使用の詳細については、185 ページの「階層データプロバイダの使用」を参照してください。

データプロバイダと uid プロパティ

Flex データ駆動型コントロール (List クラスのサブクラスであるすべてのコントロールなど) は、一意の識別子 (uid) を使用してデータプロバイダのアイテムを追跡します。Flex では、uid を自動的に作成し管理できます。ただし、IUID インターフェイスを実装して独自の uid プロパティを提供しなければならない場合や、独自の uid プロパティを提供することで処理効率が向上する場合があります。

メモ

Flex はオブジェクト (たとえば ArrayCollection のアイテムなど) の UID を作成するとき、アイテムの `mx_internal_uid` プロパティとして UID を追加します。`mx_internal_uid` プロパティは、バインド可能プロパティを持たない動的なオブジェクトに対して作成されます。`mx_internal_uid` プロパティが作成されないようにするには、オブジェクトクラスで、少なくとも 1 つのプロパティに `[Bindable]` メタデータタグを付けるか、IUID インターフェイスを実装するか、`uid` プロパティに値を設定します。

複数の異なるオブジェクトを同一と見なす必要がある場合は、IUID インターフェイスを実装して、複数のオブジェクトに同じ uid を割り当てられるようにする必要があります。IUID インターフェイスを実装しなければならない代表的な例は、アプリケーションでウィンドウページングコレクションを使用する場合です。ウィンドウページングコレクション内をカーソルが移動すると、サーバーからの特定のアイテムの取得とメモリからの解放が何度も行われます。アイテムがメモリに取り込まれるたびに、そのアイテムを表す新しいオブジェクトが作成されます。アイテムが等しいかどうかを比較する場合は、同じアイテムを表すすべてのオブジェクトが Flex によって同じものと見なされる必要があります。

IUID インターフェイスを実装しなければならないケースより、そうすることで処理効率が向上するケースの方がよくあります。原則として、データプロバイダエレメントが動的クラスのメンバーである場合、IUID インターフェイスは実装しません。これらのクラスについては uid プロパティを自動的に作成できます。しかしそれでも非効率的な面があるため、処理効率が特に重視される場合は、IUID インターフェイスの実装を検討してもかまいません。

その他のケースでは、Flex はディクショナリメカニズムを使用して uid を管理しますが、これは独自の UID を提供する場合ほど効率的ではありません。

Object クラスと Array クラスは動的なので、データプロバイダのアイテムがこれらのクラスに属する場合、通常はデータプロバイダに対して特別なことを行う必要はありません。ただし、データプロバイダのメンバーが独自に定義したカスタムクラスに属する場合は、IUID インターフェイスの実装を検討してください。

IUID インターフェイスには、クラスメンバーの一意の識別子を表す `uid` プロパティが 1 つ含まれているだけで、メソッドはありません。Flex には、疑似乱数生成器を使用して標準の GUID 形式に準拠した識別子を作成する UIDUtil クラスが用意されています。この識別子は、ユニバーサルに一意であることは保証されていませんが、作成したクラスの全メンバーの中では一意になります。UIDUtil クラスを使用するクラス、たとえば姓、名、id の各フィールドを持つ Person クラスを実装するには、次のパターンを使用できます。

```
package {
  import mx.core.IUID;
  import mx.utils.UIDUtil;

  [Bindable]
  public class Person implements IUID {
    public var id:String;
    public var firstName:String;
    public var lastName:String;
    private var _uid:String;

    public function Person() {
      _uid = createUID();
    }

    public function get uid():String {
      return _uid;
    }

    public function set uid(value:String):void {
      // コンストラクタによって uid が作成されるので、何もしない
    }
  }
}
```

オブジェクトに従業員 ID などの一意のフィールドが含まれている場合には、UIDUtil クラスを使用する必要はありません。uid プロパティの値はデータプロバイダの中だけでオブジェクトを一意に識別できればよいので、この場合は従業員 ID を uid プロパティとして使用できます。次の例ではこの方法を採っています。

```
package
{
  import mx.core.IUID;

  [Bindable]
  public class Person implements IUID {
    public var employee_id:String;
    public var firstName:String;
    public var lastName:String;

    public function get uid(): String {
      return employee_id;
    }

    public function set uid(value: String): void {
      employee_id=value;
    }
  }
}
```

メモ

オブジェクトのクローン作成では UID は管理されず、UID と関連付けられることもありません。したがって、内部 UID を持つ対象のクローンを作成する場合、その内部 UID を変更する必要があります。UID は、ダイナミックオブジェクトの場合のみ、mx_internal_uid に格納されます。IUID を実装するデータクラスのインスタンスは UID を .uid プロパティに格納するので、クローンの作成後、このプロパティを変更する必要があります。
ただし、データプロバイダでは IUID を実装する必要はありません。データプロバイダが IUID を実装しないデータクラスのインスタンスでも、ダイナミックオブジェクトではない場合、クローン技術は適切に動作しますが、ほとんどのデータプロバイダアイテムは、特にそれらのオブジェクトのクローン作成が必須ではない場合には、IUID を実装する必要があります。

コレクションについて

" コレクション " は、データプロバイダのデータへのアクセスとその表現に対して一貫した方法を提供します。コレクションでは、Flex コンポーネントとそこに格納されるデータとの間で一定レベルの抽象関係が作成されます。コレクションのインターフェイスと実装は次の機能を提供します。

- 基になるデータが変化したときに、コントロールが適切に更新されることを保証します。非コレクションデータプロバイダが変更されてもコントロールは更新されません。この場合は、次回リフレッシュされたときにようやくコントロールが更新され、新しいデータが反映されます。データプロバイダがコレクションの場合、コントロールはコレクションが変更された直後に更新されます。
- リモートデータプロバイダから到達するページングデータを処理するためのメカニズムを提供します。リモートデータプロバイダからのデータは最初から使用できるとは限らず、すべてが到達するのに時間がかかる場合があります。
- 生データプロバイダオブジェクトにより提供される処理から独立した一貫したデータ処理セットを提供します。たとえば、IList インターフェイスを実装したカスタムクラスで、基になるデータが配列やオブジェクトなどの何に格納されているかにかかわらず、コレクションのインデックスを使用してオブジェクトの挿入と削除を行うことができます。
- ICollectionView インターフェイスに準拠するコレクションは、ソートしたデータや開発者の提供するメソッドによってフィルタしたデータを表す、特定の " ビュー " を提供します。こうした制限は、コレクションビューにのみ適用されます。基になるデータが変更されることはありません。
- 1 つのコレクションを使用して、同じデータプロバイダから複数のコンポーネントにデータを格納できます。
- コレクションを使用することで、コンポーネントのデータプロバイダを実行時に切り替えることができます。また、データプロバイダを変更して、そのデータプロバイダを使用するすべてのコンポーネントに変更を反映することもできます。
- コレクションのメソッドを使用して、基になるデータオブジェクト内のデータにアクセスできます。

メモ

Array などの生オブジェクトをコントロールのデータプロバイダとして使用する場合、そのオブジェクトは自動的にコレクションラッパーにラップされます。この場合、コントロールは生オブジェクトに直接加えられた変更を自動的に検出できません。たとえば、配列の長さを変更してもコントロールは更新されません。ただし、オブジェクトプロキシ、リスナーインターフェイス、または itemUpdated プロパティを使用して、特定の変更をビューに通知することは可能です。

コレクションのインターフェイス

Flex フレームワークコレクションモデルは、次のインターフェイスを使用して、コレクションによるデータの表現方法とデータへのアクセス方法を定義します。

インターフェイス	説明
IList	順序に従って構成されたアイテムの直接的な表現。このインターフェイスは、データプロバイダ内に存在するのと同じ順序でデータプロバイダのデータを提示し、インデックスに基づいたアクセスメソッドと操作メソッドを提供します。IList クラスには、ソート、フィルタ、またはカーソルの機能はありません。
ICollectionView	アイテムコレクションのビュー。ビューを変更することで、データをソートして表示したり、フィルタ関数を適用してデータプロバイダ内のアイテムのサブセットを表示したりできます。このインターフェイスを実装するクラスは、基になるコレクションとして IList インターフェイスを使用できます。このインターフェイスは、アイテムへのアクセスのために、IViewCursor オブジェクトへのアクセスを提供します。
IViewCursor	ICollectionView オブジェクトを双方向に列挙します。" ビューカーソル " は、検索、シーク、およびブックマークの各機能を提供します。アイテムを挿入および削除することにより基になるデータ (およびビュー) を変更できます。

IList インターフェイスと ICollectionView インターフェイスは、データにアクセスし、それを変更するための代替手段を提供します。IList インターフェイスの方がシンプルで、その追加、設定、取得、削除の各操作はリニアデータに対して直接作用します。

ICollectionView インターフェイス (" コレクションビュー " とも呼ばれます) は、IList インターフェイスよりも複雑な処理セットを提供し、基になるデータがリニアに構成されていない場合に適切です。ただし、そのデータアクセス手法は IList インターフェイスよりも複雑になるため、リニアコレクションへの簡単なインデックスアクセスのみが必要な場合は IList インターフェイスを使用してください。コレクションビューでは、ソート処理またはフィルタ処理に従って、基になるデータのサブセットを表すことができます。

標準の Flex コレクションクラスである ArrayCollection および XMLListCollection は、コレクションビュー (ICollectionView) インターフェイスおよび IList インターフェイスの両方を実装しています。

IList インターフェイスのメソッドとプロパティの使用の詳細については、164 ページの「IList インターフェイスのメソッドとプロパティの使用」を参照してください。ICollectionView インターフェイスのメソッドとプロパティ、およびこのインターフェイスが公開する IViewCursor オブジェクトの使用の詳細については、166 ページの「ICollectionView インターフェイスのメソッドとプロパティの使用」を参照してください。

コレクションクラス

mx.collections パッケージに含まれるパブリッククラスを次の表に示します。この表には、定数クラス、イベントおよびエラークラスは含まれません。コレクション関連クラスの詳細については、『Adobe Flex 2 リファレンスガイド』の collections および collections.errors パッケージと CollectionEvent および CollectionEventKind クラスを参照してください。

クラス	説明
ArrayCollection	配列データプロバイダで使用するための IList インターフェイスと ICollectionView インターフェイスを実装します。
XMLListCollection	IList インターフェイスと ICollectionView インターフェイスに加えて、XML データプロバイダで使用するための XMLList メソッドのサブセットを実装します。
CursorBookmark	ICollectionView インスタンス内の IViewCursor インスタンスの位置を表します。CursorBookmark オブジェクト内にビューカーソルの位置を保存しておき、後でこのオブジェクトを使用してビューカーソルをその位置に戻すことができます。
Sort	ICollectionView インスタンスをソートするのに必要な情報およびメソッドを提供します。
SortField	ICollectionView インスタンス内でのデータのソート方法に特定のフィールドがどのように影響するかを決定するプロパティおよびメソッドを提供します。
ItemResponder	このクラスはデータソースがリモートの場合のみ使用されます。要求されたデータがまだ使用できないケースを処理します。
ListCollectionView	IList インターフェイスを実装するオブジェクトを ICollectionView インターフェイスに適合させます。これにより、IList または ICollectionView を想定するすべての対象に対してオブジェクトを渡すことができるようになります。このクラスは、ArrayCollection クラスと XMLListCollection クラスのスーパークラスです。

MXML アプリケーションでのデータプロバイダの指定

Flex フレームワークでは、さまざまな方法でデータプロバイダを指定し、それにアクセスできます。たとえば、MXML コントロールの場合と同様に、コントロールの <mx:dataProvider> プロパティ子タグを使用してデータプロバイダを定義できます。また、ActionScript の割り当てによってデータプロバイダを定義することもできます。どのアクセス手法も、次の 3 種類の主要パターンのいずれかに属します。

- Array などの生データオブジェクトの使用。
- コレクションクラスオブジェクトの直接の使用。このパターンは、オブジェクトの再使用可能性がそれほど重要でないローカルコレクションで特に役立ちます。
- インターフェイスの使用。このパターンでは、基になる実装からの独立性が最大になります。

生データオブジェクトのデータプロバイダとしての使用

データが静的な場合はいつでも、Array オブジェクトなどの生データオブジェクトを直接データプロバイダとして使用できます。たとえば、米国郵政省で定められている州識別子の静的リストを格納した配列を使用することが可能です。データオブジェクトの内容がユーザー入力やプログラムの処理などによって動的に変わる場合は、そのオブジェクトをコントロールの dataProvider プロパティとして直接使用しないでください。

HTTPService または WebService によって返された結果が Array である場合があります。そのデータを読み取り専用として扱う場合は、Array を直接データプロバイダとして使用できます。RemoteObject クラスは通常、ArrayCollection を返します。

ただし、リストコントロールは内部で配列データプロバイダをコレクションに変換するため、Array をデータプロバイダとして使用することによるパフォーマンス上の利点は得られません。Array を複数のコレクションに渡す場合は、データを受信したときに Array をコレクションに変換し、そのコレクションをコントロールに渡す方が効率的です。

生データオブジェクトを使用するには、オブジェクトをコントロールの dataProvider コントロールに割り当てるか、またはコントロールの MXML タグの本体で直接オブジェクトを定義します。例については、152 ページの「データプロバイダについて」を参照してください。

コレクションオブジェクトの直接の使用

ICollectionView インターフェイスまたは IList インターフェイスを実装するオブジェクトをデータプロバイダとして直接 MXML コントロールで使用するには、そのオブジェクトをコンポーネントの dataProvider プロパティに割り当てます。この手法はインターフェイスを使用するより直接的で、データプロバイダが常に特定のコレクションクラスに属する場合に適しています。

リストコントロールの場合は通常、データプロバイダとして ArrayCollection オブジェクトを使用し、Array を使用して ArrayCollection にデータを格納します。次の例は、ComboBox コントロールのデータを指定するデータプロバイダを示しています。

```
<?xml version="1.0"?>
<!-- dpcontrols\ArrayCollectionInComboBox.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:ComboBox id="myCB">
        <mx:ArrayCollection id="stateArray">
            <mx:Object label="AL" data="Montgomery"/>
            <mx:Object label="AK" data="Juneau"/>
            <mx:Object label="AR" data="Little Rock"/>
        </mx:ArrayCollection>
    </mx:ComboBox>
</mx:Application>
```

この例では、ComboBox の dataProvider デフォルトプロパティで ArrayCollection オブジェクトを定義しています。この ArrayCollection は source デフォルトプロパティが Object の配列であり、その各 Object がラベルフィールドとデータフィールドを持ちます。ArrayCollection の source プロパティは Array オブジェクトを受け取るため、この例では <mx:Array> タグを使用する必要はありません。

次の例では、ActionScript を使用して ArrayCollection オブジェクトを宣言および作成しています。

```
<?xml version="1.0"?>
<!-- dpcontrols\ArrayCollectionInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initData()">
    <mx:Script>
        <![CDATA[
            import mx.collections.*;
            [Bindable]
            public var stateArray:ArrayCollection;

            public function initData():void {
                stateArray=new ArrayCollection(
                [{label:"AL", data:"Montgomery"},
                {label:"AK", data:"Juneau"},
                {label:"AR", data:"Little Rock"}]);
            }
        ]]>
    </mx:Script>
```

```
    <mx:ComboBox id="myComboBox" dataProvider="{stateArray}"/>
</mx:Application>
```

ComboBox コントロールを定義した後、ComboBox コントロールの dataProvider プロパティは基になるデータプロバイダのソースオブジェクトを表すコレクションへのアクセスを提供します。そのため、このプロパティを使用してデータプロバイダを変更できます。たとえば、前のコードに次のボタンを追加すると、このボタンをクリックして ComboBox 内のリストの末尾にアリゾナ州のラベルとデータを追加できます。

```
  <mx:Button label="Add AZ"
  click="myComboBox.dataProvider.addItem({'label':'AZ','data':'Phoenix'});"/>
```

ただし通常は、次の例のように、コレクションを直接参照することをお勧めします。

```
<mx:Button label="Add AZ" click="stateArray.addItem({'label':'AZ',
  'data':'Phoenix'});"/>
```

コレクションのインターフェイスを使用したデータプロバイダへのアクセス

特定のコレクションクラスによって常にコントロールのデータプロバイダを表せることがわかっている場合は、「コレクションオブジェクトの直接の使用」で示したように、クラスを直接使用します。コードで基になるコレクション型を複数使用する場合は、次の例のように、アプリケーションコード内で ICollectionView インターフェイスを使用します。

```
public var myICV:ICollectionView = indeterminateCollection;
<mx:ComboBox id="cb1" dataProvider="{myICV}" initialize="sortICV()"/>
```

こうすると、必要に応じてインターフェイスを操作することにより、表示するデータを選択したり、基になるデータソース内のデータを取得して変更したりできます。コレクションのインターフェイスにより提供される手法の詳細については、164 ページの「IList インターフェイスのメソッドとプロパティの使用」および 166 ページの「ICollectionView インターフェイスのメソッドとプロパティの使用」を参照してください。

例 : 簡単なデータプロバイダの使用

次のサンプルコードは、基本的なコレクションクラスを使用して、コントロールで使用する Array を表し、それを操作する方法を示しています。この例では、次の機能を示しています。

- ArrayCollection を使用して配列内のデータを表す。
- ArrayCollection をソートする。
- ArrayCollection にデータを挿入する。

この例では、データの挿入操作が Array および Array の ArrayCollection での表現に対してどのように影響するかも示しています。

```
<?xml version="1.0"?>
<!-- dpcontrols\SimpleDP.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
initialize="sortAC()">
    <mx:Script>
        <![CDATA[
            import mx.collections.*;

            // Function to sort the ArrayCollection in descending order.
            public function sortAC():void {
                var sortA:Sort = new Sort();
                sortA.fields=[new SortField("label")];
                myAC.sort=sortA;
                //Refresh the collection view to show the sort.
                myAC.refresh();
            }
            // Function to add an item in the ArrayCollection.
            // Data added to the view is also added to the underlying Array.
            // The ArrayCollection must be sorted for this to work.
            public function addItemToMyAC():void {
                myAC.addItem({label:"MD", data:"Annapolis"});
            }
        ]]>
    </mx:Script>

    <!-- An ArrayCollection with an array of objects -->
    <mx:ArrayCollection id="myAC">
        <!-- Use an mx:Array tag to associate an id with the array. -->
        <mx:Array id="myArray">
            <mx:Object label="MI" data="Lansing"/>
            <mx:Object label="MO" data="Jefferson City"/>
            <mx:Object label="MA" data="Boston"/>
            <mx:Object label="MT" data="Helena"/>
            <mx:Object label="ME" data="Augusta"/>
            <mx:Object label="MS" data="Jackson"/>
            <mx:Object label="MN" data="Saint Paul"/>
        </mx:Array>
    </mx:ArrayCollection>

    <mx:HBox width="100%">
        <!-- A ComboBox populated by the underlying Array object.
            This control shows that Array retains its original order
            and MD is inserted at the end of the Array. -->
        <mx:ComboBox id="cb2" rowCount="10" dataProvider="{myArray}"/>
        <!-- A ComboBox populated by the collection view of the Array. -->
        <mx:ComboBox id="cb1" rowCount="10" dataProvider="{myAC}"/>
        <mx:Button id="b1" label="Add MD" click="addItemToMyAC();"/>
    </mx:HBox>
</mx:Application>
```

IList インターフェイスのメソッドとプロパティの使用

IList インターフェイスは、リニアデータにインデックスを使用してアクセスするための単純な手段を提供します。このインターフェイスは、基になるデータプロバイダの直接的な表現を提供します。したがって、コレクションを変更するすべての処理が、同じようにデータプロバイダを変更します。つまり、コレクションの 3 番目のアイテムを挿入した場合、基になるデータソースでもそれは 3 番目のアイテムにもなります。

メモ	ICollectionView インターフェイスを使用してコレクションをソートまたはフィルタする場合は、IList インターフェイスを使用してデータを操作しないでください。そうすると、結果が不定になります。

このインターフェイスには、次のことを実行できるプロパティとメソッドが含まれています。

- コレクション内の特定のインデックスで、アイテムの取得、設定、追加、または削除を実行する。
- コレクションの末尾にアイテムを追加する。
- コレクション内の特定のアイテムのインデックスを取得する。
- コレクション内のすべてのアイテムを削除する。
- コレクションの長さを取得する。

IList インターフェイスのメソッドとプロパティは、次のクラスまたはプロパティで直接使用できます。

- ArrayCollection クラス
- XMLList クラス
- 標準 Flex コントロールの `dataProvider` プロパティ

次のサンプルコードでは、ArrayCollection オブジェクトと、その IList インターフェイスメソッドの実装を使用して、ComboBox コントロールにエレメントの配列を表示します。IList インターフェイスのメソッドを使用して、複数のフィールドを持つオブジェクトの ArrayCollection を管理する例については、183 ページの「例 : DataGrid コントロール内のデータの変更」を参照してください。

次の例では、初期状態の Array データソースは次のエレメントで構成されています。

```
"AZ", "MA", "MZ", "MN", "MO", "MS"
```

Button コントロールをクリックすると、IList インターフェイスの `length` プロパティといくつかのメソッドを使用して次のことが行われます。

- 配列内のデータと ComboBox コントロール内の表示データを、M で始まる州の米国郵便番号を含む正しいアルファベット順リストに変更します。

  ```
  MA, ME, MI, MN, MO, MS, MT
  ```

- 実行したタスクに関する情報と配列の変更結果を TextArea コントロールに表示します。

このコードには、データプロバイダに対する変更を説明するコメントが含まれています。

```
<?xml version="1.0"?>
<!-- dpcontrols\UseIList.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initData()">

    <mx:Script>
        <![CDATA[
            import mx.collections.*;

            // The data provider is an Array of Strings
            public var myArray:Array = ["AZ", "MA", "MZ", "MN", "MO", "MS"];
            // Declare an ArrayCollection that represents the Array.
            [Bindable]
            public var myAC:ArrayCollection;

            //Initialize the ArrayCollection.
            public function initData():void {
                myAC = new ArrayCollection(myArray);
            }

            // The function to change the collection, and therefore
            // the Array.
            public function changeCollection():void {
                // Get the original collection length.
                var oldLength:int=myAC.length;

                // Remove the invalid first item, AZ.
                var removedItem:String=String(myAC.removeItemAt(0));
                // Add ME as the second item. (ILists used 0-based indexing.)
                myAC.addItemAt("ME", 1);
                // Add MT at the end of the Array and collection.
                myAC.addItem("MT");
                // Change the third item from MZ to MI.
                myAC.setItemAt("MI", 2);
                // Get the updated collection length.
                var newLength:int=myAC.length;
                // Get the index of the item with the value ME.
                var addedItemIndex:int=myAC.getItemIndex("ME");
                // Get the fifth item in the collection.
                var index4Item:String=String(myAC.getItemAt(4));

                // Display the information in the TextArea control.
                ta1.text="Start Length: " + oldLength + ". New Length: " +
                    newLength;
                ta1.text+=".\nRemoved " + removedItem;
                ta1.text+=".\nAdded ME at index " + addedItemIndex;
                ta1.text+=".\nThe item at index 4 is " + index4Item + ".";
                // Show that the base Array has been changed.
                ta1.text+="\nThe base Array is: " + myArray.join();
```

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

    <mx:ComboBox id="myCB" rowCount="7" dataProvider="{myAC}"/>
    <mx:TextArea id="ta1" height="75" width="300"/>
    <mx:Button label="rearrange list" click="changeCollection();"/>
</mx:Application>
```

ICollectionView インターフェイスのメソッドとプロパティの使用

ICollectionView インターフェイスは、基になるデータソースの " ビュー " をアイテムのコレクションとして表します。このインターフェイスの主な機能は次のとおりです。

- ビューを変更することで、基になるデータを変更することなく、データをソートして表示したり、データプロバイダ内のアイテムのサブセットを表示したりできます。詳細については、167 ページの「表示するデータのソートとフィルタ」を参照してください。
- カーソルを使用してコレクションデータにアクセスできます。カーソルを使用すると、コレクション内の移動、ブックマークを使用したコレクション内の特定の位置の保存、コレクション内のアイテムの挿入と削除が可能です。コレクション内のアイテムを挿入または削除すると、その結果として、基になるデータソースも変更されます。詳細については、170 ページの「IViewCursor インターフェイスの使用」を参照してください。
- ICollectionView オブジェクトを使用して、最初のうちは使用できない (つまり、データセットの各部分が異なるタイミングで使用可能になる) リモートデータを表すことができます。詳細については、180 ページの「コレクションの変更通知の使用」および 200 ページの「リモートデータプロバイダの使用」を参照してください。

ICollectionView インターフェイスのメソッドとプロパティは、次のクラスまたはプロパティで直接使用できます。

- ArrayCollection クラス
- XMLListCollection クラス
- NavBar クラスのサブクラス (ButtonBar、LinkBar、TabBar、ToggleButtonBar) を除くすべての標準 Flex コントロールの `dataProvider` プロパティ

以降のセクションでは、リストコレクションを使用した基本的な ICollectionView の処理について説明しますが、階層コレクションについても同様の処理を適用できる場合があります。

表示するデータのソートとフィルタ

ICollectionView インターフェイスを使用して、データプロバイダ内のデータのソートとフィルタを実行できます。こうすると、コレクションによって表されるビューが、基になるデータの並べ替えられたサブセットになります。この操作はデータプロバイダの内容には影響を与えず、コレクションビューで表されるサブセットのみを変更します。これは結果的に、コレクションを使用するコントロールの表示を変更することになります。

ソート

Sort クラスを使用すると、コレクション内のデータをソートできます。データのソートで使用する複数のフィールド、ソート結果の重複の除去、およびソートの順序付けに使用するカスタム比較関数を指定できます。また、Sort クラスを使用してコレクション内のアイテムを検索することもできます。Sort クラスを作成したとき、またはそのプロパティを変更したときは、結果を表示するコレクションに対して `refresh()` メソッドを呼び出す必要があります。

SortField クラスは、ソートで使用するフィールドを指定するために使用します。SortField オブジェクトを作成し、それを Sort クラスオブジェクトの `fields` 配列に設定します。

次の例は、コレクションをソートする関数を示します。この例の myAC は、ラベルフィールドと名前フィールドを含むオブジェクトの Array のコレクションビューです。第 1 ソートフィールドは area フィールドで、大文字と小文字を区別しない降順ソートを指定し、第 2 ソートフィールドは label フィールドで、大文字と小文字を区別した昇順ソートを指定しています。このコードは、コレクションを特定の順序でソートして表示するコントロールの initialize イベントハンドラとして呼び出すことができます。

```
// ICollectionView を降順にソートする
public function sortAC():void {
   // ソートオブジェクトを作成する
   var sortA:Sort = new Sort();
   // area フィールドでソートしてから、label フィールドでソートする
   // 2 番目のパラメータは、大文字と小文字を区別せずにソートすることを指定する
   // 3 番目のパラメータ (true) は、降順ソートを指定する
   sortA.fields=[new SortField("area", true, true),
                 new SortField("label")];
   myAC.sort=sortA;
   // コレクションビューを更新してソート結果を表示する
   myAC.refresh();
}
```

フィルタ

フィルタ関数は、ICollection ビューをデータプロバイダソースのサブセットに制限する場合に使用します。この関数は、コレクションのアイテムに対応する Object パラメータを 1 つ受け取り、コレクションビューにそのアイテムを含めるかどうかを指定する Boolean 値を返す必要があります。ソートと同様、フィルタ関数を指定または変更したときは、フィルタ結果を表示するコレクションに対して `refresh()` メソッドを呼び出す必要があります。たとえば、ストリング配列のコレクションビューの内容を M で始まるストリングのみに制限するには、次のフィルタ関数を使用します。

```
public function stateFilterFunc(item:Object):Boolean
{
  return item >= "M" && item < "N";
}
```

例 : ArrayCollection のソートとフィルタ

次の例は、フィルタ関数とソートの使用方法を示します。ボタンを使用して、コレクションのソート、フィルタ、またはその両方を実行できます。[Reset] ボタンを使用すると、コレクションビューが元の状態に戻ります。

```
<?xml version="1.0"?>
<!-- dpcontrols\SortFilterArrayCollection.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600">
    <mx:Script>
        <![CDATA[
            import mx.collections.*;

            // Function to sort the ICollectionView
            // in ascending order.
            public function sortAC():void {
                var sortA:Sort = new Sort();
                sortA.fields=[new SortField("label")];
                myAC.sort=sortA;
                //Refresh the collection view to show the sort.
                myAC.refresh();
            }

            // Function to filter out all items with labels
            // that are not in the range of M-N.
            public function stateFilterFunc(item:Object):Boolean {
                return item.label >= "M" && item.label < "O";
            }

            // Function to apply the filter function the ICollectionView.
            public function filterAC():void {
                myAC.filterFunction=stateFilterFunc;
                //Refresh the collection view to apply the filter.
                myAC.refresh();
            }
```

```
            // Function to Reset the view to its original state.
            public function resetAC():void {
                myAC.filterFunction=null;
                myAC.sort=null;
                //Refresh the collection view.
                myAC.refresh();
            }

        ]]>
    </mx:Script>

    <!-- An ArrayCollection with an array of objects. -->
    <mx:ArrayCollection id="myAC">
        <mx:Array id="myArray">
            <mx:Object label="LA" data="Baton Rouge"/>
            <mx:Object label="NH" data="Concord"/>
            <mx:Object label="TX" data="Austin"/>
            <mx:Object label="MA" data="Boston"/>
            <mx:Object label="AZ" data="Phoenix"/>
            <mx:Object label="OR" data="Salem"/>
            <mx:Object label="FL" data="Tallahassee"/>
            <mx:Object label="MN" data="Saint Paul"/>
            <mx:Object label="NY" data="Albany"/>
        </mx:Array>
    </mx:ArrayCollection>

    <!-- Buttons to filter, sort, or reset the view in the second ComboBox
            control. -->
    <mx:HBox width="100%">
        <mx:Button id="sortButton" label="Sort" click="sortAC();"/>
        <mx:Button id="filterButton" label="Filter" click="filterAC();"/>
        <mx:Button id="resetButton" label="Reset" click="resetAC();"/>
    </mx:HBox>
    <mx:HBox width="100%">
        <!-- A ComboBox populated by the underlying Array object.
            This control shows that Array retains its original order. -->
        <mx:ComboBox id="cb2" rowCount="10" dataProvider="{myArray}"/>
        <!-- A ComboBox populated by the collection view of the Array. -->
        <mx:ComboBox id="cb1" rowCount="10" dataProvider="{myAC}"/>
    </mx:HBox>
</mx:Application>
```

もっと複雑な、DataGrid コントロールをソートする例については、446 ページの「例 : DataGrid の複数列のソート」を参照してください。この例では、データの初期ソートを行い、列見出しをクリックしたときにカスタムソートを実行します。

IViewCursor インターフェイスの使用

ICollectionView インターフェイスには、IViewCursor オブジェクトを返す createCursor() メソッドが含まれます。IViewCursor オブジェクトは " カーソル " とも呼ばれます。カーソルを使用すると、ビュー内のアイテム間を移動し、コレクション内のデータにアクセスして変更できます。カーソルは位置インジケータであり、コレクション内の特定のアイテムを指し示します。IViewCursor のメソッドとプロパティを使用して次の処理を実行できます。

- カーソルを前後に移動する。
- カーソルを特定のアイテムに移動する。
- カーソル位置にあるアイテムを取得する。
- アイテムを追加、削除、変更する。
- ブックマークを使用してカーソルの位置を保存し、後でその位置にカーソルを戻す。

標準 Flex コレクションクラスである ArrayCollection および XMLListCollection を使用する場合は、次のように、オブジェクトインスタンスを参照せず、IViewCursor インターフェイスを直接使用します。

```
public var myAC:ICollectionView = new ArrayCollection(myArray);
public var myCursor:IViewCursor;
.
.
myCursor=myAC.createCursor();
```

ビューカーソルの操作

IViewCursor インターフェイスには、カーソルを移動するための次のメソッドとプロパティが含まれます。

- moveNext() メソッドと movePrevious() メソッドは、カーソルを 1 アイテムずつ前後に移動します。ビューの境界に達したかどうかをチェックするには、beforeFirst プロパティと afterLast プロパティを使用します。次の例では、ビュー内の最後のアイテムまでカーソルを移動しています。

  ```
  while (! myCursor.afterLast) {
       myCursor.moveNext();
  }
  ```

- findAny()、findFirst()、findLast() の各メソッドは、パラメータに一致するアイテムにカーソルを移動します。これらのメソッドを使用するには、ICollectionView 実装に Sort を適用する必要があります。これは、これらの関数が Sort メソッドを使用するためです。

  一意ではないインデックス内で最初のアイテムまたは最後のアイテムを検索することが重視されない場合は、findFirst() または findLast() メソッドよりも findAny() メソッドの方が若干効率的です。

関連付けられたコレクションがリモートで、必ずしもすべてのアイテムがローカルでキャッシュされていない場合、find メソッドはリモートコレクションから非同期取得を開始します。非同期取得が実行中の場合は、完了するまで待ってから新しい取得要求を行います。

次の例では、単純なオブジェクトのコレクション (州の郵便番号ストリングの ArrayCollection) の中でアイテムを検索します。ここでは、デフォルトの Sort オブジェクトを作成し、その Sort オブジェクトを ArrayCollection オブジェクトに適用して、単純なストリング配列内で文字列 "MZ" の最初のインスタンスを検索しています。

```
var sortD:Sort = new Sort();
// SortField コンストラクタの最初のパラメータ（null）は、単純なオブジェクト
// （ストリング、数値、ブール値 ）のコレクションを指定する
// 2 番目のパラメータ（true）は、大文字と小文字を区別せずにソートすることを指定する
sortD.fields = [new SortField(null, true)];
myAC.sort=sortD;
myAC.refresh();
myCursor.findFirst("MZ");
```

複雑なオブジェクトを検索する場合は、findFirst() メソッドで複数のソートフィールドに対して検索を実行できます。ただし、どの find メソッドでもフィールドをスキップして指定することはできません。たとえば、オブジェクトに 3 つのフィールドがある場合、1、1,2、1,2,3 の組み合わせでパラメータにフィールドを指定することはできますが、1 と 3 のみを指定することはできません。

次の行はどちらも、ラベル値 "ME" とデータ値 "Augusta" を持つオブジェクトを検索します。

```
myCursor.findFirst({label:"ME"});
myCursor.findFirst({label:"ME", data:"Augusta"});
```

- seek() メソッドは、ブックマークに対する相対的な位置にカーソルを移動します。このメソッドを使用して、ビュー内の先頭または末尾のアイテム、保存したブックマークの位置にカーソルを移動します。ブックマークと seek() メソッドの使用の詳細については、173 ページの「ブックマークの使用」を参照してください。

アイテムの取得、追加、および削除

IViewCursor インターフェイスには、ビュー内のデータにアクセスし、それを変更するための次のメソッドとプロパティが含まれます。

- current プロパティは、現在のカーソル位置にあるアイテムへの参照です。
- insert() メソッドは、現在のカーソル位置の前にアイテムを挿入します。ただし、(たとえば find() を実行するために) コレクションがソートされている場合、挿入したアイテムはカーソル位置ではなくソート順の位置に移動することに注意してください。
- remove() メソッドは、現在のカーソル位置にあるアイテムを削除します。削除されたアイテムが最後のアイテムでない場合、カーソルは削除されたアイテムの次の位置を指し示します。

次の例は、current プロパティに対して insert() と remove() を使用した結果を示します。

```
<?xml version="1.0"?>
<!-- dpcontrols\GetAddRemoveItems.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
        initialize="initData();">
    <mx:Script>
        <![CDATA[
            import mx.collections.*;
            public var myArray:Array = [{label:"MA", data:"Massachusetts"},
            {label:"MN", data:"Minnesota"}, {label:"MO", data:"Missouri"}];
            [Bindable]
            public var myAC:ArrayCollection;
            public var myCursor:IViewCursor;

            // Initialize the ArrayCollection when you
            // initialize the application.
            public function initData():void {
                myAC = new ArrayCollection(myArray);
            }

            // The function to change the collection,
            // and therefore the Array.
            public function testCollection():void {
                // Get an IViewCursor object for accessing the collection data.
                myCursor=myAC.createCursor();
                ta1.text="At start. cursor is at: " + myCursor.current.label;
                var removedItem:String=String(myCursor.remove());
               ta1.text+="\nAfter removing the current item, the cursor is at: "
                    + myCursor.current.label;
                myCursor.insert({label:"ME", data:"Augusta"});
                ta1.text+="\nAfter adding an item, the cursor is at: "
                    + myCursor.current.label;
            }
        ]]>
    </mx:Script>

    <mx:ComboBox id="myCB" rowCount="7" dataProvider="{myAC}"/>
    <mx:TextArea id="ta1" height="75" width="350"/>
    <mx:Button label="run test" click="testCollection();"/>
</mx:Application>
```

ブックマークの使用

ブックマークは、後で使用するためにカーソル位置を保存するときに使用します。また、組み込みの FIRST および LAST の各ブックマークプロパティを使用することにより、ビュー内の先頭または末尾のアイテムにカーソルを移動できます。

ブックマークを作成および使用するには :

1. ビュー内の目的の位置にカーソルを移動します。
2. 次のように、bookmark プロパティの現在値を変数に割り当てます。

```
var myBookmark:CursorBookmark=myIViewCursor.bookmark;
```

3. カーソルを移動する処理を行います。
4. ブックマークを付けたカーソル位置 (またはブックマーク位置からの特定のオフセット位置) に戻るときは、次のように IViewCursor seek() メソッドを呼び出します。

```
myIViewCursor.seek(myBookmark);
```

次の例では、ComboBox で選択したアイテムからコレクションの末尾までのアイテムの数をカウントした後、カーソルを最初の位置に戻します。

```
<?xml version="1.0"?>
<!-- dpcontrols\UseBookmarks.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
        initialize="run();">
    <mx:Script>
        <![CDATA[
            import mx.collections.*;
            private var myCursor:IViewCursor;

            // Initialize variables.
            public function run():void {
                // Initialize the cursor.
                myCursor=myAC.createCursor();
                // The findFirst() method, used in
                // countFromSelection() requires a
                // sorted view.
                var sort:Sort = new Sort();
                sort.fields=[new SortField("label")];
                myAC.sort=sort;
                //You must refresh the view to apply the sort.
                myAC.refresh();
            }

            // Count the items following the current
            // cursor location.
            public function countLast(theCursor:IViewCursor):int {
                var counter:int=0;
                // Set a bookmark at the current cursor location.
                var mark:CursorBookmark=theCursor.bookmark;
```

```
                // Move the cursor to the end of the Array.
                // The moveNext() method returns false when the cursor
                // is after the last item.
                    while (theCursor.moveNext()) {
                    counter++;
                }
                // Return the cursor to the initial location.
                theCursor.seek(mark);
                return counter;
            }

            // Function triggered by ComboBox change event.
            // Calls the countLast() function to count the
            // number of items to the end of the collection.
            public function countFromSelection():void {
                myCursor.findFirst(myCB.selectedItem);
                var count:int = countLast(myCursor);
                ta1.text += myCursor.current.label + " is " + count +
                                " from the last item.\n";
            }
        ]]>
    </mx:Script>

    <!-- The data provider, an ArrayCollection with an array of objects. -->
    <mx:ArrayCollection id="myAC">
        <mx:Object label="MA" data="Boston"/>
        <mx:Object label="ME" data="Augusta"/>
        <mx:Object label="MI" data="Lansing"/>
        <mx:Object label="MN" data="Saint Paul"/>
        <mx:Object label="MO" data="Jefferson City"/>
        <mx:Object label="MS" data="Jackson"/>
        <mx:Object label="MT" data="Helena"/>
    </mx:ArrayCollection>

    <mx:ComboBox id="myCB" rowCount="7" dataProvider="{myAC}"
change="countFromSelection();"/>
    <mx:TextArea id="ta1" height="200" width="175"/>
</mx:Application>
```

例 : ICollectionView インターフェイスのメソッドとプロパティを使用した配列の更新

次の例では、ArrayCollection オブジェクトの ICollectionView のメソッドとプロパティを使用して、次のエレメントを持つ配列を ComboBox コントロールに表示します。

```
"AZ", "MA", "MZ", "MN", "MO", "MS"
```

[Update View] ボタンをクリックすると、ICollectionView インターフェイスの length プロパティといくつかのメソッドを使用して次のことが行われます。

- 配列内のデータと ComboBox コントロール内の表示データを、M で始まる州の米国郵便番号を含む正しいアルファベット順リストに変更します。

  ```
  MA, ME, MI, MN, MO, MS, MT
  ```

- 追加した ME アイテムを参照するブックマークを保存し、後でこの位置にカーソルを戻します。
- 実行したタスクに関する情報と配列の変更結果を TextArea コントロールに表示します。

[Sort] ボタンをクリックすると、ビュー内のアイテムの順序が逆になり、ME ～ MO の範囲のみが表示されます。

[Reset] ボタンをクリックすると、データプロバイダの配列とコレクションビューがリセットされます。

```
<?xml version="1.0"?>
<!-- dpcontrols\UpdateArrayViaICollectionView.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
        initialize="initData();">

    <mx:Script>
        <![CDATA[
            import mx.collections.*;
            // データプロバイダは String の配列
            public var myArray:Array = ["AZ", "MA", "MZ", "MN", "MO", "MS"];
            // Array を表す ArrayCollection を宣言する
            // ComboBox を適切に更新できるようにするため、変数をバインド可能にする
            [Bindable]
            public var myAC:ArrayCollection;
            // 更新ルーチンが以前に実行されていないことを確認するための Boolean フラグ
            public var runBefore:Boolean=false;

            // アプリケーションによって初期化された ArrayCollection を初期化する
            public function initData():void {
                myAC = new ArrayCollection(myArray);
            }

            // コレクションを変更する関数
            public function changeCollection():void {
```

```
// リセットせずに 2 回実行するとエラーが発生する
if (! runBefore) {
    runBefore=true;
     // コレクションのデータにアクセスするために IViewCursor オブジェクト
を取得する
    var myCursor:IViewCursor=myAC.createCursor();
    // 元のコレクションの長さを取得する
    var oldLength:int=myAC.length;

   // カーソルは初期状態では先頭のアイテムにあります。このアイテムを削除する
    var removedItem:String=String(myCursor.remove());

    // 2 番目のアイテムとして ME を追加する。
    // カーソルは ( 新しい ) 先頭のアイテムにある
    // カーソルを 2 番目のアイテムに移動する
    myCursor.moveNext();
    // 2 番目のアイテムの前に ME を挿入する
    myCursor.insert("ME");

    // コレクションの末尾に MT を追加する
    // LAST ブックマークプロパティを使用してビューの末尾に移動する
    // 最後のアイテムの後にカーソルを配置するため、オフセット 1 を加算する
    myCursor.seek(CursorBookmark.LAST, 1);
    myCursor.insert("MT");
    // MZ を MI に変更する
    // findFirst() メソッドのためにビューをソートする
    var sort:Sort = new Sort();
    myAC.sort=sort;
    // コレクションビューを更新してソートを適用する
    myAC.refresh();
    // 配列内に MZ アイテムがあり、MI アイテムがないことを確認する
    if (myCursor.findFirst("MZ") && !myCursor.findFirst("MI")) {
        // IViewCursor には置換機能はない
        // まず、"MZ" を削除する
        myCursor.remove();
        // ビューがソートされているため、insert はこのアイテムを
        // 基になる Array データプロバイダの末尾ではなく、
        // ソートされたビュー内の正しい位置に挿入する
        myCursor.insert("MI");
    }

    // 更新後のコレクションの長さを取得する
    var newLength:int=myAC.length;
```

```
            // 値 ME を持つアイテムにブックマークを設定する
            myCursor.findFirst("ME");
            var MEMark:CursorBookmark=myCursor.bookmark;
            // Array の最後のアイテムにカーソルを移動する
            myCursor.seek(CursorBookmark.LAST);
            // コレクション内の最後のアイテムを取得する
            var lastItem:String=String(myCursor.current);
            // ブックマーク位置にカーソルを戻す
            myCursor.seek(MEMark);
            // カーソル位置にあるアイテムを取得する
            var MEItem:String=String(myCursor.current);

            // TextArea コントロールに情報を表示する
            ta1.text="Start Length: " + oldLength + ".End Length: "
                            + newLength;
            ta1.text+=".\nRemoved " + removedItem;
            ta1.text+=".\nLast Item is " + lastItem;
            ta1.text+=".\nItem at MEMark is " + MEItem;
            // 基になる Array が変更されたことを示す
            // Array の順序はソートされていない
            ta1.text+="\nThe base Array is: " + myArray.join();
        } // runBefore 条件の終わり
    }

    // 範囲を制限するために sortICV メソッドで使用するフィルタ関数
    public function MEMOFilter(item:Object):Boolean {
        return item >= "ME" && item <= "MO";
    }

    // コレクションビューを降順にソートし、
    // アイテムの範囲を ME 〜 MO に制限する
    public function sortICV():void {
        var sort:Sort = new Sort();
        sort.fields=[new SortField(null, false, true)];
        myAC.filterFunction=MEMOFilter;
        myAC.sort=sort;
        // ArrayCollection を更新して、ソートとフィルタ関数を
        // 適用する
        myAC.refresh();
        // ComboBox の selectedIndex() メソッドを呼び出して、表示内の "MA" を
        // ソートされたビューの最初のアイテムに置換する
        myCB.selectedIndex=0;
        ta1.text="Sorted";
    }
```

```
            // Array をリセットして表示を更新し、この例をもう一度実行する
            public function resetView():void {
                myArray = ["AZ", "MA", "MZ", "MN", "MO", "MS"];
                myAC = new ArrayCollection(myArray);
                ta1.text="Reset";
                runBefore=false;
            }
        ]]>
    </mx:Script>

    <mx:ComboBox id="myCB" rowCount="7" dataProvider="{myAC}"/>
    <mx:TextArea id="ta1" height="75" width="300"/>
    <mx:HBox>
        <mx:Button label="Update View" click="changeCollection();"/>
        <mx:Button label="Sort View" click="sortICV();"/>
        <mx:Button label="Reset View" click="resetView();"/>
    </mx:HBox>
</mx:Application>
```

イベントおよび更新通知の使用

Flex コレクションメカニズムには、コレクションの変更を表すイベントがあり、イベントの配信を制御するためのメソッドが用意されています。以降のセクションでは、これらのイベントについて説明し、イベントの配信を制御する方法を示します。

コレクションイベントの使用

Flex には、コレクション関連イベントで使用するクラスがあります。

- IList または ICollectionView インターフェイスを実装するクラスは、コレクションが変更されるたびに、CollectionEvent (mx.events.CollectionEvent) クラスイベントを送出します。コレクションイベントはすべて、type プロパティの値として CollectionEvent.COLLECTION_CHANGE を持ちます。
- CollectionEvent オブジェクトには、コレクションがどのようにして変更されたかを示す kind プロパティが含まれます。kind プロパティの値と CollectionEventKind 定数 (例 : UPDATE) を比較することにより、コレクションの変更方法を判断できます。

- CollectionEvent オブジェクトには items プロパティが含まれます。このプロパティは、イベントの種類によってその型が変わるオブジェクトの Array です。イベントの種類が ADD および REMOVE の場合、この配列には追加または削除されたアイテムが格納されます。UPDATE イベントの場合、items プロパティには PropertyChangeEvent イベントオブジェクトの Array が格納されます。このオブジェクトのプロパティは、変更のタイプと、変更前と変更後のプロパティ値を示します。
- PropertyChangeEvent クラスの kind プロパティは、プロパティがどのようにして変更されたかを示します。kind プロパティの値と PropertyChangeEventKind 定数 (例 : UPDATE) を比較することにより、プロパティの変更タイプを判断できます。
- IViewCursor インターフェイスを実装するクラスは、カーソル位置が変更されたときに、type プロパティ値として mx.events.FlexEvent.CURSOR_UPDATE を持つ FlexEvent クラスイベントを送出します。

コレクションイベントは、コレクションの変更を監視して表示を更新するために使用します。たとえば、カスタムコントロールのデータプロバイダとしてコレクションを使用し、コレクションが変更されるたびにコントロールを動的に更新して変更後のデータを表示する場合は、コントロールでコレクションイベントを監視し、イベントに応じてコントロールの内容を更新できます。

たとえば、コレクションイベントを使用する単純なレンタカー予約システムを開発する場合を考えます。このアプリケーションは、COLLECTION_CHANGE イベントタイプリスナーを使用して、reservations および cars データコレクションの変更を監視します。

reservationsChanged という名前の CollectionEvent リスナーメソッドは、イベントの kind フィールドを調べて次のことを行います。

- イベントの kind プロパティが ADD の場合は、イベントの items プロパティ内のオブジェクトを反復処理し、各予約の期間を表すボックスで予約情報の表示を更新する関数を呼び出します。
- イベントの kind プロパティが REMOVE の場合は、イベントの items プロパティ内のオブジェクトを反復処理し、各アイテムの予約ボックスを削除する関数を呼び出します。
- イベントの kind プロパティが UPDATE の場合は、イベントの items プロパティ内の PropertyChangeEvent オブジェクトを反復処理し、各アイテムを更新する関数を呼び出します。
- イベントの kind プロパティが RESET の場合は、予約情報をリセットする関数を呼び出します。

次の例は、reservationsChanged CollectionEvent イベントリスナー関数を示します。

```
private function reservationsChanged(event:CollectionEvent):void {
  switch (event.kind) {
    case CollectionEventKind.ADD:
      for (var i:uint = 0; i < event.items.length; i++) {
           updateReservationBox(Reservation(event.items[i]));
      }
      break;

    case CollectionEventKind.REMOVE:
      for (var i:uint = 0; i < event.items.length; i++) {
```

```
            removeReservationBox(Reservation(event.items[i]));
        }
        break;

    case CollectionEventKind.UPDATE:
        for (var i:uint = 0; i < event.items.length; i++) {
            if (event.items[i] is PropertyChangeEvent) {
                if (PropertyChangeEvent(event.items[i]) != null) {
                    updateReservationBox(Reservation(PropertyChangeEvent(
                     event.items[i]).source));
                }
            }
            else if (event.items[i] is Reservation) {
                updateReservationBox(Reservation(event.items[i]));
            }
    }
    break;

    case CollectionEventKind.RESET:
        refreshReservations();
        break;
  }
}
```

updateReservationBox() メソッドは、予約の期間を表すボックスを表示または非表示にします。removeReservationBox() メソッドは、予約ボックスを削除します。refreshReservations() メソッドは、現在の予約情報をすべて再表示します。

アプリケーションと各メソッドの詳細については、サンプルコードを参照してください。

コレクションの変更通知の使用

IList および ICollectionView インターフェイスに含まれる itemUpdated() メソッドは、基になるデータが変更されたことをコレクションに通知するとともに、データのコレクションビューが最新であることを保証します。このメソッドは、変更されたアイテム、更新されたアイテムのプロパティ、および変更前と変更後の値をパラメータとして受け取ります。

ICollectionView インターフェイスでは、基になるデータプロバイダが変更されたときのコレクションビューの自動更新を有効および無効にする、enableAutoUpdate() メソッドと disableAutoUpdate() メソッドも提供されています。

itemUpdated メソッドの使用

itemUpdated() メソッドは、オブジェクトが IEventDispatcher インターフェイスを実装していない場合に、データプロバイダオブジェクトの変更をコレクションに通知するために使用します (IEventDispatcher インターフェイスを実装していないオブジェクトは監視できません)。Flash と Flex Object、およびその他の基本データ型は、このインターフェイスを実装していません。したがって、Array などのデータプロバイダのプロパティを変更する場合や、表示オブジェクトを通じてデータプロバイダのプロパティを変更する場合は、itemUpdated メソッドを使用してコレクションを更新する必要があります。

また、コントロールのデータプロバイダとしてコレクションではなく Array を使用しなければならない場合にも、itemUpdated() メソッドを使用できます。この場合、Array はコレクションラッパーにラップされます。ラッパーは基になる Array データオブジェクトの変更を手動で通知する必要がありますが、この通知に itemUpdated() を使用できます。

コレクション内のアイテムを直接追加または削除するか、ICollectionView または IList のいずれかのメソッドを使用してコレクションを変更する場合は、itemUpdated() メソッドを使用する必要はありません。

また、クラス定義の上かクラス内の変数定義の上に [Bindable] メタデータタグを指定すると、そのクラスに IEventDispatcher インターフェイスが確実に実装され、propertyChange イベントがそのクラスから送出されるようになります。クラス宣言の上に [Bindable] タグを指定すると、すべてのプロパティについて propertyChange イベントが送出されます。特定のプロパティのみに [Bindable] タグを付けると、そのプロパティのみについてイベントが送出されます。コレクションは propertyChange イベントをリッスンします。したがって、myCollection というコレクションが、[Bindable] タグの付いた myVariable 変数を持つクラスのインスタンスで構成されている場合は、myCollection.getItemAt(0).myVariable="myText" のような式を実行すると、アイテムからイベントが送出されます。そのため、itemUpdated() メソッドを使用する必要はありません。[Bindable] メタデータタグとその使用方法の詳細については、1133 ページ、第 38 章の「データバインディング」を参照してください。

itemUpdate() メソッドの最も一般的な使い方は、バインド可能にできない、または IEventDispatcher インターフェイスを実装するように変更できないカスタムクラスデータソースの変更をコレクションに通知することです。次の概念的な例は、このような状況で itemUpdated() メソッドをどのように使用すればよいかを示しています。

次のような、何も制御せず何も編集しないクラスがあるとします。

```
public class ClassICantEdit {
   public var field1:String;
   public var field2:String;
}
```

これらのオブジェクトを使用する次のような ArrayCollection を定義します。この ArrayCollection に classICantEdit オブジェクトを格納します。

```
public var myCollection:ArrayCollection = new ArrayCollection();
```

次のような DataGrid コントロールがあります。

```
<mx:DataGrid dataProvider="{myCollection}"/>
```

myCollection ArrayCollection のフィールドを次のように更新しても、DataGrid は自動的に更新されません。

```
myCollection.getItemAt(0).field1="someOtherValue";
```

DataGrid コントロールを更新するためには、コレクションの itemUpdated() メソッドを使用する必要があります。

```
myCollection.itemUpdated(collectionOfThoseClasses.getItemAt(0));
```

自動更新の無効化と有効化

ICollectionView の disableAutoUpdate() メソッドは、基になるデータの変更を表すイベントがビューによってブロードキャストされないようにします。またこのメソッドは、ICollectionView がこれらの変更の結果として更新されないようにもします。

このメソッドを使用すると、ICollectionView、ひいては ICollectionView を DataProvider として使用するコントロールにおいて、一連の変更の中間結果が表示されなくなります。たとえば、DataGrid クラスは disableAutoUpdate() メソッドを使用して、特定のアイテムが選択されている間、ICollectionView オブジェクトが更新されないようにします。そのアイテムの選択が解除されると、enableAutoUpdate() メソッドを呼び出します。これにより、DataGrid がソートされたコレクションビューを使用する場合に、編集中のアイテムが移動しないことが保証されます。

また、コレクション内の複数のアイテムが同時に編集される場合、disableAutoUpdate() メソッドを使用してパフォーマンスを最適化できます。DataGrid のようなコントロールでは、すべての変更が終わるまで自動更新を無効にすることにより、複数のイベントに応答する必要がなくなり、更新イベントを 1 回のバッチとして受け取ることができます。

次のコードは、disableAutoUpdate() メソッドと enableAutoUpdate() メソッドの使用方法を示しています。

```
var obj:myObject = myCollection.getItemAt(0);
myCollection.disableAutoUpdate();
obj.prop1 = 'foo';
obj.prop2 = 'bar';
myCollection.enableAutoUpdate();
```

例 : DataGrid コントロール内のデータの変更

次の例では、DataGrid コントロール内のデータを追加、削除、または変更できます。

```
<?xml version="1.0"?>
<!-- dpcontrols\ModifyDataGridData.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="500"
    height="600" >

    <mx:Script>
        <![CDATA[
            import mx.events.*;
            import mx.collections.*;

            // Add event information to a log (displayed in the TextArea).
            public function collectionEventHandler(event:CollectionEvent):void {
                switch(event.kind) {
                    case CollectionEventKind.ADD:
                        addLog("Item "+ event.location + " added");
                        break;
                    case CollectionEventKind.REMOVE:
                        addLog("Item "+ event.location + " removed");
                        break;
                    case CollectionEventKind.REPLACE:
                        addLog("Item "+ event.location + " Replaced");
                        break;
                    case CollectionEventKind.UPDATE:
                        addLog("Item updated");
                        break;
                }
            }
            // Helper function for adding information to the log.
            public function addLog(str:String):void {
                log.text += str + "\n";
            }

            // Add a person to the ArrayCollection.
            public function addPerson():void {
                ac.addItem({first:firstInput.text, last:lastInput.text,
                    email:emailInput.text});
                    clearInputs();
            }

            // Remove a person from the ArrayCollection.
            public function removePerson():void {
                // Make sure an item is selected.
                if (dg.selectedIndex >= 0) {
                    ac.removeItemAt(dg.selectedIndex);
                }
            }
```

```
        // Update an existing person in the ArrayCollection.
        public function updatePerson():void {
            // Make sure an item is selected.
            if (dg.selectedItem !== null) {
                ac.setItemAt({first:firstInput.text, last:lastInput.text,
                    email:emailInput.text}, dg.selectedIndex);
            }
        }

        // The change event listener for the DataGrid.
        // Clears the text input controls and updates them with the contents
        // of the selected item.
        public function dgChangeHandler():void {
            clearInputs();
            firstInput.text = dg.selectedItem.first;
            lastInput.text = dg.selectedItem.last;
            emailInput.text = dg.selectedItem.email;
        }

        // Clear the text from the input controls.
        public function clearInputs():void {
            firstInput.text = "";
            lastInput.text = "";
            emailInput.text = "";
        }

        // The labelFunction for the ComboBox;
        // Puts first and last names in the ComboBox.
        public function myLabelFunc(item:Object):String {
            return item.first + " " + item.last;
        }
        ]]>
    </mx:Script>

    <!-- The ArrayCollection used by the DataGrid and ComboBox. -->
    <mx:ArrayCollection id="ac"
            collectionChange="collectionEventHandler(event)">
        <mx:source>
            <mx:Object first="Matt" last="Matthews" email="matt@myco.com"/>
            <mx:Object first="Sue" last="Sanderson" email="sue@myco.com"/>
            <mx:Object first="Harry" last="Harrison" email="harry@myco.com"/>
        </mx:source>
    </mx:ArrayCollection>

    <mx:DataGrid width="450" id="dg" dataProvider="{ac}"
            change="dgChangeHandler()">
        <mx:columns>
            <mx:DataGridColumn dataField="first" headerText="First Name"/>
            <mx:DataGridColumn dataField="last" headerText="Last Name"/>
            <mx:DataGridColumn dataField="email" headerText="Email"/>
        </mx:columns>
```

```
    </mx:DataGrid>

    <!-- The ComboBox and DataGrid controls share an ArrayCollection as their
        data provider.
        The ComboBox control uses the labelFunction property to construct the
        labels from the dataProvider fields. -->
    <mx:ComboBox id="cb" dataProvider="{ac}" labelFunction="myLabelFunc"/>

    <!-- Form for data to add or change in the ArrayCollection. -->
    <mx:Form>
       <mx:FormItem label="First Name">
            <mx:TextInput id="firstInput"/>
       </mx:FormItem>
       <mx:FormItem label="Last Name">
            <mx:TextInput id="lastInput"/>
       </mx:FormItem>
       <mx:FormItem label="Email">
            <mx:TextInput id="emailInput"/>
       </mx:FormItem>
    </mx:Form>

    <mx:HBox>
        <!-- Buttons to initiate operations on the collection. -->
        <mx:Button label="Add New" click="addPerson()"/>
        <mx:Button label="Update Selected" click="updatePerson()"/>
        <mx:Button label="Remove Selected" click="removePerson()"/>
        <!-- Clear the text input fields. -->
        <mx:Button label="Clear" click="clearInputs()"/>
    </mx:HBox>

    <!-- The application displays event information here -->
    <mx:Label text="Log"/>
    <mx:TextArea id="log" width="100" height="100%"/>
</mx:Application>
```

階層データプロバイダの使用

ツリーのブランチとリーフ、Menu サブメニューとアイテムなど、ノードとサブノードから成るネストした階層を表示するコントロールでは、階層データプロバイダを使用します。次のコントロールは、階層データプロバイダを利用します。

- Menu
- MenuBar
- PopUpMenuButton
- Tree

階層コンポーネントはすべて、データプロバイダの操作に同じメカニズムを使用します。このセクションの例では Tree コントロールを使用していますが、これは他のコンポーネントにも適用できます。

階層データプロバイダについて

Flex フレームワークはデフォルトで、次の 2 種類の階層データソースをサポートします。

XML は次のいずれかにすることができます。整形式の XML を含む String、XML、XMLList、XMLListCollection オブジェクト (<mx:XML> および <mx:XMLList> コンパイル時タグによって生成されるオブジェクトを含みます。これらのタグはデータバインディングをサポートします。データバインディングを ActionScript で直接実行することはできません)。Flex では、整形式 XML のネスト階層を反映する Tree またはメニューコントロールを自動的に構成できます。

Object は、ノードの子が children フィールドに格納された構造を持つ、ネストされた Object または Object サブクラス (Array または ArrayCollection オブジェクトを含む) の任意の組み合わせにすることができます。詳細については、「カスタムデータ記述子の作成」を参照してください。<mx:Model> コンパイル時タグを使用して、データバインディングをサポートするネストされたオブジェクトを作成できますが、その場合は 189 ページの「Tree およびメニューコントロールでの <mx:Model> タグの使用」で定義された構造に従う必要があります。

子を可変名のフィールドに格納できるネストされた Object など、上記以外の階層データプロバイダ構造のサポートも追加できます。

データ記述子と階層データプロバイダ構造

Tree およびメニューコントロールで使用する階層データは、データ記述子クラスを使用して解析および操作できる形式であることが必要です。" データ記述子 " は、階層コントロールとデータプロバイダオブジェクト間のインターフェイスを提供するクラスです。データ記述子はコントロールに固有のメソッドのセットを実装しており、データプロバイダの内容と構造を決定するほか、データの取得、追加、削除、およびコントロールに固有のデータプロパティの変更を行います。

Flex では、階層コントロールのために次の 2 種類のデータ記述子インターフェイスが定義されています。

ITreeDataDescriptor　Tree コントロールで使用されるメソッド

IMenuDataDescriptor　Menu、MenuBar、PopUpMenuButton の各コントロールで使用されるメソッド

Flex フレームワーク には、両方のインターフェイスを実装する DefaultDataDescriptor クラスが用意されています。dataDescriptor プロパティを使用して、デフォルトの記述子構造に適合しないデータモデルを処理するカスタムデータ記述子クラスを指定できます。

データ記述子メソッドとソースの要件

両インターフェイスのメソッドと DefaultDataDescriptor クラスの動作を次の表に示します。インターフェイス / メソッド項目の 1 行目は、そのメソッドが ITreeDataDescriptor インターフェイス、IMenuDataDescriptor インターフェイス、またはその両方のいずれに属するか、つまりそのメソッドがツリー、メニュー、またはその両方のいずれで使用されるかを示します。

インターフェイス / メソッド	戻り値	DefaultDataDescriptor の動作
両方 hasChildrent(node, [model])	ノードが子に分岐しているかどうかを示すブール値	XML では、ノードが少なくとも 1 つの子エレメントを持つ場合に `true` を返します。 その他のオブジェクトでは、ノードの `children` フィールドが空でない場合に `true` を返します。
両方 getChildren(node, *[collection]*)	ノードの子	XML では、子エレメントを持つ XMLListCollection を返します。 その他のオブジェクトでは、ノードの `children` フィールドの内容を返します。
両方 isBranch(node, *[collection]*)	ノードがブランチであるかどうか	XML では、ノードが少なくとも 1 つの子を持つ場合、またはノードが `isBranch` 属性を持つ場合に true を返します。 その他のオブジェクトでは、ノードが `isBranch` フィールドを持つ場合に `true` を返します。
両方 getData(node, *[collection]*)	ノードデータ	ノードを返します。
両方 addChildAt(node, child, index, [model])	処理が成功したかどうかを示すブール値	すべての場合で、現在インデックスの位置にあるノードの前に子オブジェクトとしてノードを挿入します。
両方 removeChildAt (node, index, [model])	処理が成功したかどうかを示すブール値	すべての場合で、インデックス位置にあるノードの子を削除します。
IMenuDataDescriptor getType(node)	メニューノードタイプを表すストリング。有効な値は `check`、`radio`、および `separator` です。	XML では、ノードの `type` 属性の値を返します。 その他のオブジェクトでは、ノードの `type` フィールドの内容を返します。
IMenuDataDescriptor isEnabled(node)	メニューノードが有効かどうかを示すブール値	XML では、ノードの `enabled` 属性の値を返します。 その他のオブジェクトでは、ノードの `enabled` フィールドの内容を返します。

インターフェイス / メソッド	戻り値	DefaultDataDescriptor の動作
IMenuDataDescriptor setEnabled(node, value)		XML では、ノードの enabled 属性の値を true または false に設定します。 その他のオブジェクトでは、ノードの enabled フィールドの内容を設定します。
IMenuDataDescriptor isToggled(node)	メニューノードが選択されているかどうかを示すブール値	XML では、ノードの selected 属性の値を返します。 その他のオブジェクトでは、ノードの enabled フィールドの内容を返します。
IMenuDataDescriptor setToggled(node, value)		XML では、ノードの selected 属性の値を true または false に設定します。 その他のオブジェクトでは、ノードの enabled フィールドの内容を設定します。
IMenuDataDescriptor getGroupName (node)	ノードが属するラジオボタングループの名前	XML では、ノードの groupName 属性の値を返します。 その他のオブジェクトでは、ノードの groupName フィールドの内容を返します。

次の例の Object は、Tree コントロールのデフォルトのデータプロバイダ構造に従っており、DefaultDataDescriptor クラスによって正しく処理されます。

```
[Bindable]
public var fileSystemStructure:Object =
  {label:"mx", children: [
    {label:"Containers", children: [
      {label:"Accordian", children:[]},
      {label:"DividedBox", children: [
        {label:"BoxDivider.as", data:"BoxDivider.as"},
        {label:"BoxUniter.as", data:"BoxUniter.as"}]},
      {label: "Grid", children:[]}]},
    {label: "Controls", children: [
      {label: "Alert", data: "Alert.as"},
      {label: "Styles", children: [
        {label: "AlertForm.as", data:"AlertForm.as"}]},
      {label: "Tree", data: "Tree.as"},
      {label: "Button", data: "Button.as"}]},
    {label: "Core", children:[]}
  ]};
```

オブジェクトの場合、ルートは Object インスタンスであるため、(XML と同様に) 常に単一のルートが存在する必要があります。ネストされた Array を含む Array をデータプロバイダとして使用することもできます。この場合、プロバイダにはルートは存在せず、最上位の配列の各エレメントがコントロールの最上位に表示されます。

DefaultDataDescriptor は、整形式の XML ノードを適切に処理できます。ただし、isBranch() メソッドが true を返すのは、パラメータノードが子ノードを持つ場合、またはノードが値 true の isBranch 属性を持つ場合のみです。したがって、XML オブジェクトで、isBranch 属性を true に設定する以外の方法で空のブランチを示す場合は、カスタムデータ記述子を作成する必要があります。

DefaultDataDescriptor はコレクションを適切に処理します。たとえば、ノードの children プロパティが ICollectionView である場合、getChildren() メソッドは子を ICollectionView オブジェクトとして返します。

Tree およびメニューコントロールでの <mx:Model> タグの使用

<mx:Model> タグを使用して、MXML でデータプロバイダ構造を定義できます。Flex コンパイラは、このタグの内容を ActionScript オブジェクトの階層グラフに変換します。<mx:Model> タグには、ActionScript で Object データプロバイダを定義する場合と比べて、次の 2 つの利点があります。

- XML によく似た読みやすい形式で構造を定義できます。
- 構造の各項目を ActionScript 変数にバインドできるため、<mx:Model> を使用して、そのデータを複数の動的ソースから取得するオブジェクトベースのデータプロバイダを作成できます。

データ記述子を使用するコントロールで <mx:Model> タグを使用するためには、コンパイラによって生成されるオブジェクトが、186 ページの「データ記述子と階層データプロバイダ構造」にあるデータ記述子要件に適合している必要があります。また、XML オブジェクトと同様、タグが単一のルートエレメントを持つ必要があります。

ほとんどの場合、<mx:Model> タグを使用する代わりに、<mx:XML> タグまたは <mx:XMLList> タグを使用することを検討してください。これらのタグの説明は、196 ページの「XML データプロバイダの使用」にあります。XML ベースのタグはエレメントへのデータバインディングをサポートしており、DefaultDataDescriptor クラスは整構造の XML をすべてサポートしています。したがって、ノード名がその機能を表すような、より自然な構造を使用でき、“children” という不自然なノード名にする必要がなくなります。

DefaultDataDescriptor クラスを使用するコントロールのデータプロバイダとして <mx:Model> タグを使用するためには、子ノードの名前がすべて “children” であることが必要です。この要件は Object を使用する場合とは異なります。Object を使用する場合は、子オブジェクトを含む配列の名前を children にします。

次のコードは、メニューのデータプロバイダとして <mx:Model> タグを使用し、タグ内でデータバインディングを指定する例を示します。また、メニュー構造を動的に変更する方法も示しています。

```
<?xml version="1.0"?>
<!-- dpcontrols\ModelWithMenu.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*">
    <mx:Script>
        <![CDATA[

            import mx.controls.Menu;
```

```
        public var productMenu:Menu;

        public function initMenu(): void
        {
            productMenu = Menu.createMenu(null, Products.Department);
            productMenu.setStyle("disabledColor", 0xCC3366);
            productMenu.show(10,10);
        }

    ]]>
</mx:Script>

<mx:Model id="Products">
    <Root>
        <Department label="Toys">
            <children label="Care Bear"/>
            <children label="GI Joe"/>
            <children label="Telly Tubbies"/>
        </Department>
        <Department label="Kitchen">
            <children label="Electronics">
                <children label="Crock Pot"/>
                <children label="Panini Grill"/>
            </children>
            <children label="Cookware">
                <children label="Grill Pan"/>
                <children label="Iron Skillet" enabled="false"/>
            </children>
        </Department>
        <!-- The items in this entry are bound to the form data -->
        <Department label="{menuName.text}">
            <children label="{item1.text}"/>
            <children label="{item2.text}"/>
            <children label="{item3.text}"/>
        </Department>
    </Root>
</mx:Model>

<mx:Button label="Show Products" click="initMenu()"/>
<!-- If you change the contents of the form, the next time you
    display the Menu, it will show the updated data in the last
    main menu item. -->
<mx:Form>
    <mx:FormItem label="Third Submenu title">
        <mx:TextInput id="menuName" text="Clothing"/>
    </mx:FormItem>
    <mx:FormItem label="Item 1">
        <mx:TextInput id="item1" text="Sweaters"/>
    </mx:FormItem>
    <mx:FormItem label="Item 2">
```

```
            <mx:TextInput id="item2" text="Shoes"/>
        </mx:FormItem>
        <mx:FormItem label="Item 3">
            <mx:TextInput id="item3" text="Jackets"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

カスタムデータ記述子の作成

階層データが DefaultDataDescriptor クラスでサポートされる形式に適合しない場合、たとえばデータが子フィールドを使用しないオブジェクト内にある場合、カスタムデータ記述子を記述し、Tree コントロールの dataDescriptor プロパティにカスタムデータ記述子を指定できます。カスタムデータ記述子は、ITreeDataDescriptor インターフェイスのすべてのメソッドを実装する必要があります。

次の例は、カスタムデータ記述子の作成方法を示します。ここでは Tree コントロールを使用しています。このデータ記述子は、ネストされた ArrayCollection オブジェクトから成るデータプロバイダを適切に処理します。

次の例の MyCustomTreeDataDescriptor クラスは ITreeDataDescriptor インターフェイスのみを実装しているため、Tree コントロールはサポートしますが、メニューコントロールはサポートしません。このカスタムクラスは、子フィールドが ArrayCollection または Object であるツリーノードをサポートします。ノードの子を取得するとき、子オブジェクトが ArrayCollection の場合には、その子オブジェクトを返します。それ以外の場合は、子オブジェクトを ArrayCollection でラップしてから返します。ノードを追加するときは、子フィールドのタイプによって異なるメソッドを使用します。

```
package myComponents
// myComponents/MyCustomTreeDataDescriptor.as
{
import mx.collections.ArrayCollection;
import mx.collections.CursorBookmark;
import mx.collections.ICollectionView;
import mx.collections.IViewCursor;
import mx.events.CollectionEvent;
import mx.events.CollectionEventKind;
import mx.controls.treeClasses.*;

public class MyCustomTreeDataDescriptor implements ITreeDataDescriptor
{

    // The getChildren method requires the node to be an Object
    // with a children field.
    // If the field contains an ArrayCollection, it returns the field
    // Otherwise, it wraps the field in an ArrayCollection.
    public function getChildren(node:Object,
        model:Object=null):ICollectionView
    {
```

```
        try
        {
            if (node is Object) {
                if(node.children is ArrayCollection){
                    return node.children;
                }else{
                    return new ArrayCollection(node.children);
                }
            }
        }
        catch (e:Error) {
            trace("[Descriptor] exception checking for getChildren");
        }
        return null;
    }

    // The isBranch method simply returns true if the node is an
    // Object with a children field.
    // It does not support empty branches, but does support null children
    // fields.
    public function isBranch(node:Object, model:Object=null):Boolean {
        try {
            if (node is Object) {
                if (node.children != null)  {
                    return true;
                }
            }
        }
        catch (e:Error) {
            trace("[Descriptor] exception checking for isBranch");
        }
        return false;
    }

    // The hasChildren method Returns true if the node actually has children.
    public function hasChildren(node:Object, model:Object=null):Boolean {
        if (node == null)
            return false;
        var children:ICollectionView = getChildren(node, model);
        try {
            if (children.length > 0)
                return true;
        }
        catch (e:Error) {
        }
        return false;
    }
    // The getData method simply returns the node as an Object.
    public function getData(node:Object, model:Object=null):Object {
        try {
```

```
            return node;
        }
        catch (e:Error) {
        }
        return null;
    }

    // The addChildAt method does the following:
    // If the parent parameter is null or undefined, inserts
    // the child parameter as the first child of the model parameter.
    // If the parent parameter is an Object and has a children field,
    // adds the child parameter to it at the index parameter location.
    // It does not add a child to a terminal node if it does not have
    // a children field.
    public function addChildAt(parent:Object, child:Object, index:int,
            model:Object=null):Boolean {
        var event:CollectionEvent = new
CollectionEvent(CollectionEvent.COLLECTION_CHANGE);
        event.kind = CollectionEventKind.ADD;
        event.items = [child];
        event.location = index;
        if (!parent) {
            var iterator:IViewCursor = model.createCursor();
            iterator.seek(CursorBookmark.FIRST, index);
            iterator.insert(child);
        }
        else if (parent is Object) {
            if (parent.children != null) {
                if(parent.children is ArrayCollection) {
                    parent.children.addItemAt(child, index);
                    if (model){
                        model.dispatchEvent(event);
                        model.itemUpdated(parent);
                    }
                    return true;
                }
                else {
                    parent.children.splice(index, 0, child);
                    if (model)
                        model.dispatchEvent(event);
                    return true;
                }
            }
        }
        return false;
    }

    // The removeChildAt method does the following:
    // If the parent parameter is null or undefined, removes
    // the child at the specified index in the model.
```

```
    // If the parent parameter is an Object and has a children field,
    // removes the child at the index parameter location in the parent.
    public function removeChildAt(parent:Object, child:Object, index:int,
model:Object=null):Boolean
    {
        var event:CollectionEvent = new
CollectionEvent(CollectionEvent.COLLECTION_CHANGE);
        event.kind = CollectionEventKind.REMOVE;
        event.items = [child];
        event.location = index;

        //handle top level where there is no parent
        if (!parent)
        {
            var iterator:IViewCursor = model.createCursor();
            iterator.seek(CursorBookmark.FIRST, index);
            iterator.remove();
            if (model)
                model.dispatchEvent(event);
            return true;
        }
        else if (parent is Object)
        {
            if (parent.children != undefined)
            {
                parent.children.splice(index, 1);
                if (model)
                    model.dispatchEvent(event);
                return true;
            }
        }
        return false;
    }

}
}
```

次の例では、MyCustomTreeDataDescriptor を使用して、階層的なネストされた ArrayCollection とオブジェクトを処理します。ボタンがクリックされると、データ記述子の addChildAt() メソッドを呼び出してツリーにノードを追加します。通常は addChildAt() メソッドを直接使用することはありません。その代わりに、Tree またはメニューコントロールのメソッドを呼び出します。これらのメソッドは内部でデータ記述子メソッドを使用して、データプロバイダを変更します。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- dpcontrols\CustDataDescriptor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
creationComplete="initCollections()">

    <mx:Script>
        <![CDATA[
```

```
import mx.collections.*;
import mx.controls.treeClasses.*;
import myComponents.*;

//Variables used to construct the ArrayCollection data provider
//First top-level node and its children.
public var nestArray1:Array = [
    {label:"item1", children: [
        {label:"item1 child", children:     [
            {label:"item 1 child child", data:"child data"}
        ]}
    ]}
];
//Second top-level node and its children.
public var nestArray2:Array = [
    {label:"item2", children: [
        {label:"item2 child", children: [
            {label:"item 2 child child", data:"child data"}
        ]}
    ]}
];
//Second top-level node and its children.
public var nestArray3:Array = [
    {label:"item3", children: [
        {label:"item3 child", children: [
            {label:"item 3 child child", data:"child data"}
        ]}
    ]}
];
//Variable for the tree array.
public var treeArray:Array
//Variables for the three Array collections that correspond to the
//top-level nodes.
public var col1:ArrayCollection;
public var col2:ArrayCollection;
public var col3:ArrayCollection;
//Variable for the ArrayCollection used as the Tree data provider.
[Bindable]
public var ac:ArrayCollection;

//build the ac ArrayCollection from its parts.
public function initCollections():void{
    // Wrap each top-level node in an ArrayCollection.
    col1 = new ArrayCollection(nestArray1);
    col2 = new ArrayCollection(nestArray2);
    col3 = new ArrayCollection(nestArray3);
   // Put the three top-level node ArrayCollections in the treeArray.
    treeArray = [
        {label:"first thing", children: col1},
        {label:"second thing", children: col2},
```

```
                    {label:"third thing", children: col3},
                ];
                //Wrap the treeArray in an ArrayCollection.
                ac = new ArrayCollection(treeArray);
            }

            // Adds a child node as the first child of the selected node,
            // if any. The default selectedItem is null, which causes the
            // data descriptor addChild method to add it as the first child
            // of the ac ArrayCollection.
            public function clickAddChildren():void {
                var newChild:Object = new Object();
                newChild.label = "New Child";
                newChild.children = new ArrayCollection();
                tree.dataDescriptor.addChildAt(tree.selectedItem, newChild, 0,
ac);
            }

        ]]>
    </mx:Script>

    <mx:Tree width="200" id="tree" dataProvider="{ac}"
        dataDescriptor="{new MyCustomTreeDataDescriptor()}"/>
    <mx:Button label="add children" click="clickAddChildren()"/>
</mx:Application>
```

XML データプロバイダの使用

ツリーのデータは、通常は XML 形式でサーバーから取得しますが、<mx:Tree> タグ内に整形式の XML を記述して定義することもできます。DefaultDataDescriptor クラスは、整形式 XML データプロバイダを処理できます。

MXML で XML オブジェクトまたは XMLList オブジェクトを定義するには、<mx:XML> タグまたは <mx:XMLList> タグを使用します。ActionScript の XML および XMLList クラスとは異なり、これらのタグでは、XML テキスト内で MXML バインディング式を使用して、可変データからノードの内容を取得できます。たとえば次のように、ノードの名前属性をテキスト入力値にバインドできます。

```
<mx:XMLList id="myXMLList">
  <child name="{textInput1.text}"/>
  <child name="{textInput2.text}"/>
</mx:XMLList>
```

XML オブジェクトは、階層データコントロールの dataProvider として直接使用できます。ただし、オブジェクトが動的に変更される場合は、次のようにする必要があります。

1. XML オブジェクトまたは XMLList オブジェクトを XMLListCollection オブジェクトに変換する。
2. XMLListCollection オブジェクトを変更することにより、データのすべての更新を行う。

こうすると、コンポーネントで動的データを確実に表すことができます。XMLListCollection クラスは、IList インターフェイスと ICollectionView インターフェイスのすべてのメソッドの使用をサポートしており、XMLList クラスの最もよく使用されるメソッドの多くを追加しています。XMLListCollection の使用の詳細については、198 ページの「XMLListCollection クラスの使用」を参照してください。

次のコード例では、2 つの Tree コントロールを定義しています。1 つ目のコントロールでは XML オブジェクトを直接使用し、2 つ目のコントロールではデータソースとして XMLListCollection オブジェクトを使用しています。

```
<?xml version="1.0"?>
<!-- dpcontrols\UseXMLDP.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:XML id="capitals">
        <root>
            <Capitals label="U.S. State Capitals">
                <capital label="AL" value="Montgomery"/>
                <capital label="AK" value="Juneau"/>
                <capital label="AR" value="Little Rock"/>
                <capital label="AZ" value="Phoenix"/>
            </Capitals>
            <Capitals label="Canadian Province Capitals">
                <capital label="AB" value="Edmonton"/>
                <capital label="BC" value="Victoria"/>
                <capital label="MB" value="Winnipeg"/>
                <capital label="NB" value="Fredericton"/>
            </Capitals>
        </root>
    </mx:XML>

    <!-- Create an XMLListCollection representing the Tree nodes.
         capitals.Capitals is an XMLList with both Capitals elements. -->
    <mx:XMLListCollection id="capitalColl" source="{capitals.Capitals}"/>

    <!-- When you use an XML-based data provider with a tree
         you must specify the label field, even if it
         is "label". The XML object includes the root,
         so you must set showRoot="false". Remember that
         the Tree will not, by default, reflect dynamic changes
         to the XML object. -->
    <mx:Tree id="Tree1" dataProvider="{capitals}" labelField="@label"
        showRoot="false" width="300"/>
    <!-- The XMLListCollection does not include the XML root. -->
    <mx:Tree id="Tree2" dataProvider="{capitalColl}" labelField="@label"
        width="300"/>
</mx:Application>
```

この例は、Tree コントロールで階層データプロバイダを使用する場合の 2 つの重要な機能を示しています。

- ECMAScript for XML (E4X) オブジェクトは単一のルートノードを持つ必要がありますが、これは Tree に表示するのに適切でない場合があります。また、ツリーは最上位に複数のエレメントを保持できます。ツリーでルートノードが表示されないようにするには、showRoot プロパティに false を指定します。Tree コントロールのデフォルトの showRoot 値は true です。ただし、XMLList コレクションは単一のルートを持たないので、通常は showRoot プロパティを使用する必要はありません。
- ツリーデータプロバイダとして XML、XMLList、XMLListCollection のいずれかのオブジェクトを使用する場合、フィールドが XML 属性であれば、そのフィールドが "label" であっても labelField プロパティを指定する必要があります。これは、属性を表すために @ 記号を使用しなければならないためです。

XMLListCollection クラスの使用

XMLListCollection クラスは、XMLList オブジェクトにコレクション機能を提供するもので、attributes()、children()、elements() など、ネイティブ XMLList クラスの一部の XML 操作メソッドを使用できます。サポートされるメソッドの詳細については、『Adobe Flex 2 リファレンスガイド』の XMLListCollection を参照してください。

次の簡単な例では、XMLListCollection オブジェクトを List コントロールのデータプロバイダとして使用しています。ここでは、XMLListCollection メソッドを使用して、データプロバイダおよび List コントロールにおけるその表現にアイテムを動的に追加および削除しています。選択可能な商品の表現には Tree コントロールを、買物リストの表現には List コントロールを使用しています。

ユーザーは Tree コントロール (データプロバイダとして静的 XML オブジェクトを使用) でアイテムを選択してボタンをクリックし、List コントロールにアイテムを追加します。ユーザーがボタンをクリックすると、イベントリスナーが XMLListCollection の addItem() メソッドを使用して、選択されている XML ノードを XMLListCollection に追加します。データプロバイダはコレクションなので、List コントロールが更新されて新しいデータが表示されます。

ユーザーがアイテムを削除するときも同様に、リストでアイテムを選択して [Remove] ボタンをクリックします。イベントリスナーは XMLListCollection の removeItemAt() メソッドを使用して、データプロバイダおよび List コントロールにおけるその表現からアイテムを削除します。

```
<?xml version="1.0"?>
<!-- dpcontrols\XMLListCollectionWithList.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="400">
    <mx:Script>
        <![CDATA[
            import mx.collections.XMLListCollection;
            import mx.collections.ArrayCollection;
```

```
            // An XML object with categorized produce.
            [Bindable]
            public var myData:XML=
                <catalog>
                  <category name="Meat">
                      <product name="Buffalo" cost="4" isOrganic="No"
                          isLowFat="Yes"/>
                      <product name="T Bone Steak" cost="6" isOrganic="No"
                          isLowFat="No"/>
                      <product name="Whole Chicken" cost="1.5" isOrganic="Yes"
                           isLowFat="No"/>
                  </category>
                  <category name="Vegetables">
                      <product name="Broccoli" cost="2.16" isOrganic="Yes"
                          isLowFat="Yes"/>
                      <product name="Vine Ripened Tomatoes" cost="1.69"
isOrganic="No"
                          isLowFat="Yes"/>
                     <product name="Yellow Peppers" cost="1.25" isOrganic="Yes"
                          isLowFat="Yes"/>
                  </category>
                  <category name="Fruit">
                      <product name="Bananas" cost="0.95" isOrganic="Yes"
                          isLowFat="Yes"/>
                      <product name="Grapes" cost="1.34" isOrganic="No"
                          isLowFat="Yes" />
                      <product name="Strawberries" cost="2.5" isOrganic="Yes"
                          isLowFat="Yes"/>
                  </category>
                </catalog>;
            // An XMLListCollection representing the data
            // for the shopping List.
           [Bindable]
           public var listDP:XMLListCollection = new XMLListCollection(new
XMLList());

          // Add the item selected in the Tree to the List XMLList data provider.
           private function doTreeSelect():void
           {
               if (prodTree.selectedItem)
               listDP.addItem(prodTree.selectedItem.copy());
           }
           // Remove the selected in the List from the XMLList data provider.
           private function doListRemove():void
           {
               if (prodList.selectedItem)
                   listDP.removeItemAt(prodList.selectedIndex);
           }
        ]]>
    </mx:Script>
```

```
    <mx:Tree id="prodTree" dataProvider="{myData}" width="200"
        showRoot="false" labelField="@name"/>

    <mx:HBox>
        <mx:Button id="treeSelect" label="Add to List"
            click="doTreeSelect()"/>
        <mx:Button id="listRemove" label="Remove from List"
            click="doListRemove()"/>
    </mx:HBox>

    <mx:List id="prodList" dataProvider="{listDP}" width="200"
        labelField="@name"/>
</mx:Application>
```

リモートデータプロバイダの使用

Flex コンポーネントでは、次の種類のリモートデータプロバイダを使用できます。

- RPC データソース：HTTPService コンポーネント、WebService コンポーネント、および RemoteObject コンポーネント。
- Flex 複数のクライアントアプリケーション間でデータを配布、同期するために、サーバーサイド Flex Data Management Service とともに使用する DataService コンポーネント。

以降のセクションでは、これらのリモートデータソースを使用してデータを提供する方法について説明します。リモートデータプロバイダの使用の詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

RPC データソースの使用

RPC データソースを使用するには、次のように適切なクラスを使用してリモートサービスの結果を表します。

- RemoteObject クラスは、サーバーで java.util.List オブジェクトとして表されたデータを自動的に ArrayCollection として返します。返されたオブジェクトを直接使用できます。
- それ以外の HTTPService および WebService の結果では、データを変更する場合、または同じ結果を複数の場所で使用する場合には、返されたデータをコレクションクラスに変換します (後者の場合は効率が向上します)。原則として、直列化された (リストベースの) オブジェクトでは ArrayCollection を、E4X 形式のデータでは XMLListCollection を使用します。

次の抜粋されたコードはこの使い方を示しており、Web サービスから返されたリストを ArrayCollection に変換しています。

```
  <mx:WebService id="employeeWS" destination"employeeWS"
    showBusyCursor="true"
    fault="alert(event.fault.faultstring)">
    <mx:operation name="getList">
      <mx:request>
        <deptId>{dept.selectedItem.data}</deptId>
      </mx:request>
    </mx:operation>
.
.
  </mx:WebService>

  <mx:ArrayCollection id="ac"
    source="mx.utils.ArrayUtil.toArray(employeeWS.getList.lastResult)"/>
  <mx:DataGrid dataProvider="{ac}" width="100%">
```

RPC データソースの使用の詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

DataService コンポーネントの使用

データプロバイダとして DataService コンポーネントを使用するには、次のように、コンポーネントの `fill()` メソッドを呼び出して、Data Management Service の宛先からのデータを ArrayCollection オブジェクトに格納します。

```
<mx:Script>
  <![CDATA[
    import mx.data.DataService;
    import mx.collections.ArrayCollection;

    public var ds:DataService;
    [Bindable]
    public var contacts:ArrayCollection;

    public function initApp()
    {
      contacts = new ArrayCollection();
      ds = new DataService("contact");
      ds.fill(contacts);
    }
  ]]>
</mx:Script>
.
.
    <mx:DataGrid id="dg" dataProvider="{contacts}" editable="true">
```

DataService コンポーネントの使用の詳細については、1375 ページ、第 51 章の「アプリケーションでのデータの分散」を参照してください。

ページング対応のリモートデータプロバイダの使用

DataService クラスを使用してリモートデータを取得する場合、初期段階ではクライアント側のコレクションにすべてのデータをロードしないようにすることができます。この手法を使用すると、大量のデータがネットワーク上を転送されて、そのデータ処理のためにアプリケーションの速度が低下するような事態を避けることができます。段階的に取得するデータのことを " ページング " データ、まだ受信されていないデータのことを " 保留 " データと呼びます。

ItemPendingError エラーについて

ページングデータを取得しているとき、プログラムコードが保留データにアクセスしようとすることがあります。この場合はコレクションによって ItemPendingError エラーがスローされます。

Flex コントロールの中には、dataProvider コレクションによってスローされた ItemPendingError エラーを自動的にキャッチして処理するものがあります。この場合、アプリケーションでエラーを管理する必要はありません。このような機能を持つコントロールは次のとおりです。List、HorizontalList、TileList、DataGrid、Menu、Tree。

その他のほとんどのクラス (すべてのチャートコントロールを含む) は ItemPendingError エラーを処理しません。そのため、エラー処理コードを独自に記述する必要があります。

ItemPendingError エラーを処理しなければならないのは次のような場合です。

- **コントロールが ItemPendingError エラーを処理しない**　コントロールのデータプロバイダとしてページングコレクションを使用する場合は、データが完全にロードされてからコントロールにデータを渡すようにするか、またはコントロールで予期せぬ動作が起きていないかどうかをアプリケーションで監視する必要があります。
- **ページングコレクションを直接使用するコードを記述している**　コードの中で保留データを処理しようとしていないかどうかを調べます。該当する箇所がある場合は、コレクションのデータにアクセスするコードを ItemPendingError の try/catch ブロックで囲みます。たとえば、コレクション全体を反復処理している場合は、`while` ループを try/catch ブロックで囲み、反復処理が完了するまで呼び出し続けます。

ItemPendingError エラーの処理

ICollectionView および IList インターフェイスを実装するクラスはすべて、まだ使用できないページングコレクションのデータにアクセスしようとすると、ItemPendingError をスローします。ItemPendingError クラスには、要求されたデータのステータスを確認する手段として、次のような方法が用意されています。

- ItemPendingError オブジェクトの `addResponder()` メソッドを使用して、IResponder オブジェクトの配列を指定できます。Flex フレームワークには IResponder インターフェイスを実装する ItemResponder クラスがあり、独自の実装クラスを作成することも可能です。
- IResponder オブジェクトにはそれぞれ、result と fault の 2 つの関数が必要です。前者はデータが正常に取得されたときに呼び出され、後者は取得が失敗したときに呼び出されます。これらの状況を処理する関数を必要に応じて記述します。ItemResponder クラスのコンストラクタはこれら 2 つの関数をパラメータとして受け取り、3 番目にオプションパラメータとして、両関数がその処理で使用できる Object を受け取ります。

ItemPendingError を処理するには :

1. エラーを生成する可能性のあるコードを `try` ブロックに配置します。
2. `try` ブロックの直後に、次のシグネチャで `catch` ブロックを記述します。

```
catch (e:ItemPendingError) {
```

3. 新しい Responder オブジェクトを必要な数だけ作成します。個々の Responder オブジェクトの形式は次のようにします。

```
responder1 = new ItemResponder(
  // result 関数
  function (data:Object, token:Object=null) {
    // 新しく受信したデータを処理するコードをここに記述する
    }
  // fault 関数
    function (info:Object, token:Object=null) {
    // データの受信に失敗したときのコードをここに
    // 記述する
    }
    // この関数にはオプションの Object パラメータが必要
    // 詳細については、『ActionScript 3.0 リファレンスガイド』の ItemResponder を参照
);
```

4. 次のように、Responder オブジェクトを ItemPendingError オブジェクトに追加します。

```
e.addResponder(responder1);
```

Responder コンストラクタに渡す 2 つの関数は、IResponder メソッド実装を定義します。1 つ目の関数は IResponder `fault` メソッドの実装を定義し、2 つ目の関数は IResponder `result` メソッドの実装を定義します。前の例は Flex ItemResponder クラスのメンバーを Responder オブジェクトとして使用しているため、もう 1 つのオプションパラメータを受け取るメソッドを定義しています。

次のコードは、ページングコレクションを反復処理する関数で ItemPendingError エラーを処理する方法を示します。

```
private var myCursor:IViewCursor;
private var total:int = 0;

// myView コレクションを反復処理する
private function iterate():void
{
  if (myCursor == null)
       myCursor = myView.createCursor();
  // カーソルを移動するコードを try ブロックに配置する
  try
  {
    while(!myCursor.afterLast)
    {
      total += myCursor.current.amount;
      myCursor.moveNext();
    }
    trace('Total amount is:', total);
  }
  // catch ブロックでは、要求されたデータが保留中のときに生成されるエラーを
  // 処理する
  catch (e:ItemPendingError)
  {
    // 新しい Responder オブジェクトを作成し、それをエラーオブジェクトの
    // responder プロパティに割り当てる
    responder = new ItemResponder(
      // データが使用可能になった場合を処理する関数を定義する
      function (data:Object, token:Object=null) {
        myCursor.moveNext();
        iterate();
      },
      // " 実際の " エラーが発生した場合を処理する関数を定義する
      function (info:Object, token:Object=null) {
        trace('fault when retrieving data', info.toString());
      });
  }
}
```

ItemPendingError に関する注意

カーソル関数はほとんどすべて ItemPendingError をスローする可能性があります。ItemPendingError がスローされた場合、カーソルはエラーでない最後の値を保持します。つまり、moveNext() を呼び出したときにエラーがスローされた場合、カーソルの current には移動前の値が保持されています。IList のメソッドでエラーをスローするものは少数です (主に getItemAt と getIndexOf)。詳細については、ASDoc を参照してください。

データは先頭から順にロードされるとは限りません。そのため、ICollectionView の中央でカーソルを前に移動したときに ItemPendingError が発生する可能性もあります。

すべてのデータが使用可能になってからコントロールを表示する

チャートコントロールのように、データプロバイダに完全なデータセットが含まれていることを必要とするコントロールを使用する場合は、コレクションのすべてのデータが使用可能になってからコントロールの `dataProvider` プロパティを割り当てるようにする必要があります。そのためには、ほとんどの場合、ページングを使用しないように DataService を設定します。ただし、Flex アプリケーションで異なる目的のために同じ Data Management Service の宛先を使用する場合など、場合によってはコントロールでページングデータを使用しなければならないことがあります。

完全なデータが必要なコントロールでページングデータを使用する方法の 1 つとして、IList インターフェイスの `toArray()` メソッドを使用する方法があります。このメソッドは、指定されたパラメータオブジェクト内のすべてのデータを Array にロードしようとします。一部のデータが使用できない場合は ItemPendingError エラーがスローされるので、203 ページの「ItemPendingError エラーの処理」に示された方法でエラーを処理できます。また、それとは別に、コントロールでデータを使用する前にデータの最初から最後までを反復処理する方法も考えられます。

次の例ではこの反復処理法を使用して、ページング対応のデータプロバイダ (`theDP` 変数で指定されている) にデータが完全にロードされてから、チャートコントロールでそのデータが使用されるようにしています。

```
private function loadDP():void
{
   var cursor:IViewCursor = theDP.createCursor();
   try
   {
      while (cursor.moveNext()) {}
      chart.dataProvider = theDP;
   }
   catch (e:ItemPendingError)
   {
      e.addResponder(new ItemResponder(
         function (result:Object, token:Object=null)
         {
            loadDP();
         },
```

```
        function (fault:Object, token:Object=null)
        {
          trace('Error while loading');
        }
      ));
      cursor = null; // ガベージコレクションで回収してもよい
    }
}
```

第 8 章

コンポーネントのサイズと位置の制御

8

Adobe Flex は、コンポーネントのサイズと位置を決定することにより、コンポーネントをレイアウトします。サイズと位置の両方を決定するために複数のオプションが用意されています。このトピックでは、Flex でのコンポーネントのレイアウト方法、およびコンポーネントのサイズと位置の制御方法について説明します。

目次

サイズと位置の設定について

Flex は、一連の規則に従ってコンポーネントのレイアウトを制御します。レイアウト規則は、個々のコンポーネントのサイズ設定規則とコンテナのサイズ設定および位置設定規則が組み合わされたものです。Flex では自動レイアウトがサポートされているため、通常はコンポーネントのサイズまたは位置を初期設定する必要はありません。レイアウトの制御は Flex に任せて、開発者自身はアプリケーションのロジックに専念することができます。必要であれば、後でインスタンスのサイズを調整することも可能です。

コンテナにはそれぞれレイアウトを制御するための独自の規則があります。たとえば、VBox コンテナでは、子が単一の列にレイアウトされます。Grid コンテナでは、セルの行と列として子がレイアウトされます。パディングは、Application コンテナでは 24 ピクセル、他の多くのコンテナでは 0 ピクセルです。

Flex にはビルトインのデフォルトレイアウト規則がありますが、コンポーネントのプロパティとメソッドを使用してレイアウトをカスタマイズできます。すべてのコンポーネントには、height や width など、コンポーネントのサイズを絶対値またはコンテナに対する相対値で指定するためのプロパティがあります。各コンテナにも、レイアウト属性の設定に使用できるプロパティとスタイルがあります。たとえば、Tile コンテナの verticalGap や horizontalGap スタイルなどの設定を使用して子と子の間隔を設定したり、direction プロパティで行または列のレイアウトを指定したりできます。コンテナ内のコンポーネントのレイアウトに異なる位置設定方法を使用することも可能です。たとえば一部のコンテナでは、x 座標と y 座標の絶対値による位置設定をサポートしています。

Flex でのレイアウトについて

Flex では、Layout Manager によってレイアウトが制御されます。Layout Manager は次の 3 段階のプロセスを経て、アプリケーション内の各コンポーネントのサイズと位置を決定します。

1. **処理パス**　アプリケーションの各コンポーネントのプロパティ設定を決定します。この段階では、プロパティ設定によって内容が変わるコンポーネントを、そのサイズと位置が Flex によって決定される前に構成できます。

   処理パスの間、各コンポーネントは Layout Manager の指示で commitProperties() メソッドを実行し、プロパティ値を決定します。

2. **計算パス**　アプリケーション内のすべてのコンポーネントのデフォルトサイズを計算します。このパスは最も深くネストされたコンポーネントから始まり、Application コンテナの方へ進みます。計算パスでは、各コンポーネントの正確に調整された (つまりデフォルトの) サイズが決定されます。各コンテナのサイズは、その子のデフォルトサイズまたは明示的なサイズ (指定されている場合) に基づきます。たとえば、Box コンテナのデフォルトの幅は、すべての子のデフォルトの幅または明示的な幅の合計に、境界線の太さ、パディング、子同士の間隔を加えたものと等しくなります。

   計算パスの間、各コンポーネントは Layout Manager の指示で measureSizes() メソッドを実行します。このメソッドは、measure() メソッドを呼び出してコンポーネントのデフォルトサイズを決定します。

3. **レイアウトパス**　コンポーネントの移動やサイズ変更を行って、アプリケーションを実際にレイアウトします。このパスは最も外側のコンテナから始まり、最も内側のコンポーネントの方へ進みます。レイアウトパスにより、各コンポーネントの実際のサイズと位置が決定されます。また、lineTo() メソッドや drawRect() メソッドの呼び出しなど、プログラムによる描画も実行されます。

   レイアウトパスの間、Flex は、コンポーネントのサイズ設定プロパティが親に対するパーセント値で指定されているかどうかを判断し、その設定を使用して子コンポーネントの実際のサイズを決定します。各コンポーネントは Layout Manager の指示で updateDisplayList() メソッドを実行し、コンポーネントの子をレイアウトします。そのため、このパスのことを " 更新パス " と呼ぶこともあります。

Flex 基準系について

Flex では、位置とサイズを決定する際に次の基準系が使用されます。

- ローカル領域とローカル座標系は、コンポーネントの外側の境界に対して相対的に表されます。コンポーネントの境界線やスクロールバーなどのビジュアルエレメントはローカル座標に含まれます。
- 可視領域は、コンポーネントのビジュアルエレメントの内側にあるコンポーネントの領域です。つまり、この領域は表示されているコンポーネントの、子コントロール、テキスト、イメージなどの内容を格納する部分を表します。Flex には、この領域に対する固有の座標系はありません。
- コンテンツ領域とコンテンツ座標系には、ビジュアルエレメントを除くコンポーネントのすべての内容が含まれます。この領域には、現在クリッピングされていてコンポーネントをスクロールしなければ見えない領域もすべて含まれています。たとえば、スクロール可能な TextArea コントロールのコンテンツ領域には、現在画面からスクロールアウトされているテキストの領域が含まれます。

パーセント値に基づいてサイズを決定するとき、および制約ベースのレイアウトを実行するときには、可視領域が使用されます。

絶対配置を指定するときに使用する Flex コンポーネントの x および y プロパティは、コンテンツ座標系にあります。

メモ

Flex 座標は基準系の左上隅を原点とします。したがって、ローカル座標系の (100,300) は、コンポーネントの左上隅から右に 100 ピクセル、下に 300 ピクセルの位置にあります。

Flex 座標系の詳細については、480 ページの「Flex 座標の使用」を参照してください。

コンポーネントのサイズ設定について

コンポーネントの高さと幅は計算パスとレイアウトパスによって決定されます。これらのサイズを取得するには、`height` プロパティと `width` プロパティを使用します。これらのプロパティやその他のプロパティを使用して、コンポーネントのサイズを制御できます。

Flex でコントロールおよびコンテナのサイズを制御するには、次のような方法があります。

デフォルトサイズ設定　コントロールとコンテナのサイズを自動的に決定します。

明示的サイズ設定　`height` プロパティと `width` プロパティに絶対値を設定します。

パーセント値ベースのサイズ設定　コンポーネントのサイズをコンテナサイズのパーセント値で指定します。

制約ベースのレイアウト　コンポーネントの端をコンテナの特定の位置に固定することで、サイズと位置を制御します。

コンポーネントのサイズの制御の詳細については、214 ページの「コンポーネントのサイズ設定」を参照してください。

コンポーネントの位置設定について

Flex では、アプリケーションの初期化時にコンポーネントが配置されます。また、アプリケーションまたはユーザーがビジュアルエレメントのサイズや位置に影響を与える何らかの操作を実行した場合には、レイアウトパスが実行され、コンポーネントが配置または再配置されます。これが起こる状況としては、たとえば次のような場合があります。

- アプリケーションによって x、y、width、height、scaleX、scaleY などのサイズを指定するプロパティが変更された場合。
- 何らかの変更によって、一度計算されたコンポーネントの幅または高さが影響を受ける場合。たとえば、Button コントロールのラベルテキストを変更した場合や、ユーザーがコンポーネントのサイズを変更した場合など。
- 子の追加、削除、サイズ変更、移動が生じた場合。たとえば、アプリケーションでコンポーネントのサイズを変更できる場合、コンテナのレイアウトが自動的に更新され、子の新しいサイズに基づいて各々の子が再配置されます。
- horizontalScrollPolicy や fontFamily など、計算や描画を必要とするプロパティまたはスタイルが変更された場合。

まれに、アプリケーションプログラマが強制的にコンポーネントのレイアウトを実行しなければならない場合があります。詳細については、213 ページの「手動でのレイアウトの強制実行」を参照してください。

Flex は、コントロールの配置とレイアウトを行う 2 つのメカニズムを備えています。

自動配置　コンテナおよびコンポーネントに固有の規則のセットに基づいて、自動的にコンテナの子を配置します。Box、Grid、Form などのほとんどのコンテナは、自動配置を使用しています。自動配置は " 自動レイアウト " と呼ばれることもあります。

絶対配置　各々の子の x プロパティと y プロパティを指定するか、制約ベースのレイアウトを使用します。制約ベースのレイアウトでは、コンテナの端と子の端または中心との間の距離を指定します。絶対配置は " 絶対レイアウト " と呼ばれることもあります。

絶対配置は次の 3 つのコンテナでサポートされています。

- Application コンテナと Panel コンテナは、デフォルトでは自動配置を使用し、layout プロパティに "absolute" が指定された場合は絶対配置を使用します。
- Canvas コンテナは常に絶対配置を使用します。

コントロールの位置制御の詳細については、232 ページの「コントロールの配置とレイアウト」を参照してください。

コンポーネントのレイアウトパターン

コンテナとその子のレイアウトにはいくつかのパターンがあります。これらのパターンは通常、次の表に分類されたタイプのいずれかに一致します。この表は、各タイプの一般的なレイアウト動作、コンテナのデフォルトサイズの決定方法、およびパーセント値でサイズ指定された子のサイズ設定方法をまとめたものです。

コンテナのタイプ	デフォルトのレイアウト動作
絶対配置 : Canvas コンテナ、または `layout="absolute"` が指定されている Application または Panel コンテナ	**一般的なレイアウト：** コンテナの子が相互に影響し合うことはありません。つまり、ある子の位置がその他の子の位置に影響を与えることはなく、子同士が重なることがあります。子の位置を明示的に指定するか、子の端または中心を親コンテナに対して相対的に固定する制約を使用します。 **デフォルトサイズ設定：** 計算パスによって下端が最も下にある子と右端が最も右にある子が特定され、それらの値を使用してコンテナのサイズが決定されます。 **パーセント値でサイズ指定された子：** 制約ベースのレイアウトを使用するか、x および y 座標による位置設定を使用するかによって、使用される規則が異なります。詳細については、224 ページの「絶対配置を使用するコンテナでのパーセント値ベースの子のサイズ設定」を参照してください。
Box、HBox、VBox など、すべての子を直線的に配置するコントロール	**一般的なレイアウト：** コンテナのすべての子が単一の行または列に配置されます。各々の子の高さと幅は他の子の高さまたは幅と異なる場合があります。 **デフォルトサイズ設定：** すべての子のデフォルトサイズまたは明示的なサイズ、すべての間隔、境界線、およびパディングが収まるようにコンテナのサイズが決定されます。 **パーセント値でサイズ指定された子：** パーセント値でサイズ指定された子の要求するサイズが使用可能な領域より大きい場合、実際には、要求されたパーセント値に比例した割合で領域を配分し、そこに収まるようにサイズが設定されます。
Grid	**一般的なレイアウト：** このコンテナは実質的に、すべてのアイテムが互いに整列した HBox 子コントロールの行を持つ VBox コントロールと見なすことができます。1 行に含まれるセルの高さはすべて同じですが、行ごとに高さを変えることは可能です。ある 1 列に含まれるセルの幅はすべて同じですが、列ごとに幅を変えることは可能です。Grid コンテナの各行または各列には異なる数のセルを定義でき、個々のセルが複数の列または行にまたがることもできます。 **デフォルトサイズ設定：** 個々の行および子のデフォルトサイズが収まるようにグリッドのサイズが決定されます。 **パーセント値でサイズ指定された子：** 子のサイズがパーセント値で指定されている場合は、直線的なコンテナサイズ設定規則に従って、子 GridItem コンポーネントはそれが属する行内に、GridRow コンポーネントはグリッドサイズ内に、それぞれ収められます。

コンテナのタイプ	デフォルトのレイアウト動作
Tile	**一般的なレイアウト：** このコンテナは同じサイズのセルがグリッド状に並んだものになります。セルの順序は行優先にも列優先にもできます。明示的なサイズまたはパーセント値によるサイズを指定しない場合、行数と列数はできる限り同数に近づきます。必要であれば、`direction` プロパティを使用して、アイテムの多数を占める方向を指定できます。 **デフォルトサイズ設定：** `tileWidth` プロパティと `tileHeight` プロパティを指定しない場合は、最大の子セルのデフォルトサイズまたは明示的なサイズが各子セルのサイズに使用されます。 **パーセント値でサイズ指定された子：** パーセント値で指定された子コンポーネントのサイズは、Tile コンテナではなく、個々のセルのパーセント値を指定します。
ナビゲータ：ViewStack、Accordion、TabNavigator	**一般的なレイアウト：** このコンテナは一度に 1 つの子を表示します。 **デフォルトサイズ設定：** コンテナのサイズは、最初に選択された子のデフォルトサイズまたは明示的なサイズによって決定されます。その後、残りの子もすべてその初期サイズに固定されます。`resizeToChild` プロパティを `true` に設定した場合は、各々の子が表示されたときにその子のデフォルトサイズまたは明示的なサイズに合わせてコンテナのサイズが変わります。 **パーセント値でサイズ指定された子：** 原則として、高さと幅の両方に 100% を指定するか、またはパーセント値ベースのサイズ設定を使用しないようにします。前者の場合は、子がナビゲータの範囲全体を満たすようになります。

基本的なレイアウト規則と注意事項

レイアウトは次の基本規則に従って実行されます。これらの規則を知っておくと、Flex でのレイアウトの詳細が理解しやすくなり、アプリケーションがなぜそのようにレイアウトされたか、アプリケーションの外観をどのように変更すればよいかを判断する上での助けとなります。

Flex でのコンポーネントのサイズ設定方法の詳細については、216 ページの「コンポーネントのサイズの決定と制御」を参照してください。コンポーネントの位置設定の詳細については、232 ページの「コントロールの配置とレイアウト」を参照してください。

- Flex はまず、すべてのコンポーネントの計算 (デフォルト) サイズまたは明示的に設定されたサイズを、最も内側の子コントロールから最も外側のコントロール (Application コントロール) の順に決定します。これは計算パスで行われます。
- 計算パスが終わると、次にパーセント値ベースのサイズをすべて決定し、最も外側のコンテナから最も内側のコントロールの順にコンポーネントをレイアウトします。これはレイアウトパスで行われます。
- 開発者がピクセル値で設定したサイズは固定的で強制力があり、コンポーネントに対して設定した最大または最小サイズ指定よりも優先します。

- 計算パスで決定されたデフォルトサイズは、明示的なサイズまたはパーセント値ベースのサイズが指定されていない (または制約ベースのレイアウトを使用する) コンポーネントのサイズを決定します。このデフォルトサイズは固定され、変更されません。
- パーセント値ベースのサイズ指定はおおよその基準となるものです。レイアウトアルゴリズムは可能な限りこの要求を満たし、パーセント値を使用して比例的にサイズを決定しますが、実際のサイズが要求されたサイズより小さくなる場合があります。パーセント値ベースのサイズは常にコンポーネントの最大サイズと最小サイズの範囲内にあり、それらの範囲の制約を受けます。パーセント値ベースのサイズを指定したコンテナの子がコンテナのサイズを超えることはありません。

手動でのレイアウトの強制実行

ときどき、プログラムによってコンポーネントを再レイアウトしなければならないことがあります。Flex は通常、多くの計算を必要とするプロパティの処理を、そのプロパティを設定したスクリプトの実行が終了するまで遅らせます。たとえば、`width` プロパティを設定すると、オブジェクトの子またはその親の幅の再計算が必要となる可能性があるため、このプロパティの設定は遅延されます。処理を遅延すると、オブジェクトの `width` プロパティがスクリプトによって複数回設定された場合でも、その処理を繰り返す必要がなくなります。ただし、場合によっては、スクリプトが完了する前に強制的にレイアウトを実行しなければならないことがあります。

レイアウトを強制的に実行しなければならない状況としては、次の場合が考えられます。

- PrintDataGrid クラスを使用して複数ページのデータグリッドを印刷しているとき。
- エフェクトを再生する前に、開始値がターゲットに設定された場合。
- プロパティを変更した後にビットマップデータをキャプチャするとき。

レイアウトを強制的に実行するには、レイアウトするコンポーネントの `validateNow()` メソッドを呼び出します。このメソッドを呼び出すと、必要に応じてオブジェクトとそのすべての子のプロパティ、サイズ、およびレイアウトが検証および更新され、再描画されます。このメソッドは計算負荷が高いので、必要な場合のみ呼び出すようにしてください。

`validateNow()` メソッドの使用例については、1067 ページの「PrintDataGrid レイアウトの更新」を参照してください。

コンポーネントのサイズ設定

Flex には、コンポーネントのサイズを制御する方法がいくつか用意されています。次の操作を実行できます。

- コンポーネントのデフォルトサイズを自動的に決定し、そのサイズを使用する。
- ピクセル値を指定する。
- 親コンテナのパーセント値としてコンポーネントのサイズを指定する。
- 制約ベースのレイアウトを指定して、レイアウトとサイズ設定を一度に行う。

以降のセクションでは、基本的なサイズ設定プロパティ、Flex でのコンポーネントのサイズの決定方法、自動的、明示的、およびパーセント値ベースのサイズ設定の使用方法、およびコンポーネントのサイズの制御方法について説明します。制約ベースのレイアウトの詳細については、238 ページの「制約ベースのレイアウトの使用」を参照してください。

Flex サイズ設定プロパティ

Flex のプロパティの中には、コンポーネントのサイズに影響を与えるものがあります。通常はほとんどのアプリケーションにおいて、ごく少数のプロパティしか使用しませんが、これらのプロパティをより深く理解すれば、Flex の基本的なサイズ設定メカニズムや Flex サイズ設定プロパティ間の相互作用を理解するのに役立ちます。Flex コンポーネントのサイズの詳細については、216 ページの「コンポーネントのサイズの決定と制御」を参照してください。

次のセクションでは最もよく使用するサイズ設定プロパティについて説明し、その後にコンポーネントのサイズ設定方法を決定するすべての特性とプロパティをまとめた表を示します。各表には、多くのアプリケーション開発者が普通は使用しないプロパティも含まれています。このようなプロパティは、カスタムコンポーネントの開発者、特にカスタムの `measure()` メソッドを実装する開発者が使用します。最後のサブセクションでは、アプリケーションの開発に役立つ具体的なサイズ設定動作とサイズ設定手法について説明します。

よく使用されるサイズ設定プロパティ

カスタムコンポーネントを作成しない場合は通常、次の基本的なプロパティを使用してコンポーネントのサイズ設定方法を指定します。

- `height`、`width`、`percentHeight`、`percentWidth` の各プロパティは、コンポーネントの高さと幅を指定します。MXML タグでは、`height` プロパティと `width` プロパティを使用して、ピクセル単位または親コンテナサイズに対するパーセント値でサイズを指定します。ActionScript では、ピクセル単位でサイズを指定するときは `height` プロパティと `width` プロパティを使用し、親コンテナに対するパーセント値で指定するときは `percentHeight` プロパティと `percentWidth` プロパティを使用します。

- minHeight、minWidth、maxHeight、maxWidth の各プロパティは、Flex によってコンポーネントのサイズが決定される場合に、コンポーネントがとりうる最小サイズと最大サイズを指定します。幅または高さをピクセル単位で明示的に指定した場合、これらのプロパティを使用しても効果はありません。

以降のセクションでは、これらのプロパティの詳細と互いの関係について説明します。Flex は強力で精巧なサイズ設定機能を多数備えており、上記のプロパティしか使用しない場合でも、このサイズ設定機能の概念を理解しておくと役に立つことがあります。216 ページの「コンポーネントのサイズの決定と制御」では、プロパティに基づいてコンポーネントのサイズを決定する際に適用される規則について説明します。

基本的なサイズ設定特性とプロパティ

コンポーネントのサイズは、次の特性とそれに関連するプロパティによって決定されます。

特性	関連するプロパティ	説明
実際のサイズ	height プロパティと width プロパティから返されます。	レイアウト段階で決定される表示コントロールの高さと幅 (ピクセル単位)。 明示的な値を設定すると、その値によって対応する実際の値が決定されます。
明示的なサイズ	explicitHeight、explicitWidth explicitHeight プロパティと explicitWidth プロパティは、height プロパティと width プロパティを整数値に設定した場合にも設定されます。	ピクセル数で具体的に指定したサイズ。これらのサイズはオーバーライドできません。 アプリケーション開発者は通常、height プロパティと width プロパティを使用して明示的なサイズを設定します。 明示的なサイズとパーセント値ベースのサイズを両方設定することはできません。一方を設定すると、他方の設定が解除されます。
パーセント値ベースのサイズ	percentHeight、percentWidth MXML タグの場合のみ、height プロパティと width プロパティをパーセントストリング値 (“50%” など) に設定すると、percentHeight プロパティと percentWidth プロパティも設定されます。	親コンテナの可視領域に対するパーセント値で具体的に指定したサイズ。0 ～ 100 の数値で指定します。 パーセント値ベースのサイズを設定すると、コンポーネントのサイズが固定でなくなり、親のサイズの変化に応じて拡大または縮小します。
デフォルトサイズ	measuredHeight、measuredWidth	アプリケーション開発者が直接使用することはありません。 コンポーネントの measure() メソッドによって決定されるコンポーネントのサイズ。 これらの値がコンポーネントの高さと幅の最大値から最小値の範囲外になることはありません。デフォルトサイズの最大値と最小値の詳細については、「最大サイズと最小サイズ」を参照してください。

最大サイズと最小サイズ

コンポーネントがとりうる " デフォルト " または " パーセント値ベース " の最大サイズと最小サイズは、次の特性によって決まります。これらの特性が、明示的に設定した値、または制約ベースのレイアウトを使用して決定されたサイズに影響を与えることはありません。

特性	関連するプロパティ	説明
最小サイズ	`minHeight`、`minWidth` `minHeight` プロパティと `minWidth` プロパティは、最小サイズを明示的に設定した場合にも設定されます。	コンポーネントがとりうる最小サイズ。 デフォルトでは、これらのサイズは最小デフォルトサイズの値に設定されます。
最大サイズ	`maxHeight`、`maxWidthmaxHeight` プロパティと `maxWidth` プロパティは、最大サイズを明示的に設定した場合にも設定されます。	コンポーネントがとりうる最大サイズ。 これらのプロパティのデフォルト値はコンポーネントに固有ですが、通常は 10000 ピクセルです。
最小デフォルトサイズ	`measuredMinHeight`、`measuredMinWidth`	アプリケーション開発者が使用することはありません。 `measure()` メソッドによって決定される最小の有効サイズ。これらのプロパティのデフォルト値はコンポーネントに固有ですが、多くのコントロールでデフォルト値は 0 です。

コンポーネントのサイズの決定と制御

以降のセクションでは、Flex においてコントロールとコンテナのサイズがコンポーネントとそのプロパティに基づいてどのように決定されるかと、Flex プロパティを使用してサイズを制御する方法について詳しく説明します。最初のセクションでは、各種サイズ設定プロパティを使用してコンポーネントのサイズを制御する方法について説明します。2 番目のセクションでは、コントロールとコンテナの両方に適用される一般的なサイズ設定規則について説明します。3 番目のセクションでは、コンテナのサイズを設定するための規則について説明します。

メモ

コンポーネントのサイズ設定に関する基本的な規則の概要については、212 ページの「基本的なレイアウト規則と注意事項」を参照してください。

基本的なサイズ設定プロパティの規則

Flex サイズ設定プロパティを使用してコンポーネントのサイズを指定するときの規則を次に示します。

- いずれかのプロパティを設定すると、対応するデフォルト値がその値によってオーバーライドされます。たとえば、height プロパティを明示的に設定すると、デフォルトの高さがオーバーライドされます。
- MXML または ActionScript で width、height、maxWidth、maxHeight、minWidth、または minHeight プロパティをピクセル値に設定した場合にも、explicitHeight や explicitMinHeight などの対応する明示的プロパティが設定されます。
- 明示的な高さと幅、およびパーセント値ベースの高さと幅は、相互に排他的です。一方の値を設定すると、他方の値が NaN に設定されます。たとえば、height または explicitHeight を 50 に設定してから percentHeight を 33 に設定すると、explicitHeight プロパティの値は、50 でなく NaN になります。また、height プロパティは percentHeight の設定によって決定された値を返します。
- MXML タグで height プロパティまたは width プロパティをパーセント値に設定した場合、実際にはパーセント値ベースの値を設定したことになり、明示的な値ではなく、percentHeight プロパティまたは percentWidth プロパティが設定されます。ActionScript では、height プロパティまたは width プロパティをパーセント値に設定することはできません。その代わりに、percentHeight プロパティまたは percentWidth プロパティを設定する必要があります。
- height プロパティと width プロパティを取得すると、その値は常にコントロールの実際の高さまたは幅になります。

コンポーネントのサイズの決定

Flex は、計算パスの間にコンポーネントのデフォルトサイズ (計算サイズともいいます) を決定します。そしてレイアウトパスの間に、明示的またはデフォルトサイズ、およびパーセント値ベースのサイズ指定に基づいて、コンポーネントの実際のサイズを決定します。

次のリストは、コントロールとコンテナをどちらも含むすべてのコンポーネントに適用されるサイズ設定規則と動作について説明したものです。コンテナに固有のサイズ設定規則については、218 ページの「コンテナのサイズの決定」を参照してください。パーセント値ベースのサイズ設定の詳細については、222 ページの「パーセント値ベースのサイズ設定の使用」を参照してください。

- コンポーネントに対して (コンポーネントの最小 / 最大範囲内の) 明示的なサイズを指定すると、そのサイズが常に使用されます。
- コンポーネントに対してパーセント値ベースのサイズを指定した場合、コンポーネントの実際のサイズは、親コンテナのサイズ設定手順の一環として、親のサイズ、コンポーネントの要求されたパーセント値、およびコンテナに固有のサイズ設定およびレイアウト規則に基づいて決定されます。

- デフォルトサイズとパーセント値ベースのサイズは常に、最低でも最小サイズと同じ大きさになります。
- パーセント値を使用してコンポーネントのサイズを指定し、そのコンテナに対して明示的なサイズまたはパーセント値ベースのサイズを指定しない場合、コンポーネントのサイズはデフォルトサイズになり、パーセント値の指定は無視されます。そうでなければ、無限反復が発生する可能性があります。
- 子または子のセットの要求するサイズが親コンテナの使用可能な領域より大きい場合、親の境界線を越えた部分はクリッピングされます。また、デフォルトではコンテナにスクロールバーが表示されるため、ユーザーはスクロールしてクリッピングされた内容を表示できます。親の境界線を越えて子を描画できるようにするには、親の `clipContent` プロパティを `false` に設定します。スクロールバーの表示を制御するには、`scrollPolicy` プロパティを使用します。
- コンポーネントに対してパーセント値ベースのサイズを指定した場合は、サイズを決定する際にコンテナの可視領域が使用されます。
- コンポーネントのサイズと位置を計算する際に、そのコンポーネントが可視であるか不可視であるかは区別されません。デフォルトでは、不可視コンポーネントも可視コンポーネントと同じようにサイズと位置が計算されます。他のコンポーネントのサイズと位置を計算する際に不可視コンポーネントが考慮されないようにするには、コンポーネントの `includeInLayout` プロパティを `false` に設定します。このプロパティは、Accordion、FormItem、ViewStack 以外のすべてのコンテナの子のレイアウトに影響を与えます。詳細については、235 ページの「非表示コントロールのレイアウトの防止」を参照してください。

メモ

コンポーネントの `includeInLayout` プロパティを `false` に設定しても、そのコンポーネントのレイアウトや表示ができなくなるわけではありません。あくまでも、他のコンポーネントのレイアウト時にそのコンポーネントが考慮されなくなるだけです。結果として、表示リスト内の次のコンポーネントが該当するコンポーネントに重なって表示されます。コンポーネントが表示されないようにするには、`visible` プロパティも `false` に設定してください。

コンテナのサイズの決定

コンテナのサイズが決定されるときは、コンポーネントの基本的なサイズ設定規則に加えて、次の規則も使用されます。

- 計算パスの間にすべてのコンポーネントのデフォルトサイズが決定され、その値がレイアウトパスでコンテナのサイズを計算するときに使用されます。
- コンテナに対して明示的なサイズを指定すると、コンポーネントの場合と同様、そのサイズが常に使用されます。
- コンテナに対してパーセント値ベースのサイズを指定すると、コンポーネントの場合と同様、親コンテナのサイズ設定手順の一環として、コンテナの実際のサイズが決定されます。

- パーセント値ベースのコンテナのサイズはおおよその基準となるものです。コンテナは少なくとも、その子が最小サイズで収まる大きさになります。パーセント値ベースのサイズ設定の詳細については、222 ページの「パーセント値ベースのサイズ設定の使用」を参照してください。
- コンテナに対して明示的なサイズまたはパーセント値ベースのサイズを指定しない場合は、そのいずれかの子に指定した明示的なサイズと、その他すべての子のデフォルトサイズを使用して、コンテナのサイズが決定されます。
- コンテナのサイズを設定するとき、コンテナの子のパーセント値ベースの設定は考慮されません。その代わりに、子のデフォルトサイズが使用されます。
- 自動スクロールバーを使用するコンテナでは、計算パスでコンテナのデフォルトサイズを決定するときにスクロールバーのサイズは考慮されません。したがって、スクロールバーが必要な場合、デフォルトサイズのコンテナでは小さすぎて適切な形にならないことがあります。

各コンテナには、コンテナのデフォルトサイズを決定する一連の規則があります。各コントロールおよびコンテナのデフォルトサイズの詳細については、『Adobe Flex 2 リファレンスガイド』の特定のコンテナのセクション、および『Flex 2 開発ガイド』の 495 ページ、第 14 章の「Application コンテナの使用」、517 ページ、第 15 章の「レイアウトコンテナの使用」および 583 ページ、第 16 章の「ナビゲータコンテナの使用」を参照してください。

例 : HBox コンテナと子のサイズの決定

次のサンプルコードは、HBox コンテナとその子のサイズがどのように決定されるかを示しています。この例では、HBox コンテナの幅は、1 つ目と 3 つ目のボタンのデフォルト幅、2 つ目のボタンの最小幅 (デフォルト幅の方が小さいため)、および 2 つの間隔を表す 16 の合計になります。ボタンのデフォルト幅はラベルテキストの幅に基づきます。この例ではどのボタンも 66 ピクセルです。したがって、HBox の幅は 66 + 70 + 66 + 16 = 218 になります。2 つ目のボタンの `minWidth` プロパティを 50 に設定すると、このボタンのデフォルト幅が 66 になり、HBox の幅は 214 になります。

アプリケーションがレイアウトされるときは、1 つ目と 3 つ目のボタンの幅がデフォルト値の 66、2 つ目のボタンの幅が最小幅の 70 に設定されます。最終的なレイアウトの計算時には、パーセント値ベースの指定は無視されます。

```
<?xml version="1.0"?>
<!-- components\HBoxSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:HBox id="h21">
        <mx:Button id="bG1"
            label="Label 1"
            width="50%"/>
        <mx:Button id="bG2"
            label="Label 2"
            width="40%"
            minWidth="70"/>
        <mx:Button id="bG3"
            label="Label 3"/>
```

```
    </mx:HBox>

    <mx:TextArea height="50" width="100%">
        <mx:text>
            HBox: {h21.width}  Button1: {bG1.width}  Button2: {bG2.width}
                Button3: {bG3.width}
        </mx:text>
    </mx:TextArea>
</mx:Application>
```

次の例では、HBox の幅は 276 ピクセル (552 ピクセルの 50%) になります。この 552 は、Application コンテナの幅 600 からコンテナの左右のパディング 24 ピクセルずつを引いた値です。ボタンのサイズはそれぞれ 106、85、66 ピクセルです。3 つ目のボタンはデフォルトサイズを使用します。可変幅のボタンのサイズは、デフォルト幅のボタン、ボタンの間隔、および 1 ピクセルの幅を引いた残りの領域の 5/9 と 4/9 になります。

ただし、HBox の `width` プロパティを 20% に設定しても、HBox の幅は 120 ピクセル (Application コンテナの幅の 20%) にはなりません。これは、この値が HBox コンテナの子を収めるには小さすぎるためです。この場合、HBox の幅は、ボタン 1 と 3 の 66 ピクセル (デフォルトサイズ)、ボタン 2 の 50 ピクセル (指定された最小サイズ)、ボタンの間隔を表す 16 ピクセル、および境界線の 2 ピクセルを合計した 200 ピクセルになります。ボタンの幅はそれぞれ 66、50、66 ピクセルです。

```
<?xml version="1.0"?>
<!-- components\HBoxSizePercent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox id="h21" width="50%" borderStyle="solid">
        <mx:Button id="bG1"
            label="Label 1"
            width="50%"/>
        <mx:Button id="bG2"
            label="Label 2"
            width="40%"
            minWidth="50"/>
        <mx:Button id="bG3"
            label="Label 3"/>
    </mx:HBox>

    <mx:TextArea height="50" width="100%">
        <mx:text>
            HBox: {h21.width}  Button1: {bG1.width}  Button2: {bG2.width}
                Button3: {bG3.width}
        </mx:text>
    </mx:TextArea>
</mx:Application>
```

コンテナと子のサイズ設定の詳細とその例については、221 ページの「Flex コンポーネントのサイズ設定手法の使用」を参照してください。パーセント値ベースのサイズ設定の詳細については、222 ページの「パーセント値ベースのサイズ設定の使用」を参照してください。

Flex コンポーネントのサイズ設定手法の使用

以降のセクションでは、デフォルトサイズ設定、明示的サイズ設定、パーセント値ベースのサイズ設定の各手法を使用してコンポーネントのサイズを制御する方法について簡単に説明します。コンポーネントのサイズ設定手法として制約ベースのレイアウトを使用する際の詳細については、238 ページの「制約ベースのレイアウトの使用」を参照してください。

デフォルトサイズ設定の使用

サイズを特に指定しない場合は、コンポーネントの measure() メソッドにより、特定のコンポーネントのデフォルトサイズ設定特性、およびコンポーネントの子コントロールのデフォルトサイズまたは明示的なサイズに基づいてサイズが計算されます。

原則として、『Flex 2 開発ガイド』に記載されているコンポーネントのデフォルトサイズがアプリケーションに適しているかどうかを判断してください。適している場合は、明示的なサイズまたはパーセント値ベースのサイズを指定する必要はありません。

次の例は、HBox コンテナの子の Button に対してデフォルトサイズ設定を使用する方法を示しています。この例では、HBox コンテナのいずれの子にも幅が指定されていません。

```
<?xml version="1.0"?>
<!-- components\DefaultButtonSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="400" borderStyle="solid">
        <mx:Button label="Label 1"/>
        <mx:Button label="Label 2"/>
        <mx:Button label="Label 3"/>
    </mx:HBox>
</mx:Application>
```

したがって、ボタンのデフォルトサイズを使用して次のようなアプリケーションが描画されます。ボタンのデフォルトサイズは、ボタンのラベルとデフォルトのパディングが収まるサイズです。

3 つ目のボタンの右側に空白の領域があります。これは、デフォルトサイズの合計が使用可能な領域より小さいためです。

明示的なサイズの指定

コンポーネントのサイズを明示的に設定するには、次のように、width プロパティと height プロパティを使用します。

```
<?xml version="1.0"?>
<!-- components\ExplicitTextSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox id="myHBox">
        <mx:TextInput id="myInput"
            width="200"
            height="40"/>
    </mx:HBox>
</mx:Application>
```

この例では、コンポーネントのサイズが 200 x 40 ピクセルに設定されます。

次の例は、コンテナとその子のサイズの設定方法を示しています。

```
<?xml version="1.0"?>
<!-- components\ExplicitHBoxSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox id="myHBox" width="150" height="150">
        <mx:TextInput id="myInput"
            text="Enter the zip code"
            width="200"
            height="40"/>
    </mx:HBox>
</mx:Application>
```

TextInput コントロールのサイズは親 HBox コンテナのサイズよりも大きく設定されています。そのため、TextInput コントロールはコンテナの境界線でクリッピングされます。子のクリッピングされた内容は、スクロールバーを無効にしない限り、コンテナのスクロールバーを使用して表示できます。スクロールバーのサイズ設定に関する注意事項については、228 ページの「コンテナのサイズを超えるコンポーネントの処理」を参照してください。

パーセント値ベースのサイズ設定の使用

パーセント値ベースのサイズ設定では、コンポーネントのサイズがコンテナに対する相対的なサイズとして動的に決定され、維持されます。たとえば、コンポーネントの幅をコンテナの 75% とするように指定できます。このサイズ設定手法には、固定的なデフォルトサイズ設定または明示的サイズ設定と比べて次のような利点があります。

- コンテナに対する相対的なサイズを指定するだけで済みます。正確なサイズを決定する必要はありません。
- コンテナサイズの変化に応じてコンポーネントのサイズが動的に変更されます。

■ 残りの領域が自動的に考慮され、要求されたサイズがその領域を超えていても、コンポーネントが収まります。

パーセント値を指定するには、次のいずれかのコーディング手法を使用します。

■ MXML タグでは、次のように height プロパティまたは width プロパティをパーセント値に設定します。

```
<mx:TextArea id="ta1" width="70%" height="40%"/>
```

■ MXML タグまたは ActionScript ステートメントでは、次のように percentHeight プロパティまたは percentWidth プロパティを数値に設定します。

```
ta1.percentWidth=70;
```

パーセント値ベースのサイズ設定を使用するコンポーネントのサイズを決定する際に正確にどのような手法が採用されるかは、そのコンポーネントを格納するコンテナのタイプによって異なります。たとえば、Tile コンテナのセルはすべて、最大の子の最大のデフォルトサイズまたは明示的なサイズになります。子コントロールのパーセント値は、Tile コントロールのサイズではなく、タイルのセルサイズのパーセントを指定します。それに対して Box、HBox、および VBox コンテナのパーセント値サイズは、コンテナのサイズに対する相対的なサイズになります。

自動配置を使用するリニアコンテナでのパーセント値ベースの子のサイズ設定

Flex は、HBox や VBox コンテナなど、自動配置を使用して単一方向に子をレイアウトするコンテナの子のサイズを設定するときに、次のことを行います。

1. 親コンテナの可視領域のサイズを決定し、サイズ計算のためのコンテナのサイズとして、対応するサイズを使用します。可視領域とは、表示されているコンポーネントの、子コントロール、テキスト、イメージなどの内容を格納する部分を表します。コンテナサイズの計算の詳細については、217 ページの「コンポーネントのサイズの決定」を参照してください。
2. パーセント値でサイズ指定された子の必要なサイズを算出するため、コンテナの可視領域のサイズからパディングと子同士の間隔を引いた値に小数値を掛けます。
3. 明示的なサイズまたはデフォルトサイズを持つすべての子に対して領域を確保します。
4. パーセント値の要求が残りの領域 (親コンテナのサイズ - すべての確保済み領域。確保済み領域には境界線、パディング、および子同士の間隔が含まれます) に収まらない場合は、指定されたパーセント値に従って残りの領域を比例配分します。
5. 計算値が最小または最大の高さまたは幅の指定と衝突する場合は、最小値または最大値の方を使用し、その他すべてのパーセント値ベースのコンポーネントを、少なくなった残りの領域に基づいて再計算します。
6. サイズを最も近い整数に切り捨てます。

次の例は、要求されたパーセント値がコンポーネントのレイアウト時のサイズと異なる場合を示しています。

- 明示的なサイズが指定されたコンポーネントとデフォルトサイズのコンポーネントのすべてと、すべての間隔およびパディングに対して領域を確保した後、HBox 親コンテナの残りの領域が 50% であったとします。ここで、あるコンポーネントが親の 20% を要求し、別のコンポーネントが 60% を要求した場合、最初のコンポーネントのサイズは親コンテナの 12.5% ((20 / 20+60) * 50%) となり、2 番目のコンポーネントのサイズは親コンテナの 37.5% となります。
- いずれかのコンポーネント (たとえば Tile コンテナなど) が親 Application コンテナの領域の 100% を要求した場合、そのコンポーネントは、Application コンテナのパディング設定を明示的に変更しない限り、Application コンテナの上下左右 24 ピクセルのパディングを除く、コンテナのすべての領域を占めます。

絶対配置を使用するコンテナでのパーセント値ベースの子のサイズ設定

Flex は、絶対配置を使用するコンテナの子のサイズを設定するときに、次のことを行います。

1. 親コンテナの可視領域を決定し、サイズ計算のためのコンテナのサイズとして、対応するサイズを使用します。コンテナサイズの計算の詳細については、217 ページの「コンポーネントのサイズの決定」を参照してください。
2. パーセント値でサイズ指定された子のサイズを算出するため、コンテナのサイズからその方向のコントロールの位置を引いた値に小数値を掛けます。たとえば、子に対して `x="10"` と `width="100%"` を指定した場合、その子のサイズは可視領域の端まで伸びます。端を越えることはありません。

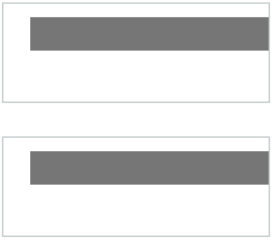
   子のサイズを決定するときにパディングや他の子は考慮されません。そのため、コントロールが他のコントロールやパディングと重なる場合があります。
3. 計算値が最小または最大の高さまたは幅の指定と衝突する場合は、最小値または最大値の方を使用します。
4. サイズを最も近い整数に切り捨てます。

次のコードは、絶対配置を使用したときのパーセント値ベースのサイズ設定動作を示しています。

```
<?xml version="1.0"?>
<!-- components\PercentSizeAbsPosit.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundGradientColors="[#FFFFFF, #FFFFFF]"
    verticalGap="25">

    <mx:Canvas
        width="200" height="75"
        borderStyle="solid">

        <mx:HBox
            x="20" y="10"
```

```
            width="100%" height="25"
            backgroundColor="#666666"/>
    </mx:Canvas>

    <mx:Canvas
        width="200" height="75"
        borderStyle="solid">

        <mx:HBox
            left="20" top="10"
            width="100%" height="25"
            backgroundColor="#666666"/>
    </mx:Canvas>
</mx:Application>
```

この例では、アプリケーションは次のように描画されます。



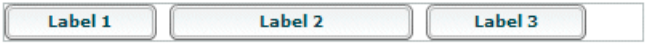
例 : HBox コンテナでのパーセント値ベースの子の使用

次の例では、HBox コンテナ内の 3 つのボタンのうち、最初の 2 つのボタンでパーセント値ベースのサイズを指定してします。

```
<?xml version="1.0"?>
<!-- components\PercentHBoxChildren.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="400">
        <mx:Button label="Label 1" width="25%"/>
        <mx:Button label="Label 2" width="40%"/>
        <mx:Button label="Label 3"/>
    </mx:HBox>
</mx:Application>
```

この例では、3 つ目のボタンのデフォルトの幅は 66 ピクセルになります。デフォルトでは、HBox コンテナにパディングはありませんが、コンポーネントの間隔は水平方向に 8 ピクセル空けられます。このアプリケーションには 3 つのコンポーネントがあるため、これらの間隔として 16 ピクセルが使用されます。最初のボタンは使用可能な領域の 25% (96 ピクセル) を要求します。2 つ目のボタンは 384 ピクセルの 40% (小数点以下を切り捨てて 153 ピクセル) を要求します。それでも 3 つ目のボタンの右には未使用の領域が残ります。

この例では、アプリケーションは次のように描画されます。

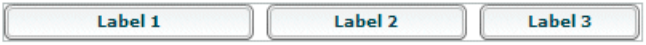


今度は、パーセント値をそれぞれ 50% と 40% に変更してみます。

```
<?xml version="1.0"?>
<!-- components\PercentHBoxChildren5040.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="400">
        <mx:Button label="Label 1"
            width="50%"/>
        <mx:Button label="Label 2"
            width="40%"/>
        <mx:Button label="Label 3"/>
    </mx:HBox>
</mx:Application>
```

この例では、1 つ目のボタンは HBox の使用可能な領域の 50% (192 ピクセル) を要求しています。2 つ目のボタンは 40% (153 ピクセル) を要求し、合計で 345 ピクセルとなります。しかし、デフォルト幅のボタンとコンポーネント間の間隔に必要な 66 ピクセルが既に確保されているので、HBox の残りの領域は 318 ピクセルしかありません。この場合、2 つのボタンで残りの領域が比例配分されます。1 つ目のボタンには .5/(.5 + .4) * 318 = 176 ピクセルが割り当てられ、2 つ目のボタンには .4/(.5 + .4) * 318 = 141 ピクセルが割り当てられます。計算値はすべて、最も近いピクセル数に切り捨てられます。

この例では、アプリケーションは次のように描画されます。



最小サイズまたは最大サイズの使用

パーセント値ベースのコンポーネントに minWidth、minHeight、maxWidth、maxHeight の各プロパティを使用して、サイズの上限と下限を指定することもできます。次の例を考えます。

```
<?xml version="1.0"?>
<!-- components\PercentHBoxChildrenMin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="400">
        <mx:Button label="Label 1"
            width="50%"/>
        <mx:Button label="Label 2"
            width="40%"
            minWidth="150"/>
        <mx:Button label="Label 3"/>
    </mx:HBox>
</mx:Application>
```

パーセント値ベースのボタンのサイズを決定するため、Flex はまず、225 ページの「例 : HBox コンテナでのパーセント値ベースの子の使用」の 2 番目の例で示された方法でサイズを計算します。その結果、最初のボタンの要求値は 176、2 つ目のボタンの要求値は 141 になります。しかし、2 つ目のボタンの最小の幅は 150 なので、そのサイズを 150 ピクセルに設定し、残りの領域を最初のボタンに割り当てます。その結果、最初のボタンの幅は 168 ピクセルになります。

この例では、アプリケーションは次のように描画されます。

コンテナとコンポーネントのサイズ設定に関するテクニック

次のセクションでは、サイズ設定を制御するためのテクニックとして、Application コンテナのサイズ設定、コンテナのサイズを超えるコンポーネントの処理、およびパディングとカスタム間隔の使用について説明します。

Application コンテナのサイズ設定

アプリケーションのサイズを設定するときは、まず Application コンテナのサイズを設定します。Application コンテナは、Adobe Flash Player 9 におけるアプリケーションの境界線を決定します。

Flex Builder を使用する場合、または MXML アプリケーションをサーバー上でコンパイルする場合は、HTML ラッパーページが自動的に生成されます。HTML ラッパーページでは、<mx:Application> タグで指定された width プロパティと height プロパティを使用して、<object> タグと <embed> タグの幅と高さが設定されます。これらの数値は、HTML ページの中の、Flash プラグインに割り当てられる部分を決定します。

HTML ラッパーを自動生成しない場合は、<mx:Application> タグの width プロパティと height プロパティを 100% に設定します。こうすると、Flash プラグインに割り当てられた領域に合わせて Flex アプリケーションが拡大 / 縮小します。

Application コンテナのサイズは、次のように、<mx:Application> タグを使用して設定します。

```
<?xml version="1.0"?>
<!-- components\AppExplicit.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="100"
    width="150">

        <!-- Application children go here. -->
</mx:Application>
```

この例では、Application コンテナのサイズを 100 x 150 ピクセルに設定しています。アプリケーションの中の、このウィンドウ領域を超える部分はすべて、ウィンドウの境界線でクリッピングされます。たとえば、200 x 200 ピクセルの DataGrid コントロールを定義した場合は、Application コンテナのサイズを超える部分がクリッピングされ、スクロールバーが表示されます。スクロールバーを無効にするか、または常に表示するには、コンテナの horizontalScrollPolicy プロパティと verticalScrollPolicy プロパティを設定します。

Application コンテナのサイズ設定の詳細については、495 ページ、第 14 章の「Application コンテナの使用」を参照してください。

コンテナのサイズを超えるコンポーネントの処理

コンテナの子の実際のサイズと間隔およびパディングの合計がコンテナのサイズを超えると、デフォルトでは、コンテナの内容がコンテナの境界線でクリッピングされます。また、コンテナにスクロールバーが表示されるため、スクロールして残りの内容を表示できます。horizontalScrollPolicy プロパティと verticalScrollPolicy プロパティを ScrollPolicy.OFF に設定すると、スクロールバーが表示されず、ユーザーはクリッピングされた内容にアクセスできなくなります。clipContent プロパティを false に設定すると、コンテナの内容がコンテナの境界を超えて描画されます。

スクロールバーの使用

コンテナの中にすべてのコンポーネントを同時に表示できない場合は、スクロールバーが使用されます。ただし、horizontalScrollPolicy プロパティまたは verticalScrollPolicy プロパティを ScrollPolicy.OFF に設定するか、clipContent プロパティを false に設定してスクロールバーを無効にした場合は別です。次の例を考えます。

```
<?xml version="1.0"?>
<!-- components\ScrollHBox.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="400">
        <mx:Button label="Label 1"
            width="50%"
            minWidth="200"/>
        <mx:Button label="Label 2"
            width="40%"
            minWidth="150"/>
        <mx:Button label="Label 3"/>
    </mx:HBox>
</mx:Application>
```

この例では、固定サイズのボタンの幅は 66 ピクセルなので、コンポーネントの間隔を考慮した結果、パーセント値ベースのボタンで使用可能な領域は 324 ピクセルになります。最初のボタンと 2 つ目のボタンの最小幅がパーセント値ベースの値より大きいため、HBox コンテナの使用可能な領域が 324 ピクセルしかないにもかかわらず、これらのボタンにそれぞれ 200 ピクセルと 150 ピクセルの幅が割り当てられます。これらの値は HBox コンテナ自体のサイズを超えるため、HBox コンテナにスクロールバーが表示されます。

スクロールバーが追加されても、コンテナの高さが初期値から増加していないことに注目してください。サイズ計算時にスクロールバーが考慮されるのは、スクロールポリシーを明示的に ScrollPolicy.ON に設定した場合のみに限られます。そのため、デフォルトの auto スクロールポリシーを使用した場合、スクロールバーはボタンに重なって表示されます。これを防ぐには、HBox コンテナの height プロパティを設定するか、またはパーセント値ベースの幅を設定して HBox コンテナのサイズを変更できるようにします。HBox コンテナの高さを変更した場合、アプリケーション内の他のコンポーネントが、サイズ設定規則に従って移動およびサイズ変更されるという点に注意する必要があります。次の例では、高さを明示的に指定することによって、ボタンとスクロールバーの両方が同時に表示されるようにしています。

```
<?xml version="1.0"?>
<!-- components\ScrollHBoxExplicitHeight.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="400" height="42">
        <mx:Button label="Label 1"
            width="50%"
            minWidth="200"/>
        <mx:Button label="Label 2"
            width="40%"
            minWidth="150"/>
        <mx:Button label="Label 3"/>
    </mx:HBox>
</mx:Application>
```

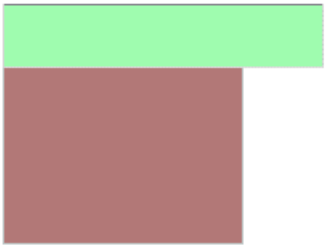
別の方法として、HBox コントロールの horizontalScrollPolicy プロパティを ScrollPolicy.ON に設定する方法もあります。こうすると、初期レイアウトパスでスクロールバーに必要な領域が確保されるため、明示的に高さを設定しなくても、スクロールバーがボタンに重なることはありません。また、スキンやスタイルの変更時にスクロールバーのサイズが変化するような状況でも正しく処理されます。ただし、スクロールが必要でない場合にも、コンテナに空白のスクロールバー領域が確保されます。

clipContent プロパティの使用

次のように、親コンテナの clipContent プロパティを false に設定すると、親の境界線を越えて内容を描画できます。この場合、スクロールバーは表示されません。

```
<?xml version="1.0"?>
<!-- components\ClipHBox.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="600"
    height="400"
    backgroundGradientColors="[#FFFFFF, #FFFFFF]">

    <mx:HBox id="myHBox"
        width="150"
        height="150"
        borderStyle="solid"
        backgroundColor="#996666"
        clipContent="false">

        <mx:TextInput id="myInput"
            width="200" height="40"
            backgroundColor="#99FF99"/>
    </mx:HBox>
</mx:Application>
```

次の図は、TextInput コントロールが HBox コントロールの右端を越えて描画された場合を示しています。



コンポーネントがコンテナの中に確実に収まるようにするには、子コンポーネントのサイズを小さくします。そのためには、コンテナに収まる明示的なサイズを設定するか、またはパーセント値ベースのサイズを指定します。パーセント値ベースのサイズを設定した場合、その子は、残りの領域と最小サイズのいずれか大きい方に収まるように縮小されます。デフォルトでは、ほとんどのコンポーネントにおいて、その幅と高さの最小値は 0 に設定されます。コンポーネントの最小値プロパティをゼロ以外の値に設定することで、コンポーネントの表示領域を確保できます。

パディングとカスタム間隔の使用

コンテナの端にパディングを設定したい場合があります。以前のリリースの Flex では " 余白 " という用語を使用していましたが、Flex 2 ではカスケーディングスタイルシートの表記に合わせて " パディング " という用語を使用しています。Application コンテナなど、一部のコンテナでは、デフォルトでパディングが設定されています。その他の HBox コンテナなどでは、デフォルトで 0 のパディングが設定されています。また、一部のコンテナには子の間隔という属性があり、これをデフォルト値から変更することもできます。アプリケーションに 0 以外のパディングと間隔が設定されている場合は、パーセント値ベースのコンポーネントのサイズを設定する前に必要なピクセル数が確保されます。次の例を考えます。

```
<?xml version="1.0"?>
<!-- components\PadHBox.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox
        width="400"
        borderStyle="solid"
        paddingLeft="5"
        paddingRight="5"
        horizontalGap="5">

        <mx:Button label="Label 1"
            width="50%"/>
        <mx:Button label="Label 2"
            width="40%"
            minWidth="150"/>
        <mx:Button label="Label 3"/>
    </mx:HBox>
</mx:Application>
```

固定サイズのボタンのデフォルト幅は 66 ピクセルです。HBox コントロールの水平方向のパディングと間隔がすべて 5 ピクセル幅に設定されているため、左右のパディングとしてそれぞれ 5 ピクセルと、コンポーネントの 2 つの間隔として計 10 ピクセルが確保され、パーセント値ベースの 2 つのコンポーネントで使用可能な領域は 314 ピクセルになります。デフォルトサイズのボタン (3 つ目のボタン) には 66 ピクセルが確保されます。2 つ目のボタンは最小サイズとして 150 ピクセルを要求し、パディングと間隔には 20 ピクセルが割り当てられます。したがって、最初のボタンで使用可能な領域は 164 ピクセルとなります。最初のボタンは 200 ピクセルを要求しているため、残りの領域がすべて割り当てられ、164 ピクセルの幅になります。

この例では、アプリケーションは次のように描画されます。

コントロールの配置とレイアウト

デフォルトでは、Canvas コンテナの子を除くすべてのコンポーネントが自動的に配置されます。Canvas コンテナ、または `layout` プロパティが `absolute` に設定されている Application コンテナまたは Panel コンテナを使用する場合は、その子に絶対位置を指定するか、制約ベースのレイアウトを使用します。以降のセクションでは、自動配置の使用方法と、`x` および `y` プロパティを使用した絶対配置の使用方法について説明します。配置とサイズ設定を両方とも制御できる制約ベースのレイアウトを使用する際の詳細については、238 ページの「制約ベースのレイアウトの使用」を参照してください。

自動配置の使用

ほとんどのコンテナでは、レイアウト方向、コンテナのパディング、コンテナの子同士の間隔など、コンテナのレイアウト規則に従って、コンテナの子が自動的に配置されます。

自動配置を使用するコンテナでは、`x` プロパティまたは `y` プロパティを直接設定するか、`move()` を呼び出しても、何も効果がないか、効果があっても一時的に過ぎません。これは、これらのコンテナの x 位置が、指定値ではなくレイアウト計算によって設定されるためです。ただし、場合によっては、これらの子の絶対位置を指定できることもあります。詳細については、233 ページの「自動配置の一時的な無効化」を参照してください。

コンテナのプロパティを指定してレイアウト属性を制御できます。プロパティの詳細については、『Adobe Flex 2 リファレンスガイド』にあるコンテナのプロパティ説明を参照してください。コンポーネントのサイズを制御するか、スペーサの追加などの手法を使用してレイアウトを制御することも可能です。

以降のセクションでは、自動配置を制御するための手法について説明します。

- Spacer コントロールを使用したレイアウトの制御
- 自動配置の一時的な無効化
- 非表示コントロールのレイアウトの防止

Spacer コントロールを使用したレイアウトの制御

Flex には、親コンテナ内での子のレイアウトに役立つ Spacer コントロールが用意されています。Spacer コントロールは実際には表示されませんが、他のコントロールと同様、親の内側の領域が確保されます。

次の例では、Button コントロールを HBox コンテナの右端に配置するため、パーセント値ベースの Spacer コントロールを使用して、Button コントロールを右側に寄せています。

```
<?xml version="1.0"?>
<!-- components\SpacerHBox.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="400">
        <mx:Image source="assets/flexlogo.jpg"/>
        <mx:Label text="Company XYZ"/>
        <mx:Spacer width="100%"/>
        <mx:Button label="Close"/>
    </mx:HBox>
</mx:Application>
```

この例では、HBox コンテナに含まれるパーセント値ベースのコンポーネントは Spacer コントロールだけです。そのため、Spacer コントロールは、他のコンポーネントが必要としていない HBox コンテナのすべての領域を占めます。Flex は、Spacer コントロールを拡大することによって、Button コントロールをコンテナの右端に寄せています。

Spacer コントロールでは、`width`、`height`、`maxWidth`、`maxHeight`、`minWidth`、`minHeight` など、サイズ設定や配置に関するすべてのプロパティを使用できます。

自動配置の一時的な無効化

Move や Zoom などのエフェクトを使用し、ユーザーアクションに基づいて子のサイズや位置を変更できます。たとえば、ユーザーが子を選択したときに、その子をコンテナの上部に移動し、サイズを 2 倍に拡大することも可能です。このようなエフェクトでは、その実行過程で子の x プロパティと y プロパティが変更されます。同様に、ボタンのクリックなどに応じてコントロールの x 座標値または y 座標値を変更することで、コントロールの位置を変更することもできます。

自動配置を使用するコンテナでは、レイアウトの更新中、子の x プロパティと y プロパティの値は無視されます。つまり、エフェクトの実行過程で行われた x プロパティと y プロパティに対する変更は、レイアウト更新時にすべて破棄され、子の新しい位置は維持されません。

コンテナの `autoLayout` プロパティを `false` に設定すると、期待した動作と矛盾するような自動配置の更新を防ぐことができます。このプロパティを `false` に設定すると、子の移動またはサイズ変更時にコンテナの内容はレイアウトされません。`autoLayout` プロパティは、Container クラスで定義されているため、この設定はすべてのコンテナに継承されます。デフォルト値は `true` で、レイアウトの更新が有効になっています。

コンテナの autoLayout プロパティを false に設定した場合でも、コンテナに子を追加するか、コンテナから子を削除したときは、レイアウトが更新されます。アプリケーションの初期化、遅延インスタンス化、および <mx:Repeater> タグは子の追加または削除を伴うため、これらのプロセスでは、autoLayout プロパティの値とは無関係に、必ずレイアウトの更新が実行されます。したがって、コンテナの初期化時には、autoLayout プロパティの値にかかわらず、コンテナの子の初期レイアウトが自動的に定義されます。

次の例は、VBox コンテナのレイアウト更新を無効にしています。

```
<?xml version="1.0"?>
<!-- components\DisableVBoxLayout.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox autoLayout="false"
        width="200"
        height="200">

        <mx:Button/>
        <mx:Button id="btn"
            click="btn.x += 10;"/>
        <mx:Button id="btn2"
            creationComplete="btn2.x = 100; btn2.y = 75;"/>
    </mx:VBox>
</mx:Application>
```

この例では、3 つの Button コントロールはすべて、初期化時に VBox コンテナの規則に従ってレイアウトされます。VBox コントロールの子がレイアウトされた後、ボタンが表示される前に、3 つ目のボタンの creationComplete イベントリスナーが送出されます。そのため、3 つ目のボタンが表示されるとき、その位置は creationComplete リスナーで指定された x と y の位置になります。ボタンがすべて表示された後、ユーザーが 2 つ目のボタンをクリックすると、そのたびに 2 つ目のボタンが右に 10 ピクセルずつ移動します。

コンテナの autoLayout プロパティを false に設定すると、子を移動またはサイズ変更した後にコンテナのレイアウトが更新されなくなります。どうしても必要な場合以外は、このプロパティを false に設定するのは避けてください。まず、autoLayout プロパティをデフォルト値の true に設定してアプリケーションをテストし、特定のコンテナおよびその子の特定のアクションに対して必要な場合にだけ、false に設定することをお勧めします。

エフェクトの詳細については、603 ページ、第 17 章の「ビヘイビアの使用」を参照してください。

非表示コントロールのレイアウトの防止

デフォルトでは、レイアウトと領域の確保は非表示コンポーネントを含むすべてのコンポーネントに対して行われます。ただし、非表示コントロールは表示されません。非表示コントロールを表示可能にしたときにそれが表示される場所は、空白の領域として表示されます。非表示コントロールに対応する場所には、代わりにコンテナの背景が表示されます。ただし、コンテナが次のいずれかのコンポーネントである場合は、コンポーネントの子コンポーネントの includeInLayout プロパティを false に設定することで、コンテナ内の他のコンポーネントをレイアウトするときにその子コンポーネントが考慮されないようにすることができます。

- Box、またはそのサブクラスのいずれか : HBox、VBox、DividedBox、HDividedBox、VdividedBox、Grid、GridItem、GridRow、ControlBar、ApplicationControlBar
- Form
- Tile とそのサブクラスの Legend
- ToolBar

コンポーネントの includeInLayout プロパティが false である場合、そのコンポーネントは他のコンポーネントのレイアウト計算には含められませんが、レイアウトは行われます。つまり、そのコンポーネントの領域が確保されないまま、描画されます。結果として、そのコンポーネントは、レイアウト順では後続のコンポーネントの下に表示される可能性があります。そのコンポーネントが描画されないようにするには、visible プロパティも false に設定する必要があります。

次の例は、includeInLayout プロパティと visible プロパティの効果を示しています。この例では、VBox コントロールの真ん中の Panel コントロールで、両方のプロパティを個別に切り替えることができます。

```
<?xml version="1.0"?>
<!-- components\HiddenBoxLayout.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox>
        <mx:Panel id="p1"
            title="Panel 1"
            backgroundColor="#FF0000"/>
        <mx:Panel id="p2"
            title="Panel 2"
            backgroundColor="#00FF00"/>
        <mx:Panel id="p3"
            title="Panel 3"
            backgroundColor="#0000FF"/>
    </mx:VBox>

    <mx:HBox>
        <mx:Button label="Toggle Panel 2 Visible"
            click="{p2.visible=!p2.visible;}"/>
        <mx:Button label="Toggle Panel 2 in Layout"
            click="{p2.includeInLayout=!p2.includeInLayout;}"/>
    </mx:HBox>
</mx:Application>
```

このアプリケーションを実行してボタンをクリックすると、visible プロパティと includeInLayout プロパティの組み合わせによって結果がどのようになるかを確認できます。この例は次のように動作します。

- レイアウトに 2 つ目の Panel コントロールを含め、そのコントロールを不可視にすると、コントロールの領域が確保され、その場所に VBox コンテナの背景が表示されます。
- レイアウトに 2 つ目の Panel コントロールを含めない場合、VBox のサイズが変更され、HBox がボタンとともに上に移動します。その後、レイアウトに 2 つ目の Panel コントロールを含めると、VBox のサイズが再度変更され、HBox とボタンが下に移動します。
- レイアウトに 2 つ目の Panel コントロールを含めずに、そのコントロールを可視にすると、コントロールは描画されますが、3 つ目の Panel コントロールのレイアウト時に考慮されないため、2 つのパネルが重なって表示されます。Panel コントロールのタイトルのアルファはデフォルトで 0.5 に設定されているため、2 つ目の Panel の位置で 2 つ目の Panel コントロールと 3 つ目の Panel コントロールが合成されされます。

絶対配置の使用

絶対配置は次の 3 つのコンテナでサポートされています。

- Application コントロールと Panel コントロールは、layout プロパティを "absolute" (ContainerLayout.ABSOLUTE) に設定した場合、絶対配置を使用します。
- Canvas コンテナは常に絶対配置を使用します。

絶対配置では、x プロパティと y プロパティを使用して子コントロールの位置を指定するか、制約ベースのレイアウトを指定します。それ以外の場合、子は親コンテナの 0,0 の位置に配置されます。x 座標と y 座標を指定した場合は、プロパティ値の変更時にのみコントロールが再配置されます。次の例は、絶対配置を使用して VBox コントロールを Canvas コントロール内に配置しています。

```
<?xml version="1.0"?>
<!-- components\CanvasLayout.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundGradientColors="[#FFFFFF, #FFFFFF]">

    <mx:Canvas
        width="100" height="100"
        backgroundColor="#999999">

        <mx:VBox id="b1"
            width="80" height="80"
            x="20" y="20"
            backgroundColor="#A9C0E7">
        </mx:VBox>
    </mx:Canvas>
</mx:Application>
```

この例では、次のイメージが作成されます。

絶対配置を使用すると、コンテナの子の位置を完全に制御できます。そのため、コンポーネントを重ねることも可能です。次の例は、前の例に 2 つ目の VBox を追加して、最初のボックスと部分的に重ねて表示しています。

```
<?xml version="1.0"?>
<!-- components\CanvasLayoutOverlap.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundGradientColors="[#FFFFFF, #FFFFFF]">

    <mx:Canvas
        width="100" height="100"
        backgroundColor="#999999">

        <mx:VBox id="b1"
            width="80" height="80"
            x="20" y="20"
            backgroundColor="#A9C0E7">
        </mx:VBox>

        <mx:VBox id="b2"
            width="50" height="50"
            x="0" y="50"
            backgroundColor="#FF0000">
        </mx:VBox>
    </mx:Canvas>
</mx:Application>
```

この例では、次のイメージが作成されます。

メモ

絶対配置を使用するコントロールの子に対してパーセント値ベースのサイズ設定を使用する場合は、親コンテナがサイズ変更されるとパーセント値ベースのコンポーネントもサイズ変更され、その結果、コントロールの無用な重複が起こる可能性がある点に注意してください。

制約ベースのレイアウトの使用

子コンポーネントのサイズと位置を同時に管理するには、制約ベースのレイアウトを使用します。この場合は、コンポーネントの端または中心をコンポーネントのコンテナの可視領域に対して相対的に固定します。可視領域とは、表示されているコンポーネントの、子コントロール、テキスト、イメージなどの内容を格納する部分を表します。

メモ

Flex Builder で制約ベースのレイアウトを使用する際の概要については、『Flex 2 ファーストステップガイド』を参照してください。

制約ベースのレイアウトを使用すると、絶対配置をサポートしている任意のコンテナの直下の子の位置とサイズを決定できます。

制約ベースのレイアウトでは、次のことができます。

- コンポーネントの端をコンテナの可視領域の対応する端から指定したピクセル離れた位置に固定します。固定された子の端は、コンテナのサイズが変更されても、親の端から等距離に維持されます。上端と下端などの両端を固定すると、コンテナのサイズが変更されたときにコンポーネントのサイズも変更されます。
- 子の水平方向または垂直方向の中心をコンテナの可視領域の中心から指定したピクセル離れた位置に固定します。この場合、パーセント値ベースのサイズ設定も併用しない限り、子のサイズは指定されたサイズから変わりません。

制約ベースのレイアウトの作成

制約ベースのレイアウトを使用してコンポーネントの位置とサイズを決定する規則を次に示します。

- Canvas コンテナ、または `layout` プロパティが `absolute` に設定されている Application コンテナまたは Panel コンテナの中にコンポーネントを直接配置します。
- 制約ベースのレイアウトは、任意の Flex フレームワークコンポーネント、つまり UIComponent クラスを拡張している任意のコンポーネントに対して指定できます。
- 制約を指定するときは、`top`、`bottom`、`left`、`right`、`horizontalCenter`、`verticalCenter` のいずれかのスタイルを使用します。

  `top`、`bottom`、`left`、`right` の各スタイルでは、コンポーネントの端と対応するコンテナの端との距離を指定します。

  `horizontalCenter` スタイルと `verticalCenter` スタイルでは、コントロールの中心点とコンテナの中心との特定の方向の距離を指定します。負の数値を指定すると、コンポーネントが中心から左または上に移動します。

  次の例は、Form コントロールの左右の端をコンテナの端から 20 ピクセル離れた位置に固定しています。

```
<mx:Form id="myForm" left="20" right="20"/>
```

- top スタイルまたは bottom スタイルを verticalCenter スタイルと同時に指定しないでください。verticalCenter の値は他のプロパティをオーバーライドします。同様に、left スタイルまたは right スタイルを horizontalCenter スタイルと同時に指定しないでください。
- 制約ベースのレイアウトによって決定されたサイズは、明示的なサイズ指定またはパーセント値ベースのサイズ指定をオーバーライドします。たとえば、left 制約と right 制約を指定すると、制約ベースによって決定された幅により、width プロパティまたは percentWidth プロパティで設定された幅がオーバーライドされます。

例 : フォームに対する制約ベースのレイアウトの使用

次のコード例は、フォームに対する制約ベースのレイアウトの使用方法を示しています。この例では、Form コントロールで制約ベースのレイアウトを使用して、その上端をキャンバスのパディング内のすぐ下に配置しています。フォームの左端と右端は、Canvas コンテナの左右の端から 20 ピクセル離れた位置になります。ボタンを含む HBox は、制約ベースのレイアウトを使用して、Canvas の右端から 20 ピクセル、Canvas の下端から 10 ピクセル離れた位置に配置されます。

ブラウザまたはスタンドアローン Flash Player のサイズを変更すると、Form レイアウト上の Application コンテナのサイズが動的に変更される効果が見られます。たとえば、アプリケーションのサイズが小さくなるにつれて、フォームとボタンが重なります。アプリケーションでは、ボタンはフォームの最後の FormItem に格納してください。次の例では、サイズ変更の効果がよくわかるように、意図的にボタンをフォームから切り離しています。

```
<?xml version="1.0"?>
<!-- components\ConstraintLayout.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Use a Canvas container in the Application to prevent
        unnecessary scroll bars on the Application. -->
    <mx:Canvas width="100%" height="100%">

         <!-- Anchor the top of the form at the top of the canvas.
            Anchor the form sides 20 pixels from the canvas sides. -->
        <mx:Form id="myForm"
                backgroundColor="#DDDDDD"
                top="0"
                left="20"
                right="20">

            <mx:FormItem label="Product:" width="100%">
                <!-- Specify a fixed width to keep the ComboBox control from
                    resizing as you change the application size. -->
                <mx:ComboBox width="200"/>
            </mx:FormItem>

            <mx:FormItem label="User" width="100%">
```

```
            <mx:ComboBox width="200"/>
        </mx:FormItem>

        <mx:FormItem label="Date">
            <mx:DateField/>
        </mx:FormItem>

        <mx:FormItem width="100%"
                direction="horizontal"
                label="Hours:">
            <mx:TextInput width="75"/>
            <mx:Label text="Minutes" width="48"/>
            <mx:TextInput width="75"/>
        </mx:FormItem>
    </mx:Form>

    <!-- Anchor the box with the buttons 20 pixels from the canvas
        right edge and 10 pixels from the bottom. -->
    <mx:HBox id="okCancelBox"
            right="20"
            bottom="10">
        <mx:Button label="OK"/>
        <mx:Button label="Cancel"/>
    </mx:HBox>
  </mx:Canvas>
</mx:Application>
```

第 9 章

コントロールの使用

9

コントロールとは、Button、TextArea、ComboBox コントロールなどのユーザーインターフェイスコンポーネントです。このトピックでは、Flex アプリケーションでのコントロールの使用方法について説明します。

Adobe Flex には、基本コントロールとデータプロバイダコントロールの 2 種類のコントロールがあります。このトピックでは、Flex のすべてのコントロールの概要を示し、基本コントロールについて詳しく説明します。データプロバイダコントロールの詳細については、411 ページ、第 12 章の「データ駆動型コントロールの使用」を参照してください。

目次

コントロールについて

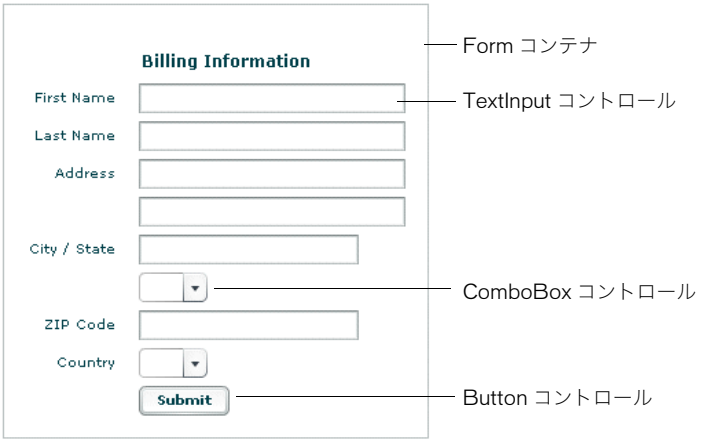
" コントロール " とは、Button、TextArea、ComboBox コントロールなど、ユーザーインターフェイス用のコンポーネントのことです。コントロールは、同じくユーザーインターフェイス用のコンポーネントで、コントロールや他のコンテナに対する階層構造を提供するコンテナに配置されます。コンテナを定義した後で、そこに含めるコントロールや他のコンテナを挿入するのが一般的です。

Flex アプリケーションのルートに位置する <mx:Application> タグは、Flash Player の描画面全体をカバーする基本コンテナを表します。コントロールまたはコンテナは、<mx:Application> タグの直下に配置することも、他のコンテナに配置することもできます。コンテナの詳細については、461 ページ、第 13 章の「コンテナについて」を参照してください。

コントロールの多くは、次のような特徴を備えています。

- コントロールとそのプロパティやイベントの値を宣言するための MXML API
- コントロールのメソッドを呼び出したり、そのプロパティを実行時に設定するための ActionScript API
- スタイル、スキン、フォントなどを使用してカスタマイズ可能な外観

次の図では、Form コンテナにいくつかのコントロールが使用されています。



コントロールの作成と設定は、MXML および ActionScript の API を使用して実行できます。次の MXML コードは、Form コンテナに TextInput コントロールを作成する例です。

```
<?xml version="1.0"?>
<!-- controls\TextInputInForm.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form width="300" height="100">
        <mx:FormItem label="Card Name">
            <mx:TextInput id="cardName"/>
        </mx:FormItem>
```

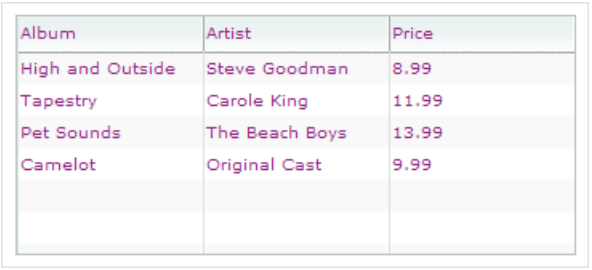
```
    </mx:Form>
</mx:Application>
```

この例では、次のイメージが作成されます。

Flex アプリケーションを構築するための言語としては MXML が一般的ですが、ActionScript を使用してコントロールを設定することもできます。たとえば、次のコード例では、アイテムの配列を DataGrid コントロールの dataProvider プロパティに設定することにより、DataGrid コントロールにデータを挿入しています。

```
<?xml version="1.0"?>
<!-- controls\DataGridConfigAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

        private function myGrid_initialize():void {
          myGrid.dataProvider = [
            {Artist:'Steve Goodman', Album:'High and Outside', Price:8.99},
            {Artist:'Carole King', Album:'Tapestry', Price:11.99},
            {Artist:'The Beach Boys', Album:'Pet Sounds', Price:13.99},
            {Artist:'Original Cast', Album:'Camelot', Price:9.99} ];
        }
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        width="350" height="150"
        color="#7B0974"
        creationComplete="myGrid_initialize();"/>
</mx:Application>
```

この例では、次のイメージが作成されます。



テキストコントロールについて

次の表に示すように、いくつかの Flex コンポーネントはテキストを表示するか、テキスト入力を受け取ります。

コントロール	テキストの種類
ラベル	編集不可能な単一行のテキストフィールド
Text	編集不可能な複数行のテキストフィールド
TextInput	(オプション) 編集可能な単一行のテキストフィールド
TextArea	(オプション) 編集可能な複数行のテキストフィールド
RichTextEditor	複数行のテキストフィールド、およびフォント、サイズ、太さ、整列などの特性を選択することで、ユーザーにテキストのフォーマットを可能にするコントロールを含む複合コントロール

これらのコントロールは、すべてが同じ外観のプレーンテキストを表示できます。フォーマット用の標準 HTML タグのサブセットを使用してフォーマットした、リッチテキストも表示できます。テキストコントロールの使用方法については、343 ページ、第 10 章の「テキストコントロールの使用」を参照してください。

データプロバイダコントロールの使用

DataGrid、Tree、ComboBox コントロールなど、いくつかの Flex コンポーネントは、データプロバイダから入力データを受け取ります。データプロバイダは配列と同様、アイテムのコレクションです。たとえば、Tree コントロールはデータプロバイダからデータを読み取ることによって、ツリー構造と各ツリーノードに割り当てられたデータを定義します。

データプロバイダでは、Flex コンポーネントとそのコンポーネントの生成に使用するデータのと間に抽象レベルが作成されます。同じデータプロバイダから複数のコンポーネントを作成できます。また、実行時にコンポーネントに使用するデータプロバイダを切り替えることもできます。さらに、データプロバイダを変更した場合に、そのデータプロバイダを使用するすべてのコンポーネントに変更が反映されるようにすることもできます。

データプロバイダをモデルとすれば、Flex コンポーネントはそのモデルに対するビューと考えることができます。モデルとビューを分離することにより、変更が生じたときに両方を修正するのではなく、どちらか一方だけを修正するだけで済むというメリットが生まれます。

このトピックでは、基本コントロールについて説明します。データプロバイダコントロールの詳細については、411 ページ、第 12 章の「データ駆動型コントロールの使用」を参照してください。

メニューコントロールの使用

次の表に示すように、いくつかの Flex コントロールはメニューを作成するか、操作します。

コントロール	説明
Menu	カスケーディングサブメニューを設定できるビジュアルメニュー
MenuBar	複数のサブメニューを持つ水平バー
PopUpMenuButton	ボタンをクリックすると開くメニューコントロール

メニューコントロールの詳細については、381 ページ、第 11 章の「メニューベースのコントロールの使用」を参照してください。

Flex のコントロール

次の表に、Flex で使用できるすべてのコントロールを示します。

コントロール	説明	詳細情報
Alert	警告のポップアップを表示します。	327 ページの「Alert コントロール」
ボタン	可変サイズのボタンを表示します。ラベルやアイコンイメージを追加できます。	251 ページの「Button コントロール」
ButtonBar	共通の外観を持つ関連するボタンの列を表示します。	259 ページの「ButtonBar コントロールと ToggleButtonBar コントロール」
CheckBox	特定の Boolean 値が `true` (チェックがオンの状態) であるか、`false` (チェックがオフの状態) であるかを示します。	268 ページの「CheckBox コントロール」
ComboBox	一連の値を保持するテキストフィールドに関連付けられたドロップダウンリストを表示します。	429 ページの「ComboBox コントロール」
ColorPicker	選択可能なドロップダウンカラー色見本パネル (パレット) を表示します。	276 ページの「DateChooser コントロールと DateField コントロール」
DataGrid	データを表形式で表示します。	438 ページの「DataGrid コントロール」
DateChooser	日付を選択できるような 1 か月単位のカレンダーを表示します。	276 ページの「DateChooser コントロールと DateField コントロール」

コントロール	説明	詳細情報
DateField	日付を表示します。右側にカレンダーアイコンが表示されます。このコントロール内の任意の場所をクリックすると、DateChooser コントロールの 1 か月単位のカレンダーがポップアップ表示されます。	276 ページの「DateChooser コントロールと DateField コントロール」
HorizontalList	アイテムの一覧を水平方向に並べて表示します。	421 ページの「HorizontalList コントロール」
HRule/VRule	水平方向 (HRule) または垂直方向 (VRule) の罫線を 1 本表示します。	336 ページの「HRule コントロールと VRule コントロール」
HSlider/VSlider	スライダトラックのいずれかの方向にサムを移動することによって値を選択できます。	289 ページの「HSlider コントロールと VSlider コントロール」
イメージ	GIF、JPEG、PNG、SVG、および SWF ファイルをインポートします。	301 ページの「Image コントロール」
Label	単一行のフィールドラベルを表示します。編集することはできません。	286 ページの「LinkButton コントロール」
LinkBar	一連のリンク先を指定する LinkButton コントロールの水平行を表示します。	262 ページの「LinkBar コントロール」
LinkButton	シンプルなハイパーリンクを表示します。	286 ページの「LinkButton コントロール」
List	スクロール可能な一連の選択肢を表示します。	412 ページの「List コントロール」
Menu	ほとんどのソフトウェアに実装されたファイルメニューや編集メニューと同様の、選択肢を含むポップアップメニューを表示します。	389 ページの「Menu コントロールのイベントの処理」
MenuBar	1 つまたは複数の Menu コントロールを含む水平メニューバーを表示します。	402 ページの「MenuBar コントロール」
NumericStepper	値を増減させるための 2 つのボタンが一体となったコントロールを表示します。	275 ページの「NumericStepper コントロール」
ProgressBar	現在の操作に要する残り時間を視覚的に表示します。	332 ページの「ProgressBar コントロール」
RadioButton	常にいずれか 1 つだけが選択状態になる複数のボタンを表示します。	270 ページの「RadioButton コントロール」
RadioButton Group	click イベントリスナーを 1 つだけ実装した RadioButton コントロールのグループを表示します。	272 ページの「<mx:RadioButtonGroup> タグを使用したグループの作成」

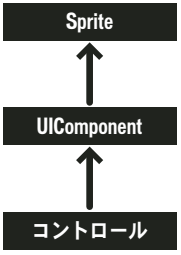
コントロール	説明	詳細情報
RichTextEditor	複数行に対応した編集可能なテキストフィールド、およびテキストフォーマットを指定するコントロールがあります。	372 ページの「RichTextEditor コントロール」
ScrollBar (HScrollBar および VScrollBar)	水平スクロールバーおよび垂直スクロールバーを表示します。	340 ページの「ScrollBar コントロール」
SWFLoader	指定された SWF ファイルまたは JPEG ファイルのコンテンツを表示します。	296 ページの「SWFLoader コントロール」
TabBar	タブを水平方向に並べて表示します。	264 ページの「TabBar コントロール」
Text	複数行のテキストフィールドを表示します。編集することはできません。	367 ページの「TextInput コントロール」
TextArea	ユーザー入力用の編集可能なテキストフィールドを表示します。複数の行を入力することもできます。	343 ページの「テキストコントロールの使用」
TextInput	ユーザー入力用の編集可能なテキストフィールドを表示します。単一行のみ入力できます。英数字データを保持することもできますが、入力データは String データ型として解釈されます。	367 ページの「TextInput コントロール」
TileList	アイテムの一覧をタイル状に並べて表示します。アイテムは垂直列または水平行の向きで順に配置されます。	425 ページの「TileList コントロール」
ToggleButtonBar	共通の外観を持つ関連するボタンの列を表示します。	259 ページの「ButtonBar コントロールと ToggleButtonBar コントロール」
Tree	展開可能なツリーとして配置された階層データを表示します。	449 ページの「Tree コントロール」
VideoDisplay	ストリーミングメディアを Flex アプリケーションに組み込みます。	312 ページの「VideoDisplay コントロール」

コントロールの操作

Flex のコントロールは、共通のクラス階層を共有しています。したがって、すべてのコントロールは同様の手順で設定することができます。このセクションでは、次のトピックについて説明します。

コントロールのクラス階層

次の図に示したように、Flex コントロールは flash.display.Sprite クラスと mx.core.UIComponent クラスから派生した ActionScript オブジェクトです。コントロールは、それらのスーパークラスのプロパティ、メソッド、イベント、スタイル、エフェクトを継承します。



Sprite クラスと UIComponent クラスは、すべての Flex コンポーネントの基本クラスです。UIComponent クラスのサブクラスは形状を持つことができ、それ自体を表示することも非表示にすることもできます。各サブクラスはタブ順序の管理に対応し、低レベルのイベント (キーボードやマウスによる入力など) を受け付けます。また、キーボードやマウスによる入力を受け付けないように無効にすることができます。

コントロールが Sprite クラスと UIComponent クラスから継承するインターフェイスの詳細については、127 ページ、第 6 章の「Flex ビジュアルコンポーネントの使用」を参照してください。

コントロールのサイズ設定

このセクションでは、コントロールのサイズ設定方法について説明します。コンポーネントのサイズの詳細については、207 ページ、第 8 章の「コンポーネントのサイズと位置の制御」を参照してください。

すべてのコントロールには、Flex アプリケーションにおけるサイズを決定するための規則が定義されています。たとえば、Button コントロールの場合は、ラベルテキストとオプションのアイコンイメージを表示できるようなサイズが基準になります。同様に、Image コントロールの場合は、読み込むイメージの大きさに合わせてサイズが計算されます。各コントロールには、デフォルトの高さと幅があります。各標準コントロールのデフォルトサイズについては、各コントロールの説明を参照してください。

コントロールのデフォルトサイズは、固定値とは限りません。たとえば、Button コントロールの場合は、ラベルテキストとオプションのアイコンイメージを表示できるだけの大きさがデフォルトサイズになります。Flex は、実行時に各コントロールのデフォルトサイズを計算します。デフォルトでは、コントロールのサイズがデフォルトサイズから変更されることはありません。

MXML で `height` 属性または `width` 属性を 50% などのパーセントに設定するか、ActionScript で `percentHeight` プロパティまたは `percentWidth` プロパティを 50 などのパーセント値に設定すると、コントロールがそれぞれ縦方向または横方向に自動的にサイズ変更されます。つまり、Flex が、親コンテナに対するパーセント値として指定されたサイズでコントロールを収めようと試みます。コントロールに対して割り当てるだけの十分な領域が存在しない場合、指定された比率は維持しながら、残された領域が比例配分されます。

次のようにして、コメントボックスの幅を親コンテナのサイズ変化に合わせて拡大 / 縮小させることも可能です。

```
<mx:TextInput id="comments" width="100%" height ="20"/>
```

また、コントロールのサイズを明示的に指定することもできます。この場合は、コントロールの `height` プロパティと `width` プロパティをピクセル数値に設定します。次の例は、`addr2` TextInput コントロールの高さと幅をそれぞれ 20 ピクセルと 100 ピクセルに設定しています。

```
<mx:TextInput id="addr2" width="100" height ="20"/>
```

コントロールのサイズを実行時に変更するには、ActionScript を使用して `width` プロパティと `height` プロパティを設定します。たとえば、次の例では、Button コントロールのクリックイベントリスナーで、`addr2` TextInput コントロールの `width` プロパティについて、幅を 10 ピクセル分増やしています。

```
<mx:Button id="button1" label="Slide" height="20"
  click="addr2.width+=10;"/>
```

メモ

このテクニックは、元々の `width` プロパティがパーセント値で設定されていた場合でも正常に動作します。`width` プロパティと `height` プロパティには、常にピクセル単位の値が格納されます。

多くのコンポーネントは最大サイズが任意の大きさに設定されています。つまり、アプリケーションの要件に合わせて、どのような大きさでも適用できるという意味です。ゼロ以外の最小サイズが定義されたコンポーネントも一部存在しますが、ほとんどのコンポーネントでは最小サイズが 0 として定義されています。maxHeight、maxWidth、minHeight、minWidth の各プロパティを使用して、各コンポーネントのサイズ変更範囲を明示的に設定できます。

コントロールの配置

コントロールはコンテナ内に配置します。ほとんどのコンテナには、自動的に子の位置を決定するレイアウト規則があらかじめ定義されています。Canvas コンテナではその子に絶対位置が指定されますが、Application コンテナと Panel コンテナでは、オプションで絶対位置とコンテナに対する相対位置を使用できます。

コントロールの絶対位置を指定するには、x プロパティと y プロパティにコンテナにおける座標をピクセル単位で設定します。この座標は、コンテナの左上隅を (0,0) とする相対位置になります。x と y には、正負の整数を指定できます。たとえば、負数を指定することによってコントロールをコンテナの表示領域外に配置しておき、ActionScript を使用して、イベントに対する応答として子を表示領域に移動するようなことも可能です。

次の例では、Canvas コンテナの左上隅を基準とし、TextInput コントロールを 150 ピクセル右、150 ピクセル下の位置に配置しています。

```
<mx:TextInput id="addr2" width="100" height ="20" x="150" y="150"/>
```

絶対位置が指定されたコンテナに配置されたコントロールを実行時に移動するには、コントロールの x プロパティと y プロパティを使用します。たとえば、次の Button コントロールの click イベントリスナーは、TextInput コントロールを現在の位置から 10 ピクセル下に移動しています。

```
<mx:Button id="button1" label="Slide" height="20" x="0" y="250"
  click="addr2.y = addr2.y+10;"/>
```

コンテナに対する相対位置を含めた、コントロールの配置の詳細については、207 ページ、第 8 章の「コンポーネントのサイズと位置の制御」を参照してください。

コントロールの外観の変更

コントロールの外観は、スタイル、スキン、およびフォントを使用してカスタマイズできます。これらは、コンポーネントに共通して適用させる特性を定義するものです。各コントロールには、設定可能な一連のスタイル、スキン、フォントが定義されています。特定のコントロールタイプに固有のものと、さまざまなコントロールで汎用的に使用されるものとがあります。

Flex では、さまざまな方法でコントロールの外観を設定できます。たとえば、コントロールの MXML タグまたは ActionScript を使用して特定のコントロールに対してのみスタイルを設定できる場合と、<mx:Style> タグを使用してアプリケーションの特定のコントロールのすべてのインスタンスに対してグローバルにスタイルを設定できる場合とがあります。

テーマでは、Flex アプリケーションの外観と雰囲気を定義します。テーマには、アプリケーションのカラースキーマや共通フォントなどの単純なものから、全 Flex コンポーネントの完全な再スキンのような複雑なものまで定義できます。コントロールに対して設定できるスタイルは、そのアプリケーションの現在のテーマによって定義されます。したがって、スタイルのプロパティは必ずしも設定できるとは限りません。詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

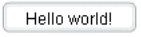
Button コントロール

Button コントロールはきわめて使用頻度の高い四角形のボタンです。Button コントロールの表面にはテキストラベルやアイコンが表示され、実際に押せるような外観を備えています。オプションで複数の Button 状態のそれぞれにグラフィックスキンを指定できます。

標準 Button コントロールまたはトグル Button コントロールを作成できます。標準 Button コントロールは、コントロールの選択後マウスボタンが押されている限りダウン状態に留まります。トグル Button コントロールは、2 回目に選択されるまでダウン状態に留まります。

通常、ボタンはイベントリスナーを使用して、ユーザーがコントロールを選択したときのアクションを実行します。Button コントロールのマウスをクリックしたときに Button コントロールが有効である場合は、`click` イベントと `buttonDown` イベントが送出されます。ボタンは、有効か無効かにかかわらず、`mouseMove`、`mouseOver`、`mouseOut`、`rollOver`、`rollOut`、`mouseDown`、`mouseUp` などのイベントを必ず送出します。

次に Button コントロールの例を示します。



ボタンをアプリケーションの外観と機能に合わせてカスタマイズするには、カスタマイズしたグラフィックスキンを使用します。Button コントロールには、アップ状態、ダウン状態、および無効状態に対応して異なるイメージスキンを設定でき、ボタンが選択されているかどうかに応じてこれらの状態のスキンを変えることができます。このコントロールは、イメージスキンを動的に変更できます。

次の例は、HBox レイアウトコンテナに配置されているビデオの録画と再生を制御する 7 つの Button コントロールを示しています。すべてのボタンはアップ状態となっています。

Button コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	ラベルテキストやアイコンを表示するために必要なサイズです。
最小サイズ	0
最大サイズ	制限なし

Button コントロールの作成

Button コントロールは、次のように、<mx:Button> タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。次のコードは、Button コントロールを作成し、"Hello world!" というラベルを指定する例です。

```
<?xml version="1.0"?>
<!-- controls\button\ButtonLabel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button id="button1" label="Hello world!" width="100"/>
</mx:Application>
```

Button コントロールのアイコン (指定されている場合) およびラベルは、Button コントロールの領域内の中央に配置されます。labelPlacement プロパティに right、left、bottom、top のいずれかを指定することにより、テキストラベルをアイコンとの位置関係で配置できます。

Button コントロールへのアイコンの埋め込み

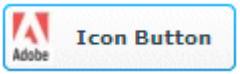
Flex では、コンパイル時でも実行時でも、グラフィックをアプリケーションに読み込むことができます。ただし、ボタンのアイコンは、実行時に参照するのではなく、コンパイル時に埋め込む必要があります。icon プロパティの値に @Embed シンタックスを使用して、GIF、JPEG、PNG、SVG、SWF のファイルを埋め込むことができます。あるいは、[Embed] メタデータを使用して、スクリプトブロック内で定義したイメージをバインドすることも可能です。ボタンのグラフィックをどうしても実行時に参照したい場合は、<mx:Button> タグの代わりに <mx:Image> タグを使用してください。

リソースを埋め込む方法の詳細については、1011 ページ、第 30 章の「アセットの埋め込み」を参照してください。

次のコード例は、Button コントロールを作成し、ラベルとアイコンを指定しています。

```
<?xml version="1.0"?>
<!-- controls\button\ButtonLabelIcon.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button
        label="Icon Button"
        icon="@Embed(source='assets/logo.jpg')"/>
</mx:Application>
```

アイコンは、アプリケーションファイルを含むディレクトリの assets サブディレクトリ内にあります。これで、ボタンのアイコンがラベルテキストの左側に表示されます。



リソース埋め込みの概要については、1011 ページ、第 30 章の「アセットの埋め込み」を参照してください。

Button コントロールのサイズ設定

デフォルトでは、ラベルとアイコンのサイズ、およびアイコンの周囲に確保される 6 ピクセル分の余白に合わせて Button コントロールの幅が調整されます。このデフォルトの幅は、Button コントロールの `width` プロパティに特定の値または親コンテナに対する比率を明示的に設定することによりオーバーライドできます。パーセント値を指定した場合、親コンテナのサイズ変更に合わせて、ボタンのサイズが幅の最大値と最小値の範囲内で変化します。

サイズを明示的に指定した Button コントロールにラベル全体を表示できるだけの領域が存在しない場合、ラベルは省略記号 (...) で切り詰められ終了されます。マウスを Button コントロール上に移動すると、ラベル全体がツールヒントとして表示されます。`tooltip` プロパティを使用してツールヒントを設定した場合にも、ラベルテキストでなくツールヒントが表示されます。

たとえば、ラベルテキストのうち、Button コントロールの高さを越える部分も描画されません。サイズを明示的に指定した Button コントロールにアイコン全体を表示できるだけの領域が存在しない場合、そのアイコンは Button コントロールの境界ボックスを越えて描画されます。

Button コントロールのユーザー操作

ユーザーが Button コントロール上でマウスをクリックすると、Button コントロールから `click` イベントが送出されます。次にその例を示します。

```
<?xml version="1.0"?>
<!-- controls\button\ButtonClick.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Input field. -->
    <mx:TextInput id="myInput" width="150" text=""/>

    <!-- Button control that triggers the copy. -->
    <mx:Button id="myButton" label="Copy Text"
        click="myText.text=myInput.text;"/>

    <!-- Output text box. -->
    <mx:TextArea id="myText" text="" width="150" height="20"/>
</mx:Application>
```

この例では、Button コントロールがクリックされると TextInput コントロールから TextArea コントロールにテキストがコピーされます。

Button コントロールが有効になっている場合、次のように動作します。

- マウスポインタを Button コントロール上に移動すると、Button コントロールがロールオーバー状態であることを示す外観に変わります。
- Button コントロールをクリックすると、その Button コントロールにフォーカスが移動し、ボタンが押された状態であることを示す外観に変わります。マウスボタンを離すと、Button コントロールの外観がロールオーバー状態に戻ります。
- マウスポインタを押し下げたまま Button コントロールの範囲外に移動すると、Button コントロールは元の状態に戻ります。フォーカスは維持されます。
- `toggle` プロパティが `true` に設定されている場合は、Button コントロール上でマウスボタンを離さない限り、ボタンの状態は変化しません。

無効状態の Button コントロールは、ユーザーの操作に関係なく無効状態の外観で表示されます。無効状態では、マウスやキーボードのすべての操作が無視されます。

Button コントロールのスキン適用

各プロパティが異なるボタン状態に対応する、最大 8 つの異なるイメージスキンプロパティのセットを指定できます。これらのスキンによって、ボタンの基本的な外観が決まります。次のボタン状態のそれぞれについてイメージを指定できます。

- アップ (マウスがコントロール上にない)
- ダウン (マウスがコントロール上にあり、マウスボタンが押されている)
- オーバー (マウスがコントロール上を移動している)
- 無効
- 選択済みでアップ
- 選択済みでダウン
- 選択済みでオーバー
- 選択済みで無効

イメージスキンの指定

Button コントロールのデフォルトの外観を指定するには、アップ状態のイメージ (upSkin プロパティ) を指定します。他のすべての状態は、デフォルトとしてアップ状態のイメージまたは別の状態のイメージを使用します。たとえば、ダウン状態のイメージを指定しなかった場合、オーバー状態に指定されたイメージが使用されます。オーバー状態のイメージを指定しなかった場合、アップ状態のイメージが使用されます。選択した状態は、選択した (押した) トグルボタンにのみ使用されます。

スキンイメージによって、シェイプを含めたボタンの外観が決まります。The image can be GIF, JPEG, PNG, SVG, or SWF file. スキンを独立したイメージファイルとして作成するか、複数のイメージを 1 つの SWF ファイルに組み込むことができます。

ボタンイメージは、コンパイル時にアプリケーションの SWF ファイルに埋め込む必要があります。イメージを実行時にサーバーからダウンロードすることはできません。イメージを埋め込むには、@Embed MXML コンパイラディレクティブを使用します。次のコード例は、GIF ファイルをアップ (デフォルト) 状態のボタンイメージとして使用する方法を示しています。

```
upSkin="@Embed(source='assets/buttonUp.gif')"
```

次のコード例は、トグルボタンを作成し、アップ、ダウン、オーバー、無効の各状態のイメージを指定しています。ボタンにはラベルとアイコンの両方が指定されています。

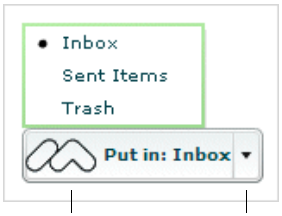
```
<?xml version="1.0"?>
<!-- controls\button\ButtonSkin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Button label="Image Button"
    toggle="true"
    color="0xFFFFAA"
    textRollOverColor="0xAAAA55"
    textSelectedColor="0xFFFF00"
    upSkin="@Embed(source='assets/buttonUp.gif')"
    overSkin="@Embed(source='assets/buttonOver.gif')"
    downSkin="@Embed(source='assets/buttonDown.gif')"
    disabledSkin="@Embed(source='assets/buttonDisabled.gif')"
    icon="@Embed(source='assets/logo.gif')"/>
</mx:Application>
```

PopUpButton コントロール

PopUpButton コントロールは、次の 2 つの水平方向のボタンから構成されるコントロールです。つまり、メインボタン、およびアイコンのみのポップアップボタンと呼ばれる小さなボタンです。メインボタンは Button コントロールです。

ポップアップボタンは、クリックすると、ポップアップコントロールと呼ばれる 2 番目のコントロールが開きます。PopUpButton コントロール外の場所またはポップアップコントロール内をクリックすると、ポップアップコントロールが閉じます。

PopUpButton コントロールは、Button コントロールにポップアップコントロールのインターフェイスを追加します。PopUpButton コントロールの一般的な使用法の 1 つとして、ポップアップボタンで、メインボタンの機能やラベルを変更する List コントロールや Menu コントロールを開くことが挙げられます。次の例は、Menu コントロールを使用しています。



この例では、メールを "Inbox" フォルダ、"Sent Items" フォルダ、または "Trash" フォルダに配置するかどうかを選択できます。この選択は、メインボタンの右側にある小さなポップアップボタンをクリックして、表示されるポップアップメニューからボタンをクリックして行います。メインボタンのテキストは、実行するアクションを示します。ユーザーがメニューの別のアイテムを選択するたびにこのテキストは変わります。

PopUpButton コントロールは、メニューの表示だけでなく、任意の Flex コントロールをポップアップコントロールとして表示できます。たとえば、ユーザーがドキュメントを確認用に送信できるワークフローアプリケーションでは、部門構成を視覚的に表したものとして Tree コントロールを使用できます。PopUpButton コントロールのポップアップボタンでは、メッセージの受信者を選択できるツリーが表示されます。

ポップアップするコントロールは、メインボタンの外観やアクションに影響を与えないようにする必要があります。代わりに、このコントロールに独自のアクションを設定できます。たとえば、メインボタンが最後のアクションのみを取り消し、ポップアップコントロールが複数のアクションを選択によって取り消すことができる List コントロールとなっている場合、取り消し用の PopUpButton コントロールを作成できます。

この PopUpButton コントロールは、Button コントロールのサブクラスであり、toggle プロパティや選択したボタンのスタイルを除き、Button コントロールのプロパティ、スタイル、イベント、およびメソッドのすべてを継承します。

コントロールには、次のような特性があります。

- popUp プロパティは、ポップアップコントロール (List または Menu など) を指定します。
- open() メソッドと close() メソッドでは、ポップアップボタンを使用せずに、ポップアップコントロールをプログラムで開いたり閉じたりできます。
- ポップアップコントロールが開いたり閉じたりすると、open イベントと close イベントが送出されます。
- popUpSkin と arrowButtonWidth のスタイルプロパティを使用すると、PopUpButton コントロールの外観を定義できます。

詳細については、『Adobe Flex 2 リファレンスガイド』の PopUpButton を参照してください。

PopUpButton コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	メインボタンのラベルとアイコン、およびポップアップボタンのアイコンを表示するのに十分なサイズです。
最小サイズ	0
最大サイズ	未定義

PopUpButton コントロールの作成

次の例のように、MXML で PopUpButton コントロールを定義するには、<mx:PopUpButton> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

次の例では、PopUpButton コントロールを使用して Menu コントロールを開いています。Menu コントロールやポップアップコントロールは、一度開くと通常どおり機能します。ユーザーがメニューアイテムを選択する時点を認識するには、次の例に示すように、Menu コントロールの change イベントにイベントリスナーを定義します。

```
<?xml version="1.0"?>
<!-- controls\button\PopUpButtonMenu.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="600" width="600">

    <mx:Script>
        <![CDATA[
            import mx.controls.*;
            import mx.events.*;
```

```
            private var myMenu:Menu;

            // Initialize the Menu control,
            // and specify it as the pop up object
            // of the PopUpButton control.
            private function initMenu():void {
                myMenu = new Menu();
                var dp:Object = [
                  {label: "New Folder"},
                  {label: "Sent Items"},
                  {label: "Inbox"}
                ];
                myMenu.dataProvider = dp;
                myMenu.addEventListener("itemClick", changeHandler);
                popB.popUp = myMenu;
            }

            // Define the event listener for the Menu control's change event.
            private function changeHandler(event:MenuEvent):void {
                var label:String = event.label;
                popTypeB.text=String("Moved to " + label);
                popB.label = "Put in: " + label;
                popB.close();
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Label text="Main button mimics the last selected menuItem."/>
        <mx:PopUpButton id="popB"
            label="Edit"
            width="135"
            creationComplete="initMenu();"/>

        <mx:Spacer height="50"/>
        <mx:TextInput id="popTypeB"/>
    </mx:VBox>
</mx:Application>
```

ユーザーの操作

PopUpButton コントロールを移動するには、次のようにマウスを使用します。

- マウスを PopUpButton コントロールの任意の場所の上に移動すると、ボタンの境界とメインボタンやポップアップボタンがハイライト表示されます。
- ボタンをクリックすると、click イベントが送出されます。
- ポップアップボタンをクリックすると、ポップアップコントロールがポップアップ表示され、open イベントが送出されます。

- PopUpButton コントロール外の場所またはポップアップコントロール内をクリックすると、ポップアップコントロールが閉じ、close イベントが送出されます。

次のキーストロークを使用すると、ユーザーは PopUpButton コントロールを移動できます。

キー	用途
スペースバー	メインボタンをクリックしたときと同様に動作します。
Ctrl + ↓ (下矢印)	ポップアップコントロールを開き、open イベントを起動します。ポップアップコントロールのキーボード処理が有効になります。
Ctrl + ↑ (上矢印)	ポップアップコントロールを閉じ、close イベントを起動します。

メモ

開いたポップアップコントロールを終了するために、Tab キーを使用することはできません。選択するか、Ctrl + ↑ (上矢印) キーの組み合わせでコントロールを閉じる必要があります。

ButtonBar コントロールと ToggleButtonBar コントロール

ButtonBar コントロールと ToggleButtonBar コントロールは、共通の外観を持つ関連するボタンの水平または垂直の列を定義します。これらのコントロールは単一のイベントである itemClick イベントを定義し、このイベントはコントロールのボタンのいずれかが選択されると送出されます。

ButtonBar コントロールは、選択状態を保持しないボタンのグループを定義します。ButtonBar コントロールのボタンが選択されると、ボタンの外観が選択状態に変わり、ボタンの選択が解除されると、ボタンの外観が選択されない状態に戻ります。

ToggleButtonBar コントロールは、選択状態または非選択状態を保持するボタンのグループを定義します。ToggleButtonBar コントロールで 1 つのボタンだけが選択状態になることができます。つまり、ToggleButtonBar コントロールの 1 つのボタンを選択すると、そのボタンは他のボタンが選択されるまで、選択状態を継続します。

ToggleButtonBar コントロールの unselectable プロパティを true に設定した場合、現在選択されているボタンを選択すると、そのボタンの選択が解除されます。unselectable プロパティのデフォルト値は false です。

次の図は、一連のボタンが定義されている ButtonBar コントロールの例です。

次の図は、一連のボタンが定義されている ToggleButtonBar コントロールの例です。コントロールの Dreamweaver ボタンが選択されています。

ButtonBar コントロールと ToggleButtonBar コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
推奨サイズ	ラベルテキストを含むすべてのボタンとアイコンを、必要な追加スペースおよびセパレータとともに表示できる幅です。
コントロールのサイズ変更規則	デフォルトでは、コントロールのサイズは変更されません。親コンテナのサイズに合わせて ButtonBar のサイズを変更する場合は、パーセント値でサイズを指定します。
追加スペース	top、bottom、left、および right の各プロパティは 0 ピクセルに設定されています。

ButtonBar コントロールの作成

ButtonBar コントロールは、次の例のように、<mx:ButtonBar> タグを使用して MXML 内で作成します。

```
<?xml version="1.0"?>
<!-- controls\bar\BBarSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:ButtonBar borderStyle="solid" horizontalGap="5">
        <mx:dataProvider>
            <mx:String>Flash</mx:String>
            <mx:String>Director</mx:String>
            <mx:String>Dreamweaver</mx:String>
            <mx:String>ColdFusion</mx:String>
        </mx:dataProvider>
    </mx:ButtonBar>
</mx:Application>
```

この例では、259 ページの「ButtonBar コントロールと ToggleButtonBar コントロール」の図のような 4 つの Button コントロールが並ぶ列を作成します。

ToggleButtonBar コントロールを作成する場合は、<mx:ButtonBar> タグを <mx:ToggleButtonBar> に置き換えます。それ以外のシンタックスは同じです。

dataProvider プロパティを使用して 4 つのボタンのラベルを指定します。dataProvider プロパティには、オブジェクトの配列を格納できます。各オブジェクトには、label、icon、および tooltip の最大 3 つのフィールドを含めることができます。

次の例では、オブジェクトの配列を使用して、各ボタンのラベルとアイコンを指定しています。

```
<?xml version="1.0"?>
<!-- controls\bar\BBarLogo.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ButtonBar borderStyle="solid" horizontalGap="5">
        <mx:dataProvider>
            <mx:Object label="Flash"
                icon="@Embed(source='assets/Flashlogo.gif')"/>
            <mx:Object label="Director"
                icon="@Embed(source='assets/Dirlogo.gif')"/>
            <mx:Object label="Dreamweaver"
                icon="@Embed(source='assets/Dlogo.gif')"/>
            <mx:Object label="ColdFusion"
                icon="@Embed(source='assets/CFlogo.gif')"/>
        </mx:dataProvider>
    </mx:ButtonBar>
</mx:Application>
```

ButtonBar コントロールまたは ToggleButtonBar コントロールは、その dataProvider プロパティの値に基づいて Button コントロールを作成します。ButtonBar および ToggleButtonBar は Container のサブクラスですが、Container.addChild()、Container.removeChild() などのメソッドを使用して Button コントロールを追加または削除しないでください。代わりに、addItem()、removeItem() などのメソッドを使用して dataProvider プロパティを操作してください。ButtonBar コントロールまたは ToggleButtonBar コントロールは、dataProvider プロパティへの変更に基づいて、自動的に必要な子を追加または削除します。

ButtonBar イベントの処理

ButtonBar コントロールと ToggleButtonBar コントロールは、ユーザーがボタンを選択すると、itemClick イベントを送出します。イベントリスナーに渡されるイベントオブジェクトは、ItemClickEvent タイプです。イベントリスナーからイベントオブジェクトのプロパティにアクセスして、選択されたボタンのインデックスや最初のボタンのインデックスが 0 であることなどを確認できます。イベントオブジェクトの詳細については、『Adobe Flex 2 リファレンスガイド』の ItemClickEvent クラスの説明を参照してください。

次の例では、ButtonBar コントロールが itemClick イベントのイベントリスナーを定義します。

```
<?xml version="1.0"?>
<!-- controls\bar\BBarEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

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

            private function clickHandler(event:ItemClickEvent):void {
```

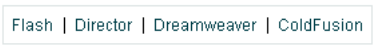
```
                myTA.text="Selected button index: " +
                    String(event.index) + "\n" +
                    "Selected button label: " +
                    event.label;
            }
        ]]>
    </mx:Script>

    <mx:TextArea id="myTA" width="200" height="100"/>

    <mx:ButtonBar
        borderStyle="solid"
        horizontalGap="5"
        itemClick="clickHandler(event);">
        <mx:dataProvider>
            <mx:String>Flash</mx:String>
            <mx:String>Director</mx:String>
            <mx:String>Dreamweaver</mx:String>
            <mx:String>ColdFusion</mx:String>
        </mx:dataProvider>
    </mx:ButtonBar>
</mx:Application>
```

この例では、イベントリスナーが、itemClick イベントに応じて、選択されたボタンのインデックスとラベルを TextArea コントロールに表示します。

LinkBar コントロール

LinkBar コントロールは、一連のリンク先を指定する LinkButton コントロールの水平または垂直方向の行を定義します。LinkBar コントロールは通常、ViewStack コンテナのアクティブな子コンテナを制御するため、またはリンクの単独のセットを作成するために使用します。

次は、一連のリンクが定義されている LinkBar コントロールの例です。



LinkBar コントロールには次のデフォルトプロパティがあります。

プロパティ	デフォルト値
推奨サイズ	すべてのラベルテキストを設定された追加スペースおよびセパレータで表示できる幅と、最も高い子の高さです。
コントロールのサイズ変更規則	デフォルトでは、LinkBar コントロールのサイズは変更されません。親コンテナのサイズに合わせて LinkBar のサイズを変更する場合は、パーセント値でサイズを指定します。
パディング	top、bottom、left、および right の各プロパティは 2 ピクセルに設定されています。

LinkBar コントロールの作成

LinkBar コントロールの最も一般的な使用方法の1つは、ViewStack コンテナでアクティブにする子の制御です。例については、584 ページの「ViewStack ナビゲータコンテナ」を参照してください。

LinkBar コントロールを単独で使用して、アプリケーションにリンクを作成することもできます。次の例では、ユーザー入力に応答する itemClick ハンドラを LinkBar コントロールに対して定義し、LinkBar の dataProvider プロパティを使用してそのラベルテキストを指定します。この例のコードを使用すると、前の図に示した LinkBar コントロールが作成されます。

```
<?xml version="1.0"?>
<!-- controls\bar\LBarSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:LinkBar borderStyle="solid"
        itemClick="navigateToURL(new URLRequest('http://www.adobe.com/' +
            String(event.label).toLowerCase()), '_blank');">
        <mx:dataProvider>
            <mx:String>Flash</mx:String>
            <mx:String>Director</mx:String>
            <mx:String>Dreamweaver</mx:String>
            <mx:String>ColdFusion</mx:String>
        </mx:dataProvider>
    </mx:LinkBar>
</mx:Application>
```

この例では、<mx:dataProvider> タグおよび <mx:Array> タグを使用してラベルテキストを定義します。itemClick ハンドラに渡されるイベントオブジェクトには、ユーザーが選択したラベルが含まれます。itemClick イベントのハンドラは、ラベルに基づいて Adobe の Web サイトへの HTTP 要求を生成し、そのページを新しいブラウザウィンドウに開きます。

次の例のように、<mx:dataProvider> タグにデータをバインドして、LinkBar コントロールを作成することもできます。

```
<?xml version="1.0"?>
<!-- controls\bar\LBarBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var linkData:ArrayCollection = new ArrayCollection([
                "Flash", "Director", "Dreamweaver", "ColdFusion"
            ]);
        ]]>
    </mx:Script>

    <mx:LinkBar
```

```
        horizontalAlign="right"
        borderStyle="solid"
        itemClick="navigateToURL(new URLRequest('http://www.adobe.com/' +
            String(event.label).toLowerCase()), '_blank');">
        <mx:dataProvider>
            {linkData}
        </mx:dataProvider>
    </mx:LinkBar>
</mx:Application>
```

この例では、LinkBar コントロール用のデータを ActionScript で変数として定義し、その変数を <mx:dataProvider> タグにバインドします。<mx:dataProvider> タグには、Flex のデータモデル、Web サービスの応答、または他のタイプのデータモデルからもバインドできます。

LinkBar コントロールは、その dataProvider プロパティの値に基づいて LinkButton コントロールを作成します。LinkBar は Container のサブクラスですが、Container.addChild()、Container.removeChild() などのメソッドを使用して LinkButton コントロールを追加または削除しないでください。代わりに、addItem()、removeItem() などのメソッドを使用して dataProvider プロパティを操作してください。LinkBar コントロールは、dataProvider プロパティへの変更に基づいて、自動的に必要な子を追加または削除します。

TabBar コントロール

TabBar コントロールは、タブを定義して水平方向または垂直方向に配置します。次は、TabBar コントロールの例です。

LinkBar コントロールと同様に、TabBar コントロールを使用して ViewStack コンテナでアクティブにする子コンテナを制御できます。TabBar コントロールを使用して ViewStack コンテナでアクティブにする子コンテナを制御するには、LinkBar コントロールと同じシンタックスを使用します。例については、584 ページの「ViewStack ナビゲータコンテナ」を参照してください。

TabBar コントロールは TabNavigator コンテナに似ていますが、TabBar コンテナは子を持てません。たとえば、TabNavigator コンテナのタブは表示する子コンテナの選択に使用しますが、TabBar コントロールは 1 つのコンテナの表示内容を設定し、選択されたタブに基づいて、そのコンテナの子の表示 / 非表示を切り替えます。

TabBar コントロールには次のデフォルトプロパティがあります。

プロパティ	デフォルト値
推奨サイズ	すべてのラベルテキストを設定された追加スペースとともに表示できる幅と、ラベルテキストを表示できる高さです。 デフォルトのタブの高さは、コントロールに適用されるフォント、スタイル、およびスキンによって決まります。tabHeight プロパティを使用して明示的に高さを設定すると、その値によってデフォルト値がオーバーライドされます。
コントロールのサイズ変更規則	デフォルトでは、TabBar コントロールのサイズは変更されません。親コンテナのサイズに合わせて TabBar のサイズを変更する場合は、パーセント値でサイズを指定します。
パディング	left および right の各プロパティは、0 ピクセルに設定されています。

TabBar コントロールの作成

<mx:TabBar> タグを使用して、MXML で TabBar コントロールを定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

TabBar コントロールのデータは、<mx:TabBar> タグの子タグである <mx:dataProvider> および <mx:Array> を使用して指定します。<mx:dataProvider> タグでのデータの指定方法はいくつかあります。TabBar コントロールを作成する最も簡単な方法では、次の例のように、<mx:dataProvider> タグ、<mx:Array> タグ、および <mx:String> タグを使用して、各タブのテキストを指定します。

```
<?xml version="1.0"?>
<!-- controls\bar\TBarSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:TabBar>
        <mx:dataProvider>
            <mx:String>Alabama</mx:String>
            <mx:String>Alaska</mx:String>
            <mx:String>Arkansas</mx:String>
        </mx:dataProvider>
    </mx:TabBar>
</mx:Application>
```

<mx:String> タグは、TabBar コントロールの各タブのテキストを定義します。

また次の例のように、<mx:Object> タグを使用して、項目をオブジェクトの配列として定義することができます。各オブジェクトには label プロパティと関連するデータ値が含まれます。

```
<?xml version="1.0"?>
<!-- controls\bar\TBarObject.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:TabBar>
```

```
        <mx:dataProvider>
            <mx:Object label="Alabama" data="Montgomery"/>
            <mx:Object label="Alaska" data="Juneau"/>
            <mx:Object label="Arkansas" data="Little Rock"/>
        </mx:dataProvider>
    </mx:TabBar>
</mx:Application>
```

label プロパティには州の名前が設定され、data プロパティにはその州都が設定されています。data プロパティを使用すると、データ値とテキストラベルを関連付けることができます。たとえば、label テキストを色の名前として使用し、その色の数値表現をデータ値として関連付けることができます。

デフォルトでは、Flex は label プロパティの値を使用してタブテキストを定義します。オブジェクトに label プロパティがない場合、次の例のように、TabBar コントロールの labelField プロパティを使用して、タブテキストを含むプロパティ名を指定できます。

```
<?xml version="1.0"?>
<!-- controls\bar\TBarLabel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:TabBar labelField="state">
        <mx:dataProvider>
            <mx:Object state="Alabama" data="Montgomery"/>
            <mx:Object state="Alaska" data="Juneau"/>
            <mx:Object state="Arkansas" data="Little Rock"/>
        </mx:dataProvider>
    </mx:TabBar>
</mx:Application>
```

TabBar コントロールへのデータの受け渡し

Flex では、ActionScript の変数定義または Flex のデータモデルから TabBar コントロールを作成できます。変数を使用する場合は、その定義で次のいずれかを設定します。

- ラベル (ストリング)
- ラベル (ストリング) とデータ (スカラー値またはオブジェクト) のペア

次の例では、変数を使用して TabBar コントロールを作成しています。

```
<?xml version="1.0"?>
<!-- controls\bar\TBarVar.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var STATE_ARRAY:ArrayCollection = new ArrayCollection([
```

```
                {label:"Alabama", data:"Montgomery"},
                {label:"Alaska", data:"Juneau"},
                {label:"Arkansas", data:"LittleRock"}
            ]);
        ]]>
    </mx:Script>

    <mx:TabBar >
        <mx:dataProvider>
            {STATE_ARRAY}
        </mx:dataProvider>
    </mx:TabBar>
</mx:Application>
```

Flex のデータモデルを dataProvider プロパティにバインドすることもできます。データモデルの使用の詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

TabBar コントロールのイベントの処理

TabBar コントロールは、ユーザーがタブを選択したときにブロードキャストされる itemClick イベントを定義します。イベントオブジェクトには次のプロパティがあります。

- label　選択されたタブのラベルを表すストリングです。
- index　選択されたタブのインデックスを表す数値です。インデックスは、0 ～ n - 1 の範囲の整数です。n は、タブの総数です。デフォルト値は 0 で、1 つ目のタブを表します。

次の例のコードは、TabBar コントロールの itemClick イベントハンドラです。

```
<?xml version="1.0"?>
<!-- controls\bar\TBarEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.events.ItemClickEvent;
            import mx.controls.TabBar;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var STATE_ARRAY:ArrayCollection = new ArrayCollection([
                {label:"Alabama", data:"Montgomery"},
                {label:"Alaska", data:"Juneau"},
                {label:"Arkansas", data:"LittleRock"}
            ]);

            private function clickEvt(event:ItemClickEvent):void {
                // Access target TabBar control.
                var targetComp:TabBar = TabBar(event.currentTarget);
                forClick.text="label is: " + event.label + " index is: " +
```

```
                    event.index + " capital is: " +
                    targetComp.dataProvider[event.index].data;
            }
        ]]>
    </mx:Script>

    <mx:TabBar id="myTB" itemClick="clickEvt(event);">
        <mx:dataProvider>
            {STATE_ARRAY}
        </mx:dataProvider>
    </mx:TabBar>

    <mx:TextArea id="forClick" width="150"/>
</mx:Application>
```

この例では、itemClick イベントが発生するたびに、タブラベル、選択されたインデックス、および TabBar コントロールの dataProvider 配列の選択されたデータで、TextArea コントロールが更新されます。

CheckBox コントロール

CheckBox コントロールは、チェックマークの表示と非表示 (空) の状態を保持するグラフィカルコントロールです。相互排他的でない一連のブール値 (true または false) を収集する必要があるときに、CheckBox コントロールを使用します。

CheckBox コントロールにはテキストラベルを 1 つ追加して、左、右、上、下のいずれかの位置に表示することができます。CheckBox コントロールのラベルは、コントロールの境界内に収まるようにクリッピングされます。

次の図に checked CheckBox コントロールの例を示します。

この例を生成するためのコードの詳細については、269 ページの「CheckBox コントロールの作成」を参照してください。

CheckBox コントロールまたは対応するテキストをクリックすると、チェックがオンの状態からチェックがオフの状態へ、またはチェックがオフの状態からチェックがオンの状態に変化します。

無効状態の CheckBox コントロールには、チェックがオンとチェックがオフの 2 つの状態があります。デフォルトでは、無効状態の CheckBox コントロールでは、有効状態の CheckBox コントロールとは異なる背景とチェックマークの色が表示されます。

CheckBox コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	ラベルを表示するために必要なサイズです。
最小サイズ	0
最大サイズ	制限なし

CheckBox コントロールの作成

次の例のように、MXML で CheckBox コントロールを定義するには、<mx:CheckBox> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- controls\checkbox\CBSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:VBox>
        <mx:CheckBox width="100" label="Employee?"/>
    </mx:VBox>
</mx:Application>
```

selected プロパティを使用して、デフォルトでチェックがオンになるチェックボックスを生成することもできます。

```
<?xml version="1.0"?>
<!-- controls\checkbox\CBSelected.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:VBox>
        <mx:CheckBox width="100" label="Employee?" selected="true"/>
    </mx:VBox>
</mx:Application>
```

CheckBox コントロールのユーザー操作

CheckBox コントロールが有効な状態でクリックすると、その CheckBox コントロールにフォーカスが移動し、最初の状態に応じてチェックがオンの状態またはチェックがオフの状態の外観が表示されます。CheckBox コントロールの領域全体をクリック領域といいます。CheckBox コントロールのテキストがアイコンよりも大きい場合、クリック可能な領域がアイコンの上下に拡張されます。

ユーザーがマウスボタンを押し下げ、そのままの状態でマウスポインタを CheckBox コントロール (またはそのラベル) の領域外に移動すると、その CheckBox コントロールはフォーカスを保持したまま、最初の状態に戻ります。ユーザーが CheckBox コントロール上でマウスボタンを離さない限り、チェックボックスの状態は変化しません。

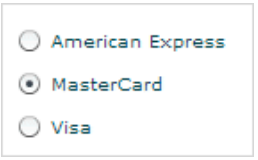
無効状態の CheckBox コントロールに対して操作を実行することはできません。

RadioButton コントロール

RadioButton コントロールを使用すると、相互排他的な選択肢の中から 1 つだけをユーザーに選択させることができます。RadioButton グループは、同じグループ名を持つ複数の RadioButton コントロールで構成されます。グループ内のメンバーは一度に 1 つしか選択できません。まだ選択されていないラジオボタンを選択すると、同じグループ内でそれまで選択されていたラジオボタンは選択解除されます。

RadioButton コントロールについて

次の例は、3 つの RadioButton コントロールから成る RadioButton グループを示しています。



この例を生成するためのコードの詳細については、270 ページの「RadioButton コントロールの作成」を参照してください。

RadioButton コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	コントロールのテキストラベル全体を表示できる幅です。
最小サイズ	0
最大サイズ	未定義

RadioButton コントロールの作成

RadioButton コントロールは、次のように、<mx:RadioButton> タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- controls\button\RBSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:RadioButton groupName="cardtype"
        id="americanExpress"
        label="American Express"
        width="150"/>
    <mx:RadioButton groupName="cardtype"
        id="masterCard"
```

```
        label="MasterCard"
        width="150"/>
    <mx:RadioButton groupName="cardtype"
        id="visa"
        label="Visa"
        width="150"/>
</mx:Application>
```

結果的に、270 ページの「RadioButton コントロールについて」のようなアプリケーションになります。

グループ内の RadioButton コントロールごとに、ボタンの click イベントに反応するイベントリスナーを定義することもできます。次のコード例では、ユーザーが RadioButton コントロールを選択すると、対応する click イベントのイベントリスナーが呼び出されます。

```
<?xml version="1.0"?>
<!-- controls\button\RBEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import flash.events.Event;

            private function handleAmEx(event:Event):void {
                // Handle event.
                myTA.text="Got Amex";
            }

            private function handleMC(event:Event):void {
                // Handle event.
                myTA.text="Got Amex";
            }

            private function handleVisa(event:Event):void {
                // Handle event.
                myTA.text="Got Amex";
            }
        ]]>
    </mx:Script>

    <mx:RadioButton groupName="cardtype"
        id="americanExpress"
        label="American Express"
        width="150"
        click="handleAmEx(event);"/>
    <mx:RadioButton groupName="cardtype"
        id="masterCard"
        label="MasterCard"
        width="150"
        click="handleMC(event);"/>
```

```
    <mx:RadioButton groupName="cardtype"
        id="visa"
        label="Visa"
        width="150"
        click="handleVisa(event);"/>

    <mx:TextArea id="myTA"/>
</mx:Application>
```

RadioButton のユーザーの操作

RadioButton コントロールが有効な場合、選択されていない RadioButton コントロール上にマウスポインタを移動すると、そのボタンがロールオーバー状態であることを示す外観に変わります。選択されていない RadioButton コントロールをクリックすると、そのコントロールに入力フォーカスが移動し、非選択のダウン状態を示す外観に変わります。マウスボタンを離すと、ボタンの外観が選択状態に変わります。同じグループ内でそれまで選択されていた RadioButton コントロールは非選択状態の外観に戻ります。

マウスポインタを押し下げたまま RadioButton コントロールの範囲外に移動すると、Button コントロールは非選択状態に戻ります。入力フォーカスは維持されます。

RadioButton コントロールが無効にされている場合は、ユーザーの操作に関係なく、RadioButton コントロールおよび RadioButton グループが無効状態の外観で表示されます。無効状態では、マウスやキーボードのすべての操作が無視されます。

RadioButton コントロールおよび RadioButtonGroup コントロールでは、次のキーボード操作ができます。

キー	アクション
Ctrl + 矢印キー	ボタンを選択せずに、フォーカスをボタンからボタンへ移動できます。
スペースバー	ボタンを選択します。

<mx:RadioButtonGroup> タグを使用したグループの作成

前の例では、各 RadioButton コントロールの groupName プロパティを使用して RadioButton グループを作成しました。次の例に示すように、RadioButtonGroup コントロールを使って RadioButton グループを作成することもできます。

```
<?xml version="1.0"?>
<!-- controls\button\RBGroupSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
```

```
            import mx.events.ItemClickEvent;

            private function handleCard(event:ItemClickEvent):void {
                //Handle event.
            }
        ]]>
    </mx:Script>

    <mx:RadioButtonGroup id="cardtype" itemClick="handleCard(event);"/>
    <mx:RadioButton groupName="cardtype"
        id="americanExpress"
        value="AmEx"
        label="American Express"
        width="150"/>
    <mx:RadioButton groupName="cardtype"
        id="masterCard"
        value="MC"
        label="MasterCard"
        width="150"/>
    <mx:RadioButton groupName="cardtype"
        id="visa"
        value="Visa"
        label="Visa"
        width="150"/>
</mx:Application>
```

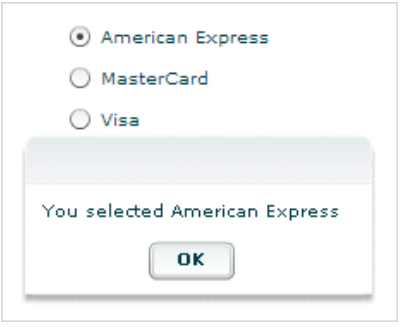
この例では、<mx:RadioButtonGroup> タグの id プロパティを使用して、グループ内のすべてのボタンに対してグループ名と itemClick イベントリスナーを 1 つ定義しています。

<mx:RadioButtonGroup> タグを使用する場合は、id プロパティが必要です。次の例のように、グループに対して itemClick イベントリスナーを指定することにより、選択されたボタンを判別することができます。

```
<?xml version="1.0"?>
<!-- controls\button\RBGroupEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.controls.Alert;
            import mx.events.ItemClickEvent;

            private function handleCard(event:ItemClickEvent):void {
                if (event.currentTarget.selectedValue == "AmEx") {
                    Alert.show("You selected American Express")
                }
                else {
                    if (event.currentTarget.selectedValue == "MC") {
                        Alert.show("You selected Master Card")
                    }
```

```
                    else {
                        Alert.show("You selected Visa")
                    }
                }
            }
        ]]>
    </mx:Script>

    <mx:RadioButtonGroup id="cardtype" itemClick="handleCard(event);"/>
    <mx:RadioButton groupName="cardtype"
        id="americanExpress"
        value="AmEx"
        label="American Express"
        width="150"/>
    <mx:RadioButton groupName="cardtype"
        id="masterCard"
        value="MC"
        label="MasterCard"
        width="150"/>
    <mx:RadioButton groupName="cardtype"
        id="visa"
        value="Visa"
        label="Visa"
        width="150"/>
</mx:Application>
```

[American Express] ボタンをクリックすると次のようになります。



itemClick イベントリスナーの中でイベントオブジェクトを使用しています。RadioButtonGroup コントロールの selectedValue プロパティには、選択された RadioButton コントロールの value プロパティが設定されます。value プロパティを省略した場合、selectedValue プロパティには、label プロパティの値が設定されます。

click イベントリスナーは、グループに対して定義した上で、さらに、個々のボタンに対して定義することもできます。

NumericStepper コントロール

NumericStepper コントロールを使用すると、順序付けられた数値の集合から特定の値を選択できます。NumericStepper コントロールは、単一行のテキスト入力フィールドと、値を段階的に増減するための 2 つの矢印ボタンで構成されます。また、キーボードの上下矢印キーを使って値を増減することもできます。

次の例は NumericStepper コントロールを示しています。

このサンプルを作成するためのコードの詳細については、275 ページの「NumericStepper コントロールの作成」を参照してください。

ユーザーが上矢印をクリックすると、表示された値が 1 単位分増加します。矢印を押し下げたままの状態にすると、ユーザーがマウスボタンを離すまで値が増加または減少し続けます。矢印をクリックすると、それがハイライト表示されてユーザーに示されます。

有効な値であれば、直接テキストフィールドに入力することもできます。編集可能な ComboBox コントロールも同様の機能を実装していますが、ドロップダウンリストが展開されることによって重要なデータが覆い隠されてしまうことがあるため、用途によっては NumericStepper コントロールのほうが都合がよい場合があります。

NumericStepper コントロールの矢印は、常にテキストフィールドの右側に表示されます。

NumericStepper コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	コントロールで使用される最大桁数を表示できる幅です。
最小サイズ	テキストのサイズに基づいて決定されます。
最大サイズ	未定義。

NumericStepper コントロールの作成

NumericStepper コントロールは、次のように、<mx:NumericStepper> タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- controls\numericstepper\NumStepSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:NumericStepper id="nstepper1" value="6" stepSize="2"/>
</mx:Application>
```

NumericStepper コントロールのサイズ設定

NumericStepper コントロール自体のサイズが変わっても、上下矢印ボタンの大きさは変化しません。NumericStepper コントロールにデフォルトの高さを超えるサイズを設定した場合は、ステッパーボタンがコントロールの上下に固定されます。

ユーザーの操作

ユーザーが上下矢印ボタンをクリックすると、表示された値が 1 単位分増減します。いずれかの矢印ボタンを 200 ミリ秒以上、押し続けた場合、ユーザーがマウスボタンを離すか、最大値または最小値に到達するまで、指定された増分単位に基づいて入力フィールドの値が増減を続けます。

キーボード操作

NumericStepper コントロールでは、次のようなキーボード操作を実行できます。

キー	説明
↓ (下矢印)	値を 1 単位ずつ減らします。
↑ (上矢印)	値を 1 単位ずつ増やします。
← (左矢印)	NumericStepper コントロールのテキストフィールド内で挿入ポインタを左に移動します。
→ (右矢印)	NumericStepper コントロールのテキストフィールド内で挿入ポインタを右に移動します。

キーボードを使用して NumericStepper 内を移動するには、キーボードにフォーカスがある必要があります。

DateChooser コントロールと DateField コントロール

DateChooser コントロールと DateField コントロールを使用すると、グラフィックカレンダーから日付を選択することができます。DateChooser コントロールのユーザーインターフェイスは、カレンダーです。DateField コントロールには、日付選択用カレンダーのポップアップで結果として日付を選択する、テキストフィールドがあります。DateField プロパティは、DateChooser プロパティのスーパーセットです。

詳細については、『Adobe Flex 2 リファレンスガイド』の DateChooser および DateField を参照してください。

DateChooser コントロールについて

DateChooser コントロールは、年と月の名前、および対応する月のカレンダーを、曜日を示すラベル付きで表示します。このコントロールは、カレンダーを常時表示しておくアプリケーションで役に立ちます。ユーザーは、カレンダーから特定の日付を選択できます。前後に移動するための矢印ボタンを使って、月や年を切り替えることができます。特定の日付を選択できないようにしたり、表示する日付範囲を制限することも可能です。

次の図に DateChooser コントロールの例を示します。

月の表示を切り替えても、日付の選択状態は変化しません。つまり、現在選択されている日付が必ずしも表示状態になるとは限りません。DateChooser コントロールは、曜日の見出しの幅に合わせてサイズが変更されます。したがって、見出しとして曜日の略語でなく曜日名を使用する場合、カレンダーには完全に曜日名を表示できるだけの幅があります。

DateChooser コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	カレンダーを表示するのに十分なサイズで、曜日名を表示するのに十分な幅です。
最小サイズ	0
最大サイズ	制限なし

DateField コントロールについて

DateField コントロールは日付を表示するためのテキストフィールドで、右側にカレンダーアイコンが表示されます。ユーザーがこのコントロールの境界ボックス内をクリックすると、DateChooser コントロールと同一の日付選択用カレンダーがポップアップ表示されます。日を選択しない場合、このテキストフィールドは空になり、日付選択用カレンダーには現在の月が表示されます。

日付選択用カレンダーが開くと、ユーザーは月のスクロールボタンをクリックして月や年をスクロールし、日付を選択することができます。ユーザーが日付を選択すると、日付選択用カレンダーが閉じ、選択した日付がテキストフィールドに表示されます。

このコントロールは、カレンダー選択ツールが必要だが、日付情報が占有する領域を最小化したい場合に便利です。

次の例に、DateField コントロールの 2 つのイメージを示します。左側は、日付選択用カレンダーが閉じた状態のコントロールです。カレンダーアイコンは、テキストボックスの右側に表示されています。右側は、日付選択用カレンダーが開いた状態の DateField コントロールです。

DateField コントロールは、ユーザーに日付を選択させる必要のあるあらゆる場所で使うことができます。たとえば、ホテルの予約システムで DateField コントロールを使用し、特定の日付を選択可能に、その他の日付を無効にすることができます。また、ユーザーが日付を選択したときに、催し物や会議などの現在のイベントを表示するアプリケーションで DateField コントロールを使うことができます。

DateField には、拡張された日付選択用カレンダー向けに DateChooser と同じデフォルトのプロパティがあります。畳んだ状態のコントロール用には、次のデフォルトのプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	フォーマットした日付とカレンダーアイコンを表示するのに十分なサイズです。
最小サイズ	0
最大サイズ	制限なし

DateChooser コントロールまたは DateField コントロールの作成

MXML で DateChooser コントロールを定義するには、<mx:DateChooser> タグを使用します。MXML で DateField コントロールを定義するには、<mx:DateField> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

次の例は DateChooser コントロールを作成しています。DateField コントロールを作成するには、<mx:DateChooser> を <mx:DateField> に変更します。この例は、DateChooser コントロールの change イベントを使用して、選択された日付をいくつかの異なる形式で表示しています。

```
<?xml version="1.0"?>
<!-- controls\date\DateChooserEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:Script>
        <![CDATA[

        import mx.events.CalendarLayoutChangeEvent;

        private function useDate(eventObj:CalendarLayoutChangeEvent):void {
            // Make sure selectedDate is not null.
            if (eventObj.currentTarget.selectedDate == null) {
                return
            }

            //Access the Date object from the event object.
            day.text=eventObj.currentTarget.selectedDate.getDay();
            date.text=eventObj.currentTarget.selectedDate.getDate();
            month.text=eventObj.currentTarget.selectedDate.getMonth();
            year.text=eventObj.currentTarget.selectedDate.getFullYear();
            wholeDate.text=
                eventObj.currentTarget.selectedDate.getFullYear() +
                "/" +
                (eventObj.currentTarget.selectedDate.getMonth()+1) +
                "/" + eventObj.currentTarget.selectedDate.getDate();
        }
        ]]>
    </mx:Script>

    <mx:DateChooser id="date1" change="useDate(event)"/>

    <mx:Form>
        <mx:FormItem label="Day">
            <mx:TextInput id="day" width="100"/>
        </mx:FormItem>
        <mx:FormItem label="Day of month">
            <mx:TextInput id="date" width="100"/>
        </mx:FormItem>
        <mx:FormItem label="Month">
            <mx:TextInput id="month" width="100"/>
        </mx:FormItem>
        <mx:FormItem label="Year">
            <mx:TextInput id="year" width="100"/>
        </mx:FormItem>
        <mx:FormItem label="Date">
            <mx:TextInput id="wholeDate" width="300"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

イベントリスナーの1行目で、selectedDate プロパティが null かどうかが示されます。これをチェックする必要があるのは、現在選択されている日付を選択するとその日付の選択が解除され、selectedDate プロパティが null に設定されて、change イベントが送出されるためです。

メモ

wholeDate フィールドの値を取得するコードで、月の数値に 1 が加算されています。これは、DateChooser コントロールの月のインデックスが 0 から始まるためです。つまり、1 月は 0 で、12 月は 11 で表されます。

Date クラスの使用

DateChooser コントロールと DateField コントロールは、selectedDate プロパティを使用して、現在選択されている日付を Date 型のオブジェクトとして格納します。日付と時刻の値を表す Date オブジェクト、または selectedDate プロパティの Date にアクセスする Date オブジェクトを作成できます。

Date クラスには、日付を操作できる多数のメソッドがあります。Date クラスの詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

Date オブジェクトは、MXML で <mx:Date> タグを使って作成および設定できます。このタグにより、Date クラスの setter メソッドが MXML のプロパティとして実装され、Date オブジェクトを初期化できるようになっています。たとえば、次のコードは DateChooser コントロールを作成し、選択された日付として、"April 10, 2005" を設定しています。DateChooser コントロールでは、月は 0 から始まるインデックスとして表現されます。

```
<mx:DateChooser id="date1">
     <mx:selectedDate>
        <mx:Date month="9" date="10" year="2005"/>
     </mx:selectedDate>
</mx:DateChooser>
```

次の例は、DateField コントロール用に最初に選択した日付を設定しています。

```
<mx:DateField id="date3" selectedDate="{new Date (2005, 9, 10)}"/>
```

次の例のように、関数の selectedDate プロパティを設定することもできます。

```
<mx:Script>
  <![CDATA[
     private function initDC():void {
        date2.selectedDate=new Date (2003, 3, 10);
     }
  ]]>
</mx:Script>

<mx:DateChooser id="date2" creationComplete="initDC();"/>
```

プロパティ表記を使用すると、Date オブジェクトの selectedDate プロパティの ActionScript の setter メソッドと getter メソッドにアクセスできます。たとえば、次の行は、選択した日付の 4 桁の年をテキストボックスに表示しています。

```
<mx:TextInput text="{date1.selectedDate.fullYear}"/>
```

ヘッダー、曜日、および今日のテキストスタイルの指定

次の日付選択用カレンダーのプロパティを使用すると、コントロールの領域のテキストスタイルを指定できます。

- headerStyleName
- weekDayStyleName
- todayStyleName

これらのプロパティを使用すると、ヘッダー、曜日リスト、および今日の日付のテキストスタイルを指定できます。これらのプロパティは、todayColor など、テキスト以外のスタイルの設定には使用できません。

次の例は、ヘッダーテキストがボールドで青色、16 ピクセルの Times New Roman フォントの DateChooser コントロールを定義しています。曜日のヘッダーはボールドでイタリック、緑色、15 ピクセルの Courier テキスト、今日の日付はボールドでオレンジ色、12 ピクセルの Times New Roman テキストになっています。今日の日付の背景色はグレーで、直接 mx:DateChooser タグに設定されています。

```
<?xml version="1.0"?>
<!-- controls\date\DateChooserStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        .myHeaderStyle{
            color:#6666CC;
            font-family:Times New Roman, Times, serif;
            font-size:16px; font-weight:bold;}
        .myTodayStyle{
            color:#CC6633;
            font-family:Times New Roman, Times, serif;
            font-size:12px; font-weight:bold;}
        .myDayStyle{
            color:#006600;
            font-family:Courier New, Courier, mono;
            font-size:15px; font-style:italic; font-weight:bold;}
    </mx:Style>

    <mx:DateChooser
        headerStyleName="myHeaderStyle"
        todayStyleName="myTodayStyle"
        todayColor="#CCCCCC"
        weekDayStyleName="myDayStyle"/>
</mx:Application>
```

選択可能な日付の指定

DateChooser コントロールには、ユーザーが選択可能な日付を指定できる次のプロパティがあります。

プロパティ	説明
disabledDays	ユーザーが選択できない曜日の配列。曜日を無効にするためによく使用されます。
disabledRange	ユーザーが選択できない日付の配列。この配列には、個々の Date オブジェクト、日付範囲を指定するオブジェクト、またはこの両方を設定できます。
selectableRange	ユーザーが選択できる 1 つの日付範囲。ユーザーは、この範囲を含む月のみ移動できます。これらの月で範囲外の日付は無効になります。選択可能範囲内の日付を無効にするには、disabledRange プロパティを使用します。

次の例は、次の特性を持つ DateChooser コントロールを示しています。

- selectableRange プロパティは、ユーザーが選択できる日付の範囲を 2006 年 1 月 1 日～2006 年 3 月 15 日に制限しています。ユーザーが移動できる月は 2006 年 1 月～2006 年 3 月の間のみです。
- disabledRanges プロパティは、ユーザーが 1 月 11 日または 1 月 23 日～2 月 10 日間の任意の日付を選択できないようにしています。
- disabledDays プロパティは、ユーザーが土曜日または日曜日を選択できないようにしています。

```
<?xml version="1.0"?>
<!-- controls\date\DateChooserStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:DateChooser
    selectableRange="{{rangeStart: new Date(2006,0,1),
        rangeEnd: new Date(2006,2,15)}}"
    disabledRanges="{[new Date(2006,0,11),
        {rangeStart: new Date(2006,0,23), rangeEnd: new Date(2006,1,10)}]}"
    disabledDays="{[0,6]}"/>
</mx:Application>
```

ActionScript での DateChooser プロパティと DateField プロパティの設定

DateChooser コントロールと DateField コントロールのプロパティには、スカラー値、配列、および Date オブジェクトを設定できます。プロパティの大半は MXML で設定できますが、ActionScript のほうが簡単に設定できる場合があります。

たとえば、次のコード例は、配列を使用して disabledDays プロパティを設定して土曜日と日曜日を無効化、つまり、カレンダーで選択できないようにしています。この例では、タグを使用した方法とタグ属性を使用した方法の、異なる 2 つの方法で disabledDays プロパティを設定しています。

```
<?xml version="1.0"?>
```

```
<!-- controls\date\DateChooserDisabledOption.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Use tags.-->
    <mx:DateField>
            <mx:disabledDays>
                <mx:Number>0</mx:Number>
                <mx:Number>6</mx:Number>
            </mx:disabledDays>
    </mx:DateField>

    <!-- Use tag attributes.-->
    <mx:DateField disabledDays="[0,6]"/>
</mx:Application>
```

次の例は、DateChooser コントロールの dayNames、firstDayOfWeek、headerColor、selectableRange の各プロパティを、initialize イベントを使用して設定しています。

```
<?xml version="1.0"?>
<!-- controls\date\DateChooserInitializeEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

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

            private function dateChooser_init():void {
                myDC.dayNames=['Sun', 'Mon', 'Tue',
                    'Wed', 'Th', 'Fri', 'Sat'];
                myDC.firstDayOfWeek = 3;
                myDC.setStyle("headerColor", 0xff0000);
                myDC.selectableRange = {rangeStart: new Date(2004,0,1),
                    rangeEnd: new Date(2007,0,10)};
            }

            private function onScroll():void {
                myDC.setStyle("fontStyle", "italic");
            }
        ]]>
    </mx:Script>

    <mx:DateChooser id="myDC"
        width="200"
        creationComplete="dateChooser_init();"
        scroll="onScroll();"/>
</mx:Application>
```

このコードでは、selectableRange プロパティを設定するため、日付範囲の始まりと終わりを表す 2 つの Date オブジェクトを作成しています。ユーザーは、指定された範囲内でしか日付を選択できません。この例では、ユーザーが DateChooser コントロールを初めてスクロールした後に、fontStyle を italics に変更しています。

DateField コントロールによる日付のフォーマット

DateField コントロールの formatString プロパティを使用して、コントロールのテキストフィールドのストリングをフォーマットできます。formatString プロパティでは、"MM"、"DD"、"YY"、"YYYY"、区切り文字、および句読点を組み合わせて指定できます。デフォルト値は "MM/DD/YYYY" です。

次の例では、formatString プロパティを "MM/DD/YY" に設定して 2 桁の年を表示しています。

```
<?xml version="1.0"?>
<!-- controls\date\DateFieldFormat.mxml -->
<mx:Application  xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Label text="{date2.formatString}"/>
    <mx:DateField id="date2"
        editable="true"
        width="100"
        formatString="MM/DD/YY"/>
</mx:Application>
```

また DateField コントロールを使用すると、日付を任意の形式のストリングに変換して、コントロールのテキストフィールドに表示する formatter 関数を指定できます。DateField の labelFunction プロパティと DateFormatter クラスは、日付の形式を設定する際便利です。

デフォルトでは、DateField コントロールのテキストフィールドの日付が "MM/DD/YYYY" という形式でフォーマットされます。DateField コントロールの labelFunction プロパティを使用すると、テキストフィールドに表示される日付をフォーマットして、日付を含むストリングを返す関数を指定できます。この関数のシグネチャは次のとおりです。

```
public function formatDate(currentDate:Date):String {
  ...
  return dateString;
}
```

関数には別の名前を選択することもできますが、必ず Date 型の引数を 1 つ含み、テキストフィールドに表示するストリングとして日付を返す必要があります。次の例は、日付を "yyyy/mm/dd" ("2005/11/24" など) の形式で表示する formatDate() 関数を定義しています。この関数は、DateFormatter オブジェクトを使用して形式を設定しています。

```
<?xml version="1.0"?>
<!-- controls\date\DateChooserFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function formatDate(date:Date):String {
                return dfconv.format(date);
            }
        ]]>
    </mx:Script>
```

```
    <mx:DateFormatter id="dfconv" formatString="YYYY/MM/DD"/>
    <mx:DateField id="df" labelFunction="formatDate" parseFunction="null"/>
</mx:Application>
```

parseFunction プロパティは、DateField コントロールのテキストフィールドにテキストとして入力された日付を解析して、コントロールに Date オブジェクトを返す関数を指定します。ユーザーによるテキストフィールドへの日付入力を許可しない場合は、labelFunction プロパティを設定する際に、parseFunction プロパティを null に設定します。

コントロールのテキストフィールドへのユーザーによる日付入力を許可する場合は、parseFunction プロパティに、テキストストリングを DateField コントロールで使用される Date オブジェクトに変換する関数を指定する必要があります。parseFunction プロパティが設定されている場合、通常、labelFunction プロパティに指定されている関数の反転を実行する必要があります。

parseFunction プロパティに指定されている関数のシグネチャを次に示します。

```
public function parseDate(valueString:String, inputFormat:String):Date {
   ...
   return newDate
}
```

ここで、valueString 引数は、ユーザーがテキストフィールドに入力するテキストストリングを含み、inputFormat 引数はストリング形式を含みます。たとえば、月、日、年の 2 文字を使用してテキストストリングを入力できる場合は、inputFormat 引数に "MM/DD/YY" を渡します。

ユーザーの操作

日付選択用カレンダーには、ユーザーが月を選択できる矢印ボタンがあります。特定の日付をマウスでクリックして選択することができます。

前方移動用の矢印ボタンをクリックすると翌月のカレンダーが、後方移動用の矢印ボタンをクリックすると前月のカレンダーが表示されます。12 月のカレンダーで翌月表示用の矢印をクリックした場合、次の年の表示に切り替わります。同様に 1 月のカレンダーで前月表示用の矢印をクリックした場合は、前の年の表示に切り替わります。特定の日付をクリックすると、その日付が選択されます。デフォルトでは、選択された日付の周囲には緑の背景色が表示されます。現在の日付は、日付が白で背景が黒で表されます。現在選択されている日付をクリックすると、その選択が解除されます。

次のキーストロークを使用すると、ユーザーは DateChooser コントロールと DateField コントロールを移動できます。

キー	用途
← (左矢印)	選択した日付をその月の前の有効日に移動します。前の月には移動しません。
→ (右矢印)	選択した日付をその月の次の有効日に移動します。次の月には移動しません。

キー	用途
↑ (上矢印)	選択した日付を現在の曜日列の上にある前の有効日に移動します。前の月には移動しません。
↓ (下矢印)	選択した日付を現在の曜日列の下にある次の有効日に移動します。次の月には移動しません。
PageUp	前の月のカレンダーを表示します。
PageDown	次の月のカレンダーを表示します。
Home	選択した日付をその月の最初の有効日に移動します。
End	選択した日付をその月の最後の有効日に移動します。
+	次の年に移動します。
-	前の年に移動します。
Ctrl + ↓ (下矢印)	DateField のみ : DateChooser コントロールを開きます。
Ctrl + ↑ (上矢印)	DateField のみ : DateChooser コントロールを閉じます。
Esc	DateField のみ : 処理をキャンセルします。
Enter	DateField のみ : 日付を選択して、DateChooser コントロールを閉じます。

メモ

ユーザーは、ナビゲーションキーストロークを使用する前にコントロールを選択する必要があります。DateField コントロールでは、記載されているキーストロークはすべて、日付選択用カレンダーが表示されている場合にのみ動作します。

LinkButton コントロール

LinkButton コントロールは、オプションでアイコンをサポートする単一行のハイパーリンクを作成します。LinkButton コントロールを使用すると、Web ブラウザで特定の URL を開くことができます。

次の例は 3 つの LinkButton コントロールを示しています。

LinkButton コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	テキスト全体を表示できるだけの幅と高さです。
最小サイズ	0
最大サイズ	未定義

LinkButton コントロールの作成

LinkButton コントロールは、次のように、<mx:LinkButton> タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- controls\button\LBSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:HBox>
        <mx:LinkButton label="link1"/>
        <mx:LinkButton label="link2"/>
        <mx:LinkButton label="link3"/>
    </mx:HBox>
</mx:Application>
```

次のコードには、特定の URL を Web ブラウザのウィンドウに表示する LinkButton コントロールが1つ含まれています。

```
<?xml version="1.0"?>
<!-- controls\button\LBSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:LinkButton label="ADBE"
        width="100"
        click="navigateToURL(new URLRequest('http://quote.yahoo.com/q?s=ADBE'),
'quote')"/>
</mx:Application>
```

この例では、navigateToURL() メソッドを使用して URL を開いています。

マウスポインタを LinkButton コントロール上に移動するか、Link コントロールをクリックすると、自動的に視覚的なキューが表示されます。前のコード例にはリンク処理ロジックはありませんが、マウスポインタをリンク上に移動するかリンクをクリックすると、色が変わります。

次のコード例には、ViewStack ナビゲーションコンテナ内を移動するための複数の LinkButton コントロールが含まれています。

```
<?xml version="1.0"?>
<!-- controls\button\LBViewStack.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox>

        <!-- Put the links in an HBox container across the top. -->
        <mx:HBox>
            <mx:LinkButton label="Link1"
                click="viewStack.selectedIndex=0;"/>
            <mx:LinkButton label="Link2"
                click="viewStack.selectedIndex=1;"/>
            <mx:LinkButton label="Link3"
                click="viewStack.selectedIndex=2;"/>
```

```
        </mx:HBox>

        <!-- This ViewStack container has three children. -->
        <mx:ViewStack id="viewStack">
            <mx:VBox width="150">
                <mx:Label text="One"/>
            </mx:VBox>
            <mx:VBox width="150">
                <mx:Label text="Two"/>
            </mx:VBox>
            <mx:VBox width="150">
                <mx:Label text="Three"/>
            </mx:VBox>
        </mx:ViewStack>
    </mx:VBox>
</mx:Application>
```

LinkButton コントロールのラベルは、その LinkButton コントロールの境界の中央に配置されます。`labelPlacement` プロパティに `right`、`left`、`bottom`、`top` のいずれかを指定することにより、テキストラベルをアイコンとの位置関係で配置できます。

LinkButton コントロールのユーザー操作

ユーザーが LinkButton コントロールをクリックすると、`click` イベントが送出されます。
LinkButton コントロールが有効になっている場合は、次のことが行われます。

- マウスポインタを LinkButton コントロール上に移動すると、LinkButton コントロールがロールオーバー状態であることを示す外観に変わります。
- LinkButton コントロールをクリックすると、その LinkButton コントロールに入力フォーカスが移動し、リンクが押された状態であることを示す外観に変わります。マウスボタンを離すと、LinkButton コントロールの外観がロールオーバー状態に戻ります。
- マウスボタンを押し下げたまま LinkButton コントロールの範囲外に移動すると、Link コントロールは元の状態に戻ります。入力フォーカスは維持されます。
- toggle プロパティが `true` に設定されている場合は、LinkButton コントロール上でマウスボタンを離さない限り、ボタンの状態は変化しません。

無効状態の LinkButton コントロールは、ユーザーの操作に関係なく無効状態で表示されます。無効状態では、マウスやキーボードのすべての操作が無視されます。

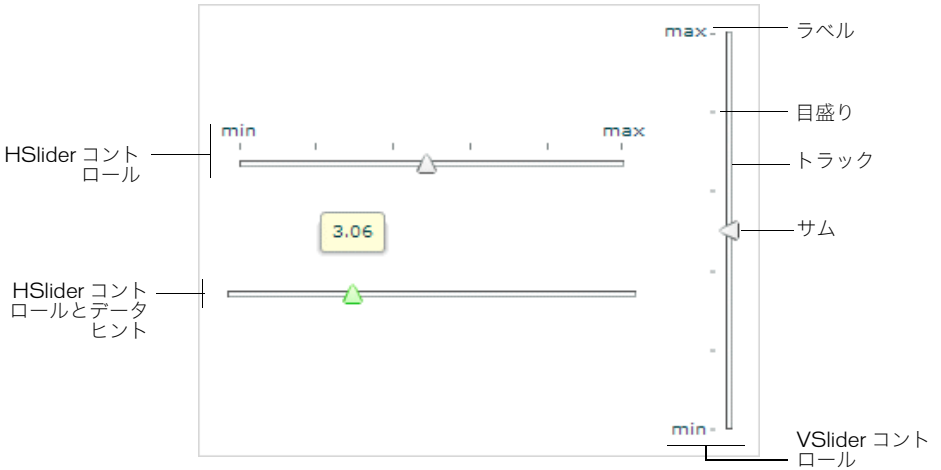
HSlider コントロールと VSlider コントロール

スライダコントロールを使用すると、スライダトラックの両端の間でスライダサムを移動することで値を選択できます。スライダの現在の値は、その両端 (スライダの最小値と最大値に対応) とサムの相対的な位置関係によって決まります。

デフォルトでは、スライダの最小値は 0、最大値は 10 に設定されています。スライダが最小値から最大値までの範囲を無段階に移動できるか、または段階的にのみ移動できるかは、スライダコントロールの設定に依存します。

スライダコントロールについて

Flex には 2 つのスライダが用意されています。HSlider (Horizontal Slider) コントロールは水平方向のスライダを、VSlider (Vertical Slider) コントロールは垂直方向のスライダを生成します。次の例は、HSlider コントロールと VSlider コントロールを示しています。



この例では、データヒント、スライダサム、トラック、目盛り、およびラベルが使われています。データヒント、目盛り、およびラベルは表示することも、非表示にすることも可能です。

次のコード例は、この図を再生成しています (注釈なし)。

```
<?xml version="1.0"?>
<!-- controls\slider\HSliderSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox>
        <mx:VBox>
            <mx:HSlider
                tickInterval="2"
```

```
                labels="['min', 'max']" height="150"/>
            <mx:HSlider/>
        </mx:VBox>

        <mx:VSlider
            tickInterval="2"
            labels="['min', 'max']"/>
    </mx:HBox>
</mx:Application>
```

HSlider コントロールと VSlider コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	**水平スライダ**　幅は 250 ピクセル、高さはスライダとそのラベルを表示できるだけのサイズになります。 **垂直スライダ**　高さは 250 ピクセル、幅はスライダとそのラベルを表示できるだけのサイズになります。
最小サイズ	なし
最大サイズ	なし

Slider コントロールの作成

HSlider コントロールは `<mx:HSlider>` タグを、VSlider コントロールは `<mx:VSlider>` タグを使用して MXML 内で定義します。コンポーネントを他のタグまたは ActionScript ブロックから参照する場合は、`id` の値を指定する必要があります。

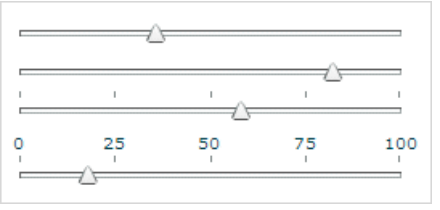
次のコード例は、4 つの HSlider コントロールを作成してます。

- 1 つ目のスライダは、最大値を 100 に設定し、ユーザーがスライダのサムを移動して 0 から 100 までの値を無段階に選択できるようにします。
- 2 つ目のスライダは、`snapInterval` プロパティを使用して、最小値から最大値の範囲に段階的な値を定義し、ユーザーが決められた値のみ選択できるようにします。この例では、`snapInterval` を 5 に設定しています。つまり、ユーザーは 0、5、10、15 のように段階的に値を選択できます。
- 3 つ目のスライダは、`tickInterval` プロパティを使用して、目盛りを追加し、その間隔を 25 に設定しています。したがって、0、25、50、75、100 の値に対応する位置に目盛りが表示されます。目盛りは `tickInterval` プロパティに 0 以外の値を設定すると表示されます。

■ 4 つ目のスライダは、labels プロパティを使用してラベルを追加し、各目盛りに設定しています。labels プロパティは、表示する値の配列を受け取り、自動的にそれらの値をスライダに沿って均等に配分します。最初の値は常にスライダの左端に対応し、最後の値は常にスライダの右端に対応します。

```
<?xml version="1.0"?>
<!-- controls\slider\HSliderSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox>
        <mx:HSlider
            maximum="100"/>
        <mx:HSlider
            maximum="100"
            snapInterval="5"/>
        <mx:HSlider
            maximum="100"
            snapInterval="5"
            tickInterval="25"/>
        <mx:HSlider
            maximum="100"
            snapInterval="5"
            tickInterval="25"
            labels="[0,25,50,75,100]"/>
    </mx:VBox>
</mx:Application>
```

この例では、次のイメージが作成されます。

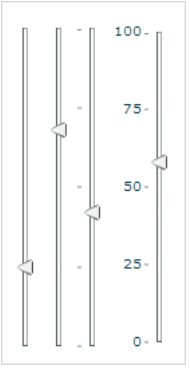


VSlider コントロールでも、同様の操作を実行できます。

```
<?xml version="1.0"?>
<!-- controls\slider\VSliderSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox>
        <mx:VSlider
            maximum="100"/>
        <mx:VSlider
            maximum="100"
            snapInterval="5"/>
        <mx:VSlider
            maximum="100"
```

```
            snapInterval="5"
            tickInterval="25"/>
        <mx:VSlider
            maximum="100"
            snapInterval="5"
            tickInterval="25"
            labels="[0,25,50,75,100]"/>
    </mx:HBox>
</mx:Application>
```

これで次のようなアプリケーションになります。



スライダの value プロパティを別のコントロールにバインドして、現在の値を表示させることもできます。次の例では value プロパティを Text コントロールにバインドしています。

```
<?xml version="1.0"?>
<!-- controls\slider\HSliderBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HSlider id="mySlider" maximum="100"/>
    <mx:Text text="{mySlider.value}"/>
</mx:Application>
```

このコードでは、次のイメージが作成されます。

スライダのイベントの使用

スライダコントロールでは、ユーザーがサムを最大値と最小値の範囲内で移動することによって値を選択できます。ユーザーがいつサムを移動したかを認識し、そのときにスライダに関連付けられている値を取得するには、スライダのイベントを使用します。

次の表に、スライダコントロールによって送出されるイベントを示します。

イベント	説明
change	ユーザーがサムを移動したときに送出されます。liveDragging プロパティが true の場合、ユーザーがサムをドラッグしている間は、イベントが継続的に送出されます。liveDragging プロパティが false の場合、ユーザーがスライダサムを離したときに、イベントが送出されます。
thumbDrag	ユーザーがサムを移動したときに送出されます。
thumbPress	ユーザーがマウスポインタを使ってサムを選択したときに送出されます。
thumbRelease	thumbPress イベントの発生後、ユーザーがマウスポインタを離したときに送出されます。

次のコード例は、ユーザーがスライダサムを離したときに change イベントを使用して、TextArea コントロールにスライダの現在の値を表示しています。

```
<?xml version="1.0"?>
<!-- controls\slider\HSliderEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.events.SliderEvent;
            import mx.controls.sliderClasses.Slider;

            private function sliderChange(event:SliderEvent):void {
                var currentSlider:Slider=Slider(event.currentTarget);
                thumb.text=String(currentSlider.value);
            }
        ]]>
    </mx:Script>

    <mx:HSlider change="sliderChange(event);"/>
    <mx:TextArea id="thumb"/>
</mx:Application>
```

スライダコントロールの liveDragging プロパティはデフォルトで false に設定されています。つまり、ユーザーがスライダサムを離したときに change イベントが送出されます。次の例のように、liveDragging に true を設定した場合は、ユーザーがサムを移動している間、change イベントが継続的に送出されます。

```
<?xml version="1.0"?>
<!-- controls\slider\HSliderEventLiveDrag.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.events.SliderEvent;
            import mx.controls.sliderClasses.Slider;

            private function sliderChangeLive(event:SliderEvent):void {
                var currentSlider:Slider=Slider(event.currentTarget);
                thumbLive.text=String(currentSlider.value);
            }
        ]]>
    </mx:Script>

    <mx:HSlider
        liveDragging="true"
        change="sliderChangeLive(event);"/>
    <mx:TextArea id="thumbLive"/>
</mx:Application>
```

複数のサムの使用

スライダコントロールで２つのサムを使用するように設定できます。サムを１つだけ使用するようにスライダを設定した場合は、スライダの端から端までの任意の位置にサムを移動できます。２つのサムを使用するように設定した場合、一方のサムがもう一方のサムを越えて移動することはできません。

２つのサムを使うようにスライダコントロールを設定した場合は、そのスライダコントロールの values プロパティを使用して、各サムの現在の値にアクセスします。values プロパティは、サムの現在の値を保持する２つのエレメントから成る配列です。次にその例を示します。

```
<?xml version="1.0"?>
<!-- controls\slider\HSliderMultThumb.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.events.SliderEvent;
            import mx.controls.sliderClasses.Slider;

            private function sliderChangeTwo(event:SliderEvent):void {
                var ct:Slider=Slider(event.currentTarget);
                thumbTwoA.text=String(ct.values[0]);
                thumbTwoB.text=String(ct.values[1]);
                thumbIndex.text=String(event.thumbIndex);
            }
        ]]>
    </mx:Script>
```

```
    <mx:HSlider thumbCount="2"
        change="sliderChangeTwo(event);"/>
    <mx:TextArea id="thumbTwoA"/>
    <mx:TextArea id="thumbTwoB"/>
    <mx:TextArea id="thumbIndex"/>
</mx:Application>
```

また、この例には、イベントオブジェクトの thumbIndex プロパティが使用されています。このプロパティは、ユーザーによって1つ目のサムの位置が変更されていた場合は 0 を、2 つ目のサムの位置が変更されていた場合は 1 を保持します。

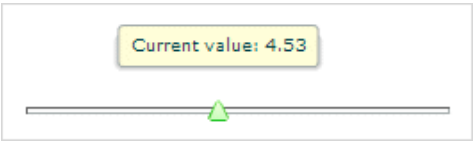
データヒントの使用

デフォルトでは、スライダのサムを選択すると、そのスライダの現在の値を示すデータヒントが表示されます。選択されたサムを移動すると、データヒントの値もそれに伴って更新されます。データヒントは、showDataTip プロパティを false に設定することによって無効にできます。

dataTipFormatFunction プロパティを使用してコールバック関数を指定することにより、データヒントテキストの形式を設定できます。この関数はデータヒントテキストを保持する String 型のパラメータを 1 つ取り、新しいデータヒントテキストが格納されたストリングを返します。次にその例を示します。

```
<?xml version="1.0"?>
<!-- controls\slider\HSliderDataTip -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function myDataTipFunc(val:String):String {
                return "Current value: " + String(val);
            }
        ]]>
    </mx:Script>

    <mx:HSlider
        height="80"
        dataTipFormatFunction="myDataTipFunc"/>
</mx:Application>
```

このコードでは、次のイメージが作成されます。



この例のデータヒント関数は、データヒントテキストの前に "Current value: " というストリングを付加します。この例を応用し、データヒントの前にドル記号 ($) が挿入されるようにすれば、商品の価格を制御するスライダを作成することもできます。

キーボード操作

スライダコントロールにフォーカスが置かれているとき、HSlider コントロールと VSlider コントロールでは、次のようなキーボード操作を実行できます。

キー	説明
← (左矢印)	HSlider コントロールの値を 1 スナップ間隔分減らします。スナップ間隔が指定されていない場合は、1 ピクセル分減らします。
→ (右矢印)	HSlider コントロールの値を 1 スナップ間隔分増やします。スナップ間隔が指定されていない場合は、1 ピクセル分増やします。
Home	HSlider コントロールのサムを最小値まで移動します。
End	HSlider コントロールのサムを最大値まで移動します。
↑ (上矢印)	VSlider コントロールの値を 1 スナップ間隔分増やします。スナップ間隔が指定されていない場合は、1 ピクセル分増やします。
↓ (下矢印)	VSlider コントロールの値を 1 スナップ間隔分減らします。スナップ間隔が指定されていない場合は、1 ピクセル分減らします。
PageDown	VSlider コントロールのサムを最小値まで移動します。
PageUp	VSlider コントロールのサムを最大値まで移動します。

SWFLoader コントロール

SWFLoader コントロールを使用すると、ある Flex 2 アプリケーションを別の Flex アプリケーションにロードできます。Loader コントロールには、そのコンテンツを拡大 / 縮小するためのプロパティが実装されています。また、表示するコンテンツに合わせて、Loader コントロールのサイズが自動的に変わるようにすることも可能です。デフォルトでは、SWFLoader コントロールのサイズに合うようにコンテンツが拡大 / 縮小されます。また、プログラムを使用して要求時にコンテンツをロードし、ロード操作の進行状況を監視できます。

SWFLoader コントロールではさらに、GIF、JPEG、PNG、SVG、または SWF ファイルをアプリケーションにロードすることができます。SWF ファイルには Flex 2 アプリケーションは含まれません。

メモ

Flex にはまた、GIF、JPEG、PNG、SVG、または SWF ファイルをロードするための Image コントロールもあります。通常、静止グラフィックファイルと SWF ファイルをロードするには Image コントロールを使用し、Flex 2 アプリケーションを SWF ファイルとしてロードするには SWFLoader コントロールを使用します。Image コントロールは、カスタムセルレンダラーとアイテムエディタでも使用できるように設計されています。

Image コントロールの詳細については、301 ページの「Image コントロール」を参照してください。SWFLoader コントロールを使用して Flex アプリケーションをロードする方法の詳細については、300 ページの「SWFLoader コントロールを使用した Flex データサービス アプリケーションのロード」を参照してください。

SWFLoader コントロールはフォーカスを取得できません。ただし、SWFLoader コントロール内にロードしたコンテンツ自体は、フォーカスの取得やフォーカス操作が可能です。

SWFLoader コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	ロードしたコンテンツ全体を表示するのに必要な幅と高さです。
最小サイズ	0
最大サイズ	未定義

SWFLoader コントロールの作成

SWFLoader コントロールは、次のように、<mx:SWFLoader> タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- controls\swfloader\SWFLoaderSimple.mxml-->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:SWFLoader id="loader1" source="FlexApp.swf"/>
</mx:Application>
```

Image コントロールと同様に、Embed ステートメントと SWFLoader コントロールを併用しても、次の例に示すように、イメージをアプリケーションに埋め込むことができます。

```
<?xml version="1.0"?>
<!-- controls\swfloader\SWFLoaderSimpleEmbed.mxml-->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:SWFLoader id="loader1" source="@Embed(source='flexapp.swf')"/>
</mx:Application>
```

SWFLoader コントロールを SVG ファイルで使用する場合、Embed ステートメントを使用してのみファイルをロードできます。SVG ファイルを実行時にロードすることはできません。埋め込みリソースの詳細については、302 ページの「イメージの読み込みについて」、および 1011 ページ、第 30 章の「アセットの埋め込み」の Image コントロールの説明を参照してください。

この手法は、主に、アプリケーションにグラフィックやアニメーションを追加する SWF ファイルを扱うものです。複雑な操作を実装した SWF ファイルには向いていません。複雑な操作を要求する SWF ファイルを読み込む場合は、カスタムコンポーネントを作成することをお勧めします。カスタムコンポーネントの詳細については、Flex 2 コンポーネントの作成と拡張を参照してください。

ロードした Flex 2 アプリケーションの操作

"FlexApp.mxml" ファイルを使用した次の例では、2 つの Label コントロール、1 つの変数、および変数を変更する 1 つのメソッドを定義する単純な Flex アプリケーションを示します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="200" width="200">

    <mx:Script>
        <![CDATA[
            [Bindable]
            public var varOne:String = "This is a public variable";

            public function setVarOne(newText:String):void {
                varOne=newText;
            }
        ]]>
    </mx:Script>

    <mx:Label id="lblOne" text="I am here"/>
    <mx:Label text="{varOne}"/>

    <mx:Button label="Nested Button" click="setVarOne('Nested button
pressed.');"/>

</mx:Application>
```

この例をまず "FlexApp.SWF" ファイルにコンパイルし、次に SWFLoader コントロールを使用して、このファイルを次の例のように別の Flex アプリケーションにロードします。

```
<?xml version="1.0"?>
<!-- controls\swfloader\SWFLoaderInteract.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.managers.SystemManager;
```

```
            import mx.controls.Label;

            [Bindable]
            public var loadedSM:SystemManager;

            // Initialize variables with information from
            // the loaded application.
            private function initNestedAppProps():void {
                loadedSM = SystemManager(myLoader.content);
            }

            // Update the Label control in the outer application
            // from the Label control in the loaded application.
            public function updateLabel():void {
                lbl.text=loadedSM.application["lblOne"].text;
            }

            // Write to the Label control in the loaded application.
            public function updateNestedLabels():void {
                loadedSM.application["lblOne"].text = "I was just updated";
                loadedSM.application["varOne"] = "I was just updated";
            }

            // Write to the varOne variable in the loaded application
            // using the setVarOne() method of the loaded application.
            public function updateNestedVarOne():void {
                FlexApp(loadedSM.application).setVarOne("Updated varOne!");
            }
        ]]>
    </mx:Script>

    <mx:Label id="lbl"/>
    <mx:SWFLoader id="myLoader" width="300"
        source="FlexApp.swf"
        creationComplete="initNestedAppProps();"/>

    <mx:Button label="Update Label Control in Outer Application"
        click="updateLabel();"/>
    <mx:Button label="Update Nested Controls"
        click="updateNestedLabels();"/>
    <mx:Button label="Update Nexted varOne"
        click="updateNestedVarOne();"/>
</mx:Application>
```

このアプリケーションは実行時に SWF ファイルをロードします。SWF ファイルは埋め込まれません。SWFLoader タグを使用して Flex 2 アプリケーションを埋め込む方法については、1011 ページ、第 30 章の「アセットの埋め込み」を参照してください。

上の例では、SWFLoader コントロールの creationComplete イベントを使用して 2 つの変数を初期化します。1 つめの変数にはロードされた Flex アプリケーションの SystemManager オブジェクトへの参照が、2 つめの変数にはロードされたアプリケーションの Label コントロールへの参照が含まれます。

ユーザーが外側のアプリケーションで最初の Button コントロールをクリックすると、ロードされたアプリケーションの Label コントロールから外側のアプリケーションの Label コントロールに、Flex によってテキストがコピーされます。

ユーザーが 2 つめの Button コントロールをクリックすると、ロードされたアプリケーションの Label コントロールと *varOne* 変数に、Flex によってテキストが書き込まれます。

3 つめの Button コントロールをクリックすると、Flex はロードされたアプリケーションの setVarOne() メソッドを使用して、ロードされたアプリケーションで定義されている *varOne* 変数にテキストを書き込みます。

SWFLoader コントロールを使用した Flex データサービス アプリケーションのロード

Flex データサービス のユーザーは、SWFLoader コントロールを使用して Flex アプリケーションをロードできます。次のコード例は、"buttonicon.mxml" ファイルをロードしています。"buttonicon.mxml" は、252 ページの「Button コントロールへのアイコンの埋め込み」にある例です。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:SWFLoader source="buttonicon.mxml.swf" scaleContent="false"/>
</mx:Application>
```

この例では、source プロパティの値を "buttonicon.mxml.swf" に設定します。Adobe Flex データサービスによって buttonicon.mxml ファイルがコンパイルされ、SWF ファイルがメインアプリケーションに返されます。この値を "buttonicon.swf" に設定すると、Flex データサービス は SWF ファイルが存在すればそれを返しますが、存在しない場合は buttonicon.mxml のコンパイルは行いません。

アプリケーションクラスの外部化

SWFLoader コントロールを使用してロードするアプリケーションのサイズを縮小するには、ロード対象のアプリケーションによって重複して組み込まれるフレームワーククラスを外部化するようにロード済みのアプリケーションに指示します。この結果、ロード済みアプリケーションには必要とするクラスのみ組み込まれるため、ロード済みアプリケーションのサイズは小さくなり、一方でフレームワークコードと他の依存関係はロード対象のアプリケーションに組み込まれます。

フレームワーククラスを外部化するには、mxmlc コマンドで link-report オプションを使用して、ロード対象のアプリケーションからリンカーレポートを生成します。次に、ロード済みアプリケーションをコンパイルするときに、mxmlc コンパイラで load-externs オプションを使用してこのレポートを指定します。

フレームワーククラスを外部化するには、以下を実行します。

1. ロード対象のアプリケーションのリンカーレポートを生成します。

```
mxmlc -link-report=report.xml MyApplication.mxml
```

2. このリンクレポートを使用してロード済みアプリケーションをコンパイルします。

```
mxmlc -load-externs=report.xml MyLoadedApplication.mxml
```

3. ロード対象のアプリケーションをコンパイルします。

```
mxmlc MyApplication.mxml
```

メモ

load-externs オプションを使用してロード済みアプリケーションの依存関係を外部化する場合、ロード済みアプリケーションが Adobe Flex の将来のバージョンと互換性がない可能性があります。したがって、アプリケーションの再コンパイルが必要となる場合があります。将来の Flex アプリケーションでアプリケーションを確実にロードできるようにするには、アプリケーションとアプリケーションが必要とするすべてのクラスをコンパイルします。

詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 9 章の「Flex コンパイラの使用」を参照してください。

SWFLoader コントロールのサイズ設定

SWFLoader コントロールは、scaleContent プロパティを使用してサイズ変更の動作を制御できます。scaleContent プロパティを true に設定した場合、コンテンツがコントロールの境界に合わせて拡大 / 縮小されます。ただし、デフォルトでは、イメージの縦横比が維持されます。

Image コントロール

Adobe Flex は、GIF、JPEG、PNG、SVG、SWF ファイルなど、複数のイメージ形式をサポートしています。Image コントロールを使用すると、これらのイメージをアプリケーションに読み込むことができます。

メモ

Flex には、Flex 2 アプリケーションをロードする SWFLoader コントロールもあります。静止グラフィックファイルと SWF ファイルのロードには、通常 Image コントロールを使用し、Flex 2 アプリケーションのロードには、SWFLoader コントロールを使用します。Image コントロールは、カスタムアイテムレンダラーとアイテムエディタでも使用できるように設計されています。SWFLoader コントロールの詳細については、296 ページの「SWFLoader コントロール」を参照してください。

イメージの読み込みについて

Flex ではイメージの読み込みについて、実行時に GIF、JPEG、PNG、および SWF ファイルを読み込む方法、およびコンパイル時に GIF、JPEG、PNG、SVG、および SWF ファイルを埋め込む方法をサポートしています。どちらの方法を選択するかは、イメージのファイルタイプとアプリケーションの特性によって異なります。

イメージを埋め込んだ場合、埋め込まれたイメージは Flex SWF ファイルの一部となるため、ロードは直接的に行われます。ただし、アプリケーションのサイズが増えるので、アプリケーションの初期化プロセスは遅くなります。また、埋め込みイメージは、イメージファイルに変更を加えるたびにアプリケーションを再コンパイルする必要があります。リソース埋め込みの概要については、1011 ページ、第 30 章の「アセットの埋め込み」を参照してください。

リソースを埋め込まない場合は、実行時にリソースをロードします。SWF ファイルが実行されるローカルなファイルシステムからリソースをロードすることもできますし、ネットワーク上の HTTP 要求を通じて (通常の方法)、リモートリソースにアクセスすることもできます。イメージは、Flex アプリケーションとは独立しているため、イメージの名前さえ同じであれば、修正を加えたとしても再コンパイルの必要はありません。イメージの参照により、アプリケーションの初期ロード時間に余分なオーバーヘッドが生じることはありません。ただし、イメージを Adobe Flash Player にロードするときに遅延が生じる場合があります。

SWF ファイルは、ローカルまたはネットワーク上の両方の外部リソースにアクセスすることができません。アクセスできるのは、いずれか一方だけです。SWF ファイルがどちらにアクセスできるようにするかは、アプリケーションのコンパイル時に `use-network` フラグを使用して決定します。`use-network` を `false` に設定すると、ローカルファイルシステムのリソースにアクセスできますが、ネットワーク上のリソースにはアクセスできません。デフォルトの `true` に設定すると、ネットワーク上のリソースにアクセスできますが、ローカルファイルシステムのリソースにはアクセスできません。

`use-network` フラグの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 9 章の「Flex コンパイラの使用」を参照してください。

実行時にイメージをロードする場合は、Flash Player のセキュリティ制限に注意する必要があります。たとえば、URL を使用してイメージを参照できますが、デフォルトのセキュリティ設定により、Flex アプリケーションは、対象アプリケーションと同じドメインに保存されたリソースにしかアクセスできません。他のサーバー上のイメージにアクセスするには、"crossdomain.xml" ファイルを使用する必要があります。

アプリケーションのセキュリティの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 4 章の「Flex セキュリティの適用」を参照してください。

SVG 描画の制限

Flex で SVG ファイルを操作する場合は、次の制限に注意する必要があります。

- SVG ファイルはアプリケーションに埋め込むことができるだけで、実行時にロードすることはできません。
- SMIL とアニメーションはサポートされていません。
- マスキングとフィルタはサポートされていません。
- パターン塗りつぶしと一部の高度なグラデーションはサポートされていません。
- ユーザー操作とスクリプトはサポートされていません。
- SVG テキストは、検索および選択不可能な SWF シェイプアウトラインとしてレンダリングされます。つまり、Flash Player のネイティブテキストとしてはレンダリングされません。

Image コントロールを使用したイメージ読み込みの制御

Image コントロールは、イメージの読み込みに関して、次のアクションをサポートしています。

- イメージパスの指定
- イメージのサイズ設定
- Canvas コンテナへのイメージの配置
- 可視性 (表示 / 非表示) の設定

イメージパスの指定

Image コントロールの source プロパティの値には、読み込むイメージファイルへの相対パスか絶対パス、または URL を指定します。相対パスを指定する場合、そのタグが使われているファイルのディレクトリが起点となります。詳細については、303 ページの「イメージパスの指定」を参照してください。

source プロパティには次の形式があります。

- `source="@Embed(source='relativeOrAbsolutePath')"`

  参照されたイメージは、Flex がアプリケーション用 SWF ファイルを生成するコンパイル時に、生成済み SWF ファイルにパッケージ化されます。埋め込むことができるのは、GIF、JPEG、PNG、SVG、SWF のファイルです。イメージを埋め込む場合、source プロパティの値は、ローカルファイルシステム上のファイルへの相対パスまたは絶対パスである必要があります。URL を指定することはできません。

  次の例は、JPEG イメージを Flex アプリケーションに読み込んでいます。

```
<?xml version="1.0"?>
<!-- controls\image\ImageSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Image id="loader1" source="@Embed(source='logo.jpg')"/>
</mx:Application>
```

  この例では、イメージのサイズは、イメージファイルのデフォルトサイズになります。

■ source="relativeOrAbsolutePathOrURL"

参照されたイメージファイルは、生成済み SWF ファイルの一部としてパッケージ化されるのではなく、実行時に Flex によってロードされます。参照できるのは、GIF、JPEG、PNG、および SWF のファイルだけです。use-network を false に設定すると、ローカルファイルシステムのリソースにアクセスできますが、ネットワーク上のリソースにはアクセスできません。デフォルトの true に設定すると、ネットワーク上のリソースにアクセスできますが、ローカルファイルシステムのリソースにはアクセスできません。

次の例は、実行時に JPEG イメージにアクセスしています。

```
<?xml version="1.0"?>
<!-- controls\image\ImageSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Image id="loader1" source="logo.jpg"/>
</mx:Application>
```

source プロパティに @Embed が指定されていないため、イメージは実行時にロードされます。

多くのアプリケーションでは、イメージを格納するための専用のディレクトリが作成されます。通常、このようなディレクトリは、アプリケーションのメインディレクトリのサブディレクトリになります。source プロパティは、イメージへの相対パスをサポートしているため、アプリケーションディレクトリに対する相対位置でイメージファイルを指定できます。

次の例では、すべてのイメージを、アプリケーションディレクトリの "assets" サブディレクトリに格納しています。

```
<?xml version="1.0"?>
<!-- controls\image\ImageSimpleAssetsDir.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Image id="loader1" source="@Embed(source='assets/logo.jpg')"/>
</mx:Application>
```

次の例では、相対パスを使用して、アプリケーションのルートディレクトリと同じレベルの "assets" ディレクトリにあるイメージを参照しています。

```
<?xml version="1.0"?>
<!-- controls\image\ImageSimpleAssetsDirTop.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Image id="loader1" source="@Embed(source='../assets/logo.jpg')"/>
</mx:Application>
```

URL を使用してイメージを参照することもできますが、デフォルトのセキュリティ設定により、Flex アプリケーションは、対象アプリケーションと同じドメインに保存されたリソースにしかアクセスできません。他のサーバー上のイメージにアクセスするには、"crossdomain.xml" ファイルを使用する必要があります。

次の例では、URL を使用してイメージを参照する方法を示します。

```
<?xml version="1.0"?>
<!-- controls\image\ImageSimpleAssetsURL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Image id="image1"
        source="http://localhost:8100/flex/assets/logo.jpg"/>
</mx:Application>
```

メモ

Flex アプリケーションと同じ Web サーバーにホストされたイメージを相対 URL で指定できます。ただし、その場合、ローカルでイメージにアクセスするのではなく、インターネットを介してロードする必要があります。

イメージの再利用

正しいイメージ読み込みシンタックスを使用することで、アプリケーション内で同じイメージを何度も使用できます。Flex はイメージを一度しかロードしませんが、ロードしたイメージを必要な回数だけ何度でも参照します。

イメージのサイズ設定

読み込んだイメージの高さと幅には、イメージファイルの設定が使われます。デフォルトでは、イメージのサイズ変更は行われません。

読み込むイメージの高さまたは幅を明示的に設定するには、Image コントロールの `height` プロパティと `width` プロパティを設定します。`height` プロパティまたは `width` プロパティを使用すると、親コントロールに依存しないサイズを設定できます。デフォルトで `scaleContent` プロパティが `true` に設定されているため、指定された高さと幅に合わせてイメージが拡大 / 縮小されます。デフォルトではイメージの縦横比が維持されるため、指定された領域全体を使用してイメージが表示されることはありません。イメージのサイズを明示的に設定する場合は、`scaleContent` プロパティを `false` に設定して、拡大 / 縮小を無効にしてください。イメージのサイズに無関係に、使用できる領域全体を使用してイメージを表示するには、`maintainAspectRatio` プロパティを `false` に設定します。イメージの縦横比の詳細については、306 ページの「サイズ設定時の縦横比の維持」を参照してください。

イメージのサイズがアプリケーションのレイアウトの一部として変更されるようにするには、`height` プロパティまたは `width` プロパティに、親コンテナに対する倍率 (%) を指定します。コンポーネントは、Flex により、指定されたプロパティに基づいてサイズ変更されます。`maxHeight`/`maxWidth` や `minHeight`/`minWidth` といったプロパティを使用して、サイズ変更の上限と下限を指定することもできます。サイズ変更の詳細については、461 ページ、第 13 章の「コンテナについて」を参照してください。

イメージのサイズ変更の一般的な用途に、イメージサムネールの作成があります。次の例に使用されているイメージの元々の高さと幅は 100 x 100 ピクセルです。高さと幅を 20 x 20 ピクセルに指定することにより、イメージのサムネールを作成できます。

```
<?xml version="1.0"?>
<!-- controls\image\ImageSimpleThumbnail.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Image id="image1"
        source="logo.jpg"
        width="20" height="20"/>
    <mx:Image id="image2"
        source="logo.jpg"/>
</mx:Application>
```

サイズ設定時の縦横比の維持

イメージの縦横比とは、イメージの高さと幅の比率のことです。たとえば、標準の NTSC テレビでは 4:3 の縦横比が使用されるのに対し、HDTV では 16:9 が使用されます。640 x 480 ピクセルの解像度に設定されたコンピュータのモニタでも、4:3 の縦横比が使用されています。正方形の縦横比は 1:1 になります。

すべてのイメージは、本来の縦横比を持っています。Image コントロールの `height` プロパティおよび `width` プロパティを使ってイメージのサイズを変更した場合、イメージが歪んで表示されないように、デフォルトで縦横比が維持されます。

イメージの縦横比を維持するため、`<mx:Image>` タグで指定された高さと幅で完全に描画することができない場合もあります。たとえば、元のイメージが 100 x 100 ピクセル、縦横比 1:1 の正方形のとき、次のステートメントでイメージをロードしたとします。

```
<mx:Image source="myImage.jpg" height="200" width="200"/>
```

イメージは、元のサイズの 4 倍に拡大され、200 x 200 ピクセル領域全体に描画されます。

次の例では、同じイメージの高さと幅を 150 x 200 ピクセル、縦横比 3:4 に設定しています。

```
<mx:Image source="myImage.jpg" height="150" width="200"/>
```

この例では、サイズ変更されたイメージに対して正方形の領域が指定されていません。デフォルトでは、Flex によりイメージの縦横比が維持されるため、イメージのサイズは、サイズ上の制約に従いながら縦横比を維持できる最大の大きさ、つまり、150 x 150 ピクセルに変更されます。残りの 50 x 150 ピエクセルの領域は空白のままになります。ただし、この空白の領域も `<mx:Image>` タグで確保されているため、他のコントロールやレイアウトエレメントが利用することはできません。

Resize エフェクトを使用することにより、トリガに反応してイメージの幅と高さを変更できます。Resize エフェクトを構成するときに、イメージの新しい高さと幅を指定します。デフォルトでイメージの縦横比が維持されるため、イメージは縦横比を維持しつつ、可能な限り新しいサイズに合うようにサイズ変更されます。次の例では、イメージの内側にマウスポインタを重ねるとイメージが拡大され、イメージの外側にマウスポインタを移動すると元のサイズに戻ります。

```
<?xml version="1.0"?>
<!-- controls\image\ImageResize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Resize id="resizeBig" widthFrom="120" widthTo="200"/>
    <mx:Resize id="resizeSmall" widthFrom="200" widthTo="120"/>

    <mx:Image width="120"
        source="@Embed('logo.jpg')"
        rollOverEffect="{resizeBig}"
        rollOutEffect="{resizeSmall}"/>
</mx:Application>
```

Resize エフェクトの詳細については、603 ページ、第 17 章の「ビヘイビアの使用」を参照してください。

イメージのサイズ変更時に縦横比を維持したくない場合は、maintainAspectRatio プロパティを false に設定します。デフォルトでは、maintainAspectRatio が true に設定され、縦横比の維持が有効になっています。

次の例は、縦横比に無関係に Flex のロゴの高さと幅のプロパティに明示的に値を指定してサイズを変更しています。

```
<?xml version="1.0"?>
<!-- controls\image\ImageResizeMaintainAR.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Image id="image1"
        source="@Embed('logo.jpg')"
        width="250" height="100"
        maintainAspectRatio="false"/>
</mx:Application>
```

縦横比を維持しない設定を選択することにより、イメージにあえて歪みを適用することもできます。たとえば、Adobe のロゴはデフォルトでは 136 x 47 ピクセルです。次の例では、イメージのサイズ変更で縦横比が維持されないため、イメージが歪んでいます。

Canvas コンテナへのイメージの配置

Canvas コンテナ、および layout プロパティが absolute に設定された Panel コンテナおよび Application コンテナを使用すると、コンテナ内でその子の位置を指定できます。イメージの絶対位置を指定するには、Image コントロールの x プロパティと y プロパティを使用します。

メモ	その他のすべてのコンテナでは、子の位置がコンテナによって制御され、x と y のプロパティは無視されます。

x プロパティと y プロパティは、Canvas コンテナのイメージの左上隅の位置を指定します。次の例では、イメージの位置を (40.40)、つまり、Canvas コンテナの左上隅から 40 ピクセル下、40 ピクセル右に設定しています。

```
<?xml version="1.0"?>
<!-- controls\image\ImageCanvas.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Canvas id="canvas0"
        borderStyle="solid"
        width="200"
        height="200">
        <mx:Image id="img0"
            source="@Embed('logo.jpg')"
            x="40" y="40"/>
    </mx:Canvas>
</mx:Application>
```

これで次のようなアプリケーションになります。

可視性 (表示 / 非表示) の設定

Image コントロールの visible プロパティを使用すると、ロードしたイメージを非表示の状態にすることができます。デフォルトでは、イメージは表示状態になります。イメージを非表示の状態にするには、visible プロパティを false に設定します。次の例は、イメージをロードし、そのイメージを非表示の状態にしています。

```
<?xml version="1.0"?>
<!-- controls\image\ImageVisible.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox id="vbox0"
        borderStyle="solid">
        <mx:Image id="img0"
```

```
            visible="false"
            source="@Embed(source='logo.jpg')"/>
    </mx:VBox>
</mx:Application>
```

イメージを非表示の状態に設定したとしても、VBox コンテナは、子をレイアウトするときに、イメージ用の領域を確保します。したがって、VBox はイメージファイルと同じサイズとなり、このアプリケーションで、<mx:Image> タグの後に Button コントロールを追加しても、イメージを表示状態にした場合と同じ位置にボタンが表示されます。

イメージを非表示にし、他の子のサイズと位置の設定時に親のコンテナでイメージを無視させる場合は、Image コントロールの includeInLayout プロパティを false に設定します。デフォルトでは、includeInLayout プロパティは true に設定されているため、イメージが非表示になっていても、表示されているかのようにサイズと位置がコンテナによって設定されます。

visible プロパティは、特定のイメージだけを表示して、それ以外を非表示にするような場合に利用します。たとえば、アプリケーションの領域に対し、3 つのイメージのうち、ユーザーの操作に基づいて、いずれか 1 つだけを表示したい場合があります。3 つあるイメージのうち、1 つだけ visible プロパティを true に設定し、それ以外のすべてのイメージは visible プロパティを false に設定して不可視状態にすることができます。

イメージのプロパティは ActionScript を使用して設定できます。次の例では、ユーザーがボタンをクリックすると、イメージの visible プロパティが true に設定されて、このイメージが表示されます。

```
<?xml version="1.0"?>
<!-- controls\image\ImageAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function showImage():void {
                image1.visible=true;
            }
        ]]>
    </mx:Script>
    <mx:VBox id="vbox0"
        width="80"
        height="80">
        <mx:Image id="image1"
            visible="false"
            source="@Embed(source='logo.jpg')"/>
        <mx:Button id="myButton"
            label="Show" click="showImage();"/>
    </mx:VBox>
</mx:Application>
```

次の例は、ユーザーがボタンをクリックする前後の状態を示しています。

visible プロパティの設定を使用して、イメージをロードするタイミングを制御することもできます。ページがロードされるときに、イメージを不可視状態にしたまま領域のみを割り当てることにより、パフォーマンスの低下を初期化段階に集中させることができます。アプリケーションが初期化された後のインターフェイスでは既にイメージがロードされているため、ユーザーはイメージを使った操作を快適に実行することができます。また、アプリケーションのサイズ変更や再レイアウトに伴う外観の乱れも、visible プロパティを設定することにより防ぐことができます。

Image コントロールの使用テクニック

製品カタログでは、ユーザーが商品をクリックすると、その商品のイメージが表示される、という手法がよく使われています。カタログ作成のテクニックの 1 つとして、カタログのイメージをすべて非表示状態にした上でアプリケーションにロードする方法があります。ユーザーが商品を選択すると該当するイメージが表示される、というものです。

ただし、この方法では、カタログに掲載するすべてのイメージについて Image コントロールを追加し、アプリケーションの起動時にイメージ (非表示) をロードしなければなりません。すべてのイメージを取り込まなければならないので、SWF ファイルのサイズが不必要に大きくなってしまうばかりか、非表示のイメージをロードすることにより、起動時間に悪影響が及ぶ可能性があります。

もっと効率のよい方法は、イメージを必要に応じてサーバーからダイナミックにロードすることです。非表示のイメージを格納する必要がないため SWF ファイルのサイズを抑えることができ、起動時間も短縮されます。

この方法を実装するためには、<mx:Image> タグを定義する ActionScript クラスが、SWFLoader クラスのサブクラスになっていることが前提です。イメージを作成した後で、イメージファイルを動的にロードする load() メソッドなど、SWFLoader コントロールのプロパティとメソッドを使用してイメージを操作できます。

メモ

SWFLoader コントロールの load() メソッドは、GIF、JPEG、PNG、および SWF のファイルに対してのみ機能します。このメソッドで SVG ファイルをロードすることはできません。

次の例は、ユーザーがボタンをクリックしたときに、load() メソッドで、"logo.jpg" のイメージを "logowithtext.gif" に置き換えています。

```
<?xml version="1.0"?>
<!-- controls\image\ImageLoad.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
  <mx:Script>
    <![CDATA[
      private function afterImage():void {
        image1.load('http://localhost:8100/flex/assets/logowithtext.jpg');
      }
    ]]>
  </mx:Script>

  <mx:VBox id="vbox0"
      width="150" height="100">
      <mx:Image id="image1" source="logo.jpg"/>
      <mx:Button id="myButton"
          label="Show Second Image"
          click="afterImage();"/>
  </mx:VBox>
</mx:Application>
```

この例では、アプリケーションがローカルファイルシステムのイメージにアクセスしています。イメージは URL によって埋め込みまたはアクセスされていません。したがって、このアプリケーションは、use-network フラグを false に設定してコンパイルする必要があります。

load() メソッドを呼び出しても、イメージを保持するコンテナは子のレイアウトを調整しません。そのため、置き換えるイメージは同じサイズにするのが普通です。新しいイメージが元のイメージと比べあまりに大きい場合、コンテナの他のコンポーネントが覆い隠されてしまいます。

置き換えるイメージは、アプリケーションでユーザーが行ったアクションに基づいて選択します。たとえば、ユーザーがリストボックスやデータグリッド内で選択した内容に基づいてイメージをロードすることができます。

次の例では、データグリッド内で選択されたアイテムのインデックス番号を使用して、ロードするイメージを決定しています。この例では、グリッド内のアイテムに対応する "1.jpg"、"2.jpg"、"3.jpg" のような名前がイメージに付けられています。

```
// グリッド内で選択されたアイテムに対応するイメージを取得します。
private function getImage():void {
   var cartGrid = dgrid;
   var imageSource:String = 'images/' + cartGrid.getSelectedIndex() + '.jpg';
   image1.load(imageSource);
}
```

この例では、イメージが "images" ディレクトリに格納されています。イメージのパスは、ディレクトリ名、インデックス番号、およびファイル拡張子 (.jpg) になります。

この関数を、次に示すように、change イベントのイベントリスナーとしてデータグリッドに登録する必要があります。

```
<mx:DataGrid id="dgrid" height="200" width="350" change="getImage();"/>
```

ユーザーがデータグリッド内で別のアイテムを選択すると、Flex によって getImage() 関数が呼び出され、表示されるイメージが更新されます。

この例では、選択されたアイテムのインデックスを使用してロードするイメージを決めていますが、選択されたアイテムの情報を使用してロードするイメージが決定されるように変更することも可能です。たとえば、グリッドに一連のオブジェクトがあり、各オブジェクトには対応するイメージ名を持つプロパティがあるとします。

`load()` メソッドの他にも、`percentLoaded` などの、SWFLoader コントロールの他のプロパティにアクセスすることもできます。特に有用なのは `percentLoaded` プロパティです。このプロパティを利用してプログレスバーを表示することにより、アプリケーションが単に処理を行っているだけで、フリーズしたわけではないことをユーザーに伝えることができます。SWFLoader のプロパティとメソッドの一覧については、『Adobe Flex 2 リファレンスガイド』を参照してください。

VideoDisplay コントロール

Flex には、ストリーミングメディアを Flex アプリケーションに組み込む方法として VideoDisplay コントロールが用意されています。これらのコントロールを使用して、Flex では FLV (Flash Video) ファイルをサポートしています。このセクションでは、アプリケーションでの VideoDisplay コントロールの使用方法について説明します。

Flex でのメディアの使用

メディア (ムービーやオーディオクリップなど) が Web ユーザーへの情報提供手段として使用される機会はますます多くなっています。そのため開発者には、メディアをユーザーにストリーミングで送信して制御させるための仕組みが必要とされています。たとえば、メディアコントロールの用途として次のような場合が考えられます。

- 会社の CEO からのビデオメッセージを再生する
- 映画や映画の予告編をストリーミングする
- 歌や歌の一部分をストリーミングする
- 学習教材をメディアの形で提供する

Flex のストリーミング VideoDisplay コントロールを使用すれば、Flash プレゼンテーションにストリーミングメディアを容易に組み込めます。それらのコントロールにより、Flex では FLV (Flash Video File) ファイルが利用できるようになります。このコントロールはビデオおよびオーディオデータで使用できます。アプリケーションで VideoDisplay コントロールだけを使用する場合、ユーザーがメディアファイルを制御する手段は提供されません。

メモ

VideoDisplay コントロールでは、早送りと巻き戻しの機能はサポートされていません。また、VideoDisplay ントロールではアクセシビリティとスタイルには対応していません。

VideoDisplay コントロールについて

Flex で作成される VideoDisplay コントロールには、メディアをユーザーがコントロールするためのユーザーインターフェイスは表示されません。これは単にメディアを格納および再生するためのコントロールです。

メモ | 何らかのメディアが再生されていない限り、ユーザーには何も表示されません。

コントロールの playheadTime プロパティには、ビデオファイル内における再生ヘッドの現在の位置が秒単位で保持されます。このコントロールが送出するイベントの大部分は、関連付けられたイベントオブジェクトに再生ヘッドの位置を含んでいます。再生ヘッド位置を一度使用すると、ビデオファイルが特定の位置に達したときにイベントが送出されます。詳細については、314 ページの「キューポイントの追加」を参照してください。

VideoDisplay コントロールでは、volume プロパティもサポートしています。このプロパティは、0.0 ～ 1.00 の値を取ります。0.0 はミュート、1.00 は最大ボリュームです。デフォルト値は 0.75 です。

メディアコンポーネントのサイズ設定

VideoDisplay コントロールで再生されるビデオメディアの外観は、次のプロパティによる影響を受けます。

- maintainAspectRatio
- height
- width

maintainAspectRatio を true (デフォルト) に設定すると、コントロールのサイズが設定された後に、メディアのサイズが再調整されます。メディアのサイズは、縦横比を維持するように設定されます。

コントロールの width および height プロパティを両方とも省略すると、コントロールのサイズは再生中のメディアと同じサイズになります。一方のプロパティのみを指定し、maintainAspectRatio プロパティが false になっている場合、他方のプロパティの値は再生中のメディアのサイズによって決まります。maintainAspectRatio プロパティが true ならば、サイズが変更されてもメディアの縦横比は維持されます。

次の例は、VideoDisplay コントロールを作成しています。

```
<?xml version="1.0"?>
<!-- controls\videodisplay\VideoDisplaySimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:VBox>
        <mx:VideoDisplay
            source="http://localhost:8100/flex/assets/MyVideo.flv"
            height="400"
            width="400"/>
    </mx:VBox>
</mx:Application>
```

デフォルトでは、VideoDisplay コントロールのサイズはメディアに合わせて調整されます。コントロールの width プロパティまたは height プロパティを指定し、いずれかがメディアのサイズよりも小さい場合、コンポーネントのサイズは変更されません。コンポーネントに合わせてメディアのサイズが調整されます。コントロールの再生領域がメディアのデフォルトサイズより小さい場合、メディアはコントロール内に収まるように縮小されます。

VideoDisplay コントロールのメソッドの使用

アプリケーションでは、VideoDisplay コントロールの close()、load()、pause()、play()、および stop() の各メソッドを使用できます。次の例では、2 つの Button コントロールのイベントリスナーで pause() メソッドと play() メソッドを使用して、FLV ファイルの一時停止と再生を行っています。

```
<?xml version="1.0"?>
<!-- controls\videodisplay\VideoDisplayStopPlay.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox>
        <mx:VideoDisplay id="myVid"
            source="http://localhost:8100/flex/assets/MyVideo.flv"/>
        <mx:Button label="Pause" click="myVid.pause();"/>
        <mx:Button label="Play" click="myVid.play();"/>
    </mx:VBox>
</mx:Application>
```

キューポイントの追加

キューポイントを使用すると、メディアの再生ヘッドが指定の位置に達した時点でイベントをトリガできます。キューポイントを使用するには、cuePointManagerClass プロパティを mx.controls.videoClasses.CuePointManager に設定してキューポイントの管理を有効にし、次にキューポイントの配列を VideoDisplay コントロールの cuePoints プロパティに渡します。この配列の各エレメントには 2 つのフィールドを含めます。1 つは name フィールドで、キューポイントに付ける任意の名前を指定します。もう 1 つは time フィールドで、キューポイントを関連付ける VideoDisplay コントロールの再生ヘッド位置を秒数で指定します。

VideoDisplay コントロールの再生ヘッドがいずれかのキューポイントに達すると、再生ヘッドから cuePoint イベントが送出されます。次に例を示します。

```
<?xml version="1.0"?>
<!-- controls\videodisplay\VideoDisplayCP.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.events.CuePointEvent;
```

```
            import mx.controls.videoClasses.CuePointManager;

            private function cpHandler(event:CuePointEvent):void {
                cp.text="got to cuepoint: " + event.cuePointName + " " +
                    String(event.cuePointTime);
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:VideoDisplay
            source="http://localhost:8100/flex/assets/MyVideo.flv"
            cuePointManagerClass="mx.controls.videoClasses.CuePointManager"
            cuePoint="cpHandler(event);">
            <mx:cuePoints>
                <mx:Object name="first" time="10"/>
                <mx:Object name="second" time="20"/>
            </mx:cuePoints>
        </mx:VideoDisplay>
        <mx:TextArea id="cp"/>
    </mx:VBox>
</mx:Application>
```

この例では、コントロールがキューポイントに達したとき、イベントリスナーで TextArea コントロールにストリングを表示しています。表示するストリングは、キューポイントの名前と時間です。

CuePointManager クラスの使用によるキューポイントの追加

VideoDisplay コントロールのキューポイントを設定するには、cuePointManager プロパティを使用します。このプロパティの型は CuePointManager であり、キューポイントをプログラムによって操作するために使用するメソッドは、次の例ように CuePointManager クラスで定義されます。

```
<?xml version="1.0"?>
<!-- controls\videodisplay\VideoDisplayCPManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.events.CuePointEvent;

            [Bindable]
            private var myCuePoints:Array = [
            { name: "first", time: 10},
            { name: "second", time: 20} ];

            // Set cue points using methods of the CuePointManager class.
            private function initCP():void {
                myVid.cuePointManager.setCuePoints(myCuePoints);
            }
```

```
            private var currentCP:Object=new Object();

            private function cpHandler(event:CuePointEvent):void {
                cp.text="go to cuepoint: " + event.cuePointName + " " +
                    String(event.cuePointTime);
                // Remove cue point.
                currentCP.name=event.cuePointName;
                currentCP.time=event.cuePointTime;
                myVid.cuePointManager.removeCuePoint(currentCP);
                // Display the number of remaining cue points.
                cp.text=cp.text + "\n" + "Cue points remaining: " +
                    String(myVid.cuePointManager.getCuePoints().length);
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:VideoDisplay id="myVid"
            initialize="initCP();"
            cuePointManagerClass="mx.controls.videoClasses.CuePointManager"
            source="http://localhost:8100/flex/assets/MyVideo.flv"
            cuePoint="cpHandler(event);"/>
        <mx:TextArea id="cp" width="200" />
    </mx:VBox>
</mx:Application>
```

カメラからのストリーミングビデオ

次の例のように、VideoDisplay.attachCamera() メソッドを使用してコントロールを設定すれば、カメラからのビデオストリームをコントロールに表示できます。

```
<?xml version="1.0"?>
<!-- controls\videodisplay\VideoDisplayCamera.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Define a variable of type Camera.
            import flash.media.Camera;
            public var cam:Camera;

            public function initCamera():void {
                // Initialize the variable.
                cam = Camera.getCamera();
                myVid.attachCamera(cam)
            }
        ]]>
```

```
    </mx:Script>

    <mx:VideoDisplay id="myVid"
        width="320" height="240"
        creationComplete="initCamera();"/>
</mx:Application>
```

この例では、VideoDisplay コントロールの creationComplete イベントのイベントハンドラに Camera オブジェクトを作成し、その Camera オブジェクトを attachCamera() メソッドに引数として渡しています。

Flash Media Server 2 での VideoDisplay コントロールの使用

VideoDisplay コントロールを使用すれば、次の例のように、Adobe の Macromedia® Flash® Media Server 2 からメディアストリームを読み込むことができます。

```
<?xml version="1.0"?>
<!-- controls\videodisplay\VideoDisplayFMS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox>
        <mx:Label text="RTMP FMS 2.0"/>
        <mx:VideoDisplay
            autoBandWidthDetection="false"
            source="rtmp://localhost/videodisplay/bike.flv"/>
    </mx:HBox>
</mx:Application>
```

この例では、"bike.flv" ファイルを Flash Media Server 2\applications\videodisplay\streams_definst_ ディレクトリに配置しています。

autoBandWidthDetection プロパティは、デフォルト値である false に明示的に設定されています。autoBandWidthDetection プロパティが true の場合、サーバーサイドファイル "main.asc" を Flash Media Server 2\applications\videodisplay\scripts ディレクトリに作成する必要があります。このファイルは次の関数を実装します。

```
application.onConnect = function(p_client, p_autoSenseBW) {}
application.calculateClientBw = function(p_client) {}
Client.prototype.getStreamLength = function(p_streamName) {}
```

以下は main.asc の実装の例を示したものです。

```
application.onConnect = function(p_client, p_autoSenseBW) {
  // ここにセキュリティコードを追加します。

  this.acceptConnection(p_client);

  if (p_autoSenseBW)
    this.calculateClientBw(p_client);
```

```
   else
      p_client.call("onBWDone");
}

Client.prototype.getStreamLength = function(p_streamName) {
   return Stream.length(p_streamName);
}

application.calculateClientBw = function(p_client) {
   // クライアントの BandWidth を設定するコードを追加します。
   // bytes_in および bytes_Out を返す p_client.getStats() を使用して
   // p_client.call("onBWCheck", result, p_client.payload) により
   // 帯域幅を確認します。
   p_client.call("onBWDone");
}
```

main.asc の詳細については、Flash Media Server 2 のマニュアルを参照してください。

Flash Media Server の AMF バージョンの指定

NetConnection、NetStream、または SharedObject の各オブジェクトをバイナリデータとして読み込みや書き込みを行うと、使用する AMF (Action Message Format) のバージョンがオブジェクトの `objectEncoding` プロパティで示されます。これは ActionScript 3.0 フォーマット、ActionScript 1.0 または ActionScript フォーマットです。`objectEncoding` プロパティの有効な値には次のようなものがあります。

AMF0　オブジェクトは ActionScript 1.0 および 2.0 の AMF 形式を使用して直列化されます。

AMF3　オブジェクトは ActionScript 3.0 の AMF 形式を使用して直列化されます。

DEFAULT　オブジェクトは Flash Player のバージョンのデフォルトの形式を使用して直列化されます。

Flash Media Server のすべてのバージョンが、`AMF0` のエンコードを使用します。そのため Flex アプリケーションでは、Flash Media Server で使用されるすべての NetConnection オブジェクトおよび SharedObject オブジェクトの `objectEncoding` プロパティを、`AMF0` に設定する必要があります。NetStream では、`objectEncoding` プロパティは読み取り専用です。自身のサーバーへの FAP または RTMP 接続を行う際は、常に AMF3 を使用できます。

ColorPicker コントロール

ColorPicker コントロールを使用すると、ユーザーはドロップダウン色見本パネル (パレット) から色を選択できます。このコントロールは、最初に選択された色付きのプレビューサンプルとして表示されます。ユーザーがコントロールを選択すると、色見本パネルが表示されます。このパネルには、選択した色のサンプルと色見本パネルが含まれています。デフォルトでは、この色見本パネルに Web セーフカラー (216 色、三原色のそれぞれの値が #CC0066 など 33 の倍数となっている) が表示されます。

詳細については、『Adobe Flex 2 リファレンスガイド』の ColorPicker を参照してください。

ColorPicker コントロールについて

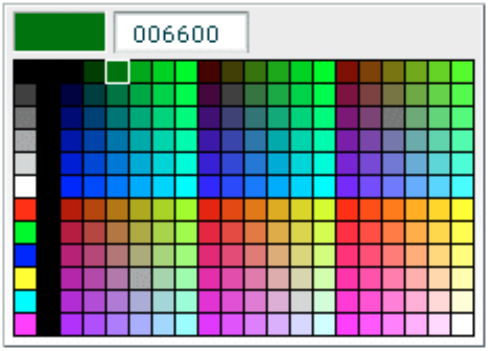
ColorPicker コントロールを開くと、アプリケーションの他のコントロール上に色見本パネルが拡張します。通常、下方向に開きます。色見本パネルがアプリケーションの下側の境界内には収まらないが、ColorPicker ボタンより上で収まる場合は、上方向に開かれます。

`showTextField` プロパティを `true` (デフォルト) に設定すると、パネルには選択した色のラベルが付いたテキストボックスが表示されます。テキストボックスを表示して、`editable` プロパティを `true` (デフォルト) に設定した場合、ユーザーは 16 進数値を入力することで色を指定できます。

次の例に、`mx:ColorPicker` タグのデフォルト設定を使用する畳まれた状態と展開された状態の ColorPicker コントロールを示します。

畳まれた状態

展開された状態



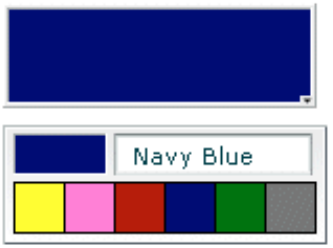
Flex は、データプロバイダから色見本パネルとテキストボックスを作成します。デフォルトでは、このコントロールは、すべての Web セーフカラーを含むデータプロバイダを使用します。独自のデータプロバイダを使用する場合、次の項目を指定できます。

表示する色　独自のデータプロバイダを使用する場合、色を指定する必要があります。

テキストボックスに表示する色用のラベル　テキストラベルを指定しない場合、16 進数のカラー値が使用されます。

各色に関する追加情報この情報には、ID や説明コメントなど、アプリケーションに使用できる任意の情報を指定できます。

次の図に、カラーラベル値を含むカスタムデータプロバイダを使用する展開された状態の ColorPicker コントロールを示します。このコントロールは、スタイルを使用して表示エレメントのサイズも設定しています。



ColorPicker コントロールには、次のデフォルトのサイズ設定特性があります。

プロパティ	デフォルト値
デフォルトサイズ	ColorPicker : 22 x 22 ピクセル 色見本パネル : ColorPicker コントロールの幅に合わせて設定される
最小サイズ	0 x 0
最大サイズ	未定義

ColorPicker コントロールの作成

ColorPicker コントロールを MXML で定義するには `<mx:ColorPicker>` タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。たとえば、図の ColorPicker コントロールは、次の最低限のコードで作成されています。

```
<mx:ColorPicker id="cp"/>
```

ColorPicker コントロールでは色にリストデータプロバイダを使用します。この種のデータプロバイダの詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。データプロバイダを省略すると、Web セーフカラーを使用するデフォルトのデータプロバイダが使用されます。データプロバイダは、色の配列でも、オブジェクトの配列でもかまいません。次の例は、単純な色の配列を持つ ColorPicker を作成しています。より複雑なデータプロバイダの使用方法については、322 ページの「オブジェクトを使用した ColorPicker コントロールの作成」を参照してください。

```
<?xml version="1.0"?>
<!-- controls\colorpicker\CPSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            [Bindable]
            public var simpleDP:Array = ['0x000000', '0xFF0000', '0xFF8800',
                '0xFFFF00', '0x88FF00', '0x00FF00', '0x00FF88', '0x00FFFF',
                '0x0088FF', '0x0000FF', '0x8800FF', '0xFF00FF', '0xFFFFFF'];
        ]]>
    </mx:Script>

    <mx:ColorPicker id="cp" dataProvider="{simpleDP}"/>
</mx:Application>
```

メモ

ColorPicker コントロールのデータは、<mx:dataProvider> 子タグを使用しても指定できます。例については、324 ページの「カスタムフィールド名の使用」を参照してください。
このセクションのいくつかの例では、配列の内容が静的なため、カスタムデータソースとして ArrayCollections ではなく単純な Arrays を使用しています。

通常、ColorPicker コントロールに対するユーザーの操作はイベントで処理します。次の例は、change イベントと open イベントのイベントリスナーを前の例の ColorPicker コントロールに追加しています。

```
<?xml version="1.0"?>
<!-- controls\colorpicker\CPEvents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            //Import the event classes.
            import mx.events.DropdownEvent;
            import mx.events.ColorPickerEvent;

            [Bindable]
            public var simpleDP:Array = ['0x000000', '0xFF0000', '0xFF8800',
               '0xFFFF00', '0x88FF00', '0x00FF00', '0x00FF88', '0x00FFFF',
               '0x0088FF', '0x0000FF', '0x8800FF', '0xFF00FF', '0xFFFFFF'];

            public function openEvt(event:DropdownEvent):void {
                forChange.text="Opened";
            }

            public function changeEvt(event:ColorPickerEvent):void {
                forChange.text="Selected Item: "
                   + event.currentTarget.selectedItem + " Selected Index: "
                   + event.currentTarget.selectedIndex;
            }
        ]]>
    </mx:Script>
```

```
    <mx:VBox>
        <mx:TextArea id="forChange"
            width="150"/>
        <mx:ColorPicker id="cp"
            dataProvider="{simpleDP}"
            open="openEvt(event);"
            change="changeEvt(event);"/>
    </mx:VBox>
</mx:Application>
```

ColorPicker コントロールは、色見本パネルが開かれたときに open イベントを送出し、ユーザーの操作によってコントロールの値が変化したときに change イベントを送出します。イベントリスナーに渡されるオブジェクトの currentTarget プロパティには、ColorPicker コントロールへの参照が設定されています。この例では、ColorPicker コントロールの selectedItem プロパティおよび selectedIndex プロパティをイベントリスナーで使用しています。change イベントが発生すると、選択されたアイテムとアイテムのインデックスでコントロール内の TextArea コントロールが更新され、open イベントでは Opened という語が表示されます。

カラー値の配列から ColorPicker コントロールを作成した場合、target.selectedItem フィールドには 16 進数のカラー値が設定されます。オブジェクトの配列から作成した場合、target.selectedItem フィールドには選択されたアイテムに対応するオブジェクトへの参照が設定されます。

ColorPicker コントロールのアイテムのインデックスは 0 から始まります。つまり、値は 0、1、2、...、n - 1 となります。n はアイテムの総数です。したがって、target.selectedIndex の値は 0 から始まり、前の例の値 2 は、カラー 0xFF8800 のデータプロバイダエントリを指します。

オブジェクトを使用した ColorPicker コントロールの作成

ColorPicker コントロールのデータは、オブジェクトの配列を使用して設定することもできます。デフォルトでは、ColorPicker は、オブジェクトの次の 2 つのフィールドを使用します。1 つは color で、もう 1 つは label です。label フィールドの値は、色見本パネルのテキストフィールドのテキストを決定します。オブジェクトに label フィールドがない場合、テキストフィールドの color フィールドの値が使用されます。ColorPicker コントロールの colorField プロパティと labelField プロパティを使用すると、color フィールドと label フィールドに別の名前を指定できます。オブジェクトには、色の説明や内部カラー ID など、ActionScript で使用できる追加フィールドを設定できます。

例 : オブジェクトを使用する ColorPicker コントロール

次の例は、color、label、descript の 3 つのフィールドを持つオブジェクトの配列を使用する ColorPicker を示しています。

```
<?xml version="1.0"?>
<!-- controls\colorpicker\CPObjects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.events.ColorPickerEvent;
            import mx.events.DropdownEvent;

            [Bindable]
            public var complexDPArray:Array = [
                {label:"Yellow", color:"0xFFFF00",
                    descript:"A bright, light color."},
                {label:"Hot Pink", color:"0xFF66CC",
                    descript:"It's HOT!"},
                {label:"Brick Red", color:"0x990000",
                    descript:"Goes well with warm colors."},
                {label:"Navy Blue", color:"0x000066",
                    descript:"The conservative favorite."},
                {label:"Forest Green", color:"0x006600",
                    descript:"Great outdoorsy look."},
                {label:"Grey", color:"0x666666",
                    descript:"An old reliable."}]

            public function openEvt(event:DropdownEvent):void {
                descriptBox.text="";
            }

            public function changeEvt(event:ColorPickerEvent):void {
                descriptBox.text=event.currentTarget.selectedItem.label
                    + ": " + event.currentTarget.selectedItem.descript;
            }
        ]]>
    </mx:Script>

    <!-- Convert the Array to an ArrayCollection. Do this if
    you might change the colors in the panel dynamically. -->
    <mx:ArrayCollection id="complexDP" source="{complexDPArray}"/>

    <mx:VBox>
        <mx:TextArea id="descriptBox"
            width="150" height="50"/>
        <mx:ColorPicker id="cp"
            height="50" width="150"
            dataProvider="{complexDP}"
```

```
            change="changeEvt(event);"
            open="openEvt(event);"
            swatchWidth="25"
            swatchHeight="25"
            textFieldWidth="95"
            editable="false"/>
    </mx:VBox>
</mx:Application>
```

この例では、selectedItem プロパティには、選択されたアイテムを定義するオブジェクトへの参照が設定されています。selectedItem.label を使用してオブジェクトの label プロパティ (カラー名) にアクセスし、selectedItem.descript を使用してオブジェクトの descript プロパティ (カラーの説明) にアクセスしています。change イベントが発生すると、選択されたアイテムの label プロパティとアイテムの説明で TextArea コントロールが更新されます。open イベントでは、ユーザーが ColorPicker を開いて色見本パネルを表示するたびに、TextArea コントロールの現在のテキストがクリアされます。

この例ではまた、ColorPicker の複数のプロパティとスタイルを使用して、コントロールのビヘイビアと外観を指定しています。editable プロパティは、ユーザーがカラーラベルボックスに値を入力できないようにしています。このため、ユーザーはデータプロバイダからしか色を選択できなくなります。swatchWidth スタイルと swatchHeight スタイルは、色見本パネル内のカラーサンプルのサイズを制御し、textFieldWidth スタイルは、テキストフィールドで最も長いカラー名全体を収められるようにしています。

カスタムフィールド名の使用

color フィールドと label フィールドにカスタム名を使用する場合もあります。たとえば、データがカスタムの列名が設定されている外部データソースの場合が挙げられます。次のコードは、前の例を変更して、cName および cVal というカスタムのカラーフィールドとラベルフィールドを使用するようにしています。<mx:dataProvider> タグを使用してデータプロバイダを作成する方法も示しています。

```
<?xml version="1.0"?>
<!-- controls\colorpicker\CPCustomFieldNames.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.events.ColorPickerEvent;
            import mx.events.DropdownEvent;

            public function openEvt(event:DropdownEvent):void {
                descriptBox.text="";
            }

            public function changeEvt(event:ColorPickerEvent):void {
                descriptBox.text=event.currentTarget.selectedItem.cName
```

```
                    + ": " + event.currentTarget.selectedItem.cDescript;
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:TextArea id="descriptBox"
            width="150" height="50"/>
        <mx:ColorPicker id="cp"
            height="50" width="150"
            labelField="cName"
            colorField="cVal"
            change="changeEvt(event)"
            open="openEvt(event)"
            swatchWidth="25"
            swatchHeight="25"
            textFieldWidth="95"
            editable="false">
            <mx:dataProvider>
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object cName="Yellow" cVal="0xFFFF00"
                            cDescript="A bright, light color."/>
                        <mx:Object cName="Hot Pink" cVal="0xFF66CC"
                            cDescript="It's HOT!"/>
                        <mx:Object cName="Brick Red" cVal="0x990000"
                            cDescript="Goes well with warm colors."/>
                        <mx:Object cName="Navy Blue" cVal="0x000066"
                            cDescript="The conservative favorite."/>
                        <mx:Object cName="Forest Green" cVal="0x006600"
                            cDescript="Great outdoorsy look."/>
                        <mx:Object cName="Grey" cVal="0x666666"
                            cDescript="An old reliable."/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ColorPicker>
    </mx:VBox>
</mx:Application>
```

ユーザーの操作

ColorPicker コントロールは編集できる場合とできない場合があります。編集不可能な ColorPicker コントロールでは、ユーザーは色見本パネルオプションの中から色を選択する必要があります。編集可能な ColorPicker コントロールでは、ユーザーは色見本パネルアイテムを選択するか、色見本パネル上部にあるラベルテキストフィールドに 16 進数のカラー値を直接入力できます。テキストボックスには、数字および範囲が a ～ f や A ～ F の大文字または小文字を入力できます。他の数値以外の文字はすべて無視されます。

マウスの操作

マウスを使用すると、移動してコントロールから選択できます。

- 畳まれた状態のコントロールをクリックすると、色見本パネルを表示または非表示にできます。
- 色見本パネルの見本をクリックすると、色見本を選択してパネルを閉じることができます。
- パネル領域の外をクリックすると、選択せずにパネルを閉じることができます。
- テキストフィールド内をクリックすると、テキスト入力カーソルを移動できます。

キーボードショートカット

ColorPicker が編集可能で、色見本パネルにフォーカスがある場合、範囲が A ～ F と a ～ f の英字キー、数字キー、BackSpace キーおよび Delete キーで、カラーテキストボックスにテキストを入力または削除できます。また、次のキーストロークを使用してドロップダウンリストを制御できます。

キー	説明
Ctrl + ↓ (下矢印)	色見本パネルを開き、選択した色見本のフォーカスを配置します。
Ctrl + ↑ (上矢印)	色見本パネルが開いている場合、閉じます。
Home	選択した色を色見本パネルの行の最初の色の位置に移動します。列が 1 つしかない場合は無効です。
End	選択した色を色見本パネルの行の最後の色の位置に移動します。列が 1 つしかない場合は無効です。
PageUp	選択した色を色見本パネルの列の一番上の色の位置に移動します。列が 1 つしかない場合は無効です。
PageDown	選択した色を色見本パネルの列の一番下の色の位置に移動します。列が 1 つしかない場合は無効です。
Esc	ColorPicker の色を変更せずに、色見本パネルを閉じます。 このキーはほとんどの Web ブラウザでサポートされていません。
Enter	色見本パネルから現在の色を選択して、色見本パネルを閉じます。色見本をクリックするのと同じ効果です。フォーカスが編集可能な ColorPicker のテキストフィールドにある場合、フィールドテキストに指定された色を選択します。
矢印	色見本パネルが開いている場合、フォーカスを色見本グリッドの上下左右にある次の色に移動します。単一行の色見本パネルでは、上矢印キーと右矢印キーが同等で、下矢印キーと左矢印キーが同等となります。 複数行の色見本パネルでは、選択した色が前後の行の先頭または末尾に折り返されます。単一行の色見本パネルでは、行の先頭または末尾を超えてキーを押すと、行が折り返されます。 色見本パネルを閉じたもののまだフォーカスがある場合、上矢印キーと下矢印キーが無効になります。左矢印キーと右矢印キーでは、ColorPicker の選択を変更し、パネルが開いている場合と同様に色の間を移動できます。

メモ	色見本パネルが開いている場合、Tab キーや Shift + Tab キーを使用してフォーカスを別のオブジェクトに移動させることはできません。

Alert コントロール

Flex のあらゆるコンポーネントから Alert クラスの静的な show() メソッドを呼び出すことによって、モーダルなポップアップダイアログボックスを表示できます。このポップアップダイアログボックスには、メッセージの他、オプションでタイトル、ボタン、アイコンなどを実装できます。次の例は、Alert コントロールのポップアップダイアログボックスを示しています。

Alert コントロールは、ユーザーがこのコントロールのボタンを選択したとき、または Esc キーを押したときに閉じます。

Alert.show() メソッドのシンタックスは次のとおりです。

```
public static show(
  text:String,
  title:String = null,
  flags:uint = mx.controls.Alert.OK,
  parent:Sprite = null,
  clickListener:Function = null,
  iconClass:Class = null,
  defaultButton:uint = mx.controls.Alert.OK) : Alert
```

このメソッドは、Alert コントロールオブジェクトを返します。

次の表に、show() メソッドのパラメータを示します。

パラメータ	説明
text	(必須) ダイアログボックスに表示されるテキストメッセージを指定します。
title	ダイアログボックスのタイトルを指定します。省略した場合は、空白のタイトルバーが表示されます。
flags	ダイアログボックスに表示するボタンを指定します。オプションは次のとおりです。 mx.controls.Alert.OK [OK] ボタン mx.controls.Alert.YES [Yes] ボタン mx.controls.Alert.NO [No] ボタン mx.controls.Alert.CANCEL [Cancel] ボタン 各オプションはビット値であり、パイプ (|) 演算子を使用して、他のオプションと組み合わせることができます。ボタンの表示順序は、コードで指定した順序とは無関係にここに記載した順となります。デフォルト値は mx.controls.Alert.OK です。
parent	Alert コントロールの親オブジェクトです。
clickListener	ボタンの click イベントに対するリスナーを指定します。 このハンドラに渡されるイベントオブジェクトは、CloseEvent クラスのインスタンスです。イベントオブジェクトには、detail というフィールドがあります。このフィールドは、クリックされたボタンの flag 値 (mx.controls.Alert.OK、mx.controls.Alert.CANCEL、mx.controls.Alert.YES、mx.controls.Alert.NO のいずれか) になります。
iconClass	ダイアログボックスでメッセージテキストの左側に表示されるアイコンを指定します。
defaultButton	flags パラメータの有効な値のいずれかを使用して、デフォルトのボタンを指定します。ユーザーが Enter キーを押したときに選択されるボタンになります。デフォルト値は Alert.OK です。 Escape キーを押すと、まるでそのボタンをクリックしたかのように、[キャンセル] または [いいえ] ボタンがトリガされます。

Alert コントロールを使用するには、まずアプリケーションに Alert クラスを読み込んでから、次の例のように show() メソッドを呼び出します。

```
<?xml version="1.0"?>
<!-- controls\alert\AlertSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
        ]]>
    </mx:Script>

    <mx:TextInput id="myInput"
        width="150"
```

```
        text=""/>
    <mx:Button id="myButton"
        label="Copy Text"
        click="myText.text = myInput.text;
           Alert.show('Text Copied!', 'Alert Box', mx.controls.Alert.OK);"/>
    <mx:TextInput id="myText"/>
</mx:Application>
```

この例では、Button コントロールが選択されると TextInput コントロールから TextArea コントロールにテキストがコピーされ、Alert コントロールが表示されます。

次の例に示すように、Button コントロールにイベントリスナーを定義することもできます。

```
<?xml version="1.0"?>
<!-- controls\alert\AlertSimpleEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;

            private function alertListener():void {
                myText.text = myInput.text;
                Alert.show("Text Copied!", "Alert Box", Alert.OK);
            }
        ]]>
    </mx:Script>

    <mx:TextInput id="myInput"
        width="150"
        text=""/>
    <mx:Button id="myButton"
        label="Copy Text"
        click="alertListener();"/>
    <mx:TextInput id="myText"/>
</mx:Application>
```

メモ

show() メソッドでダイアログボックスが作成された後、ダイアログボックスが開いたままになっていてもアプリケーションの処理は続行されます。

Alert コントロールのサイズ設定

Alert コントロールのサイズは、表示するテキスト、ボタン、およびアイコンが収まるよう自動的に調整されます。次の例のように、show() メソッドで返される Alert オブジェクトを使用して、Alert コントロールのサイズを明示的に設定することもできます。

```
<?xml version="1.0"?>
<!-- controls\alert\AlertSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
```

```
        <![CDATA[
            import mx.controls.Alert;

            // Define variable to hold the Alert object.
            public var myAlert:Alert;

            private function openAlert():void {
                myAlert = Alert.show("Copy Text?", "Alert",
                    Alert.OK    | Alert.CANCEL);
                // Set the height and width of the Alert control.
                myAlert.height=150;
                myAlert.width=150;
            }
        ]]>
    </mx:Script>

        <mx:TextInput id="myInput"
            width="150"
            text=""/>
        <mx:Button id="myButton"
            label="Copy Text"
            click="openAlert();"/>
        <mx:TextInput id="myText"/>
</mx:Application>
```

この例では、Alert オブジェクトの height プロパティと width プロパティを指定して、コントロールのサイズを明示的に設定しています。

Alert コントロールでイベントリスナーを使用する方法

次の例では、Alert コントロールのポップアップダイアログボックスにイベントリスナーを追加します。イベントリスナーを使用すると、Alert コントロールのボタンが選択されたときに実行する処理を指定できます。イベントリスナーに渡されるイベントオブジェクトは、CloseEvent タイプです。

次の例では、ユーザーが Alert コントロールの [OK] ボタンを選択したときにテキストのみコピーします。

```
<?xml version="1.0"?>
<!-- controls\alert\AlertEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.events.CloseEvent;

            private function alertListener(eventObj:CloseEvent):void {
                // Check to see if the OK button was pressed.
                if (eventObj.detail==Alert.OK) {
                    myText.text = myInput.text;
```

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

<mx:TextInput id="myInput"
    width="150"
    text="" />
<mx:Button id="myButton"
    label="Copy Text"
    click='Alert.show("Copy Text?", "Alert",
        Alert.OK | Alert.CANCEL, this,
        alertListener, null, Alert.OK);'/>
<mx:TextInput id="myText"/>
</mx:Application>
```

この例では、Alert コントロールのイベントリスナーを定義します。イベントリスナーの本体内で、イベントオブジェクトの `detail` プロパティを確認して押されたボタンを判別します。イベントオブジェクトは CloseEvent クラスのインスタンスです。ユーザーが [OK] ボタンを押した場合は、テキストをコピーします。ユーザーがそれ以外のボタンまたは Esc キーを押した場合は、テキストをコピーしません。

Alert コントロールアイコンの指定

Alert コントロールには、メッセージテキストの左側に表示されるアイコンを追加できます。次のコードは、前のセクションの例に `Embed` メタデータタグを追加してアイコンを読み込むように修正したものです。リソースを読み込む方法の詳細については、55 ページ、第 4 章の「ActionScript の使用」を参照してください。

```
<?xml version="1.0"?>
<!-- controls\alert\AlertIcon.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.events.CloseEvent;

            [Embed(source="assets/alertIcon.jpg")]
            [Bindable]
            public var iconSymbol:Class;

            private function alertListener(eventObj:CloseEvent):void {
                // Check to see if the OK button was pressed.
                if (eventObj.detail==Alert.OK) {
                    myText.text = myInput.text;
                }
            }
```

```
        ]]>
    </mx:Script>

        <mx:TextInput id="myInput"
            width="150"
            text=""/>
        <mx:Button id="myButton"
            label="Copy Text"
            click='Alert.show("Copy Text?", "Alert",
                Alert.OK | Alert.CANCEL, this,
                alertListener, iconSymbol,  Alert.OK );'/>
        <mx:TextInput id="myText"/>
</mx:Application>
```

ProgressBar コントロール

ProgressBar コントロールは、処理の進行状況を時間経過に沿って視覚的に表現するものです。ProgressBar コントロールには、確定型と不確定型の 2 つの種類があります。確定型の ProgressBar コントロールでは、時間の経過に伴う処理の進行状況が比例直線的に表現されます。ユーザーを長時間待たせる必要があるとき、その処理の規模が事前にわかっている場合に使用します。

不確定型の ProgressBar コントロールは、処理の規模が不確定である場合に、処理の経過時間に基づいて進捗状況を表現します。処理の規模が明らかな場合は、確定型の ProgressBar コントロールを使用することをお勧めします。

次の例は、ProgressBar コントロールの両方の種類を示しています。

確定型の ProgressBar コントロール

不確定型の ProgressBar コントロール

ProgressBar コントロールは、処理の完了まで長時間にわたってユーザーを待たせる必要がある場合に使用します。ProgressBar コントロールは、あらゆるタイプのコンテンツロード処理に適用できます。どの程度のコンテンツがロード済みであるかを表示するラベルを使用することもできます。

ProgressBar コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	幅 150 ピクセル、高さ 4 ピクセル
最小サイズ	0
最大サイズ	未定義

ProgressBar コントロールのモード

ProgressBar コントロールの動作モードを指定するには、mode プロパティを使用します。ProgressBar コントロールは、次の動作モードをサポートしています。

event progress イベントと complete イベントを送出するロードプロセスを、source プロパティを使って指定します。たとえば、SWFLoader コントロールと Image コントロールはファイルのロードの過程で、これらのイベントを送出します。一般に、このモードでは、確定型の ProgressBar を使用します。これがデフォルトのモードです。

また、イメージを再ロードする場合や、SWFLoader コントロールと Image コントロールで複数のイメージをロードする場合など、複数のロードプロセスについて進捗状況を評価したいときにもこのモードを使用します。

polled getBytesLoaded() メソッドおよび getsBytesTotal() メソッドを実装するロードプロセスを、source プロパティを使って指定します。これらのメソッドは、SWFLoader コントロールと Image コントロールなどに実装されています。一般に、このモードでは、確定型の ProgressBar を使用します。

manual maximum、minimum、indeterminate の各プロパティを設定し、setProgress() メソッドを呼び出します。一般に、このモードでは、不確定型の ProgressBar を使用します。

ProgressBar コントロールの作成

次の例のように、MXML で ProgressBar コントロールを定義するには、<mx:ProgressBar> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

次の例では、Image コントロールで、デフォルトの event モードを使用して、イメージのロード状況を追跡します。

```
<?xml version="1.0"?>
<!-- controls\pbar\PBarSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function initImage():void {
              image1.load('http://localhost:8100/flex/assets/DSC00034.JPG');
            }
        ]]>
    </mx:Script>

    <mx:VBox id="vbox0"
        width="600" height="600">
        <mx:Canvas>
            <mx:ProgressBar width="200" source="image1"/>
```

```
        </mx:Canvas>
        <mx:Button id="myButton"
            label="Show"
            click="initImage();"/>
        <mx:Image id="image1"
            height="600" width="600"
            autoLoad="false"
            visible="true"/>
    </mx:VBox>
</mx:Application>
```

このモードでは、Image コントロールはロード中に progress イベントを生成し、ロードが完了したときに complete イベントを生成します。

<mx:Image> タグには getBytesLoaded() メソッドと getBytesTotal() メソッドが実装されているため、次のように、polled モードを使用することもできます。

```
<mx:ProgressBar width="200" source="image1" mode="polled"/>
```

manual モード (mode="manual") で使用するには、不確定モードの ProgressBar コントロールで maximum プロパティと minimum プロパティ、および setProgress() メソッドを使用します。setProgress() メソッドのシグネチャを次に示します。

```
setProgress(Number completed, Number total)
```

completed　処理の進行量を、maximum から minimum の範囲内の値で指定します。たとえば、ロードされたバイト数を追跡する場合は、既にロード済みのバイト数を指定します。

total　処理の合計を指定します。たとえば、ロードされたバイト数を追跡する場合は、ロードの対象となる合計バイト数を指定します。通常は、maximum と同じ値になります。

進捗状況を評価するには、setProgress() メソッドを明示的に呼び出して ProgressBar コントロールを更新する必要があります。

ProgressBar コントロールのラベルの定義

デフォルトでは、ProgressBar に LOADING xx% (xx はロード済みイメージの割合) のように表示されます。他のテキストストリングを指定するには、label プロパティを使用します。

label プロパティでは、ラベルテキストのストリングに次の特殊文字を使用できます。

%1　現在ロード済みのバイト数に相当します。

%2　合計バイト数に相当します。

%3　ロードが完了した比率に相当します。

%%　% 記号に相当します。

たとえば、ラベルに次のように表示させたいとします。

```
Loading Image 1500 out of 78000 bytes, 2%
```

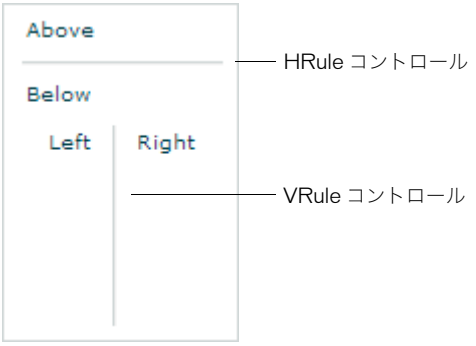
次のコードを使用します。

```
<?xml version="1.0"?>
<!-- controls\pbar\PBarLabel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function initImage():void {
              image1.load('http://localhost:8100/flex/assets/DSC00034.JPG');
            }
        ]]>
    </mx:Script>

    <mx:VBox id="vbox0"
        width="600" height="600">
        <mx:Canvas>
            <mx:ProgressBar
                width="300"
                source="image1"
                mode="polled"
                label="Loading Image %1 out of %2 bytes, %3%%"
                labelWidth="400"/>
        </mx:Canvas>
        <mx:Button id="myButton"
            label="Show"
            click="initImage();"/>
        <mx:Image id="image1"
            height="600" width="600"
            autoLoad="false"
            visible="true"/>
    </mx:VBox>
</mx:Application>
```

HRule コントロールと VRule コントロール

HRule (Horizontal Rule) コントロールは水平方向の罫線を、VRule (Vertical Rule) コントロールは垂直方向の罫線を作成するものです。一般に、この 2 つのコントロールは、コンテナ内に境界線を引くときに使用します。

次の図に HRule コントロールと VRule コントロールの例を示します。



HRule コントロールと VRule コントロールには、次のデフォルトプロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	**Horizontal Rule**　デフォルトの幅と高さは、それぞれ 100 ピクセルおよび 2 ピクセルになります。デフォルトでは、HRule コントロールのサイズは不変です。サイズ変更を有効にするには、`width` と `height` にパーセント値を指定します。 **Vertical Rule**　デフォルトでは、高さが 100 ピクセル、幅は 2 ピクセルになります。デフォルトでは、VRule コントロールのサイズは不変です。サイズ変更を有効にするには、`width` と `height` にパーセント値を指定します。
`strokeWidth`	2 ピクセル
`strokeColor`	`0xC4CCCC`
`shadowColor`	`0xEEEEEE`

HRule コントロールと VRule コントロールの作成

HRule コントロールと VRule コントロールは、次のように、`<mx:HRule>` タグおよび `<mx:VRule>` タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、`id` 値を指定します。

```
<?xml version="1.0"?>
<!-- controls\rule\RuleSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

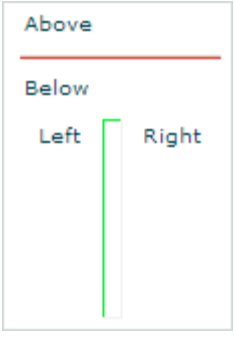
```
    <mx:VBox>
        <mx:Label text="Above"/>
        <mx:HRule/>
        <mx:Label text="Below"/>
    </mx:VBox>
    <mx:HBox>
        <mx:Label text="Left"/>
        <mx:VRule/>
        <mx:Label text="Right"/>
    </mx:HBox>
</mx:Application>
```

この例では、前出の図のような出力結果が生成されます。

次の例のように、HRule コントロールと VRule コントロールのプロパティを使用して、線幅、線のカラー、陰影色などを指定することもできます。

```
<?xml version="1.0"?>
<!-- controls\rule\RuleProps.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox>
        <mx:Label text="Above"/>
        <mx:HRule shadowColor="0xEEEEEE"/>
        <mx:Label text="Below"/>
    </mx:VBox>
    <mx:HBox>
        <mx:Label text="Left"/>
        <mx:VRule strokeWidth="10" strokeColor="0xC4CCCC"/>
        <mx:Label text="Right"/>
    </mx:HBox>
</mx:Application>
```

このコードでは、次のイメージが作成されます。



HRule コントロールと VRule コントロールのサイズ

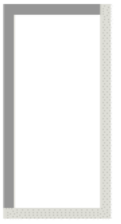
HRule コントロールと VRule コントロールでは、罫線の描画方法が strokeWidth プロパティによって次のように決定されます。

- strokeWidth プロパティを 1 に設定した場合、1 ピクセル幅の直線が描画されます。
- strokeWidth プロパティを 2 に設定した場合、1 ピクセル幅の 2 本の罫線が描画されます。この 2 本の罫線は HRule コントロールの場合は水平に、VRule コントロールの場合は垂直に、それぞれ平行に描画されます。これがデフォルト値です。
- strokeWidth プロパティに 2 より大きい値を設定した場合、1 ピクセル幅の枠を持つ、くぼみのある四角形として描画されます。

次の例に、この 3 つのパターンで描画した結果を示します。

VRule コントロール
strokeWidth = 1

VRule コントロール
strokeWidth = 2 (デフォルト)



VRule コントロール
strokeWidth = 10

HRule コントロールの height プロパティに、strokeWidth プロパティを超える値を設定した場合、罫線は指定された高さの矩形内に描画され、矩形の中央に配置されます。罫線の高さは、strokeWidth プロパティに指定した高さになります。

VRule コントロールの width プロパティに、strokeWidth プロパティを超える値を設定した場合、罫線は指定された幅の矩形内に水平に描画され、矩形の中央に配置されます。罫線の幅は、strokeWidth プロパティに指定した幅になります。

HRule コントロールの height プロパティまたは VRule コントロールの width プロパティに、strokeWidth プロパティより小さい値を設定した場合は、strokeWidth プロパティが height プロパティまたは width プロパティと等しいものとして罫線が描画されます。

メモ

height プロパティと width プロパティをパーセント値で指定した場合、実際のピクセル値は、height プロパティと width プロパティを strokeWidth プロパティと比較する前に計算されます。

HRule コントロールと VRule コントロールの色は、strokeColor プロパティと shadowColor プロパティによって決定されます。strokeColor プロパティでは、次のようにして線の色が指定されます。

- strokeWidth プロパティに 1 が設定されている場合、罫線全体に色が指定されます。
- strokeWidth プロパティに 2 が設定されている場合、HRule の場合は上側の直線に、VRule コントロールの場合は左側の直線に色が指定されます。
- strokeWidth プロパティに 2 より大きい値を設定した場合、矩形の上辺および左辺に対して色が指定されます。

shadowColor プロパティでは、次のようにして線の陰影色が指定されます。

- strokeWidth プロパティに 1 を設定した場合、処理は一切適用されません。
- strokeWidth プロパティに 2 が設定されている場合、HRule の場合は下側の直線に、VRule コントロールの場合は右側の直線に色が指定されます。
- strokeWidth プロパティに 2 より大きい値を設定した場合、矩形の底辺および右辺に対して色が指定されます。

スタイルプロパティの設定

strokeWidth、strokeColor、shadowColor は、スタイルのプロパティです。したがって、タグの定義の一部として MXML を設定することも、MXML の <mx:Style> タグを使って設定することもできます。あるいは、ActionScript から setStyle() メソッドを使って設定することも可能です。

次の例では、<mx:Style> タグを使って、すべての HRule コントロールの strokeColor プロパティと shadowColor プロパティに、デフォルト値として、それぞれ #00FF00 (ライムグリーン) と #0000FF (青) を設定しています。この例では、strokeWidth を 5 に設定した、thickRule というクラスセレクタを定義しています。次の例のように、HRule コントロールまたは VRule コントロールのあらゆるインスタンスに使用できます。

```
<?xml version="1.0"?>
<!-- controls\rule\RuleStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        .thickRule {strokeWidth:5}
        HRule {strokeColor:#00FF00; shadowColor:#0000FF}
    </mx:Style>

    <mx:HRule styleName="thickRule"/>
</mx:Application>
```

この例では、次のイメージが作成されます。

ScrollBar コントロール

VScrollBar (垂直の ScrollBar) コントロールおよび HScrollBar (水平の ScrollBar) コントロールを使用すると、データが多すぎて表示領域に収まらない場合に、表示される部分のデータを制御できます。

VScrollBar コントロールと HScrollBar コントロールをスタンドアローンのコントロールとして使用することもできますが、通常は、スクロール機能を実装したカスタムコンポーネントとして、他のコンポーネントと一体化されています。詳細については、Flex 2 コンポーネントの作成と拡張を参照してください。

ScrollBar コントロールは、2 つの矢印ボタン、1 つのトラック、および 1 つのサムの合計 4 つのパーツで構成されます。サムの位置とボタンの表示は、ScrollBar コントロールの現在の状態に依存します。ScrollBar コントロールでは、表示状態を計算するために次の 4 つのパラメータが使用されます。

- 範囲の最小値
- 範囲の最大値
- 現在の位置 (最大値と最小値の範囲内)
- ビューポートのサイズ (上記の範囲内に一度に表示できるアイテム数。この範囲と等しいかそれ以下であること)

ScrollBar コントロールの作成

次の例に示すように、ScrollBar コントロールは、<mx:VScrollbar> タグ (垂直スクロールバーの場合) または <mx:HScrollBar> タグ (水平スクロールバーの場合) を使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- controls\bar\SBarSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
       <![CDATA[

           import mx.events.ScrollEvent;

           // Event handler function to display the scroll location.
           private function myScroll(event:ScrollEvent):void {
               showPosition.text = "VScrollBar properties summary:" + '\n' +
                   "-------------------------------------" + '\n' +
                   "Current scroll position: " +
                   event.currentTarget.scrollPosition  + '\n' +
                   "The maximum scroll position: " +
                   event.currentTarget.maxScrollPosition + '\n' +
                   "The minimum scroll position: " +
                   event.currentTarget.minScrollPosition;
```

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

    <mx:Label
        width="100%"
        color="blue"
        text="Click on the scroll bar to view its properties."/>

    <mx:VScrollBar id="bar"
        height="100%"
        minScrollPosition="0"
        maxScrollPosition="{this.width - 20}"
        lineScrollSize="50"
        pageScrollSize="100"
        repeatDelay="1000"
        repeatInterval="500"
        scroll="myScroll(event);"/>

    <mx:TextArea id="showPosition"
        height="100%" width="100%"
        color="blue"/>
</mx:Application>
```

ScrollBar コントロールのサイズ設定

ScrollBar コントロールは、上下の矢印ボタンよりも小さいサイズで表示しようとした場合、正しく表示されません。この状態に対するエラーチェック機構は存在しません。このような場合は、ScrollBar コントロールを非表示にすることをお勧めします。サムを表示するだけの領域がない場合、サムが非表示になります。

ユーザーの操作

マウスボタンで ScrollBar コントロールの各部分をクリックすると、リスナーに対してイベントが送出されます。ScrollBar コントロールのイベントを待機するオブジェクトによって、表示するデータの領域が更新されます。スクロール操作が実行されると、ScrollBar コントロールの表示が新しい状態を反映するように更新されます。

第 10 章

テキストコントロールの使用

10

テキストコントロールは、テキストの表示、またはユーザーによるテキストの入力、あるいはその両方を行います。このトピックでは、Adobe Flex アプリケーションでのテキストコントロールの使用方法について説明します。

内容

テキストコントロールについて

テキストを表示し、ユーザーがアプリケーションにテキストを入力できるようにするには、テキストベースの Flex コントロールを使用します。次の表は、コントロールの一覧です。コントロールに単一行のテキストではなく複数行のテキストを入力できるかどうか、およびコントロールがユーザーによる入力を受け入れるかどうかを示しています。

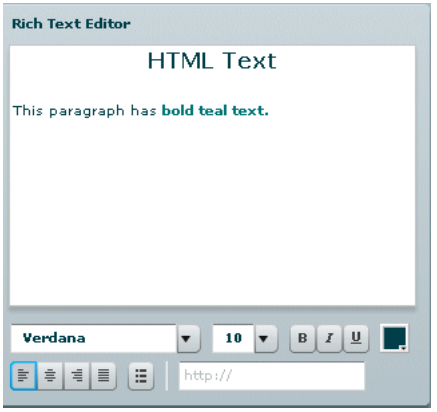
コントロール	複数行	ユーザー入力
Label	いいえ	いいえ
TextInput	いいえ	はい
Text	はい	いいえ
TextArea	はい	はい
RichTextEditor	はい	はい

RichTextEditor コントロール以外のすべてのコントロールは、単純なテキスト領域を持つ単一のコンポーネントです。たとえば、次の図は、簡単なフォームにおける TextInput コントロールを示しています。

次のコードで、上のイメージが作成されます。

```
<?xml version="1.0"?>
<!-- textcontrols/FormItemLabel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
  <mx:Form id="myForm" width="500" backgroundColor="#909090">
     <!-- Use a FormItem to label the field. -->
     <mx:FormItem label="First Name">
        <mx:TextInput id="ti1" width="150"/>
     </mx:FormItem>
  </mx:Form>
</mx:Application>
```

RichTextEditor コントロールは複合コントロールであり、TextArea コントロールを含む Panel コントロールと、テキストフォーマットと HTTP リンクを指定するための複数のコントロールを含む ControlBar で構成されます。次の図は、RichTextEditor コントロールを示しています。



次のコードで、上のイメージが作成されます。

```
<?xml version="1.0"?>
<!-- textcontrols/RTECDATA.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
  <mx:RichTextEditor id="rte1" title="Rich Text Editor">
     <mx:htmlText>
        <![CDATA[
           <p align='center'><b><font size='16'>HTML Text</font></b>
           </p>This paragraph has <font color='#006666'><b>bold teal text.
           </b></font>
        ]]>
     </mx:htmlText>
  </mx:RichTextEditor>
</mx:Application>
```

Flex のテキストベースコントロールでは、次のプロパティを使用してテキストを設定および取得できます。

text　フォーマット情報を持たないプレーンテキスト。text プロパティ使用の詳細については、346 ページの「text プロパティの使用」を参照してください。

htmlText　HTML タグのサブセットを使用したフォーマットを表すリッチテキスト。黒丸付きのテキストと URL リンクを含めることもできます。htmlText プロパティ使用の詳細については、350 ページの「htmlText プロパティの使用」を参照してください。

これらのプロパティは両方とも同一の基になるテキストを設定しますが、フォーマットは異なるものを使用できます。たとえば次のような手順で、テキストの設定、変更、および取得を行うことができます。

- htmlText プロパティを使用してフォーマット済みテキストを設定し、text プロパティを使用してプレーンテキストとして戻すことができます。
- ユーザーが編集可能なテキストコントロール (TextInput、TextArea、RichTextEditor) でフォーマット済みテキストを設定するには、text プロパティでテキストストリングを設定し、TextRange クラスを使用してそのテキストの一部をフォーマットします。htmlText プロパティを使用してテキストを再取得した場合、プロパティのストリングにはフォーマット用の HTML タグが含まれます。TextRange クラスの使用の詳細については、361 ページの「テキストの選択と修正」を参照してください。

text プロパティの使用

text プロパティを使用すると、テキストコントロールに表示するテキストストリングを指定し、コントロール内のテキストをプレーンテキストストリングとして取得することができます。このプロパティを設定すると、テキストストリング内のすべての HTML タグは、リテラルテキストとしてコントロール内に表示されます。

text プロパティを設定するときにテキストフォーマットを指定することはできませんが、コントロール内でテキストをフォーマットすることは可能です。テキストコントロールスタイルを使用してコントロール内のすべてのテキストをフォーマットし、TextRange クラスを使用してテキスト領域をフォーマットすることができます。(TextRange クラスの使用の詳細については、361 ページの「テキストの選択と修正」を参照してください。

次のコード行では、text プロパティを使用してラベルテキストを指定します。

```
<mx:Label text="This is a simple text label"/>
```

引用符、大なり記号、小なり記号、アポストロフィなどの特殊文字を指定する方法は、それらの特殊記号を MXML タグで使用するか ActionScript タグで使用するかによって異なります。また、テキストを直接指定するか、CDATA セクションにテキストをラップするかによっても異なります。

メモ

ストリングを使用して直接 MXML で text プロパティの値を指定すると、空白文字は畳んで表示されます。text プロパティの値を ActionScript で指定した場合、空白文字は畳んで表示されません。

text プロパティでの特殊文字の指定

次の規則は、テキストコントロール MXML タグの text プロパティに特殊文字を含める方法を示しています。つまり、text="the text" などのプロパティ割り当て、または <mx:text> サブタグの本体のいずれかに指定します。

標準テキストの使用　次のルールは、CDATA セクションを使用しない場合の特殊文字の使用方法を決定します。

- 左の山カッコ (<)、右の山カッコ (>)、アンパサンド (&) などの特殊文字を使用するには、それぞれ対応する <、>、& の XML 文字エンティティを挿入します。二重引用符 (") および単一引用符 (') には、" および ' を使用できます。また、円記号 (¥) には、&#165 などの数値文字参照を使用できます。その他の名前が付けられた文字エンティティは使用しないでください。これらは、Flex ではリテラルテキストとして扱われます。
- ストリング内のプロパティテキストストリングを囲んでいる文字は使用できません。ストリングを二重引用符 (") で囲む場合、ストリング内のすべての二重引用符に対して、エスケープシーケンス \" を使用します。ストリングを単一引用符で囲む場合に、ストリング内のすべての単一引用符に対して、エスケープシーケンス \' を使用します。二重引用符で囲まれたストリング内の単一引用符、および単一引用符で囲まれたストリング内の二重引用符は使用することができます。
- Flex のテキストコントロールでは、text プロパティの \t や \n などのエスケープ文字は無視されます。これらが無視されるかまたはスペース、タブ、改行に変換されるかは、プロパティ割り当てと <mx:text> サブタグのどちらを指定したかによって決まります。改行を含めるには、テキストを CDATA セクション内に入力してください。テキストコントロールの text="string" 属性指定では、たとえば改行文字を
、タブ文字を 	 というように、数値文字エンティティとして指定することができますが、<mx:text> サブタグではそのように指定できません。

次のコード例では、標準テキストに text プロパティを使用します。

```
<?xml version="1.0"?>
<!-- textcontrols/StandardText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="400">

  <mx:Text width="400" text="This string contains a less than, &lt;,
     greater than, &gt;, ampersand, &amp;, apostrophe, ', and
     quotation mark &quot;."/>

  <mx:Text width="400" text='This string contains a less than, &lt;,
     greater than, &gt;, ampersand, &amp;, apostrophe, &apos;, and
     quotation mark, "'/>

  <mx:Text width="400">
     <mx:text>
        This string contains a less than, &lt;, greater than,
        &gt;, ampersand, &amp;, apostrophe, ', and quotation mark, ".
     </mx:text>
  </mx:Text>

</mx:Application>
```

結果として生成されるアプリケーションには、ほぼ同一の 3 つのテキストコントロールが含まれます。各コントロールには、次のテキストが含まれます。ただし最初の 2 つのコントロールでは、テキスト内のすべてのタブがスペースに変換されます。

```
This string contains a less than, <, greater than, >, ampersand, &,
  apostrophe, ', and  quotation mark, ".
```

CDATA セクション　CDATA タグ内でテキストストリングを折り返す場合、次の規則が適用されます。

- CDATA セクションは、テキストコントロールの開始タブのプロパティ割り当てステートメントでは使用できません。プロパティは、<mx:text> 子タグで定義する必要があります。
- CDATA セクション内のテキストは、空白文字を含め、すべて入力されたとおりに表示されます。特殊文字には " や ‹ などのリテラル文字を使用してください。また標準の改行文字とタブ文字を使用してください。> などの文字エンティティ、および \n などのバックスラッシュスタイルエスケープ文字は、リテラルテキストとして表示されます。

次のコード例は、CDATA セクションの規則に従ったものです。<mx:text> タグ内の 2 行目および 3 行目のテキストはインデントされません。先行するタブやスペース文字があれば、表示テキスト内に表示されるからです。

```
<?xml version="1.0"?>
<!-- textcontrols/TextCDATA.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="500">
  <mx:Text width="100%">
     <mx:text>
        <![CDATA[This string contains a less than, <, greater than, >,
ampersand, &, apostrophe, ', return,
tab. and quotation mark, ".]]>
     </mx:text>
  </mx:Text>
</mx:Application>
```

表示テキストは、次のように 3 行に表示されます。

```
This string contains a less than, <, greater than, >,
ampersand, &, apostrophe, ', return,
tab.  and quotation mark, ".
```

ActionScript での特殊文字の指定

次の規則は、初期化関数などで ActionScript を使用してコントロールの text プロパティ値を指定する場合や、プロパティの設定で使用する変数にストリング値を割り当てる場合に、テキストコントロールに特殊文字を挿入する方法を指定しています。

- テキストストリングを囲んでいる文字を、ストリング内で使用することはできません。ストリングを二重引用符 (") で囲む場合は、ストリング内のすべての二重引用符に対してエスケープシーケンス (\") を使用します。ストリングを単一引用符 (') で囲む場合は、ストリング内のすべての単一引用符に対してエスケープシーケンス (\') を使用します。
- 特殊文字には、バックスラッシュエスケープ文字を使用します。たとえば、タブ文字に対して \t、改行文字の組み合わせに対して \n または \r などを使用します。二重引用符にはエスケープ文字 \"、単一引用符には \' を使用できます。

- 標準テキスト (CDATA セクションではない) で、左の山カッコ (<)、右の山カッコ (>)、アンパサンド (&) の特殊文字を追加するには、それぞれ <、>、& に相当する XML 文字エンティティを挿入します。二重引用符 (") および単一引用符 (') には、" および ' を使用できます。また、円記号 (¥) には、¥ などの数値文字参照を使用できます。その他の名前が付けられた文字エンティティは使用しないでください。これらは、Flex ではリテラルテキストとして扱われます。
- CDATA セクションのみには、< や &#165 などの文字エンティティまたは参照は使用しないでください。それらは、Flex ではリテラルテキストとして扱われます。代わりに、< などの実際の文字を使用してください。

次の例では、初期化関数を使用して、text プロパティに特殊文字を含むストリングを設定します。

```
<?xml version="1.0"?>
<!-- textcontrols/InitText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initText()">
  <mx:Script>
     public function initText():void {
        //The following is on one line.
        myText.text="This string contains a return, \n, tab, \t, and quotation
mark, \". This string also contains less than, &lt;, greater than, &gt;,
ampersand, &amp;, and apostrophe, ', characters.";
     }
  </mx:Script>
  <mx:Text width="450" id="myText" initialize="initText();"/>
</mx:Application>
```

次の例では、<mx:Script> タグと CDATA セクション内の変数を共に使用して、text プロパティを設定します。

```
<?xml version="1.0"?>
<!-- textcontrols/VarText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
  <![CDATA[
     [Bindable]
        //The following is on one line.
     public var myText:String ="This string contains a return, \n, tab, \t, and
quotation mark, \". This string also contains less than, <, greater than, <,
ampersand, <;, and apostrophe, ', characters.";
  ]]>
  </mx:Script>
  <mx:Text width="450" text="{myText}"/>
</mx:Application>
```

それぞれの例の表示テキストは、3 行に表示されます。最初の行は \n 文字により指定された改行で終わります。残りのテキストは、1 行に収めるには長すぎるため、3 行目に折り返されます。次の出力では、わかりにくいかもしれませんが、タブ文字が正しい位置に挿入されています。

```
This string contains a return,
, tab, , and quotation mark, ". This string also contains less than, <,
greater than, >, ampersand, &, and apostrophe, ', characters.
```

htmlText プロパティの使用

HTML 形式のテキストストリングを設定または取得するには、htmlText プロパティを使用します。標準の HTML には含まれていない 1 つのタグ textFormat を使用することもできます。サポートされているタグと属性の詳細については、354 ページの「HTML テキストでのタグの使用」を参照してください。

Flex スタイルを使用してテキストフォーマットを指定することもできます。スタイルを使用してフォントの特性やテキストの太さなどの基本スタイルを設定し、<font> タグなどのタグを使用して、テキストセクション内で基本スタイルをオーバーライドできます。次の例では、<mx:Text> タグスタイルは青色、イタリック、14 ポイントのテキストを指定します。<mx:htmlText> タグには、カラーやポイントサイズをオーバーライドする HTML タグが含まれています。

```
<?xml version="1.0"?>
<!-- textcontrols/HTMLTags.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="450"
backgroundGradientColors="[#FFFFFF, #FFFFFF]">
  <mx:Text width="100%" color="blue" fontStyle="italic" fontSize="14">
     <mx:htmlText>
        <![CDATA[This is 14 point blue italic text.<br><b><font color="#000000"
size="10">This text is 10 point black, italic, and bold.</font></b>]]>
     </mx:htmlText>
  </mx:Text>
</mx:Application>
```

このコードの出力は、次のようになります。

HTML タグとテキストの指定

テキスト内で HTML タグが検出されたときに Flex コンパイラがエラーを生成しないようにするには、次のいずれかの方法を使用します。

- CDATA タグ内のテキストを折り返します。
- 左の山カッコ (<)、右の山カッコ (>)、およびアンパサンド (&) という HTML の区切り文字の代わりに <、>、および & の文字エンティティを使用して、HTML マークアップを指定します。

文字エンティティの方法には多くの制限があるため、単純な HTML マークアップ以外のすべての HTML マークアップには、CDATA セクションを使用することをお勧めしています。

- 拡張された HTML マークアップは書き込みが面倒で、読み取りが困難な場合があります。
- テキストに小なり記号とアンパサンド文字を含めるときに、複雑なエスケープシーケンスを使用する必要があります。

たとえば以下のストリングを表示する場合は、次のようにします。

小なり記号 (<) および**ボールドテキスト**。

CDATA セクションを使用しない場合は、次のテキストを使用する必要があります。

```
A less than character &amp;c#060; and &lt;b&gtbold text&lt;/b&gt.
```

CDATA セクションで、次のテキストを使用します。

```
A less than character &lt; and <b>bold text</b>.
```

HTML テキストの指定

テキストコントロールに HTML テキストを指定する場合、次の規則が適用されます。

- CDATA セクションを、<mx:Text> タグのインライン htmlText プロパティで直接使用することはできません。テキストは、<mx:htmlText> サブタグまたは ActionScript コード内に入力する必要があります。
- Flex では、CDATA セクション外で MXML のプロパティ割り当てまたは ActionScript によって指定したテキストにおける、連続する空白文字 (改行、スペース、タブ) は畳んで表示されます。
- テキストを CDATA セクションで指定した場合は、テキストコントロールの condenseWhite プロパティを使用して、空白文字を畳んで表示するかどうかを制御できます。デフォルトでは、condenseWhite プロパティは false になっており、空白文字は畳んで表示されません。
- 改行と段落には、HTML の <p> および
 タグを使用します。ActionScript の CDATA セクションでは、\n エスケープ文字を使用することもできます。
- 割り当てステートメント内で、HTML テキストストリングを単一引用符または二重引用符で囲む場合 (つまり、ストリングが <mx:htmlText> タグ内に置かれていない場合)、ストリングで使用されている引用符文字をエスケープする必要があります。

- 割り当て区切り文字に対して二重引用符を使用する場合、HTML 内では二重引用符 (") に対して " を使用します。ActionScript では、エスケープシーケンス (\") を使用することもできます。

  メモ 外部ファイルからテキストをロードしている場合、二重引用符をエスケープする必要はありません。ActionScript でテキストストリングを割り当てている場合にのみ、二重引用符のエスケープが必要です。

- 割り当て区切り文字に対して単一引用符を使用する場合、HTML 内では単一引用符 (') に対して ' を使用します。ActionScript では、エスケープシーケンス (\') を使用することもできます。

- HTML フォーマット済みテキストを入力するときは、二重引用符または単一引用符内に HTML タグの属性を含める必要があります。引用符で囲んでいない属性値は、テキストの不適切なレンダリングなど、予期しない結果を生み出す可能性があります。引用符内では、引用符のエスケープ規則に従う必要があります。詳細は、353 ページの「HTML テキストでの特殊文字のエスケープ」に記載されています。

次の例では、テキストを指定するために MXML および ActionScript を使用している、いくつかの簡単な HTML フォーマット済みテキストを示しています。

```
<?xml version="1.0"?>
<!-- textcontrols/HTMLFormattedText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="500">
  <mx:Script><![CDATA[
     //The following is on one line.
     [Bindable]
     public var myHtmlText:String="This string contains <b>less than </b>, &lt;,
<b>greater than</b>, &gt;, <b>ampersand</b>, &amp;, and <b>double quotation
mark</b>, &quot;, characters.";
  ]]></mx:Script>

  <mx:Text id="htmltext2" width="450" htmlText="{myHtmlText}" />
  <mx:Text width="450">
     <mx:htmlText>
       <!-- The following is on one line. Line breaks would appear in the output.
-->
        <![CDATA[
           This string contains <b>less than</b>, &lt;, <b>greater than </b>,
&gt;, <b>ampersand</b>, &amp;, and <b>double quotation mark</b>,&quot;,
characters.
        ]]>
     </mx:htmlText>
  </mx:Text>
</mx:Application>
```

各テキストコントロールには、次のテキストが表示されます。

This string contains **less than**, <, **greater than**, >, **ampersand**, &, and **double quotation mark**, " characters.

HTML テキストでの特殊文字のエスケープ

HTML テキストでの特殊文字のエスケープに適用される規則は、CDATA セクションと標準テキストでは異なります。

CDATA セクション　htmlText ストリングを指定する場合、次の規則が適用されます。

- ActionScript (<mx:htmlText> タグを除く) では、タブに対する \t、改行文字に対する \n など、特殊文字に対して標準のバックスラッシュエスケープシーケンスを使用できます。また多くの特殊文字をエスケープするため、単一引用符に対する \'、二重引用符に対する \" など、バックスラッシュ文字を使用することもできます。\x の組み合わせは使用できません。改行文字の前にバックスラッシュを入力しても、表示テキストには何の影響もありません。改行文字の前にバックスラッシュを使用すると、割り当てステートメントを複数のテキスト行に分割することができます。
- ActionScript と <mx:htmlText> タグの両方で、HTML タグおよび数値文字エンティティを使用できます。たとえば、\n の代わりに
 タグを使用することができます。
- 表示テキストに左の山カッコ (<)、右の山カッコ (>)、またはアンパサンド (&) 文字を挿入するには、それぞれ対応する文字エンティティ (<、>、または &) を使用します。単一引用符と二重引用符に対して、" および ' エンティティを使用することもできます。これらのエンティティは、Flash Player で認識される唯一の名前付き文字エンティティです。Flash Player では、円記号 (¥) に対する ¥ などの数値エンティティは認識されますが、それに対応する文字エンティティである ¥ は認識されません。

  次のコード例では、htmlText プロパティを使用してフォーマット済みテキストを表示します。

```
<?xml version="1.0"?>
<!-- textcontrols/HTMLTags2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="500">
  <mx:Text width="100%">
     <mx:htmlText><![CDATA[<p>This string contains a <b>less than</b>, &lt;.
        </p><p>This text is in a new paragraph.<br>This is a new line.</p>]]>
     </mx:htmlText>
  </mx:Text>
</mx:Application>
```

このコードは、次のテキストを表示します。

```
This string contains a less than, <.

This text is in a new paragraph.
This is a new line.
```

標準テキスト　次の規則が適用されます。

- タグを開いたり文字エンティティを開始したりするときに、HTML で左の山カッコ (<)、右の山カッコ (>)、またはアンパサンド (&) 文字を使用するには、文字エンティティを使用する必要があります。詳細は、350 ページの「htmlText プロパティの使用」に記載されています。

- 小なり記号 (< を使用)、アンパサンド (& を使用) を表示するには、名前付きエンティティである & と HTML 数値エンティティを組み合わせたものを使用する必要があります。大なり記号、二重引用符、単一引用符には、それぞれ標準文字エンティティである >、"、' を使用します。その他のすべての文字エンティティについては、円記号 (¥) に対する ¥ , のように、数値エンティティの組み合わせを使用します。
- ActionScript (<mx:htmlText> タグまたはインライン htmlText プロパティを除く) では、バックスラッシュ文字を使用して、タブ文字、改行文字、引用符などの特殊文字 (アンパサンドを除く) をエスケープできます。すべてのケースにおいて、正しくエスケープされた HTML タグや数値エンティティを使用できます。たとえば、\n の場所に
 タグや
 エンティティを使用できます。

HTML テキストでのタグの使用

htmlText プロパティを使用するときは、Flash Player でサポートされている HTML のサブセットを使用します。Flash Player では、次のタグがサポートされています。

- アンカータグ (<a>)
- ボールドタグ (<b>)
- ブレークタグ (
)
- フォントタグ (<font>)
- イメージタグ (<img>)
- イタリックタグ (<i>)
- リスト項目タグ (<li>)
- 段落タグ (<p>)
- テキストフォーマットタグ (<textformat>)
- 下線タグ (<u>)

アンカータグ (<a>)

アンカー (<a>) タグは、ハイパーリンクを作成し、次の属性をサポートします。

href　ブラウザにロードするページの URL を指定します。URL は、ページをロードしている SWF ファイルの場所に対して絶対的でも相対的でもかまいません。

target　ページをロードする先のターゲットウィンドウの名前を指定します。

たとえば、次の HTML コードは、Adobe の Web サイトへの "Go home" というリンクを作成するものです。

```
<a href='http://www.adobe.com' target='_blank'>Go Home</a>
```

また、スタイルシートを使用すると、アンカータグに a:link、a:hover、および a:active の各スタイルを定義できます。

<a> タグでは、リンクテキストは青になりません。テキストのフォーマットを変更するには、フォーマットタグを適用する必要があります。また、スタイルシートを使用すると、アンカータグに a:link、a:hover、および a:active の各スタイルを定義できます。

Label、Text、および TextArea コントロールは、ユーザーが htmlText プロパティでハイパーリンクを選択したときでも link イベントを送出できます。link イベントを生成するには、次に例に示すのように、ハイパーリンクのリンク先の前に event: を付けます。

```
<?xml version="1.0"?>
<!-- textcontrols/LabelControlLinkEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    borderStyle="solid"
    backgroundGradientColors="[#FFFFFF, #FFFFFF]">

    <mx:Script>
        <![CDATA[
            import flash.events.TextEvent;

            public function linkHandler(event:TextEvent):void {
                myTA.text="link occured.";
                // Open the link in a new window.
                navigateToURL(new URLRequest(event.text), '_blank')
            }
        ]]>
    </mx:Script>

    <mx:Label selectable="true" link="linkHandler(event);">
        <mx:htmlText><![CDATA[<a href='event:http://www.adobe.com'>Adobe</
a>]]></mx:htmlText>
    </mx:Label>

    <mx:TextArea id="myTA"/>
</mx:Application>
```

link イベントを生成するには、Label コントロールに selectable プロパティがあり、true に設定されている必要があります。

link イベントを使用すると、イベントが生成され、ハイパーリンクのリンク先の event: に続くテキストが、イベントオブジェクトの text プロパティに含まれます。ただし、ハイパーリンクは自動的には実行されません。イベントハンドラ内から手動でハイパーリンクを実行する必要があります。これにより、アプリケーション内でハイパーリンクを変更することができ、場合によってはハイパーリンクの実行を禁止することもできます。

ボールドタグ ()

ボールド () タグは、テキストをボールドとしてレンダリングします。埋め込みフォントを使用する場合、ボールド体が使用可能になっていないとテキストは表示されません。エンドユーザーのローカルシステムに存在すると予想されるフォントを使用する場合、そのフォントが存在しなければ、システムにより近似値のボールドフォントが使用されるか、ボールドの代わりに標準フォントが使用されることがあります。いずれの場合でも、ボールドタグ内のテキストは表示されます。

次のコードの一部では、bold という語にボールド体が適用されます。

```
This word is <b>bold</b>.
```

fontWeight スタイルを使用してコントロール内のすべてのテキストに設定したボールドフォーマットを、 終了タグでオーバーライドすることはできません。

ブレークタグ (
)

ブレーク (
) タグは、テキスト内に改行を作成します。このタグは、Label または TextInput コントロールには影響しません。

次のコードの一部では、line という語句の後で改行します。

```
The next sentence is on a new line.<br>Hello there.
```

フォントタグ (<font>)

フォント (<font>) タグは、カラー、フォント、サイズなどのフォント特性を指定します。

フォントタグは、次の属性をサポートします。

color　テキストカラーを指定します。16 進数のカラー値 (#FFFFFF) を使用してください。その他の形式はサポートされていません。

face　使用するフォントの名前を指定します。カンマで区切られたフォント名の一覧を指定することもできます。その場合、Flash Player は最初に利用可能なフォントを選択します。指定したフォントがシステムにインストールされていないか、または SWF ファイルに埋め込まれていない場合、Flash Player は代替フォントを選択します。次の例は、フォントを設定する方法を示しています。

size　フォントのサイズをポイント単位で指定します。+2 または -4 など、相対サイズを使用することもできます。

次の例では、<font> tag: を使用します。

```
<?xml version="1.0"?>
<!-- textcontrols/FontTag.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
backgroundGradientColors="[#FFFFFF, #FFFFFF]">
  <mx:TextArea height="100" width="250">
     <mx:htmlText>
        <![CDATA[
```

```
            You can vary the <font size='20'>font size</font>,<br><font
color="#0000FF">color</font>,<br><font face="CourierNew, Courier,
Typewriter">face</font>, or<br><font size="18" color="#FF00FF"face="Times, Times
New Roman, _serif">any combination of the three.</font>
         ]]>
      </mx:htmlText>
   </mx:TextArea>
</mx:Application>
```

このコードの出力は、次のようになります。

イメージタグ (<img>)

メモ　Flex 2 では、<img> タグは完全にはサポートされておらず、場合によっては機能しないこともあります。

イメージ (<img>) タグは、外部の JPEG、GIF、PNG、SWF ファイルをテキストフィールドに埋め込みます。テキストは、テキストフィールドに埋め込んだイメージの周囲に自動的に配置されます。このタグがサポートされているのは、複数行でテキストを折り返す、動的テキストフィールドおよび入力テキストフィールドのみです。

デフォルトでは、Flash はテキストフィールドに埋め込まれたメディアをフルサイズで表示します。埋め込みメディアのサイズを指定するには、<img> タグの height および width 属性を使用します。

一般的に、テキストフィールドに埋め込まれたイメージは、<img> タグの次の行に表示されます。ただし、<img> タグがテキストフィールドの最初の文字の場合、イメージはテキストフィールドの最初の行に表示されます。

<img> タグには、イメージファイルへのパスを指定するための、1 つの必須の属性 src があります。その他の属性は、すべてオプションです。

<img> タグは、次の属性をサポートしています。

src　GIF、JPEG、PNG または SWF ファイルへの URL を指定します。この属性は必須です。その他の属性はすべてオプションです。外部ファイルは、完全にダウンロードされるまで表示されません。

align　テキストフィールド内の埋め込みイメージの水平方向の整列を指定します。有効な値は、left および right です。デフォルト値は、left です。

height　イメージの高さをピクセルで指定します。

hspace　イメージの周りの水平方向の空白の量 (テキストが表示されない領域) を指定します。デフォルト値は 8 です。

`id`　読み込まれたイメージの識別子を指定します。これは、ActionScript を使用して、埋め込みコンテンツを制御するときに役立ちます。

`vspace`　イメージの周りの垂直方向の空白の量 (テキストが表示されない領域) を指定します。

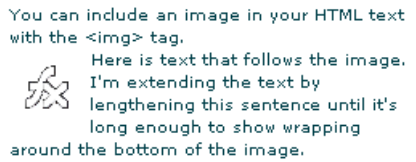
`width`　イメージの幅をピクセルで

指定します。デフォルト値は 8 です。

次の例では、<img> タグの使用方法と、テキストがイメージのイメージの周囲に流し込まれるかを示します。

```
<?xml version="1.0"?>
<!-- textcontrols/ImgTag.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
backgroundGradientColors="[#FFFFFF, #FFFFFF]" width="300" height="300">
  <mx:Text height="100%" width="100%">
     <mx:htmlText>
        <![CDATA[
           <p>You can include an image in your HTML text with the &lt;img&gt;
tag.</p><p><img src='../assets/bird.gif' width='30' height='30' align='left'
hspace='10' vspace='10'>Here is text that follows the image. I'm extending the
text by lengthening this sentence until it's long enough to show wrapping around
the bottom of the image.</p>
        ]]>
     </mx:htmlText>
  </mx:Text>
</mx:Application>
```

このコードの出力は、次のようになります。



埋め込みイメージからのハイパーリンクの作成

埋め込みイメージからハイパーリンクを作成するには、次の例に示すように、<a> タグ内に <img> タグを埋め込みます。

```
<?xml version="1.0"?>
<!-- textcontrols/ImgTagWithHyperlink.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="500"
height="500" borderStyle="solid">
  <mx:TextArea width="100%" height="100%">
     <mx:htmlText>
        <![CDATA[
           <a href='http://www.adobe.com'><img src='../assets/bird.gif'/></
a>Click the image to go to the Adobe home page.
        ]]>
```

```
    </mx:htmlText>
  </mx:TextArea>
</mx:Application>
```

ユーザーが <a> タグで囲まれたイメージ上にマウスポインタを移動させると、マウスポインタは、標準のハイパーリンクのようにハンドアイコンに変わります。マウスのクリックやキー入力などの双方向性は、<a> タグに囲まれた SWF ファイルに登録されません。

イタリックタグ ()

イタリック () タグは、指定されたテキストをイタリックフォントで表示します。埋め込みフォントを使用する場合、イタリックフォントを利用できる必要があります。利用できないなら、テキストは表示されません。エンドユーザーのローカルシステムに存在すると予想されるフォントを使用する場合、そのフォントが存在しなければ、システムにより近似値のイタリックフォントが使用されるか、イタリックの代わりに標準フォントが使用されることがあります。いずれの場合でも、イタリックタグ内のテキストは表示されます。

次のコードの一部では、italic という語にイタリックフォントが適用されます。

```
The next word is in <i>italic</i>.
```

fontStyle スタイルを使用してコントロール内のすべてのテキストに設定したイタリックフォーマットを、 終了タグでオーバーライドすることはできません。

リスト項目タグ (<li>)

リスト項目 (<li>) タグは、タグで囲まれたテキストの先頭に黒丸を付けて、新しい行に表示します。これ以外の種類の HTML リスト項目には使用できません。終了の タグは、改行を生成します (ただし、 を入力すると単一の改行を生成します)。HTML とは異なり、 タグを <ul> タグで囲むことはできません。たとえば、次の Flex コードでは、2 つの項目を持つ黒丸付きリストが生成されます。

```
<?xml version="1.0"?>
<!-- textcontrols/BulletedListExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="500">
  <mx:Text width="100%">
    <mx:htmlText >
      <![CDATA[
        <p>This is a bulleted list:<li>First Item</li><li>Second Item</li></p>
      ]]>
    </mx:htmlText>
  </mx:Text>
</mx:Application>
```

メモ

Flex 2 では、 タグは Label コントロールでは正常に機能しません。TextInput コントロールでは、テキストの最初の文字の前に挿入する必要があります。

段落タグ ( )

段落 ( ) タグは、新しい段落を作成します。開始タグ ( ) では改行は強制されませんが、終了タグ ( ) では改行が強制されます。HTML 内とは異なり、 タグでは段落間に強制的にダブルスペースは挿入されません。スペースは、
 タグで生成されるものと同一です。

 タグは、次の属性をサポートします。

align　段落内のテキストの整列を指定します。有効な値は、left、right、および center です。

次のコードの一部では、2 つの中央揃えの段落を作成します。

```
<p align="center">This is a first centered paragraph</p>
<p align="center">This is a second centered paragraph</p>
```

テキストフォーマットタグ (<textformat>)

テキストフォーマット (<textformat>) タグでは、HTML テキストフィールド内で TextFormat クラスの段落フォーマットプロパティのサブセットを使用できます。このサブセットには、行間の行送り、インデント、マージン、タブストップなどが含まれます。テキストフォーマットタグとビルトインの HTML タブを結合することができます。テキストフォーマットタグは、次の属性をサポートします。

- **blockindent**　左のマージンから <textformat> タブ本体のテキストまでのインデントを、ポイント単位で指定します。
- **indent**　左のマージンまたは本文のインデント (指定されている場合) から <textformat> タブ本体の先頭文字までのインデントを、ポイント単位で指定します。
- **leading**　行間の垂直の行送りを指定します。
- **leftmargin**　段落の左マージンをポイント単位で指定します。
- **rightmargin**　段落の右マージンをポイント単位で指定します。
- **tabstops**　カスタムのタブストップを負でない整数の配列として指定します。

下線タグ (<u>)

下線 (<u>) タグは、タグが付けられたテキストに下線を表示します。

次のコードの一部では、underlined という語句に下線を引きます。

```
The next word is <u>underlined</u>.
```

textDecoration スタイルを使用してコントロール内のすべてのテキストに設定した下線を、</u> 終了タグでオーバーライドすることはできません。

テキストの選択と修正

次の項で説明するとおり、TextArea、TextInput、および RichTextEditor コントロール内のテキストを選択および修正できます (Label または Text コントロールのテキストを変更するには、コントロールの text または HTMLtext プロパティに、新規の値を割り当てます)。HTMLText プロパティの詳細については、350 ページの「htmlText プロパティの使用」を参照してください。

テキストの選択

次の項で説明するとおり、Flex の編集可能なコントロールには、テキスト領域を選択し、選択内容を取得するためのプロパティとメソッドがあります。362 ページの「テキストの修正」で説明されているとおり、選択内容を修正することができます。

選択の作成

RichTextEditor コントロールの TextArea サブコントロールを含む、TextInput および TextArea コントロールには、次に示すテキスト選択プロパティとメソッドがあります。

- setSelection() メソッドは、テキストの範囲を選択します。テキスト内の開始文字と最終文字の直後の位置の 0 から始まるインデックスを指定します。
- selectionBeginIndex および selectionEndIndex は、テキスト内での選択内容の開始位置と、終了位置の直後の位置を設定または返します。位置は 0 から始まります。

たとえば、myTextArea TextArea コントロールの最初の 10 文字を選択する場合は、次のメソッドを使用します。

```
myTextArea.setSelection(0, 10);
```

選択内容の最後の文字を TextArea コントロールの 25 文字目に変更するには、次のステートメントを使用します。

```
myTextArea.endIndex=25;
```

RichTextEditor コントロールでテキストを選択するには、コントロールの TextArea サブコントロールを使用します。このサブコントロールには、textArea id を使用してアクセスします。たとえば、myRTE RichTextEditor コントロールで最初の 10 文字を選択するには、次のコードを使用します。

```
myRTE.textArea.setSelection(0, 10);
```

選択の取得

テキストコントロールの選択内容を取得するには、選択したテキストが含まれる TextRange オブジェクトを取得します。それから、TextRange オブジェクトを使用して、選択済みのテキストを修正します。詳細は、362 ページの「テキストの修正」に記載されています。次の項で説明するとおり、選択内容を取得するときに使用する方法は、コントロールのタイプによって異なります。

TextArea または TextInput コントロールで選択内容を取得するには

TextArea または TextInput コントロールで現在選択されているテキストが含まれる TextRange オブジェクトを取得するには、TextRange クラスコンストラクタを使用します。たとえば、myTextArea コントロールの現在の選択内容を取得するには、次の行を使用します。

```
var mySelectedTextRange:TextRange = new TextRange(myTextArea, true);
```

2 番目の `true` パラメータは、選択済みのテキストが含まれる TextRange オブジェクトを返すようコンストラクタに指示します。

RichTextEditor コントロールで選択内容を取得するには

TextArea サブコントロール内の現在選択されているテキストが含まれる TextRange オブジェクトを取得するには、RichTextEditor の読み取り専用プロパティである `selection` を使用します。TextRange オブジェクトを使用して、選択済みのテキストを変更できます。詳細は、「テキストの修正」に記載されています。たとえば、MyRTE RichTextEditor コントロールの現在の選択内容を取得するには、次の行を使用します。

```
public var mySelectedTextRange:TextRange = myRTE.selection;
```

テキストの修正

TextRange クラスを使用して、TextArea、TextInput、または RichTextEditor コントロールのテキストを修正します。このクラスは、次のテキストの特性に影響します。

- `text` または `htmltext` プロパティの内容
- テキストカラー、装飾 (下線)、および整列
- フォントファミリー、サイズ、スタイル (イタリック)、および太さ (ボールド)
- HTML <a> リンクの URL。

TextRange オブジェクトの取得

TextRange オブジェクトを取得するには、次の手法を使用します。

- 現在のテキスト選択が含まれる TextRange オブジェクトを取得します。詳細は、362 ページの「選択の取得」に記載されています。
- 特定のテキスト範囲が含まれる TextRange オブジェクトを作成します。

特定のテキスト範囲が含まれる TextRange オブジェクトを作成するには、次のフォーマットで TextRange コンストラクタを使用します。

```
new TextRange(control, modifiesSelection, beginIndex, endIndex)
```

テキストが含まれるコントロールを指定します。また、TextRange オブジェクトが選択内容に相当しているかどうか (つまり、選択済みテキストを表現および修正更するかどうか) を指定します。さらに、テキスト範囲内の最初の文字と最後の文字を表す 0 ベースのインデックスを指定します。原則として、選択内容の設定には TextRange コンストラクタを使用しないでください。代わりに setSelection() メソッドを使用します。詳細は、361 ページの「選択の作成」に記載されています。このため、開始インデックスと終了インデックスを指定するときは、2 番目のパラメータを常に false に設定する必要があります。

例として、myTextArea という名前の TextArea コントロールの 5 番目から 25 番目までの文字が含まれる TextRange オブジェクトを取得するには、次の行を使用します。

```
var myTARange:TextRange = new TextRange(myTextArea, false, 4, 25);
```

テキストの変更

TextRange オブジェクトを取得した後は、そのプロパティを使用して範囲内のテキストを修正します。TextRange に対する変更は、テキストコントロールに表示されます。

テキストは、TextRange オブジェクト内で、HTML テキストまたはプレーンテキストとして取得および設定できます。最初にテキストを設定したときに、どのプロパティを使用したかは関係ありません。たとえば、TextArea コントロールを作成して、その text プロパティを設定した場合、TextRange の htmlText プロパティを使用して、テキストを取得および変更することができます。次の例は、この使用法を示しており、TextRange クラスを使用してテキストの範囲にアクセスし、そのプロパティを変更する方法を示しています。また、String プロパティとメソッドを使用して、テキストインデックスを取得する方法も示しています。

```
<?xml version="1.0"?>
<!-- textcontrols/TextRangeExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
     height="500">
  <mx:Script><![CDATA[
     import mx.controls.textClasses.TextRange

     public function alterText():void {
        // Create a TextRange object starting with "the" and ending at the
        // first period. Replace it with new formatted HTML text.
       var tr1:TextRange = new TextRange(ta1, false, ta1.text.indexOf("the", 0),
ta1.text.indexOf(".", 0));
        tr1.htmlText="<i>italic HTML text</i>"

        // Create a TextRange object with the remaining text.
        // Select the text and change its formatting.
```

```
        var tr2:TextRange = new TextRange(ta1, true, ta1.text.indexOf("It", 0),
ta1.text.length-1);
        tr2.color=0xFF00FF;
        tr2.fontSize=18;
        tr2.fontStyle = "italic"; // any other value turns italic off
        tr2.fontWeight = "bold"; // any other value turns bold off
        ta1.setSelection(0, 0);
     }
  ]]></mx:Script>

  <mx:TextArea id="ta1" fontSize="12" fontWeight="bold" width="100%"
height="100">
     <mx:text>
        This is a test of the emergency broadcast system. It is only a test.
     </mx:text>
  </mx:TextArea>
  <mx:Button label="Alter Text" click="alterText();"/>
</mx:Application>
```

例 : RichTextEditor コントロールでの選択済みテキストの変更

次の例では、ユーザーがテキストを選択したときに、RichTextEditor コントロールの selectedText プロパティを使用して TextRange を取得する方法と、TextRange プロパティを使用して選択テキストの特性を取得および変更する方法を示しています。この例を使用するには、マウスでテキストの範囲を選択します。マウスボタンを離すと、選択範囲が fuchsia Courier の 20 ポイントフォントでフォーマットされた “This is replacement text.” というストリングに置き換わり、テキスト領域には元の置換テキストが表示されます。

```
<?xml version="1.0"?>
<!-- textcontrols/TextRangeSelectedText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="500">
  <mx:Script><![CDATA[
     import mx.controls.textClasses.TextRange;

     //The following text must be on a single line.
     [Bindable]
     public var htmlData:String="<textformat leading='2'><p
align='center'><b><font size='20'>HTML Formatted Text</font></b></p></
textformat><br><textformat leading='2'><p align='left'><font face='_sans'
size='12' color='#000000'>This paragraph contains <b>bold</b>, <i>italic</i>,
<u>underlined</u>, and <b><i><u>bold italic underlined </u></i></b>text. </
font></p></textformat><br><p><u><font face='arial' size='14'
color='#ff0000'>This a red underlined 14-point arial font with no alignment
set.</font></u></p><p align='right'><font face='verdana' size='12'
color='#006666'><b>This a teal bold 12-pt. Verdana font with alignment set to
right.</b></font></p>";
```

```
     public function changeSelectionText():void {
        //Get a TextRange with the selected text and find its length.
        var sel:TextRange = rte1.selection;
        var selLength:int = sel.endIndex - sel.beginIndex;
        //Do the following only if the user made a selection.
        if (selLength) {
           //Display the selection size and font color, size, and family.
           t1.text="Number of characters selected: " + String(selLength);
           t1.text+="\n\nOriginal Font Family: " + sel.fontFamily;
           t1.text+="\nOriginal Font Size: " + sel.fontSize;
           t1.text+="\nOriginal Font Color: " + sel.color;
           //Change font color, size, and family and replace selected text.
           sel.text="This is replacement text. "
           sel.color="fuchsia";
           sel.fontSize=20;
           sel.fontFamily="courier"
           //Show the new font color, size, and family.
           t1.text+="\n\nNew text length: " + String(sel.endIndex -
sel.beginIndex);
           t1.text+="\nNew Font Family: " + sel.fontFamily;
           t1.text+="\nNew Font Size: " + sel.fontSize;
           t1.text+="\nNew Font Color: " + sel.color;
        }
     }
  ]]></mx:Script>

  <!-- The text area. When you release the mouse after selecting text,
     it calls the func1 function. -->
  <mx:RichTextEditor id="rte1" htmlText="{htmlData}" width="100%" height="100%"
mouseUp="changeSelectionText()"/>
  <mx:TextArea editable="false" id="t1" fontSize="12" fontWeight="bold"
width="300" height="180"/>

</mx:Application>
```

Label コントロール

Label コントロールは、編集することのできない、単一行から成るテキストラベルです。次のような特性があります。

- ユーザーはテキストを変更できませんが、アプリケーションはテキストを修正できます。
- スタイルまたは HTML テキストを使用すると、テキストのフォーマットを指定できます。
- 整列とサイズ変更を制御できます。
- このコントロールは透明で backgroundColor プロパティを持たないため、コンポーネントのコンテナの背景は見えています。

- コントロールには境界線がありません。そのため、ラベルは背景に直接書き込まれたテキストのように表示されます。
- このコントロールはフォーカスを取得できません。

詳細については、『Adobe Flex 2 リファレンスガイド』の Label を参照してください。

編集することのできない、複数行から成るテキストフィールドを作成する場合は、Text コントロールを使用します。詳細については、369 ページの「Text コントロール」を参照してください。ユーザーが編集可能なテキストフィールドを作成するには、TextInput または TextArea コントロールを使用します。詳細については、367 ページの「TextInput コントロール」および 371 ページの「TextArea コントロール」を参照してください。

次の図に Label コントロールの例を示します。

このサンプルを作成するためのコードの詳細については、「Label コントロールの作成」を参照してください。

Label コントロールの作成

Label コントロールは、次の例に示すように、<mx:Label> タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- textcontrols/LabelControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="150"
height="80" borderStyle="solid" backgroundGradientColors="[#FFFFFF, #FFFFFF]">
  <mx:Label text="Label1"/>
</mx:Application>
```

純粋なテキストだけのストリングを指定する場合は text プロパティを使用し、HTML 形式のストリングを指定する場合は htmlText プロパティを使用します。これらのプロパティの使用の詳細については、346 ページの「text プロパティの使用」および 350 ページの「htmlText プロパティの使用」を参照してください。

Label コントロールのサイズ設定

Label コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	テキスト全体を表示できるだけの幅と高さです。
最小サイズ	0
最大サイズ	10000 x 10000 ピクセル

幅を指定しない場合、`text` または `htmlText` プロパティの値を変更すると、Label コントロールのサイズは自動的に変更されます。

サイズを明示的に指定した Label コントロールにテキスト全体を表示できるだけの領域が存在しない場合、テキストは省略記号 (...) で切り詰められ終了されます。マウスを Label コントロール上に移動すると、テキスト全体がツールヒントとして表示されます。`tooltip` プロパティを使用してツールヒントを設定した場合も、テキストでなくツールヒントが表示されます。

TextInput コントロール

TextInput コントロールは、オプションで編集も可能な単一行のテキストフィールドです。TextInput コントロールでは、Adobe Flash Player の HTML レンダリング機能がサポートされています。

詳細については、『Adobe Flex 2 リファレンスガイド』の TextInput を参照してください。

次の図に TextInput コントロールの例を示します。

編集可能な、複数行から成るテキストフィールドを作成する場合は、TextArea コントロールを使用します。詳細については、371 ページの「TextArea コントロール」を参照してください。編集不可のテキストフィールドを作成するには、Label および Text コントロールを使用します。詳細については、365 ページの「Label コントロール」および 369 ページの「Text コントロール」を参照してください。

TextInput コントロールにはラベルが含まれていませんが、Label コントロールを使用するか、Form レイアウトコンテナの FormItem コンテナで TextInput コントロールをネストすることにより、ラベルを追加できます。例については、344 ページの「テキストコントロールについて」を参照してください。TextInput コントロールは、`change`、`textInput`、および `enter` の各イベントを送出します。

TextInput コントロールを無効にすると、その内容は、`disabledColor` スタイルで指定した色で表示されます。TextInput コントロールの `editable` プロパティを `false` に設定すると、テキストの編集を防止できます。TextInput コントロールの `displayAsPassword` プロパティを設定すると、入力文字をアスタリスクで表示することにより、入力テキストを隠すことができます。

TextInput コントロールの作成

TextInput コントロールは、次の例に示すように、<mx:TextInput> タグを使用して MXML 内で定義します。MXML の他の場所 (他のタグまたは ActionScript ブロック) のコントロールを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- textcontrols/TextInputControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:TextInput id="text1" width="100"/>
</mx:Application>
```

Label コントロールと同様、純粋なテキストだけのストリングを指定する場合は text プロパティを使用し、HTML 形式のストリングを指定する場合は htmlText プロパティを使用します。詳細については、346 ページの「text プロパティの使用」および 350 ページの「htmlText プロパティの使用」を参照してください。

TextInput コントロールのサイズ設定

TextInput コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	テキストのサイズ。デフォルトの最小サイズは、高さ 22 ピクセル、幅 160 ピクセルです。
最小サイズ	0
最大サイズ	10000 x 10000 ピクセル

幅を指定しない場合、text または htmlText プロパティの値を変更すると、TextInput コントロールのサイズは自動的に変更されます。ユーザー入力に応じてサイズが変更されることはありません。

TextInput コントロールへのバインド

場合によっては、TextInput コントロールの text プロパティに変数をバインドすることもできます。それにより、次の例に示すように、コントロールは変数値を表すようになります。

```
<?xml version="1.0"?>
<!-- textcontrols/BoundTextInputControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     [Bindable]
     public var myProp:String="This is the initial myPropString.";
  ]]></mx:Script>
  <mx:TextInput text="{myProp}"/>
</mx:Application>
```

この例では、TextInput コントロールに myProp 変数の値が表示されます。変数の値が変わり、コントロールが値の変化を追跡する必要がある場合は、[Bindable] メタデータタグを使用してください。このメタデータタグを使用しないと、コンパイラによって警告が生成されます。

Text コントロール

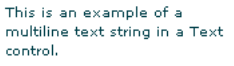
Text コントロールは複数行の編集不能なテキストを表示します。コントロールには、次のような特性があります。

- ユーザーはテキストを変更できませんが、アプリケーションはテキストを修正できます。
- コントロールでは、スクロールバーはサポートされていません。テキストの長さがコントロールのサイズを超える場合は、キーを使用してテキストをスクロールすることができます。
- コントロールはユーザーからは見えません。そのため、コンポーネントのコンテナの背景は透けて見えます。
- コントロールには境界線がありません。そのため、ラベルは背景に直接書き込まれたテキストのように表示されます。
- コントロールでは、HTML テキストのほか、各種のテキストスタイルおよびフォントスタイルをサポートしています。
- テキストはコントロールの境界で常に折り返され、コントロールに上詰めで配置されます。

詳細については、『Adobe Flex 2 リファレンスガイド』の Text を参照してください。

編集することのできない、単一行のテキストフィールドを作成する場合は、Label コントロールを使用します。詳細については、365 ページの「Label コントロール」を参照してください。ユーザーが編集可能なテキストフィールドを作成するには、TextInput または TextArea コントロールを使用します。詳細については、367 ページの「TextInput コントロール」および 371 ページの「TextArea コントロール」を参照してください。

次の図は、幅 175 ピクセルの Text コントロールの例です。



Text コントロールの作成

Text コントロールは、次の例に示すように、<mx:Text> タグを使用して MXML 内で定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- textcontrols/TextControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
  <mx:Text width="175" text="This is an example of a multiline text string in a
Text control."/>
</mx:Application>
```

純粋なテキストだけのストリングを指定する場合は text プロパティを使用し、HTML 形式のストリングを指定する場合は htmlText プロパティを使用します。詳細については、346 ページの「text プロパティの使用」および 350 ページの「htmlText プロパティの使用」を参照してください。

このコントロールでは、backgroundColor プロパティはサポートされていません。この背景は常にコントロールのコンテナの背景です。

Text コントロールのサイズ設定

Text コントロールには、次のデフォルトのサイズ設定プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	コントロールのサイズはテキストに合わせて設定されます。コントロールの幅は一番長いテキスト行の長さに、高さは行の数に合わせて調整されます。ピクセル幅を指定しない場合、高さはテキストストリング内の明示的な改行の数によって決まります。テキストの長さが変わると、コントロールのサイズも新しいテキストに合わせて変更されます。
最小サイズ	0
最大サイズ	10000 x 10000 ピクセル

Flex では次のように Text コントロールのサイズ設定が行われます。

- height および width プロパティの両方にピクセル値を指定した場合、コントロールのサイズを超えるテキストは、コントロールの境界でクリッピングされます。
- ピクセル幅を明示的に指定し、高さは指定しない場合、テキストは幅に合わせて折り返され、必要な行数を表示するための高さが計算されます。
- 幅をパーセント値ベースで指定し、高さは指定しない場合、テキストは折り返されず、高さは改行文字の数によって決まる行数と等しくなります。
- 高さのみを指定して幅は指定しない場合、高さの値は幅の計算に影響を与えず、コントロールの幅は最長の行の長さに合わせて調整されます。

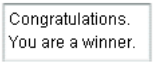
原則として、テキストが長い場合はピクセル単位で width プロパティを指定する必要があります。テキストが変更される可能性があり、アプリケーションで Text コントロールが常に同じスペースを占めるようにしたい場合は、予想されるテキストの最大長に合わせて height プロパティと width プロパティを明示的に設定してください。

TextArea コントロール

TextArea コントロールは、境界線とスクロールバー (オプション) を持つ、複数行に対応した編集可能なテキストフィールドです。TextArea コントロールでは、Flash Player の HTML とリッチテキストのレンダリング機能がサポートされています。TextArea コントロールは、change イベントおよび textInput イベントを送出します。

詳細については、『Adobe Flex 2 リファレンスガイド』の TextArea を参照してください。

次の図に TextArea コントロールの例を示します。



編集可能な単一行のテキストフィールドを作成する場合は、TextInput コントロールを使用します。詳細については、367 ページの「TextInput コントロール」を参照してください。編集不可のテキストフィールドを作成するには、Label および Text コントロールを使用します。詳細については、365 ページの「Label コントロール」および 369 ページの「Text コントロール」を参照してください。

TextArea コントロールを無効にした場合、その内容は disabledColor スタイルで指定した色で表示されます。TextArea コントロールの editable プロパティを false に設定すると、テキストの編集を防止できます。TextArea コントロールの displayAsPassword プロパティを設定すると、入力文字をアスタリスクで表示することにより、入力テキストを隠すことができます。

TextArea コントロールの作成

TextArea コントロールは、次の例に示すように、<mx:TextArea> タグを使用して MXML 内で定義します。MXML の他の場所 (他のタグまたは ActionScript ブロック) のコントロールを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- textcontrols/TextAreaControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:TextArea id="textConfirm" width="300" height="100" text="Please enter your
thoughts here."/>
</mx:Application>
```

Text コントロールと同様、純粋なテキストだけのストリングを指定する場合は text プロパティを使用し、HTML 形式のストリングを指定する場合は htmlText プロパティを使用します。詳細については、346 ページの「text プロパティの使用」および 350 ページの「htmlText プロパティの使用」を参照してください。

TextArea コントロールのサイズ設定

TextArea コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	`width` プロパティは 160 ピクセル `height` プロパティは 44 ピクセル
最小サイズ	0
最大サイズ	10000 x 10000 ピクセル

TextArea コントロールは、含まれているテキストに合わせてサイズ変更されません。新規のテキストが TextArea コントロールの容量を超え、かつ `horizontalScrollPolicy` が `true` (デフォルト値) の場合は、コントロールによりスクロールバーが追加されます。

RichTextEditor コントロール

RichTextEditor コントロールを使用すると、ユーザーがテキストを入力、編集、およびフォーマットできます。ユーザーは、RichTextEditor コントロールの下部に表示されるサブコントロールを使用して、テキストフォーマットや URL リンクを適用します。

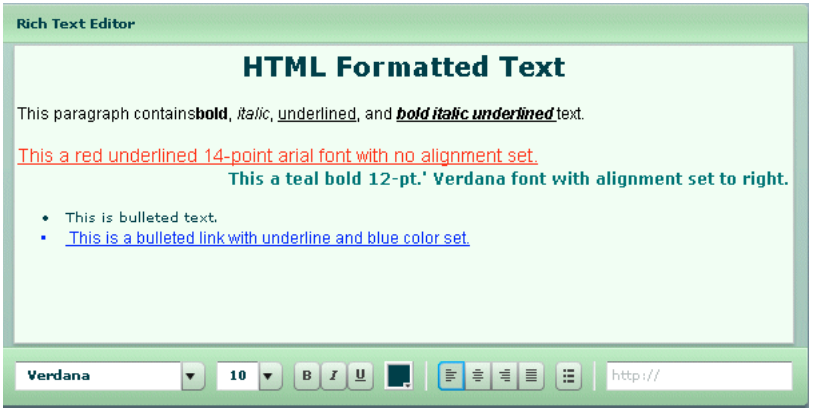
詳細については、『Adobe Flex 2 リファレンスガイド』の RichTextEditor を参照してください。

RichTextEditor コントロールについて

RichTextEditor コントロールは、Panel コントロールと 2 つの直系の子で構成されています。

- TextArea コントロール。ユーザーはその中にテキストを入力できます。
- フォーマットコントロールを持つ ToolBar コンテナ。このフォーマットコントロールを使用して、ユーザーはテキストの特性を指定します。ユーザーは、ToolBar サブコントロールを使用して、次のテキスト特性を適用することができます。
- フォントファミリー
- フォントサイズ
- ボールド、イタリック、下線のフォントスタイルの組み合わせ
- テキストカラー
- テキストの整列 : 左揃え、中央揃え、右揃え、均等割り付け
- 黒丸
- URL リンク

次の図は、フォーマットされたテキストを持つ RichTextEditor コントロールを示しています。



この例のソースの詳細については、374 ページの「RichTextEditor コントロールの作成」を参照してください。

RichTextEditor は、次に示すとおりインタラクティブに使用します。

- 入力するテキストは、コントロール設定で指定されたとおりにフォーマットされます。
- 既存のテキストに新規のフォーマットを適用するには、テキストを選択し、必要なフォーマットをコントロールに設定します。
- リンクを作成するには、テキストの範囲を選択し、右側のテキストボックスにリンクターゲットを入力してから Enter キーを押します。URL のみを指定することもできます。リンクは、_blank ターゲットでいつでも開くことができます。また、リンクを作成することにより、リンクテキストの外見が変わることはありません。任意のカラーや下線を、個別に適用する必要があります。
- 通常のキーボードコマンドを使用して、Flash HTML テキストフィールド内およびテキストフィールド間で、リッチテキストの切り取り、コピー、およびペーストを行うことができます。このテキストフィールドには、RichTextEditor コントロールの TextArea サブコントロールも含まれます。また、TextArea と任意のテキストアプリケーションの間で、プレーンテキストのコピーおよびペーストを行うこともできます。このテキストアプリケーションには、ブラウザやテキストエディタなどが含まれます。

RichTextEditor コントロールの作成

RichTextEditor コントロールは、次の例に示すように、<mx:RichTextEditor> タグを使用して MXML 内で定義します。MXML の他の場所 (他のタグまたは ActionScript ブロック) のコントロールを参照する場合は、id 値を指定します。

```
<?xml version="1.0"?>
<!-- textcontrols/RichTextEditorControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:RichTextEditor id="myRTE" text="Congratulations, winner!" />
</mx:Application>
```

フォーマットされていないテキストストリングを指定する場合は text プロパティを使用し、HTML 形式のストリングを指定する場合は htmlText プロパティを使用します。これらのプロパティの使用の詳細については、346 ページの「text プロパティの使用」および 350 ページの「htmlText プロパティの使用」を参照してください。コントロール内でのテキストの選択、置換、フォーマットの詳細については、361 ページの「テキストの選択と修正」を参照してください。

次の例では、372 ページの「RichTextEditor コントロールについて」の図を作成するためのコードを示しています。

```
<?xml version="1.0"?>
<!-- textcontrols/RichTextEditorControlWithFormattedText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="700"
height="400">
  <!-- The HTML text string used to populate the RichTextEditor control's
     TextArea subcontrol. The text is on a single line. -->
  <mx:Script><![CDATA[
     [Bindable]
     public var htmlData:String="<textformat leading='2'><p
align='center'><b><font size='20'>HTML Formatted Text</font></b></p></
textformat><br><textformat leading='2'><p align='left'><font face='_sans'
size='12' color='#000000'>This paragraph contains<b>bold</b>, <i>italic</i>,
<u>underlined</u>, and <b><i><u>bold italic underlined </u></i></b>text.</
font></p></textformat><br><p><u><font face='arial' size='14'
color='#ff0000'>This a red underlined 14-point arial font with no alignment
set.</font></u></p><p align='right'><font face='verdana' size='12'
color='#006666'><b>This a teal bold 12-pt.' Verdana font with alignment set to
right.</b></font></p><br><li>This is bulleted text.</li><li><font face='arial'
size='12' color='#0000ff'><u> <a href='http://www.adobe.com'>This is a bulleted
link with underline and blue color set.</a></u></font></li>";
  ]]></mx:Script>

  <!-- The RichTextEditor control. To reference a subcontrol prefix its ID with
the RichTextEditor control ID. -->
  <mx:RichTextEditor id="rte1"
     backgroundColor="#ccffcc"
     width="605"
     headerColors="[#88bb88, #bbeebb]"
     footerColors="[#bbeebb, #88bb88]"
```

```
    title="Rich Text Editor"
    htmlText="{htmlData}"
    initialize="rte1.textArea.setStyle('backgroundColor', '0xeeffee')"
  />
</mx:Application>
```

RichTextEditor コントロールのサイズ設定

RichTextEditor コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	幅 325 ピクセル、高さ 300 ピクセル
最小サイズ	幅 220 ピクセル、高さ 200 ピクセル
最大サイズ	10000 x 10000 ピクセル

このコントロールは、TextArea コントロールのテキストのサイズに応じてサイズ変更されることはありません。テキストが表示可能なスペースを超えた場合、デフォルトでは、TextArea コントロールによりスクロールバーが追加されます。`height` または `width` プロパティのいずれか (両方ではない) の値を指定すると、コントロールでは、設定していないプロパティに対してデフォルト値が使用されます。

設定した `width` 値により、幅が 605 ピクセル未満になる場合、RichTextEditor コントロールでは、サブコントロールが連続してスタックされます。

RichTextEditor サブコントロールのプログラミング

TextArea や ColorPicker などの RichTextEditor サブコントロールの設定、またはテキストフォーマットを制御するいずれかの ComboBox コントロールや Button コントロール設定はすべてアプリケーションで制御できます。RichTextEditor サブコントロールを参照するには、『Adobe Flex 2 リファレンスガイド』の RichTextEditor に記載されている、要求するコントロール ID の前に、RichTextEditor コントロールの ID を追加します。たとえば、参照する RichTextEditor コントロール中の ColorPicker コントロールの ID が rte1 の場合、rte1.colorPicker を使用します。

RichTextEditor コントロールに直接適用する継承可能なスタイルは、基になる Panel コントロールとそのサブコントロールに影響します。RichTextEditor コントロールに直接適用するプロパティは、基になる Panel コントロールのみに影響します。

RichTextEditor サブコントロールのプロパティとスタイルの設定

次に示す簡単な例は、RichTextEditor コントロールおよびそのサブコントロールのプロパティとスタイルを設定および変更する方法を示しています。この例では、RichTextEditor コントロールが Panel クラスから継承したスタイルを使用して、Panel コントロールのヘッダと ToolBar コンテナのカラーを設定します。また、RichTextEditor コントロールの creationComplete イベントメンバー内の TextArea コントロールの背景色を設定します。ユーザーがボタンをクリックすると、このクリックのイベントリスナーにより、TextArea コントロールの背景色と ColorPicker コントロールの選択済みカラーが変更されます。

```
<?xml version="1.0"?>
<!-- textcontrols/RTESubcontrol.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
     height="420">
  <!-- The RichTextEditor control. To set the a subcontrol's style or property,
fully qualify the control ID. The footerColors style sets the ControlBar colors.
-->
  <mx:RichTextEditor id="rte1"
     backgroundColor="#ccffcc"
     headerColors="[#88bb88, #bbeebb]"
     footerColors="[#bbeebb, #88bb88]"
     title="Rich Text Editor"
     creationComplete="rte1.textArea.setStyle('backgroundColor','0xeeffee')"
     text="Simple sample text"
  />

  <!-- Button to set a white TextArea background. -->
  <mx:Button
     label="Change appearance"
     click="rte1.textArea.setStyle('backgroundColor',
'0xffffff');rte1.colorPicker.selectedIndex=27;"
  />

  <!-- Button to reset the display to its original appearance. -->
  <mx:Button
     label="Reset Appearance"
     click="rte1.textArea.setStyle('backgroundColor',
'0xeeffee');rte1.colorPicker.selectedIndex=0;"
  />
</mx:Application>
```

RichTextEditor サブコントロールの削除と追加

整列ボタンのような標準的な RichTextEditor サブコントロールは、いずれも削除できます。また、検索および置換のダイアログボックスをポップアップ表示させるボタンのような、独自のサブコントロールを追加することもできます。

既存のサブコントロールを削除するには :

1. エディタのツールバーコンテナサブコントロールの `removeChildAt` メソッドを呼び出す関数を作成して、削除するコントロールを指定します。
2. RichTextEditor コントロールの `initialize` イベントリスナー内で、このメソッドを呼び出します。

次の例では、RichTextEditor コントロールから整列ボタンを削除し、2 番目の RichTextEditor コントロールのデフォルトの外観を表示します。

```
<?xml version="1.0"?>
<!-- textcontrols/RTERemoveAlignButtons.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     public function removeAlignButtons():void {
        rt1.toolbar.removeChild(rt1.alignButtons);
     }
  ]]></mx:Script>

  <mx:RichTextEditor id="rt1" title="RichTextEditor With No Align Buttons"
creationComplete="removeAlignButtons()"/>

  <mx:RichTextEditor id="rt2" title="Default RichTextEditor"/>

</mx:Application>
```

新しいサブコントロールを追加するには :

1. サブコントロールを定義する ActionScript 関数を作成します。また、コントロールの関数をサポートするために必要なメソッドも作成します。
2. 次に例に示すように、RichTextEditor コントロールの `initialize` イベントリスナー内でこのメソッドを呼び出します。

```
<mx:RichTextEditor id="rt" initialize="addMyControl()"
```

次の例では、RichTextEditor コントロールに検索および置換のダイアログボックスを追加します。これはアプリケーションと、検索および置換のダイアログを定義する (さらにテキスト上で検索および置換の操作も実行する) カスタムの TitleWindow コントロールという、2 つのファイルで構成されます。アプリケーションには、次のような TitleWindow をポップアップ表示させるボタンを追加する関数が含まれています。

```
<?xml version="1.0"?>
<!-- textcontrols/CustomRTE.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
     width="600" height="100%">
  <mx:Script>
     <![CDATA[
        import mx.controls.*;
        import mx.containers.*;
        import flash.events.*;
        import mx.managers.PopUpManager;
        import mx.core.IFlexDisplayObject;

        // The variable for the pop-up dialog box.
        public var w:IFlexDisplayObject;

        // Add the Find/Replace button to the Rich Text Editor control's
        // tool bar container.
        public function addFindReplaceButton():void {
           var but:Button = new Button();
           but.label = "Find/Replace";
           but.addEventListener("click",findReplaceDialog);
           rt.toolbar.addChild(but);
        }

        // The event listener for the Find/Replace button's click event
        // creates a pop-up with a MyTitleWindow custom control.
        public function findReplaceDialog(event:Event):void {
           var w:MyTitleWindow = MyTitleWindow(PopUpManager.createPopUp
            (this, MyTitleWindow, true));
           w.height=200;
           w.width=340;
           // Pass the a reference to the textArea subcontrol
           // so that the custom control can replace the text.
           w.RTETextArea = rt.textArea;
           PopUpManager.centerPopUp(w);
        }
     ]]>
  </mx:Script>

<mx:RichTextEditor id="rt" width="95%" title="RichTextEditor"
     text="This is a short text."
     initialize="addFindReplaceButton()"/>

</mx:Application>
```

次の "MyTitleWindow.mxml" ファイルは、検索および置換のインターフェイスとロジックを含むカスタムの myTitleWindow コントロールを定義します。

```
<?xml version="1.0"?>
<!-- A TitleWindow that displays the X close button. Clicking the close button
only generates a CloseEvent event, so it must handle the event to close the
control. -->
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" title="Find/Replace"
showCloseButton="true" close="closeDialog();">
<mx:Script>
<![CDATA[
import mx.controls.TextArea;
import mx.managers.PopUpManager;

// Reference to the RichTextArea textArea subcontrol.
// It is set by the application findReplaceDialog method
// and used in the replaceAndClose method, below.
public var RTETextArea:TextArea;

// The event handler for the Replace button's click event.
// Replace the text in the RichTextEditor TextArea and
// close the dialog box.
public function replaceAndClose():void{
RTETextArea.text = RTETextArea.text.replace(ti1.text, ti2.text);
PopUpManager.removePopUp(this);
}

// The event handler for the TitleWindow close button.
public function closeDialog():void {
PopUpManager.removePopUp(this);
}

]]>
</mx:Script>

<!-- The TitleWindow subcontrols: the find and replace inputs,
their labels, and a button to initiate the operation. -->
<mx:Label text="Find what:"/>
<mx:TextInput id="ti1"/>
<mx:Label text="Replace with:"/>
<mx:TextInput id="ti2"/>
<mx:Button label="Replace" click="replaceAndClose();"/>

</mx:TitleWindow>
```

第 11 章

メニューベースのコントロールの使用

11

Adobe Flex フレームワークには、メニューを作成または操作するコントロールがいくつかあります。このトピックでは、これらのコントロールの内容と使用法について説明します。

目次

メニューベースのコントロールについて

Flex フレームワークには、階層データをカスケーディングメニュー形式に表示するコントロールが 3 つあります。すべてのメニューコントロールでそれぞれのメニューアイテムのアイコンとラベルを設定でき、ユーザーの操作に対する応答として mx.events.MenuEvent クラスのイベントを送出することができます。メニューベースのコントロールには次の種類があります。

Menu コントロール　カスケーディングサブメニューを設定できるビジュアルメニュー。通常、メニューコントロールは、ボタンのクリックなどのユーザーの操作に対する応答として表示します。MXML タグでは Menu コントロールを定義できません。Menu コントロールの定義、表示、非表示は、ActionScript を使用して行います。

MenuBar コントロール　メニューアイテムが水平に並んだメニューバー。メニューバーのそれぞれのアイテムには、ユーザーが MenuBar アイテムをクリックしたときに表示されるサブメニューを設定できます。MenuBar コントロールは実質的に、メニューの最上位をバーアイテムとして表示す静的 (ポップアップしない) メニューです。

PopUpMenuButton コントロール　PopUpButton コントロールのサブクラスで、ユーザーがポップアップボタン (小さいボタン) をクリックしたときに Menu コントロールを表示します。メインボタン (大きいボタン) のラベルは、ユーザーがポップアップメニューからアイテムを選択すると変わります。

メニューの構造とデータの定義

メニューベースのコントロールでは、次のような特性を備えたデータプロバイダを使用して、メニューの構造と内容を指定します。

- ほとんどの場合データプロバイダは階層型ですが、単一レベルのメニューも指定できます。
- 個々のメニューアイテムには、その外観と動作を決定するフィールドが含まれています。メニューベースのコントロールには、ラベルテキスト、アイコン、メニューアイテムのタイプ、およびアイテムの状態を定義するフィールドがあります。有効なフィールドについては、383 ページの「メニュー項目情報の指定と使用」を参照してください。

メニューのデータプロバイダについて

メニューベースのコントロールの `dataProvider` プロパティで、メニューの構造と内容を定義するオブジェクトを指定します。メニューの内容が動的に変わる場合、データプロバイダを変更してメニューを変化させます。

通常、メニューベースのコントロールは、ネストしたオブジェクトの配列や XML などの階層的なデータプロバイダからデータを取得します。動的に変化するデータをメニューに表示するには、ArrayCollection や XMLListCollection などの ICollectionView インターフェイスを実装しているオブジェクトを使用します。

メニューベースのコントロールは、データ記述子を使用してデータプロバイダの内容を解析および操作します。デフォルトでは、メニューベースのコントロールは DefaultDataDescriptor クラスの記述子を使用しますが、独自のクラスを作成して、そのクラスを Menu コントロールの `dataDescriptor` プロパティに指定することもできます。

DefaultDataDescriptor クラスでは、次の種類のデータを使用できます。

XML　有効な XML テキストを含むストリング、あるいは有効な E4X 形式の XML データを含む `<mx:XML>` または `<mx:XMLList>` の各コンパイル時タグか XML オブジェクトまたは XMLList オブジェクトのいずれか。

その他のオブジェクト　アイテムの配列、またはアイテムの配列を含むオブジェクトで、ノードの子が `children` という名前のアイテムに含まれているもの。`<mx:Model>` コンパイル時タグを使用して、データバインディングをサポートするネストされたオブジェクトを作成できますが、その場合は 189 ページの「Tree およびメニューコントロールでの <mx:Model> タグの使用」で定義された構造に従う必要があります。

コレクション　ArrayCollection クラスや XMLListCollection クラスなどの ICollectionView インターフェイスを実装していて、そのデータソースが上記の 2 つの項目のいずれかで指定された構造に適合しているオブジェクト。DefaultDataDescriptor クラスには、コレクションを効率的に処理できるコードが含まれています。メニューのデータが動的に変わる場合は、コレクションをデータプロバイダとして使用してください。それ以外を使用すると、メニューのデータが更新されずに古くなります。

DefaultDataDescriptor でサポートされる形式の詳細など、階層オブジェクトやデータ記述子の詳細については、186 ページの「データ記述子と階層データプロバイダ構造」を参照してください。

すべてのデータ駆動型コントロールと同様に、データプロバイダの内容が動的に変化し、その変化をメニューに反映させる必要がある場合は、データソースには ArrayCollection オブジェクトや XMLListCollection オブジェクトなどのコレクションを使用してください。メニューを変更するには、基となっているコレクションを変更します。メニューの外観は、それに従って自動的に変わります。

XML データ内のノード (メニューアイテム) タグはどのような名前でもかまいません。このトピックの多くの例では、すべてのメニュー項目に <node>、最上位のアイテムに <menuItem>、サブメニューのアイテムに <subMenuItem> などのタグを使用しますが、<person> や <address> などのデータを識別できるタグ名を使用する方が現実に即しています。メニューを処理するコードは XML を読み取って、ネストされたノード間の関係に応じて階層を表示します。詳細については、「メニュー項目情報の指定と使用」を参照してください。

ほとんどのメニューの最上位には、単一のルートアイテムではなく、複数のアイテムがあります。<mx:XML> タグで作成された XML オブジェクトなどには、単一のルートノードが必要です。ルートのあるデータプロバイダを使用するメニューを表示する際に、そのルートを非表示にするには、Menu、PopUpMenuButton、または MenuBar の showRoot プロパティを false に設定します。

メニュー項目情報の指定と使用

メニュー項目の表示方法と使用法は、メニューベースのコントロールのデータプロバイダにある情報によって決まります。メニューの内容にアクセスする、またはメニューの内容を変更するには、データプロバイダの内容を変更します。

メニューベースのクラスでは、IMenuDataDescriptor クラスのメソッドを使用して、メニューの動作や内容を定義するデータプロバイダの情報にアクセスして操作します。Flex では、これらのインターフェイスを実装している DefaultDataDescriptor クラスが用意されています。dataDescriptor プロパティを設定しないと、メニューベースのコントロールでは DefaultDataDescriptor クラスを使用します。このセクションでは、ユーザーが設定できるメニュー情報と、DefaultDataDescriptor クラスを使用するときに設定するデータプロバイダのフィールドと値について説明します。

メニュー項目のタイプ

データプロバイダのそれぞれの項目では、アイテムのタイプと、タイプ特有のメニューアイテム情報を指定できます。メニューベースのクラスでは、次のアイテムのタイプ (type フィールドの値) を使用できます。

normal (デフォルト) normal タイプのアイテムを選択すると、change イベントがトリガされます。ただしアイテムに子が設定されている場合は、サブメニューが開きます。

check check タイプのアイテムを選択すると、メニューアイテムの toggled プロパティの値が true から false に、またはその逆に切り替わります。メニューアイテムに true が設定されている状態では、メニューのアイテムラベルの横にチェックマークが表示されます。

radio radio タイプのアイテムは RadioButton コントロールに非常に似ていて、グループ単位で機能します。同一グループのラジオメニューアイテムは、一度に 1 つしか選択できません。このセクションの例では、3 つのサブメニューアイテムを "one" というグループのラジオボタンとして定義しています。

選択されたラジオアイテムの toggled プロパティは true に設定され、同一グループのこれ以外のラジオアイテムの toggled プロパティは false に設定されます。Menu コントロールでは、選択されているラジオボタンの隣に塗りつぶされた円が表示されます。ラジオグループの selection プロパティが、選択されたメニューアイテムのラベルに設定されます。

separator separator タイプのアイテムは、メニュー内のアイテムを別々の表示グループに分ける単純な横線を描きます。

メニューの属性

メニューアイテムでは、アイテムの表示方法と動き方を決定する属性をいくつか指定できます。次の表では、指定できる属性とそのデータ型、さらにその目的と、メニューで DefaultDataDescriptor クラスを使用してデータプロバイダを解析する場合に、そのデータプロバイダが属性をどのように表示するかを一覧で説明します。

属性	データ型	内容
enabled	Boolean	ユーザーがメニューアイテムを選択できる場合は true、選択できない場合は false を指定します。このプロパティを指定しないと、アイテムはこの値が true の場合と同じように扱われます。 デフォルトのデータ記述子を使用する場合、データプロバイダでは enabled XML 属性またはオブジェクトフィールドを使用して、この特定を指定する必要があります。
groupName	ストリング	(radio タイプでのみ必須および有効) ラジオグループ内のラジオボタンアイテムを関連付ける識別子。デフォルトのデータ記述子を使用する場合、データプロバイダでは groupName XML 属性またはオブジェクトフィールドを使用して、この特定を指定する必要があります。

属性	データ型	内容
icon	Class	イメージアセットのクラス識別子を指定します。このアイテムは、check タイプ、radio タイプおよび separator タイプでは使用しません。選択したラジオアイテムやチェックボックスアイテムに使用するアイコンを指定するには、checkIcon スタイルや radioIcon スタイルを使用します。 アイコンを指定するデータ内のフィールド名や、アイコンを決定する関数は、メニューの iconField プロパティまたは iconFunction プロパティで指定します。
label	ストリング	コントロールに表示されるテキストを指定します。このアイテムは、separator 以外のすべてのタイプのメニューアイテムに使用します。 ラベルを指定するデータ内のフィールド名や、ラベルを決定する関数は、メニューの labelField プロパティまたは labelFunction プロパティで指定します。データプロバイダが E4X XML 形式の場合、ラベルを表示するにはこれらのプロパティのいずれかを指定する必要があります。データプロバイダがストリングの配列の場合、ストリング値がラベルとして使用されます。
toggled	Boolean	check アイテムまたは radio アイテムが選択されているかを指定します。このプロパティを選択しないと、アイテムはこの値が false の場合と同じように扱われ、アイテムは選択されません。 デフォルトのデータ記述子を使用する場合、データプロバイダでは toggled XML 属性またはオブジェクトフィールドを使用して、この特定を指定する必要があります。
type	ストリング	メニューアイテムのタイプを指定します。有効な値は separator、check、および radio です。他の値や、タイプ項目のないノードは、normal のメニュー項目として扱われます。 デフォルトのデータ記述子を使用する場合、データプロバイダでは type XML 属性またはオブジェクトフィールドを使用して、この特定を指定する必要があります。

メニューベースのコントロールは、これ以外のオブジェクトフィールドや XML 属性をすべて無視するため、これらをアプリケーション特有のデータとして使用できます。

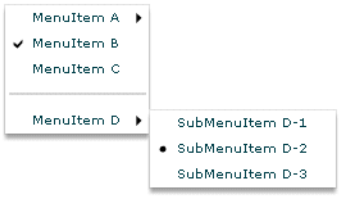
例 : 配列をメニューのデータプロバイダとして使用する

次の例では、配列をデータプロバイダとして使用するメニューと、そのメニュー特性をデータプロバイダに定義する方法を示しています。同じメニュー構造を XML に指定するアプリケーションについては、400 ページの「例 : 単純な Menu コントロールの作成」を参照してください。

```
<?xml version="1.0"?>
<!-- menus/ArrayDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
```

```
    <mx:Script>
        <![CDATA[
            import mx.controls.Menu;

            // Method to create an Array-based menu.
            private function createAndShow():void {
                // The third parameter sets the showRoot property to false.
                // You must set this property in the createMenu method,
                // not later.
                var myMenu:Menu = Menu.createMenu(null, menuData, true);
                myMenu.show(10, 10);
            }

            // The Array data provider
            [Bindable]
            public var menuData:Array = [
                {label: "MenuItem A", children: [
                    {label: "SubMenuItem A-1", enabled: false},
                    {label: "SubMenuItem A-2", type: "normal"}
                    ]},
                {label: "MenuItem B", type: "check", toggled: true},
                {label: "MenuItem C", type: "check", toggled: false},
                {type: "separator"},
                {label: "MenuItem D", children: [
                    {label: "SubMenuItem D-1", type: "radio",
                        groupName: "g1"},
                    {label: "SubMenuItem D-2", type: "radio",
                        groupName: "g1", toggled: true},
                    {label: "SubMenuItem D-3", type: "radio",
                        groupName: "g1"}
                    ]}
                ];
        ]]>
    </mx:Script>

    <!-- Button control to create and open the menu. -->
    <mx:Button x="300" y="10"
        label="Open Menu"
        click="createAndShow();"/>
</mx:Application>
```

次の図は、上記のコードによって作成されたコントロールです。MenuItem D が開き、チェックアイテム B とラジオアイテム D-2 が選択されています。



例 : XML メニューデータプロバイダおよびアイコン

次の例では、XML をデータプロバイダとして使用するメニューコントロールを表示し、コントロール内のアイテムのカスタムアイコンを指定します。

```
<?xml version="1.0"?>
<!-- menus/SimpleMenuControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[
            // Import the Menu control.
            import mx.controls.Menu;

            [Bindable]
            [Embed(source="assets/topIcon.jpg")]
            public var myTopIcon:Class;

            [Bindable]
            [Embed(source="assets/radioIcon.jpg")]
            public var myRadioIcon:Class;

            [Bindable]
            [Embed(source="assets/checkIcon.gif")]
            public var myCheckIcon:Class;

            // Create and display the Menu control.
            private function createAndShow():void {
                var myMenu:Menu = Menu.createMenu(null, myMenuData, false);
                myMenu.labelField="@label";

                // Specify the check icon.
                myMenu.setStyle('checkIcon', myCheckIcon);

                // Specify the radio button icon.
                myMenu.setStyle('radioIcon', myRadioIcon);

                // Specify the icon for the topmenu items.
                myMenu.iconField="@icon";
                myMenu.show(10, 10);
            }
        ]]>
    </mx:Script>

    <!-- Define the menu data. -->
    <mx:XML format="e4x" id="myMenuData">
        <root>
            <menuitem label="MenuItem A" icon="myTopIcon">
                <menuitem label="SubMenuItem A-1" enabled="False"/>
```

```
                <menuitem label="SubMenuItem A-2"/>
            </menuitem>
            <menuitem label="MenuItem B" type="check" toggled="true"/>
            <menuitem label="MenuItem C" type="check" toggled="false"
                icon="myTopIcon"/>
            <menuitem type="separator"/>
            <menuitem label="MenuItem D" icon="myTopIcon">
                <menuitem label="SubMenuItem D-1" type="radio"
                    groupName="one"/>
                <menuitem label="SubMenuItem D-2" type="radio"
                    groupName="one" toggled="true"/>
                <menuitem label="SubMenuItem D-3" type="radio"
                    groupName="one"/>
            </menuitem>
        </root>
    </mx:XML>

    <mx:VBox>
        <!-- Define a Button control to open the menu -->
        <mx:Button id="myButton"
            label="Open Menu"
            click="createAndShow();"/>
    </mx:VBox>
</mx:Application>
```

メニューベースのコントロールのイベント処理

Menu コントロールまたはメニューベースのコントロールのユーザー操作は、イベント駆動型です。つまり、ユーザーによるメニューやサブメニューの開閉、メニュー内での選択、メニューアイテムに対するカーソルの移動などの操作で生成されるイベントを、アプリケーションで処理するという意味です。イベントの詳細とその使用法については、81 ページ、第 5 章の「イベントの使用」を参照してください。

Menu コントロールと MenuBar コントロールは、メニュー特有の同じイベントを送出します。PopUpMenuButton コントロールのイベント処理は、他の 2 つのコントロールとは異なりますが、共有しているエレメントは多数あります。

Menu コントロールのイベントの処理

Menu コントロールは、次のような MenuEvent クラスのメニュー特有のイベントタイプを定義します。

change (MenuEvent.CHANGE) ユーザーがキーボードやマウスで現在のメニュー選択を変更したときに送出されます。

itemClick (MenuEvent.ITEM_CLICK) ユーザーが `normal` タイプ、`check` タイプ、または `radio` タイプの有効なメニューアイテムを選択したときに送出されます。ユーザーが `separator` タイプのメニューアイテム、サブメニューが開くメニューアイテム、または無効になっているメニューアイテムを選択した場合は、送出されません。

itemRollOut (MenuEvent.ITEM_ROLL_OUT) マウスポインタが Menu アイテムから離れたときに送出されます。

itemRollOver (MenuEvent.ITEM_ROLL_OVER) マウスポインタが Menu アイテム上に置かれたときに送出されます。

menuHide (MenuEvent.MENU_HIDE) メニュー全体または 1 つのサブメニューが閉じたときに送出されます。

menuShow (MenuEvent.MENU_SHOW) メニュー全体または 1 つのサブメニューが開いたときに送出されます。

イベントリスナーに渡されるイベントオブジェクトは MenuEvent タイプで、これには次のようなメニュー特有のプロパティが含まれます。複数含まれる場合もあります。

プロパティ	内容
`item`	イベントに関連するメニューアイテムに使用する、データプロバイダのアイテム
`index`	メニューまたはサブメニューに表示されるアイテムのインデックス
`label`	アイテムのラベル
`menu`	イベントが発生した Menu コントロールへの参照
`menuBar`	メニューの親である MenuBar コントロールのインスタンス。ただしメニューが MenuBar の子ではない場合は `undefined` になります。詳細については、402 ページの「MenuBar コントロール」を参照してください。

オブジェクトベースのメニューアイテムのプロパティとフィールドにアクセスするには、次のようにメニューアイテムのフィールド名を指定します。

```
ta1.text = event.item.label
```

E4X XML ベースのメニューアイテムの属性にアクセスするには、次のようにメニューアイテムの属性名を E4X シンタックスに指定します。

```
ta1.text = event.item.@label
```

メモ	メニューベースのコントロールのサブメニューにイベントリスナーを設定している場合、エレメントの削除などによってメニューのデータプロバイダーの構造が変わると、イベントリスナーが使用できなくなる場合があります。データプロバイダの構造が変わったときにイベントリスナーを確実に使用できるようにするには、サブメニューではなく、メニューベースのコントロールのイベントをリッスンするか、データプロバイダの構造を変更するイベントが発生するたびに、イベントリスナを追加します。

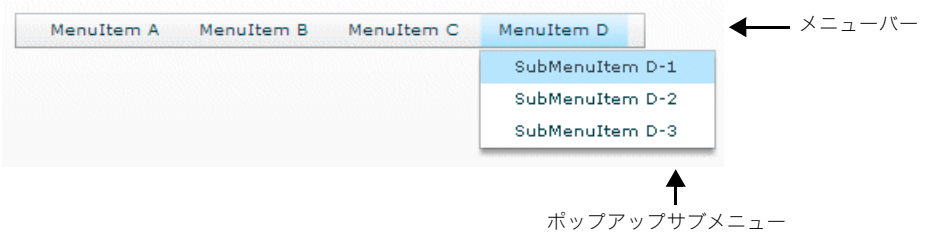
次の例は、単純なイベントリスナーを使用するメニューを示しています。複雑な例については、394 ページの「例 : Menu コントロールのイベントの使用」を参照してください。

```
<?xml version="1.0"?>
<!-- menus/EventListener.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">

    <mx:Script>
        <![CDATA[
            import mx.controls.Menu;
            import mx.events.MenuEvent;

            // Function to create and show a menu.
            private function createAndShow():void {
                var myMenu:Menu = Menu.createMenu(null, myMenuData, false);
                myMenu.labelField="@label"
                // Add an event listener for the itemClick event.
                myMenu.addEventListener(MenuEvent.ITEM_CLICK,
                    itemClickInfo);
                // Show the menu.
                myMenu.show(225, 10);
            }

            // The event listener for the itemClick event.
            private function itemClickInfo(event:MenuEvent):void {
                ta1.text="event.type: " + event.type;
                ta1.text+="\nevent.index: " + event.index;
                ta1.text+="\nItem label: " + event.item.@label
                ta1.text+="\nItem selected: " + event.item.@toggled;
                ta1.text+= "\nItem type: " + event.item.@type;
            }
        ]]>
    </mx:Script>

    <!-- The XML-based menu data provider. -->
    <mx:XML id="myMenuData">
            <xmlRoot>
```

```
            <menuitem label="MenuItem A" >
                <menuitem label="SubMenuItem A-1" enabled="false"/>
                <menuitem label="SubMenuItem A-2"/>
            </menuitem>
            <menuitem label="MenuItem B" type="check" toggled="true"/>
            <menuitem label="MenuItem C" type="check" toggled="false"/>
            <menuitem type="separator"/>
            <menuitem label="MenuItem D" >
                <menuitem label="SubMenuItem D-1" type="radio"
                    groupName="one"/>
                <menuitem label="SubMenuItem D-2" type="radio"
                    groupName="one" toggled="true"/>
                <menuitem label="SubMenuItem D-3" type="radio"
                    groupName="one"/>
            </menuitem>
            </xmlRoot>
    </mx:XML>

    <!-- Button controls to open the menus. -->
    <mx:Button x="10" y="5"
        label="Open Menu"
        click="createAndShow();"/>
    <!-- Text area to display the event information -->
    <mx:TextArea x="10" y="40"
        width="200" height="100"
        id="ta1"/>
</mx:Application>
```

MenuBar イベントの処理

次の図に MenuBar コントロールを示します。



メニューバーの場合、次のようなイベントが発生します。

change (MenuEvent.CHANGE) ユーザーがキーボードやマウスで現在のメニューバーの選択を変更したときに送出されます。また、このイベントはユーザーがポップアップサブメニューで現在のメニュー選択を変更したときにも送出されます。メニューバーでイベントが発生するとき、MenuEvent オブジェクトの menu プロパティは null です。

itemRollOut (MenuEvent.ITEM_ROLL_OUT) マウスポインタがメニューバーのアイテムから離れたときに送出されます。

itemRollOver (MenuEvent.ITEM_ROLL_OVER) マウスポインタがメニューバーのアイテム上に置かれたときに送出されます。

menuHide (MenuEvent.MENU_HIDE) サブメニューが閉じたときに送出されます。

menuShow (MenuEvent.MENU_SHOW) ポップアップサブメニューが開かれたときに送出されます。ドロップダウンメニューがない場合は、メニューバーアイテムが選択されたときに送出されます。

メモ

メニューバーのアイテムが選択されたとき、MenuBar コントロールは `itemClick` イベントを送出しません。ポップアップサブメニューのアイテムが選択されたときにのみ、`itemClick` イベントを送出します。

MenuBar はポップアップサブメニューごとに、`change`、`itemClick`、`itemRollOut`、`itemRollOver`、`menuShow`、および `menuHide` イベントを Menu コントロールの場合と同様に送出します。ポップアップメニューによってトリガされるイベントは、Menu コントロールのイベントと同じように処理します。詳細については、389 ページの「Menu コントロールのイベントの処理」を参照してください。

次の例では、メニューバーおよびポップアップサブメニューのイベントを処理します。

```
<?xml version="1.0"?>
<!-- menus/MenuBarEventInfo.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initCollections();">

    <mx:Script>
        <![CDATA[
            import mx.events.MenuEvent;
            import mx.controls.Alert;
            import mx.collections.*;

            [Bindable]
            public var menuBarCollection:XMLListCollection;

            private var menubarXML:XMLList =<>
                <menuitem label="Menu1">
                    <menuitem label="MenuItem 1-A" data="1A"/>
                    <menuitem label="MenuItem 1-B" data="1B"/>
                </menuitem>
                <menuitem label="Menu2">
                    <menuitem label="MenuItem 2-A" data="2A"/>
                    <menuitem label="MenuItem 2-B" data="2B"/>
                </menuitem>
                <menuitem label="Menu3" data="M3"/>
                </>

            // Event handler to initialize the MenuBar control.
```

```
            private function initCollections():void {
                menuBarCollection = new XMLListCollection(menubarXML);
            }

            // Event handler for the MenuBar control's change event.
            private function changeHandler(event:MenuEvent):void  {
                // Only open the Alert for a selection in a pop-up submenu.
                // The MenuEvent.menu property is null for a change event
                // dispatched by the menu bar.
                if (event.menu != null) {
                    Alert.show("Label: " + event.item.@label + "\n" +
                        "Data: " + event.item.@data, "Clicked menu item");
                }
            }

            // Event handler for the MenuBar control's itemRollOver event.
            private function rollOverHandler(event:MenuEvent):void {
                rollOverTextArea.text = "type: " + event.type + "\n";
                rollOverTextArea.text += "target menuBarIndex: " +
                    event.index + "\n";
            }

            // Event handler for the MenuBar control's itemClick event.
            private function itemClickHandler(event:MenuEvent):void {
                itemClickTextArea.text = "type: " + event.type + "\n";
                itemClickTextArea.text += "target menuBarIndex: " +
                    event.index + "\n";
            }
        ]]>
    </mx:Script>

    <mx:Panel title="MenuBar Control Example"
        height="75%" width="75%"
        paddingTop="10" paddingLeft="10">

        <mx:Label
            width="100%"
            color="blue"
            text="Select a menu item."/>
        <mx:MenuBar labelField="@label"
            dataProvider="{menuBarCollection}"
            change="changeHandler(event);"
            itemClick="itemClickHandler(event);"
            itemRollOver="rollOverHandler(event);"/>
        <mx:TextArea id="rollOverTextArea"
            width="200" height="100"/>
        <mx:TextArea id="itemClickTextArea"
            width="200" height="100"/>
    </mx:Panel>
</mx:Application>
```

例 : Menu コントロールのイベントの使用

次の例では、Menu コントロールのイベントを試すことができます。メニューを 2 つ表示することができ、1 つは XML をデータプロバイダとし、もう 1 つは配列をデータプロバイダとしています。TextArea コントロールは、ユーザーがメニューを開いたとき、マウスを動かしたとき、およびメニューアイテムを選択したときの各イベントの情報を表示します。XML ベースのメニューとオブジェクトベースのメニューを処理する場合の違いと、それぞれの Menu イベントについて取得できる情報のタイプの一部を示しています。

```
<?xml version="1.0"?>
<!-- menus/ExtendedMenuExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">

    <mx:Script>
        <![CDATA[

        // Import the Menu control and MenuEvent class.
        import mx.controls.Menu;
        import mx.events.MenuEvent;
        //Define a variable for the Menu control.
        private var myMenu:Menu;

        // The event listener that creates menu with an XML data
        // provider and adds event listeners for the menu.
        private function createAndShow():void {
            // Clear the event output display.
            ta1.text="";
            // Don't show the (single) XML root node in the menu.
            myMenu = Menu.createMenu(null, myMenuData, false);
            //You must set the labelField explicitly for XML data providers.
            myMenu.labelField="@label"
            myMenu.addEventListener(MenuEvent.ITEM_CLICK, menuShowInfo);
            myMenu.addEventListener(MenuEvent.MENU_SHOW, menuShowInfo);
            myMenu.addEventListener(MenuEvent.MENU_HIDE, menuShowInfo);
            myMenu.addEventListener(MenuEvent.ITEM_ROLL_OUT, menuShowInfo);
            myMenu.addEventListener(MenuEvent.ITEM_ROLL_OVER,
                menuShowInfo);
            myMenu.show(225, 10);
        }

        // The event listener for the menu events.
        // Retain information on all events for a menu instance.
        private function menuShowInfo(event:MenuEvent):void {
            ta1.text="event.type: " + event.type;
            ta1.text+="\nevent.label: " + event.label;
            // The index value is -1 for menuShow and menuHide events.
            ta1.text+="\nevent.index: " + event.index;
            //The item field is null for show and hide events.
```

```
    if (event.item) {
        ta1.text+="\nItem label: " + event.item.@label
        ta1.text+="\nItem selected: " + event.item.@toggled;
        ta1.text+= "\nItem type: " + event.item.@type;
    }
}

// The event listener that creates an object-based menu
// and adds event listeners for the menu.
private function createAndShow2():void {
    // Show the top (root) level objects in the menu.
    myMenu = Menu.createMenu(null, menuData, true);
    myMenu.addEventListener(MenuEvent.ITEM_CLICK, menuShowInfo2);
    myMenu.addEventListener(MenuEvent.MENU_SHOW, menuShowInfo2);
    // The following line is commented to so you can see the
    // results of an ITEM_CLICK event.
    // (The menu hides immediately after the click.)
    // myMenu.addEventListener(MenuEvent.MENU_HIDE, menuShowInfo2);
    myMenu.addEventListener(MenuEvent.ITEM_ROLL_OVER,
        menuShowInfo2);
    myMenu.addEventListener(MenuEvent.ITEM_ROLL_OUT,
        menuShowInfo2);
    myMenu.show(225, 10);
}

// The event listener for the object-based Menu events.
private function menuShowInfo2(event:MenuEvent):void {
    ta1.text="event.type: " + event.type;
    ta1.text+="\nevent.label: " + event.label;
    // The index value is -1 for menuShow and menuHide events.
    ta1.text+="\nevent.index: " + event.index;
    // The item field is null for show and hide events.
    if (event.item) {
        ta1.text+="\nItem label: " + event.item.label
        ta1.text+="\nItem selected: " + event.item.toggled;
        ta1.text+= "\ntype: " + event.item.type;
    }
}

// The object-based data provider, an Array of objects.
// Its contents is identical to that of the XML data provider.
[Bindable]
public var menuData:Array = [
    {label: "MenuItem A", children: [
        {label: "SubMenuItem A-1", enabled: false},
        {label: "SubMenuItem A-2", type: "normal"}
    ]},
    {label: "MenuItem B", type: "check", toggled: true},
    {label: "MenuItem C", type: "check", toggled: false},
            {type: "separator"},
```

```
            {label: "MenuItem D", children: [
                {label: "SubMenuItem D-1", type: "radio", groupName: "g1"},
                {label: "SubMenuItem D-2", type: "radio", groupName: "g1",
                        toggled: true},
                {label: "SubMenuItem D-3", type: "radio", groupName: "g1"}
            ]}
        ];

    ]]>
    </mx:Script>

    <!-- The XML-based menu data provider.
        The <mx:XML tag requires a single root. -->
    <mx:XML id="myMenuData">
            <xmlRoot>
            <menuitem label="MenuItem A" >
                <menuitem label="SubMenuItem A-1" enabled="false"/>
                <menuitem label="SubMenuItem A-2"/>
            </menuitem>
            <menuitem label="MenuItem B" type="check" toggled="true"/>
            <menuitem label="MenuItem C" type="check" toggled="false"/>
            <menuitem type="separator"/>
            <menuitem label="MenuItem D" >
                <menuitem label="SubMenuItem D-1" type="radio"
                    groupName="one"/>
                <menuitem label="SubMenuItem D-2" type="radio"
                    groupName="one" toggled="true"/>
                <menuitem label="SubMenuItem D-3" type="radio"
                    groupName="one"/>
            </menuitem>
            </xmlRoot>
    </mx:XML>

    <!-- Button controls to open the menus. -->
    <mx:Button x="10" y="5"
        label="Open XML Popup"
        click="createAndShow();"/>
    <mx:Button x="10" y="35"
        label="Open Object Popup"
        click="createAndShow2();"/>
    <!-- Text area to display the event information -->
    <mx:TextArea x="10" y="70"
        width="200" height="300"
        id="ta1"/>
</mx:Application>
```

PopUpMenuButton コントロールのイベント処理

PopUpMenuButton は PopUpButton コントロールのサブクラスなので、PopUpButton コントロールのすべてのイベントを使用できます。ユーザーがメインボタンをクリックすると、PopUpMenuButton コントロールが click (MouseEvent.CLICK) イベントを送出します。

ユーザーが PopUpMenuButton メインボタンをクリックすると、選択したメニューアイテムの情報を含む itemClick (MenuEvent.ITEM_CLICK) イベントが、コントロールによって送出されます。そのため、メインボタンをクリックした場合と、ポップアップメニューから現在のアイテムを選択した場合では、同じ itemClick イベントが送出されます。どちらの場合でも同じイベントが送出されるため、メインボタンをクリックすると、最後に選択した menuItem をクリックしたときと同じビヘイビアが生成されます。そのため、メインボタンは頻繁に使用されるメニューアイテムの役割を果たします。

次の例は、PopUpMenuButton によるイベントの生成方法と、アプリケーションによるイベントの処理方法を示しています。

ユーザーがポップアップメニューからアイテムを選択すると、次のことが起こります。

- PopUpMenuButton によって itemClick イベントが送出されます。
- アプリケーションの itemClickHandler() イベントリスナー関数によって itemClick イベントが処理され、イベントの情報が Alert コントロールに表示されます。

ユーザーがメインボタンをクリックすると、次のことが起こります。

- PopUpMenuButton コントロールによって click イベントが送出されます。
- PopUpMenuButton コントロールによって itemClick イベントが送出されます。
- アプリケーションの itemClickHandler() イベントリスナー関数によって itemClick イベントが処理され、選択した Menu アイテムの情報が Alert コントロールに表示されます。
- アプリケーションの clickHandler() イベントリスナー関数によって MouseEvent.CLICK イベントも処理され、Button ラベルが Alert コントロールに表示されます。

```
<?xml version="1.0"?>
<!-- menus/PopUpMenuButtonEvents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="600" width="600"
    creationComplete="initData();">

    <mx:Script>
        <![CDATA[
        import mx.events.*;
        import mx.controls.*;

        // Set the Inbox (fourth) item in the menu as the button item.
        private function initData():void {
            Menu(p1.popUp).selectedIndex=3;
        }
```

```
        // itemClick event handler, invoked when you select from the menu.
        // Shows the event's label, index properties, and the values of the
        // label and data fields of the data provider entry specified by
        // the event's item property.
        public function itemClickHandler(event:MenuEvent):void {
            Alert.show("itemClick event label: " + event.label
                + "  \nindex: " + event.index
                + "  \nitem.label: " + event.item.label
                + "  \nitem.data: " + event.item.data);
        }

        //Click event handler for the main button.
        public function clickHandler(event:MouseEvent):void {
            Alert.show(" Click Event currentTarget.label: "
                + event.currentTarget.label);
        }

        //The menu data provider
        [Bindable]
        public var menuDP:Array = [
            {label: "Inbox", data: "inbox"},
            {label: "Calendar", data: "calendar"},
            {label: "Sent", data: "sent"},
            {label: "Deleted Items", data: "deleted"},
            {label: "Spam", data: "spam"}
        ];
    ]]></mx:Script>

    <mx:PopUpMenuButton  id="p1"
        showRoot="true"
        dataProvider="{menuDP}"
        click="clickHandler(event)"
        itemClick="itemClickHandler(event);"
    />
</mx:Application>
```

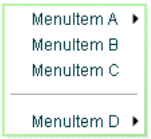
Menu コントロール

Menu コントロールは、個別に選択できる選択肢を持つポップアップメニューです。ユーザーの操作に対する応答として Menu コントロールを作成するには、ActionScript を使用します。通常、この処理はイベントリスナーの一部です。Menu コントロールは、イベントに対する応答として作成されるので、MXML タグを持ちません。Menu コントロールは、ActionScript でのみ作成できます。

詳細については、『Adobe Flex 2 リファレンスガイド』の Menu を参照してください。

Menu コントロールについて

次の例は Menu コントロールです。



この例では、MenuItem A と MenuItem D というアイテムがあり、それぞれサブメニューが開きます。サブメニューは、ユーザーが親アイテムの上にマウスポインタを移動したとき、またはキーボードのキーを使用して親アイテムにアクセスしたときに開きます。

Menu コントロールのデフォルトの場所はアプリケーションの左上隅で、x 座標と y 座標はそれぞれ 0 です。x パラメータと y パラメータを `show()` メソッドに渡すと、アプリケーションを基準とした Menu コントロールの場所を制御できます。

開いたメニューは、ユーザーが有効なメニューアイテムを選択するか、ユーザーがアプリケーションの他のコンポーネントを選択するか、またはスクリプトによって閉じられるまで、表示されたままになります。

常に表示されている " 静的な " メニューを作成するには、MenuBar コントロールまたは PopUpMenuButton コントロールを使用します。MenuBar コントロールの詳細については、402 ページの「MenuBar コントロール」を参照してください。PopUpMenuButton コントロールの詳細については、405 ページの「PopUpMenuButton コントロール」を参照してください。

Menu コントロールには、次のようなサイズ設定の特性があります。

プロパティ	デフォルト値
デフォルトサイズ	幅は Menu のテキストによって決まります。デフォルトの高さは、メニューの行数に 1 行の高さのデフォルトである 19 ピクセルをかけた数です。

Menu コントロールの作成

MXML タグでは Menu コントロールを作成できません。Menu コントロールは、ActionScript で作成する必要があります。

Menu コントロールを作成するには :

1. ActionScript の静的な `Menu.createMenu()` メソッドを呼び出し、Menu コントロールを 2 番目のパラメータとして設定しているデータプロバイダのインスタンスをこのメソッドに渡して、Menu コントロールのインスタンスを作成します。次に例を示します。

   ```
   var myMenu:Menu = Menu.createMenu(null, myMenuData);
   ```

   最初のパラメータには、オプションでメニューの親コンテナを指定できます。

データプロバイダのルートノードを表示しない場合、たとえばデータプロバイダが E4X 形式の XML ドキュメントであれば、3 番目のパラメータに false の値を使用します。このパラメータによって、メニューの showRoot プロパティが設定されます。次の例では、データプロバイダのルートを表示しないメニューを作成します。

```
var myMenu:Menu = Menu.createMenu(null, myMenuData, false);
```

メモ

ルートノードを非表示にするには、createMenu メソッドの showRoot プロパティを設定する必要があります。メニューの作成後にこのプロパティを設定しても、適用されません。

2. 次の例のように、ActionScript の Menu.show() メソッドを呼び出して、Menu のインスタンスを表示します。

```
myMenu.show(10, 10);
```

メモ

Menu オブジェクトは、show() メソッドによって自動的に表示リストに追加され、hide() メソッドによって自動的に表示リストから削除されます。メニューの外をクリックするか Esc キーを押しても、Menu オブジェクトは非表示になり、表示リストから削除されます。
Menu.createMenu() メソッドで Menu オブジェクトを作成した場合、そのメニューを閉じると Menu オブジェクトは自動的に表示リストから削除されます。このデフォルトのビヘイビアをキャンセルするには、menuHide イベントをリッスンして、このイベントオブジェクトに対して preventDefault() を呼び出します。
Menu.popUpMenu() メソッドで表示されたメニューは自動的には削除されないため、Menu オブジェクトに対して PopUpManager.removePopUp() メソッドを呼び出す必要があります。

例 : 単純な Menu コントロールの作成

この例では、<mx:XML> タグを使用して Menu コントロールのデータを定義します。また、Button コントロールを使用して、Menu コントロールを開くイベントをトリガします。

```
<?xml version="1.0"?>
<!-- menus/SimpleMenuControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[
            // Import the Menu control.
            import mx.controls.Menu;

            // Create and display the Menu control.
            private function createAndShow():void {
                var myMenu:Menu = Menu.createMenu(null, myMenuData, false);
                myMenu.labelField="@label";
                myMenu.show(10, 10);
            }
        ]]>
    </mx:Script>

    <!-- Define the menu data. -->
```

```
    <mx:XML format="e4x" id="myMenuData">
        <root>
            <menuitem label="MenuItem A" >
                <menuitem label="SubMenuItem A-1" enabled="False"/>
                <menuitem label="SubMenuItem A-2"/>
            </menuitem>
            <menuitem label="MenuItem B" type="check" toggled="true"/>
            <menuitem label="MenuItem C" type="check" toggled="false"/>
            <menuitem type="separator"/>
            <menuitem label="MenuItem D" >
                <menuitem label="SubMenuItem D-1" type="radio"
                    groupName="one"/>
                <menuitem label="SubMenuItem D-2" type="radio"
                    groupName="one" toggled="true"/>
                <menuitem label="SubMenuItem D-3" type="radio"
                    groupName="one"/>
            </menuitem>
        </root>
    </mx:XML>

    <mx:VBox>
        <!-- Define a Button control to open the menu -->
        <mx:Button id="myButton"
            label="Open Menu"
            click="createAndShow();"/>
    </mx:VBox>
</mx:Application>
```

XML データ内のノードタグには、どのような名前でも割り当てられます。前の例では、汎用タグの <menuitem> を使用してノードに名前を付けていますが、<node>、<subNode>、<person>、<address> なども使用できます。

この例では E4X 形式の XML データソースを使用しているため、E4X @ 属性指定子シンタックスでラベルフィールドを指定し、データプロバイダのルートノードを非表示にするようコントロールに指示する必要があります。

type 属性など、Menu コントロールにとって有効な属性またはフィールドがいくつかあります。Flex によるデータプロバイダのデータの解釈法と使用法については、383 ページの「メニュー項目情報の指定と使用」を参照してください。

Menu コントロールのユーザー操作

Menu コントロールは、マウスまたはキーボードを使用して操作できます。クリックにより、メニューアイテムを選択したり、メニューを閉じることができます。ただし、次のタイプのメニューアイテムは例外です。

無効な状態のアイテムまたはセパレータ　これらのアイテムの上にマウスを移動またはクリックしても、メニューは表示されたままになります。

サブメニューアンカー　アイテムの上にマウスを移動するとサブメニューが開きますが、クリックしても何も変化しません。開いているサブメニューのアイテム以外のメニューアイテムの上にマウスを移動させると、サブメニューが閉じます。

Menu コントロールにフォーカスがある場合、次のキーを使用して制御できます。

Key	内容
↓ (下矢印) ↑ (上矢印)	メニュー行の下および上のアイテムを選択します。行の先頭からは末尾に、末尾からは先頭に選択が移動します。
→ (右矢印)	サブメニューを開きます。メニューバーにフォーカスがある場合は、隣のメニューを選択します。
← (左矢印)	サブメニューを閉じてフォーカスを親メニューに戻します (親メニューがある場合)。または、メニューバーの前のメニューを選択します (メニューバーがある場合)。
Enter	サブメニューを開きます。サブメニューがない場合には、行をマウスでクリックしてボタンを離すのと同じ効果があります。
Esc	1 つのメニューレベルを閉じます。

MenuBar コントロール

MenuBar コントロールでは、最上位のメニューを水平なメニューバーとして表示します。メニューバーのそれぞれのアイテムは、サブメニューをポップアップで表示できます。MenuBar コントロールは、Menu コントロールと同じ方法でデータプロバイダを解釈し、Menu コントロールと同じイベントをサポートします。ただし、Menu コントロールとは異なり、MenuBar コントロールは静的です。つまり、ポップアップメニューとしては機能せずに、アプリケーション上に常に表示されます。MenuBar は静的のため、MXML に直接定義できます。

詳細については、『Adobe Flex 2 リファレンスガイド』の MenuBar を参照してください。Menu コントロールの詳細については、389 ページの「Menu コントロールのイベントの処理」を参照してください。

MenuBar コントロールについて

次の例は MenuBar コントロールです。

このコントロールは、データプロバイダメニューの最上位のラベルを示しています。ユーザーが最上位のメニューアイテムを選択すると、MenuBar コントロールはサブメニューを開きます。ユーザーが他の最上位メニューアイテムを選択するか、サブメニューアイテムを選択するか、MenuBar の領域外をクリックするまで、サブメニューは表示されたままになります。

MenuBar コントロールには、次のようなサイズ設定の特性があります。

プロパティ	デフォルト値
デフォルトサイズ	幅はメニューテキストによって決まります。幅の最小値は 27 ピクセルです。高さのデフォルト値は 22 ピクセルです。

MenuBar コントロールの作成

MXML では MenuBar コントロールを <mx:MenuBar> タグで定義します。MXML アプリケーションの他の場所 (他のタグまたは ActionScript ブロック) にあるコンポーネントを参照する場合は、id の値を指定します。

MenuBar コントロールのデータは、dataProvider プロパティで指定します。MenuBar コントロールでは、Menu コントロールと同じタイプのデータプロバイダを使用します。Menu コントロールと MenuBar コントロールに使用するデータプロバイダの詳細については、382 ページの「メニューの構造とデータの定義」を参照してください。階層データプロバイダの詳細については、185 ページの「階層データプロバイダの使用」を参照してください。

MenuBar コントロールを作成する単純なケースでは、<mx:XML> タグまたは <mx:XMLList> タグと標準的な XML ノードシンタックスを使用して、メニューのデータプロバイダを定義する場合があります。XML ベースのデータプロバイダを使用する場合は、次の規則に注意してください。

- <mx:XML> タグを使用する場合は単一のルートノードが必要で、MenuBar コントロールの showRoot プロパティを false に設定します。このように設定しない場合、MenuBar にはルートのみがボタンとして表示されます。<mx:XMLList> タグを使用する場合は XML ノードのリストを定義し、最上位ノードでメニューバーのボタンを定義します。
- データプロバイダにラベル属性がある場合、それが "label" という名前であっても、次の例のように MenuBar コントロールの labelField プロパティを設定して、ラベルに E4X の @ 表記を使用する必要があります。

  ```
  labelField="@label"
  ```

dataProvider プロパティは MenuBar コントロールのデフォルトプロパティです。そのため次の例のように、XML オブジェクトまたは XMLList オブジェクトを <mx:MenuBar> タグの直接の子として定義できます。

```
<?xml version="1.0"?>
<!-- menus/MenuBarControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

<!-- Define the menu; dataProvider is the default MenuBar property.
Because this uses an XML data provider, specify the labelField and
showRoot properties. -->
<mx:MenuBar id="myMenuBar" labelField="@label">
<mx:XMLList>
<menuitem label="MenuItem A" >
<menuitem label="SubMenuItem A-1" enabled="False"/>
 <menuitem label="SubMenuItem A-2"/>
</menuitem>
<menuitem label="MenuItem B" type="check" selected="true"/>
<menuitem label="MenuItem C" type="check" selected="false"/>
<menuitem label="MenuItem D" >
<menuitem label="SubMenuItem D-1" type="radio" groupName="one"/>
<menuitem label="SubMenuItem D-2" type="radio" groupName="one"
selected="true"/>
<menuitem label="SubMenuItem D-3" type="radio" groupName="one"/>
</menuitem>
</mx:XMLList>
</mx:MenuBar>
</mx:Application>
```

MenuBar コントロールの最上位ノードは、メニューバーのボタンになります。したがって、この例の MenuBar コントロールでは、前の図のように 4 つのラベルが表示されます。

XML データ内のノードタグには、どのような名前でも割り当てられます。前の例では、汎用タグの <menuitem> を使用してノードに名前を付けていますが、<node>、<subNode>、<person>、<address> なども使用できます。type 属性など、MenuBar コントロールにとって意味のある属性またはフィールドがいくつかあります。Flex によるデータプロバイダのデータの解釈法と使用法については、383 ページの「メニュー項目情報の指定と使用」を参照してください。

MenuBar コントロールのユーザー操作

MenuBar コントロールのユーザーの操作は Menu コントロールとほとんど同じですが、次の違いがあります。MenuBar コントロールにフォーカスがある場合、左矢印を押すと前のメニューが開きます。現在のメニューバーのアイテムに閉じたポップアップメニューがある場合、右矢印を押すと現在のメニューが開きます。ポップアップメニューが開いている場合は、左矢印で次のメニューが開きます。この動作は、MenuBar コントロールの端に達すると反対の端に戻ります。

詳細については、402 ページの「Menu コントロールのユーザー操作」を参照してください。

PopUpMenuButton コントロール

PopUpMenuButton は、小さなボタンで Menu コントロールをポップアップさせる PopUpButton コントロールです。ユーザーがポップアップメニューからアイテムを選択すると、PopUpButton のメインボタンの表示が、選択したメニューアイテムのアイコンとラベルに変わります。Menu コントロールや MenuBar コントロールとは異なり、PopUpMenuButton は単一レベルのメニューのみをサポートします。

詳細については、『Adobe Flex 2 リファレンスガイド』の PopUpMenuButton を参照してください。Menu コントロールの詳細については、389 ページの「Menu コントロールのイベントの処理」を参照してください。PopUpButton コントロールの詳細については、256 ページの「PopUpButton コントロール」を参照してください。

PopUpMenuButton コントロールについて

次の例は、ポップアップボタンをクリックする前とクリックした後の PopUpMenuButton コントロールです。

PopUpMenuButton は以下のように動作します。

- 小さいボタン (デフォルトでは v アイコンで表示されます) をクリックすると、ボタンの下にポップアップメニューが表示されます。
- ポップアップメニューからアイテムを選択すると、PopUpMenuButton のメインボタンのラベルが選択したアイテムのラベルに変わり、PopUpMenuButton コントロールによって `MenuEvent.CHANGE` イベントが送出されます。
- メインボタンをクリックすると、PopUpMenuButton コントロールによって `MenuEvent.CHANGE` イベントと `MouseEvent.ITEM_CLICK` イベントが送出されます。

PopUpMenuButton イベントの処理の詳細については、397 ページの「PopUpMenuButton コントロールのイベント処理」を参照してください。

PopUpMenuButton コントロールを使用すると、ポップアップメニューからアイテムを選択することによって、メインボタンの機能を変更することができます。最後に選択したアイテムが、メインボタンのアイテムになります。

ユーザー操作が多数あり、同一のオプションが頻繁に選択される傾向があるものの、アプリケーションデベロッパーはどのオプションをデフォルトにすべきか推測できない場合に、このビヘイビアは役立ちます。テキストエディタのコントロールバーでは、間隔などのオプションにこうしたコントロールが頻繁に使用されています。間隔などのオプションには、頻繁に使用される設定があるものの、デベロッパーは事前にその値を判別できないためです。たとえば Microsoft Word では、線の間隔や境界線、テキストやハイライトの色などの指定に、こうしたコントロールを使用しています。

PopUpButton コントロールを使用すると、PopUpMenuButton とはビヘイビアの異なるポップアップメニューボタンを作成できます。たとえば、ユーザーがメニューアイテムを選択したときに、メインボタンのデフォルトアクションを変更しないボタンを作成することができます。詳細については、256 ページの「PopUpButton コントロール」を参照してください。

PopUpMenuButton コントロールには、次のようなサイズ設定の特性があります。

プロパティ	デフォルト値
デフォルトサイズ	メインボタンのラベルとアイコン、およびポップアップボタンのアイコンを表示するのに十分なサイズです。このコントロールは、メニューの領域を確保しません。
最小サイズ	0
最大サイズ	10000 x 10000

PopUpMenuButton コントロールの作成

MXML では PopUpMenuButton コントロールを `<mx:PopUpMenuButton>` タグで定義します。MXML アプリケーションの他の場所 (他のタグまたは ActionScript ブロック) にあるコンポーネントを参照する場合は、`id` の値を指定します。

PopUpMenuButton コントロールのデータは、`dataProvider` プロパティで指定します。構造や内容などの有効なデータプロバイダについては、382 ページの「メニューの構造とデータの定義」を参照してください。

デフォルトでは、最初に選択されたアイテムが、ポップアップメニューの dataProvider の最初のアイテムになります。またメインボタンのデフォルトラベルが、アイテムのラベルになります。デフォルトラベルは `labelField` プロパティまたは `labelFunction` プロパティで決まります。メインボタンの最初のラベルに特定のアイテムのラベルと機能を設定するには、次のように、PopUpMenuButton コントロールの `creationComplete` イベントのリスナーを作成して、Menu サブコントロールの `selectedIndex` プロパティを設定します。

```
Menu(MyPopUpControl.popUp).selectedIndex=2;
```

PopUpMenuButton コントロールの `popUp` プロパティのタイプは Menu ではなく IUIComponent になっているため、このプロパティを Menu にキャストする必要があります。

PopUpMenuButton の label プロパティを使用して、メインボタンのラベルを設定することもできます。詳細については、408 ページの「label プロパティの使用」を参照してください。

ポップアップメニューが閉じると、選択内容や関連プロパティは失われます。

メモ

データプロバイダからメインボタンのラベルを設定する場合は、PopUpMenuButton の initialize イベントではなく、creationComplete イベントを使用してください。

例 : PopUpMenuButton コントロールの作成

次の例では、E4X 形式の XML データプロバイダを使用して PopUpMenuButton コントロールを作成します。

```
<?xml version="1.0"?>
<!-- menus/PopUpMenuButtonControl.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Menu

            // The initData function sets the initial value of the button
            // label by setting the Menu subcontrol's selectedIndex property.
            // You must cast the popUp property to a Menu.
            private function initData():void {
                Menu(pb2.popUp).selectedIndex=2;
            }
        ]]>
    </mx:Script>

    <mx:XML format="e4x" id="dp2">
        <root>
            <editItem label="Cut"/>
            <editItem label="Copy"/>
            <editItem label="Paste"/>
            <separator type="separator"/>
            <editItem label="Delete"/>
        </root>
    </mx:XML>

    <mx:PopUpMenuButton id="pb2"
        dataProvider="{dp2}"
        labelField="@label"
        showRoot="false"
        creationComplete="initData();"/>
</mx:Application>
```

この例では E4X 形式の XML データソースを使用しているため、E4X @ 属性指定子シンタックスでラベルフィールドを指定し、データプロバイダのルートノードを非表示にするようコントロールに指示する必要があります。

label プロパティの使用

PopUpMenuButton コントロールの label プロパティを使用すると、メインボタンのラベルの内容を指定できます。また、labelField プロパティまたは labelFunction プロパティで決められた、ポップアップメニューのラベルを上書きすることもできます。固定部分と可変部分のあるメインボタンのラベルを作成する場合に、label プロパティは役立ちます。たとえば、メールの "Send to :" ボタンの宛先部分のテキストのみをポップアップメニューで制御するようにすると、メインボタンのラベルは、メニューからどちらが選択されたかによって決まるため、"Send to : Inbox" または "Send to : Trash" になります。

動的な label プロパティを使用するには、PopUpMenuButton コントロールの change イベントリスナーを使用して、イベントの label プロパティを基にラベルを設定します。次に例を示します。

```
<?xml version="1.0"?>
<!-- menus/PopUpMenuButtonLabel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="600" width="600">

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

            public function itemClickHandler(event:MenuEvent):void {
                event.currentTarget.label= "Send to: " + event.label;
            }

            [Bindable]
            public var menuData:Array = [
                {label: "Inbox", data: "inbox"},
                {label: "Calendar", data: "calendar"},
                {label: "Sent", data: "sent"},
                {label: "Deleted Items", data: "deleted"},
                {label: "Spam", data: "spam"}
            ];
        ]]>
    </mx:Script>

    <mx:PopUpMenuButton id="p1"
        showRoot="true"
        dataProvider="{menuData}"
        label="Send to: Inbox"
        itemClick="itemClickHandler(event);"/>
</mx:Application>
```

PopUpMenuButton のユーザー操作

PopUpMenuButton　コントロールのメインボタンとポップアップボタンのユーザー操作は、PopUpButton コントロールの場合と同じです。ポップアップメニューのユーザー操作は Menu コントロールと同じです。PopUpButton のユーザー操作の詳細については、258 ページの「ユーザーの操作」を参照してください。Menu コントロールのユーザー操作の詳細については、402 ページの「Menu コントロールのユーザー操作」を参照してください。

第 12 章

データ駆動型コントロールの使用

12

Adobe Flex のコントロールのいくつかは、データプロバイダ (データを含むオブジェクト) から入力データを取得します。たとえば、Tree コントロールはデータプロバイダからデータを読み取ることによって、ツリー構造と各ツリーノードに割り当てられたデータを定義します。

このトピックでは、複雑なデータの視覚化を可能にするコントロールに焦点を当てながら、データプロバイダを使用するいくつかのコントロールについて説明します。データプロバイダを使用してこれらのコントロールの内容を設定するためのさまざまな方法の例も紹介します。また次の各トピックでは、データプロバイダおよびデータプロバイダを使用するコントロールについても説明します。

- 151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」では、データプロバイダの詳細、およびコレクションをデータプロバイダとして使用する方法を紹介します。
- 381 ページ、第 11 章の「メニューベースのコントロールの使用」では、Menu、MenuBar、および PopUpMenuButton の各コントロールについて説明します。
- 583 ページ、第 16 章の「ナビゲータコンテナの使用」では、TabNavigator や Accordion のような、データプロバイダを使用して構造体を作成するナビゲータコンテナについて説明します。
- 1443 ページ、第 53 章の「チャートの概要」ではチャートコントロールの使用方法を紹介します。

目次

List コントロール

List コントロールでは、アイテムを垂直方向に並べて表示します。List コントロールの機能は、HTML のフォームエレメントである SELECT に似ています。多くの場合、リストのすべてのアイテムにアクセスできるように垂直スクロールバーが表示されます。オプションの水平スクロールバーは、リストアイテムが水平方向に収まらない場合に、ユーザーがアイテムを表示するために使用します。ユーザーはリストからアイテムを選択できます。複数選択も可能です。

メモ

HorizontalList、TileList、DataGrid、Menu、および Tree の各コントロールは、List コントロールまたはその直接の親である ListBase クラスから派生します。このため、List コントロールの情報の大部分がこれらのコントロールに適用されます。

詳細については、『Adobe Flex 2 リファレンスガイド』の List を参照してください。

次の図は、List コントロールです。

List コントロールのサイズ設定

List コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	幅は表示される最初の 7 アイテム (アイテムが 7 つ以下の場合はリスト内の表示されるすべてのアイテム) 中の最も幅広いラベルが収まるだけの幅、高さは 1 行 20 ピクセルとして 7 行分の高さ
最小サイズ	0
最大サイズ	5000 x 5000

horizontalScrollPolicy="on" に指定した場合、List コントロールのデフォルトの幅は変化しませんが、それでも表示される最も幅広なラベルが収まるだけの幅はあります。

horizontalScrollPolicy="on" に設定して、List コントロールのピクセル幅を指定した場合は、次の例のように、measureWidthOfItems() メソッドを使用してスクロールバーの右端の位置が内容の右端と合うことを確認できます。5 ピクセルを追加することで、テキストの右端の文字が正しく表示されるようになります。

```
<mx:List width="200" id="li2" horizontalScrollPolicy="on"
  maxHorizontalScrollPosition="{li2.measureWidthOfItems() - li2.width +
  5}">
```

上記の行では、スクロールバーの右端の位置が、List コントロールの右端近くにある測定された最も長いリストアイテムの末尾になるようにしています。ただしこの方法を使用すると、アプリケーションの効率性が下がる場合があるため、明示的にサイズを設定したほうがよいこともあります。

リスト、および ListBase クラスのすべてのサブクラスは、スタイルまたはデータプロバイダが変化した際にサイズが決まります。

width プロパティの値を最も長いラベルの幅よりも小さくなるように設定して、horizontalScrollPolicy="off" に指定した場合、コントロールの幅を超えるラベルは切り取られます。

List コントロールの作成

List コントロールの定義には <mx:List> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

List コントロールではリストデータプロバイダを使用します。詳細については、152 ページの「データプロバイダについて」を参照してください。

List コントロールのデータを指定するには、コントロールの dataProvider プロパティを使用します。ただし dataProvider は List コントロールのデフォルトのプロパティであるため、<mx:List> タグの <mx:dataProvider> 子タグを指定する必要はありません。静的な List コントロールを作成する最も簡単な方法では、単に <mx:String> タグをコントロールの本体内部に配置します。これは、Flex では次の例のように、複数のタグが自動的にストリングの配列として解釈されるからです。

```
<?xml version="1.0"?>
<!-- dpcontrols/ListDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:List>
        <mx:dataProvider>
            <mx:String>AK</mx:String>
            <mx:String>AL</mx:String>
            <mx:String>AR</mx:String>
        </mx:dataProvider>
    </mx:List>
</mx:Application>
```

List コントロールのアイテムのインデックスは 0 から始まります。つまり、値は 0、1、2、...、n - 1 となります。n はアイテムの総数です。アイテムの値はラベルテキストです。

通常、List コントロールに対するユーザーの操作はイベントを使用して処理します。次のコード例では、List コントロールに change イベントのハンドラを追加します。Flex は、ユーザーの操作によってコントロールの値が変化したときに mx.ListEvent.CHANGE イベントをブロードキャストします。

```
<?xml version="1.0"?>
<!-- dpcontrols/ListChangeEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:Script>
        <![CDATA[
            import flash.events.Event;

            public function changeEvt(event:Event):void {
                forChange.text=event.currentTarget.selectedItem.label + " " +
                event.currentTarget.selectedIndex;
            }
        ]]>
    </mx:Script>

    <mx:List width="35" change="changeEvt(event)">
        <mx:Object label="AL" data="Montgomery"/>
        <mx:Object label="AK" data="Juneau"/>
        <mx:Object label="AR" data="Little Rock"/>
    </mx:List>
    <mx:TextArea id="forChange" width="150"/>
</mx:Application>
```

この例では、List コントロールの selectedItem プロパティおよび selectedIndex プロパティをイベントハンドラで使用しています。change イベントが発生すると、コントロールで選択されたアイテムのラベルとアイテムのインデックスで TextArea コントロールが更新されます。

イベントハンドラに渡されるオブジェクトの target プロパティには、List コントロールへの参照が設定されています。イベントの currentTarget プロパティを使用すると、コントロールのすべてのプロパティを参照できます。currentTarget.selectedItem フィールドには選択されたアイテムのコピーが設定されます。ストリングの配列を使用して List コントロールを作成した場合、currentTarget.selectedItem フィールドにはストリングが設定されます。オブジェクトの配列を使用して作成した場合は、currentTarget.selectedItem フィールドには選択されたアイテムに対応するオブジェクトが設定されます。そのためこの場合、currentTarget.selectedItem.label は選択されたアイテムのラベルフィールドを参照します。

ラベル関数の使用

List コントロールにラベル関数を渡して、コントロールに表示するテキストを決定するロジックを使用できます。ラベル関数には次のシグネチャが必要です。

labelFunction(*item*:Object):String

Label コントロールによって渡された item パラメータには、リストアイテムオブジェクトが含まれます。関数は List コントロールに表示されるストリングを返す必要があります。

メモ

ListBase の大部分のサブクラスではまた、上述のシグネチャを持つ labelFunction プロパティが使用されます。DataGrid および DataGridColumn コントロールの場合、メソッドシグネチャは labelFunction(item:Object, dataField:DataGridColumn):String です。
item には DataGrid アイテムオブジェクトが入り、dataField は DataGrid 列を指定します。

次の例では、List コントロールにアイテムを表示するために、label フィールドおよび data フィールドの値を組み合わせる関数を使用しています。

```
<?xml version="1.0"?>
<!-- dpcontrols/ListLabelFunction.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
    <mx:Script><![CDATA[
        public function myLabelFunc(item:Object):String {
            return item.data + ", " + item.label;
        }
    ]]></mx:Script>

    <mx:ArrayCollection id="myDP">
        <mx:source>
            <mx:Object label="AL" data="Montgomery"/>
            <mx:Object label="AK" data="Juneau"/>
            <mx:Object label="AR" data="Little Rock"/>
        </mx:source>
    </mx:ArrayCollection>

    <mx:List dataProvider="{myDP}" labelFunction="myLabelFunc"/>
</mx:Application>
```

この例を使用すると、次のような List コントロールが作成されます。

メモ

この例では、ArrayCollection オブジェクトをデータプロバイダとして使用しています。データが動的に変更される可能性がある場合は、コレクションをデータプロバイダとして使用する必要があります。詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

データヒントの表示

データヒントはツールヒントに似ていて、List コントロール内の行の上にマウスポインタを移動させるとテキストが表示されます。List コントロール内のテキストで長さがコントロールの幅を超えるものは、右側が切り取られるか、コントロールにスクロールバーがある場合はスクロールして表示することになります。データヒントを使用すると、マウスポインタをセルに移動させたときに、切り取られたテキストを含むテキスト全体が表示されるので、この問題を解決できます。データヒントは有効にすると、データが切り取られたフィールドにのみ表示されます。データヒントを表示するには、List コントロールの showDataTips プロパティを true に設定します。

メモ

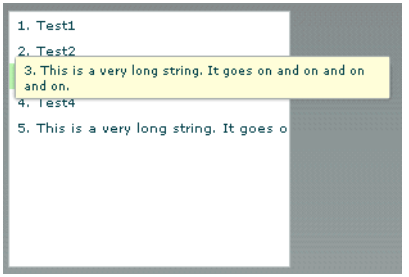
DataGrid コントロールでデータヒントを使用するには、DataGrid の各 DataGridColumn で showDataTips プロパティを設定してください。

showDataTips プロパティのデフォルトの動作では、ラベルのテキストが表示されます。ただし、dataTipField プロパティおよび dataTipFunction プロパティを使用して、データヒントの表示内容を変更することもできます。dataTipField プロパティの機能は labelField プロパティに似ています。このプロパティにはデータプロバイダのフィールド名を指定して、そのフィールドを列のセルのデータヒントとして使用します。dataTipFunction プロパティの機能は labelFunction プロパティに似ています。このプロパティには、リストアイテムに対して表示するデータヒントのストリングを指定します。

次の例では、List コントロールの showDataTips プロパティを設定しています。

```
<mx:List id="myList" dataProvider="{myDP}" width="220" height="200"
   showDataTips="true"/>
```

この例を使用すると、次のような List コントロールが作成されます。



スクロールヒントの表示

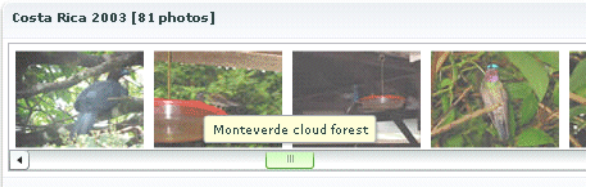
スクロールヒントは、ユーザーがリストをスクロールするのに合わせて、リストのどのアイテムを表示しているかという情報をユーザーに示すために使用します。このヒントはユーザーがスクロールしたときにのみ表示され、スクロールバーにマウスを合わせただけでは表示されません。スクロールヒントは、ライブスクロールが無効になっている (liveScrolling プロパティが false に設定されている) 場合に、スクロールサムを離すまでスクロールが発生しないようにできるため便利です。showScrollTips プロパティのデフォルト値は false です。

showScrollTips プロパティを設定すると、デフォルトでは、一番上の表示アイテムのインデックス番号が表示されます。scrollTipFunction プロパティを使用すると、スクロールヒントに表示する内容を指定できます。scrollTipFunction プロパティは、labelFunction プロパティと同じように動作します。このプロパティには、リストアイテムに対して表示するスクロールヒントのストリングを指定する関数を設定します。ただし、スクロールヒントの内容を取得するためにサーバーにアクセスすることは望ましくありません。

次の例では、HorizontalList コントロールの showScrollTips プロパティおよび scrollTipFunction プロパティを設定します。scrollTipFunction プロパティには、現在表示されているリストアイテムの description プロパティの値を取得する関数を指定しています。

```
<mx:HorizontalList id="list" dataProvider="{album.photo}" width="100%"
  itemRenderer="Thumbnail" columnWidth="108" height="100"
  selectionColor="#FFCC00" liveScrolling="false" showScrollTips="true"
  scrollTipFunction="scrollTipFunc"
  change="currentPhoto=album.photo[list.selectedIndex]"/>
```

このコードでは、次の HorizontalList コントロールが作成されます。



List コントロール行のテキストの垂直方向の整列

verticalAlign スタイルを使用すると、List のテキストの表示位置を行の垂直方向の上揃え、中央揃え、下揃えのいずれかに設定できます。デフォルト値は top です。値として、middle または bottom も設定できます。

次の例では、List コントロールの verticalAlign プロパティを bottom に設定します。

```
<mx:List id="myList" dataProvider="{myDP}" width="220" height="200"
  verticalAlign="bottom"/>
```

行の高さの調整と List のテキストの折り返し

variableRowHeight プロパティを使用すると、List コントロールの行の高さを内容に合わせて変えることができます。デフォルト値は false です。variableRowHeight プロパティを true に設定すると、rowHeight プロパティの設定が無効になり、rowCount プロパティは読み取り専用になります。

次の例では、List コントロールの variableRowHeight プロパティを true に設定します。

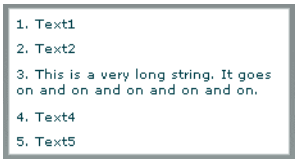
```
<mx:List id="myList" dataProvider="{myDP}" width="220" height="200"
  variableRowHeight="true"/>
```

wordWrap プロパティを variableRowHeight プロパティと組み合わせて使用すると、List の行の幅よりテキストが長い場合にテキストを折り返して複数行で表示できます。

次の例では、wordWrap プロパティおよび variableRowHeight プロパティを true に設定します。

```
<mx:List id="myList" dataProvider="{myDP}" width="220" height="200"
  variableRowHeight="true" wordWrap="true"/>
```

このコードでは、次の List コントロールが作成されます。



カスタムアイテムレンダラーの使用

アイテムレンダラーは、List コントロールのデータアイテムを表示するオブジェクトです。カスタムアイテムレンダラーを使用する最も簡単な方法は、itemRenderer プロパティの値として MXML コンポーネントを指定することです。アイテムレンダラーとして MXML コンポーネントを使用すると、複数のレベルのコンテナおよびコントロールを含めることができます。カスタムアイテムレンダラーとして ActionScript クラスを使用することもできます。カスタムアイテムレンダラーの詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

次の例は itemRenderer プロパティを FancyCellRenderer という MXML コンポーネントに設定しています。また、MXML コンポーネントの高さがデフォルトの行の高さより大きくなるので、variableRowHeight プロパティを true に設定します。

```
<mx:List id="myList1" dataProvider="{myDP}" width="220" height="200"
  itemRenderer="FancyItemRenderer" variableRowHeight="true"/>
```

List コントロールへのアイコンの指定

次の例のように、各 List アイテムと一緒に表示するアイコンを指定できます。

```
<?xml version="1.0"?>
<!-- dpcontrols/ListIcon.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
         [Embed(source="assets/radioIcon.jpg")]
         public var iconSymbol1:Class;
         [Embed(source="assets/topIcon.jpg")]
         public var iconSymbol2:Class;
      ]]>
   </mx:Script>

   <mx:List iconField="myIcon">
      <mx:dataProvider>
         <mx:Array>
            <mx:Object label="AL" data="Montgomery" myIcon="iconSymbol1"/>
            <mx:Object label="AK" data="Juneau" myIcon="iconSymbol2"/>
            <mx:Object label="AR" data="Little Rock" myIcon="iconSymbol1"/>
         </mx:Array>
      </mx:dataProvider>
```

```
   </mx:List>
</mx:Application>
```

この例では、iconField プロパティを使用して、アイコンを使用する各アイテムのフィールドを指定しています。また、Embed メタデータを使用してアイコンを読み込んでから、List コントロール定義で参照しています。

iconFunction プロパティを使用して、アイコンを決定する関数を指定することもできます。この方法は、labelFunction プロパティを使用してラベルテキストを決定する関数を指定する方法とほぼ同じです。アイコン関数には次のシグネチャが必要です。

iconFunction(*item*:Object):Class

Label コントロールによって渡された item パラメータには、リストアイテムオブジェクトが含まれます。関数は List コントロールに表示されるアイコンのクラスを返す必要があります。

次の例で表示される List コントロールでは、iconFunction プロパティを使用して、リストの各アイテムに対して表示するアイコンを決定します。

```
<?xml version="1.0"?>
<!-- dpcontrols/ListIconFunction.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
   <mx:Script>
      <![CDATA[
         // Embed icons.
         [Embed(source="assets/radioIcon.jpg")]
         public var pavementSymbol:Class;
         [Embed(source="assets/topIcon.jpg")]
         public var normalSymbol:Class;

         // Define data provider.
         private var myDP: Array;
         private function initList():void {
            myDP = [
               {Artist:'Pavement', Album:'Slanted and Enchanted', Price:11.99},
               {Artist:'Pavarotti', Album:'Twilight', Price:11.99},
               {Artist:'Other', Album:'Other', Price:5.99}];

            list1.dataProvider = myDP;
         }

         // Determine icon based on artist. Pavement gets a special icon.
         private function myiconfunction(item:Object):Class{
            var type:String = item.Artist;
            if (type == "Pavement") {
               return pavementSymbol;
            }
            return normalSymbol;
         }
      ]]>
   </mx:Script>
```

```
   <mx:VBox >
      <mx:List id="list1" initialize="initList()" labelField="Artist"
         iconFunction="myiconfunction" />
   </mx:VBox>
</mx:Application>
```

List コントロールの行の色の切り替え

alternatingItemColors スタイルプロパティを使用すると、List コントロールの各行の色を定義する配列を指定できます。配列には 2 色以上を指定する必要があります。配列で指定した色がすべて使用された後は、1番目の色に戻って同じ配色が繰り返されます。

次の例では、明るい青を表す #66FFFF と灰色がかった青を表す #33CCCC の 2 項目で構成される配列を定義しています。したがって、List コントロールの行は、この 2 色を繰り返し使用して表示されます。3 色の配列を指定すれば、行は 3 色を繰り返し使用することになります。

```
<mx:List alternatingItemColors="[#66FFFF, #33CCCC]".../ >
```

List コントロールのユーザー操作

ユーザーは、個々のリストアイテムを選択するにはアイテムをクリックします。複数のアイテムを選択するには Ctrl キーや Shift キーを押しながらアイテムをクリックします。(複数選択を可能にするには、allowMultipleSelection プロパティを true に設定する必要があります。)

マウスやキーボードを使用してアイテムの選択操作が行われるたびに、change イベントがブロードキャストされます。マウスの操作では、マウスボタンを離すと List コントロールからイベントがブロードキャストされます。

allowDragSelection プロパティを true に設定した場合、ユーザーが 1 つまたは複数の行の上でマウスボタンを押し、そのマウスボタンを押したままマウスをコントロール外へドラッグして、マウスを上下に動かすと、コントロールが上下にスクロール表示されます。

List コントロールでは、一度に表示できるレコード数のアイテムを表示します。10 行の List コントロールに表示されるデータ内で PageDown キーを押すと、0 ～ 10、9 ～ 18、18 ～ 27 番目のレコードが順に表示されます。必ず 1 つのアイテムが前後のページで重複することになります。

List コントロールでは次のキーボード操作ができます。

キー	アクション
↑ (上矢印)	1 つ上のアイテムを選択します。
↓ (下矢印)	1 つ下のアイテムを選択します。
PageUp	1 ページ上のアイテムを選択します。
PageDown	1 ページ下のアイテムを選択します。

キー	アクション
Home	リストの一番上のアイテムを選択します。
End	リストの一番下のアイテムを選択します。
英数字キー	入力された文字ではじまるラベルを持つ次のアイテムにジャンプします。
Ctrl	トグルキーです。隣接しない複数のアイテムを選択または選択解除できます。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。
Shift	連続選択キーです。隣接する複数のアイテムを選択するときに使用します。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。

HorizontalList コントロール

HorizontalList コントロールには、アイテムの横並びのリストが表示されます。HorizontalList コントロールは、カスタムアイテムレンダラーと組み合わせて、イメージなどのデータのリストを表示する場合に特に有効です。カスタムセルレンダラーの詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

詳細については、『Adobe Flex 2 リファレンスガイド』の HorizontalList を参照してください。

HorizontaList コントロールについて

HorizontalList コントロールの表示は、Repeater オブジェクトを使用して HBox コンテナ内にコンポーネントを繰り返し表示するのとよく似た外観になる場合があります。ただし、パフォーマンスの面では、HorizontalList コントロールのほうが HBox コンテナと Repeater オブジェクトの組み合わせよりも優れています。これは、HorizontalList コントロールでは表示領域内のオブジェクトだけをインスタンス化するためです。HorizontalList のスクロールは、Repeater オブジェクトを使用する場合より遅くなります。Repeater オブジェクトの詳細については、903 ページ、第 26 章の「コントロールおよびコンテナの 動的な繰り返し」を参照してください。

HorizontalList コントロールでは、アイテムは常に左から右に表示されます。通常、このコントロールには、リスト内のすべてのアイテムにアクセスするための水平スクロールバーが含まれます。オプションの垂直スクロールバーは、リストアイテムが垂直方向に収まらない場合に、ユーザーがアイテムを表示するために使用します。`allowMultipleSelection` プロパティの値に応じて、ユーザーはリストから単一または複数のアイテムを選択できます。

次の図に HorizontalList コントロールを示します。

Product images courtesy of Lavish

HorizontalList コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	4 列、サイズはセルのサイズによって決定。
最小サイズ	0
最大サイズ	5000 x 5000

詳細については、『Adobe Flex 2 リファレンスガイド』の HorizontalList を参照してください。

HorizontalList コントロールの作成

HorizontalList コントロールを定義するには、<mx:HorizontalList> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

HorizontalList コントロールは多数のプロパティおよびメソッドを List コントロールと共有しています。一部の共有プロパティの使用方法については、412 ページの「List コントロール」を参照してください。HorizontalList コントロールは、リストベースのデータプロバイダを使用します。詳細については、152 ページの「データプロバイダについて」を参照してください。

HorizontalList コントロールのデータは、次の例のように、<mx:HorizontalList> タグの dataProvider プロパティを使用して指定します。

```
<?xml version="1.0"?>
<!-- dpcontrols/HListDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="450">
   <mx:Script>
      <![CDATA[
         import mx.collections.*;
         import mx.controls.Image;

         private var catalog:ArrayCollection;
         private static var cat:Array = [
            "assets/usbfan.jpg", "assets/usbwatch.jpg",
            "assets/007camera.jpg", "assets/radiowatch.jpg"
         ];
```

```
        // Initialize the HorizontalList control by setting its dataProvider
        // property to an ArrayCollection containing the items parameter.
        private function initCatalog(items:Array):void
        {
           catalog = new ArrayCollection(items);
           myList.dataProvider = catalog;
        }
     ]]>
  </mx:Script>

  <!-- A four-column HorizontalList.
      The itemRenderer is a Flex Image control.
      When the control is created, pass the cat array to the
      initialization routine. -->
  <mx:HorizontalList id="myList" columnWidth="100" rowHeight="100"
     columnCount="4" itemRenderer="mx.controls.Image"
     creationComplete="initCatalog(cat)"/>
</mx:Application>
```

この例では、creationComplete イベントを使用してデータプロバイダにイメージファイルの ArrayCollection を取り込み、itemRenderer プロパティを使用して Image コントロールをアイテムレンダラーとして指定しています (割り当てでコントロールの完全なパッケージ名を使用しているのは、このコードが mx.controls パッケージを読み込まないためです)。こうすると、データプロバイダによって指定された 4 つのイメージが HorizontalList コントロールに表示されます。

次の例では、Flex 2 に含まれる Flex Explorer アプリケーションを使用しています。

"HorizontalListDemo.mxml" の例では、HorizontalList コントロールに製品のイメージのカタログを表示しています。HorizontalList コントロールのアイテムレンダラーは、Thumbnail という名前の MXML コンポーネントです。

```
<?xml version="1.0"?>
<!-- dpcontrols/HorizontalListDemo.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundAlpha="0" creationComplete="srv.send()">

   <mx:Script>
      <![CDATA[
         import mx.utils.ArrayUtil;
      ]]>
   </mx:Script>


   <mx:HTTPService id="srv" url="assets/catalog.xml" useProxy="false"/>

   <mx:ArrayCollection id="catalogAC"
      source="{mx.utils.ArrayUtil.toArray(srv.lastResult.catalog.product)}"/>

   <mx:HorizontalList dataProvider="{catalogAC}"
```

```
      width="100%" itemRenderer="Thumbnail"/>
</mx:Application>
```

次の例に、製品カタログアプリケーションのアイテムレンダラーとして使用される MXML コンポーネント "Thumbnail.mxml" を示します。この例では、アイテムレンダラーを定義して、1 つの Image コントロールと 2 つの Label コントロールという計 3 つのコントロールを設定します。これらのコントロールは、アイテムレンダラーに渡される data オブジェクトを調べて、表示する内容を決定します。カスタムセルレンダラーの詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

```
<?xml version="1.0"?>
<!-- dpcontrols/Thumbnail.mxml -->
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
      width="165" height="120" verticalAlign="middle" verticalGap="0"
      verticalScrollPolicy="off">
   <mx:CurrencyFormatter id="cf"/>
   <mx:Image id="img" height="100" width="50" source="../{data.image}"/>
   <mx:VBox width="100%" paddingTop="0" horizontalGap="4">
      <mx:Label text="{data.name}" fontWeight="bold"/>
      <mx:Label text="{cf.format(data.price)}" fontWeight="bold"/>
   </mx:VBox>
</mx:HBox>
```

HorizontalList コントロールのユーザー操作

ユーザーは、個々のリストアイテムを選択するにはアイテムをクリックします。複数のアイテムを選択するには Ctrl キーや Shift キーを押しながらアイテムをクリックします。

マウスやキーボードを使用してアイテムの選択操作が行われるたびに、change イベントがブロードキャストされます。マウスの操作では、マウスボタンを離すと HorizontalList コントロールからイベントがブロードキャストされます。アイテムの上からコントロールの外にマウスがドラッグされると、コントロールが上または下にスクロールします。

HorizontalList コントロールでは、一度に表示できるレコード数のアイテムを表示します。4 つのリストをページングすると、0 ～ 4 番目、5 ～ 8 番目というようにレコードが表示され、戦後のページで表示されるレコードが重複することはありません。

キーボード操作

HorizontalList コントロールでは次のキーボード操作ができます。

キー	アクション
PageUp	1ページ左のアイテムを選択します。
← (左矢印)	1つ左のアイテムを選択します。
↓ (下矢印)	1つ右のアイテムを選択します。
PageDown	1ページ右のアイテムを選択します。
Home	リストの先頭のアイテムを選択します。
End	リストの末尾のアイテムを選択します。
Ctrl	トグルキーです。`allowMultipleSelection` プロパティが `true` に設定されている場合に、隣接しない複数のアイテムを選択または選択解除できます。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。
Shift	連続選択キーです。`allowMultipleSelection` が `true` に設定されている場合に、隣接する複数のアイテムを選択できます。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。

TileList コントロール

TileList コントロールでは、アイテムをタイル状に並べて表示します。アイテムは垂直列または水平行の向きで順に配置されます。TileList コントロールは、カスタムアイテムレンダラーと組み合わせて、イメージなどのデータのリストを表示する場合に特に有効です。TileList コントロールのデフォルトのアイテムレンダラーは TileListItemRenderer であり、データプロバイダーのラベルフィールドのテキストおよびすべてのアイコンがデフォルトで表示されます。カスタムセルレンダラーの詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

詳細については、『Adobe Flex 2 リファレンスガイド』の TileList を参照してください。

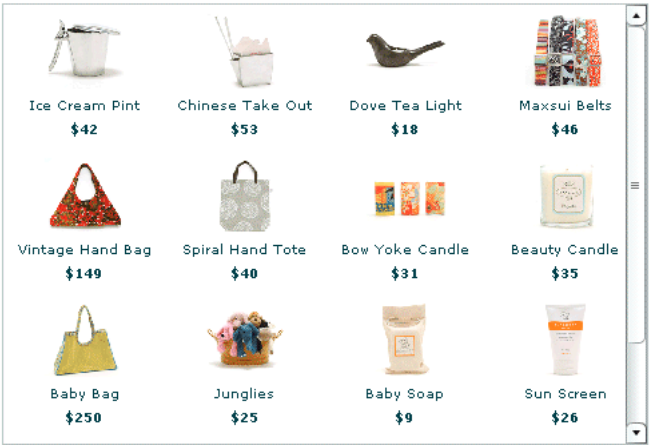
TileList コントロールについて

TileList コントロールの表示は、Repeater オブジェクトを使用して Tile コンテナ内にコンポーネントを繰り返し表示するのとよく似た外観になる場合があります。ただし、パフォーマンスの面では、TileList コントロールのほうが Tile コンテナと Repeater オブジェクトの組み合わせよりも優れています。これは、TileList コントロールでは表示領域内のオブジェクトだけをインスタンス化するためです。TileList のスクロールは、Repeater オブジェクトを使用する場合より遅くなります。Repeater オブジェクトの詳細については、903 ページ、第 26 章の「コントロールおよびコンテナの 動的な繰り返し」を参照してください。

TileList コントロールには、複数のアイテムが同じサイズで並べて表示されます。通常は、リスト内のすべてのアイテムにアクセスできるように、リストの向きに応じて縦、横いずれか一方のスクロールバーを含みます。`multipleSelection` プロパティの値に応じて、ユーザーはリストから単一または複数のアイテムを選択できます。

TileList コントロールでは、子を 1 つまたは複数の垂直列または水平行に配置し、必要に応じて新しい行や列を開始します。`direction` プロパティによってレイアウトの主な方向が決まります。`direction` プロパティに指定できる値は、1 つのレイアウトにつき `horizontal` (デフォルト) または `vertical` のいずれかです。`horizontal` レイアウトでは、タイルは行ごとに埋められ、各行がコントロール内の空きスペースを埋めていきます。表示領域に収まりきらない数のタイルがある場合、水平のコントロールには垂直スクロールバーが作成されます。`vertical` レイアウトでは、タイルは垂直の空きスペース内で列ごとに埋められ、コントロールには必要に応じて水平スクロールバーが作成されます。

次の図に TileList コントロールの例を示します。



TileList コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	4 列、4 行。デフォルトのアイテムレンダラーを使用する場合、各セルは 50 x 50 ピクセル、全体のサイズは 200 x 200 ピクセルです。
最小サイズ	0
最大サイズ	5000 x 5000

詳細については、『Adobe Flex 2 リファレンスガイド』の TileList を参照してください。

TileList コントロールの作成

TileList コントロールを定義するには、<mx:TileList> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

TileList コントロールは多数のプロパティおよびメソッドを List コントロールと共有しています。一部の共有プロパティの使用方法については、412 ページの「List コントロール」を参照してください。TileList コントロールは、リストベースのデータプロバイダを使用します。詳細については、152 ページの「データプロバイダについて」を参照してください。

TileList コントロールのデータは、次の例のように、<mx:TileList> タグの dataProvider プロパティを使用して指定します。

```
<?xml version="1.0"?>
<!-- dpcontrols/TileListDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   initialize="initData();" >
   <mx:Script>
   <![CDATA[
      import mx.controls.Button;
      import mx.collections.*;
      private var listArray:Array=[
         {label: "item0", data: 0},{label: "item1", data: 1},
         {label: "item2", data: 2},{label: "item3", data: 3},
         {label: "item4", data: 4},{label: "item5", data: 5},
         {label: "item6", data: 6},{label: "item7", data: 7},
         {label: "item8", data: 8}];
      [Bindable]
      public var TileListdp:ArrayCollection;

      private function initData():void {
         TileListdp = new ArrayCollection(listArray);
      }
   ]]>
   </mx:Script>

   <mx:TileList dataProvider="{TileListdp}"
```

```
      itemRenderer="mx.controls.Button"/>
</mx:Application>
```

この例では、ラベルとデータ値を定義するストリングの配列を含む ArrayCollection をデータプロバイダに取り込みます。次に itemRenderer プロパティを使用して、Button コントロールをアイテムレンダラーに指定します。Button コントロールにはデータプロバイダのラベル値が表示されます。TileList コントロールには、指定したラベルを持つ 9 つの Button コントロールが表示されます。

次の例は、Flex Explorer サンプルアプリケーションの "TileListDemo.mxml" ファイルです。この例は、製品のイメージのセットを TileList コントロールに表示するカタログアプリケーションの MXML コードです。TileList コントロールのアイテムレンダラーは、Thumbnail という名前の MXML コンポーネントです。この例では、アイテムレンダラーを定義して、1 つの Image コントロールと 2 つの Label コントロールという計 3 つのコントロールを設定します。これらのコントロールは、アイテムレンダラーに渡される data オブジェクトを調べて、表示する内容を決定します。カスタムセルレンダラーの詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

```
<?xml version="1.0"?>
<!-- dpcontrols/TileListDemo.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   backgroundAlpha="0" creationComplete="srv.send()">

   <mx:HTTPService id="srv" url="assets/catalog.xml" useProxy="false"/>

   <mx:TileList dataProvider="{srv.lastResult.catalog.product}"
      height="100%" width="100%" itemRenderer="Thumbnail"
      rowHeight="130" columnWidth="175"/>
</mx:Application>
```

製品カタログアプリケーションのアイテムレンダラーとして使用される MXML コンポーネント "Thumbnail.mxml" は、421 ページの「HorizontalList コントロール」で紹介したものと同一です。カスタムセルレンダラーの詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

TileList コントロールのユーザー操作

ユーザーは、個々のリストアイテムを選択するにはアイテムをクリックします。複数のアイテムを選択するには Ctrl キーや Shift キーを押しながらアイテムをクリックします。

マウスやキーボードを使用してアイテムの選択操作が行われるたびに、change イベントがブロードキャストされます。マウスの操作では、マウスボタンを離すと TileList コントロールからイベントがブロードキャストされます。アイテムの上からコントロールの外にマウスがドラッグされると、コントロールが上または下にスクロールします。

キーボード操作

TileList コントロールでは次のキーボード操作ができます。

キー	アクション
↑ (上矢印)	1つ上のアイテムを選択します。コントロールの向きが垂直で、現在のアイテムが列の一番上にある場合は、前の列の最後のアイテムに移動します。この動作は最初の列の最初のアイテムまで来ると停止します。
↓ (下矢印)	1つ下のアイテムを選択します。コントロールの向きが垂直で、現在のアイテムが列の一番下にある場合は、次の列の最初のアイテムに移動します。この動作は最後の列の最後のアイテムまで来ると停止します。
→ (右矢印)	1つ右のアイテムを選択します。コントロールの向きが水平で、現在のアイテムが行の末尾にある場合は、次の行の最初のアイテムに移動します。この動作は最後の行の最後のアイテムまで来ると停止します。
← (左矢印)	1つ左のアイテムを選択します。コントロールの向きが水平で、現在のアイテムが行の冒頭にある場合は、前の行の最後のアイテムに移動します。この動作は最初の行の最初のアイテムまで来ると停止します。
PageUp	1ページ上のアイテムを選択します。単一ページのコントロールの場合、リストの冒頭にあるアイテムを選択します。
PageDown	1ページ下のアイテムを選択します。単一ページのコントロールの場合、リストの末尾にあるアイテムを選択します。
Home	リストの先頭のアイテムを選択します。
End	リストの末尾のアイテムを選択します。
Ctrl	トグルキーです。`allowMultipleSelection` が `true` に設定されている場合に、隣接しない複数のアイテムを選択または選択解除できます。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。
Shift	連続選択キーです。`allowMultipleSelection` が `true` に設定されている場合に、隣接する複数のアイテムを選択できます。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。

ComboBox コントロール

ComboBox コントロールはドロップダウンリストで、ユーザーは値を1つ選択できます。HTML の SELECT フォームエレメントによく似た機能を果たします。

詳細については、『Adobe Flex 2 リファレンスガイド』の ComboBox を参照してください。

ComboBox コントロールについて

次の図は ComboBox コントロールです。

コントロールが編集可能な状態の場合、ユーザーはリストの最上部に直接テキストを入力できます。また、リストにあらかじめ設定されている値から 1 つを選択することもできます。編集不可能な状態の場合、ユーザーが 1 文字入力するとドロップダウンリストが開き、入力された文字に最も近い値までスクロールされます。マッチングには、ユーザーが入力した 1 文字目だけが使用されます。

ドロップダウンリストを開いたときに、アプリケーションの下側の境界内に収まらない場合は、上方向に開かれます。リストアイテムが長すぎて、表示領域の水平方向内に収まらない場合は、収まる範囲のテキストだけが表示されます。ドロップダウンリストに表示するアイテム数が多い場合は、垂直スクロールバーが表示されます。

ComboBox コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	メインコントロールの表示領域のドロップダウンリストに含まれる最も長い項目、およびドロップダウンボタンを表示するのに十分な幅。ドロップダウンリストが非表示の状態のデフォルトの高さは、ラベルのテキストサイズによって決まります。 デフォルトのドロップダウンリストの高さは 5 行またはドロップダウンリストの項目数の行のいずれか小さい方になります。ドロップダウンリストの各項目のデフォルトの高さは 22 ピクセルです。
最小サイズ	0
最大サイズ	5000 x 5000
`dropdownWidth`	ComboBox コントロールの幅。
`rowCount`	5

ComboBox コントロールの作成

ComboBox コントロールを MXML で定義するには <mx:ComboBox> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

ComboBox コントロールではリストデータプロバイダを使用します。詳細については、152 ページの「データプロバイダについて」を参照してください。

ComboBox コントロールのデータは、<mx:ComboBox> タグの dataProvider プロパティを使用して指定します。データプロバイダは配列か、ICollectionView または IList インターフェイスを実装するクラスである必要があります。多くの場合これは ArrayCollection になります。データプロバイダおよびコレクションの詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

ComboBox コントロールを作成する簡単な方法では、次の例のように、<mx:dataProvider> 子タグを使用してプロパティを指定し、<mx:ArrayCollection> タグを使用して各項目を ArrayCollection として定義します。ArrayCollection のソースはストリングの配列です。

```
<?xml version="1.0"?>
<!-- dpcontrols/ComboBoxSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:ComboBox>
      <mx:ArrayCollection>
         <mx:String>AK</mx:String>
         <mx:String>AL</mx:String>
         <mx:String>AR</mx:String>
      </mx:ArrayCollection>
   </mx:ComboBox>
</mx:Application>
```

この例は、MXML のデフォルトを活用する方法を示しています。dataProvider は ComboBox コントロールのデフォルトのプロパティなので、<mx:dataProvider> タグを使用する必要はありません。同様に、source は ArrayCollection クラスのデフォルトのプロパティなので、<mx:ArrayCollection> タグ内で <mx:source> タグを使用する必要はありません。最後に、ソース配列に <mx:Array> タグを指定する必要もありません。

データプロバイダには、次の例のように、複数のフィールドを持つオブジェクトも含めることができます。

```
<?xml version="1.0"?>
<!-- dpcontrols/ComboBoxMultiple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:ComboBox>
      <mx:ArrayCollection>
         <mx:Object label="AL" data="Montgomery"/>
         <mx:Object label="AK" data="Juneau"/>
         <mx:Object label="AR" data="Little Rock"/>
      </mx:ArrayCollection>
```

```
   </mx:ComboBox>
</mx:Application>
```

最初の例のようにデータソースがストリングの配列である場合、ComboBox にはストリングがドロップダウンリストのアイテムとして表示されます。データソースがオブジェクトで構成されている場合、ComboBox にはデフォルトでラベルフィールドの内容が使用されます。ただしこの動作は、433 ページの「ComboBox のラベルの指定」の説明にあるように、オーバーライドすることが可能です。

ComboBox コントロールのアイテムのインデックスは 0 から始まります。つまり、値は 0、1、2、...、n - 1 となります。n はアイテムの総数です。アイテムの値はラベルテキストです。

ComboBox コントロールでのイベントの使用

通常、ComboBox コントロールに対するユーザーの操作はイベントで処理します。

ComboBox コントロールは、次のようなユーザーの操作によってコントロールの `selectedIndex` プロパティまたは `selectedItem` プロパティの値が変化すると、`change` イベント (flash.events.Event クラス、`type` プロパティの値は flash.events.Event.CHANGE) をブロードキャストします。

- ユーザーが前に選択されていたアイテムと異なるアイテムを選択して、マウスクリック、Enter キー、または Ctrl + 上矢印キーを使用してドロップダウンリストを閉じた場合。
- ドロップダウンリストが閉じた状態で、ユーザーが上矢印、下矢印、PageUp、PageDown のいずれかのキーを押して新しいアイテムを選択した場合。
- ComboBox コントロールが編集可能でユーザーがコントロール内にテキストを入力した場合、コントロールのテキストフィールドが変更されるたびに `change` イベントがブロードキャストされます。

ComboBox コントロールは、開かれた際および閉じた際に、mx.events.DropdownEvent.OPEN 型 (開いたとき) および mx.events.DropdownEvent.CLOSE 型 (閉じたとき) の mx.events.DropdownEvent をブロードキャストします。この各イベントおよびその他の ComboBox イベントの詳細については、『Adobe Flex 2 リファレンスガイド』の ComboBox を参照してください。

次の例では、ComboBox イベントからの情報が表示されます。

```
<?xml version="1.0"?>
<!-- dpcontrols/ComboBoxEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
   <mx:Script>
      <![CDATA[
         import flash.events.Event;
         import mx.events.DropdownEvent;

         // Display the type of event for open and close events.
         private function dropEvt(event:DropdownEvent):void {
            forChange.text+=event.type + "\n";
         }
```

```
        // Display a selected item's label field and index for change events.
        private function changeEvt(event:Event):void {
           forChange.text+=event.currentTarget.selectedItem.label + " " +
              event.currentTarget.selectedIndex + "\n";
        }
     ]]>
  </mx:Script>

  <mx:ComboBox open="dropEvt(event)" close="dropEvt(event)"
        change="changeEvt(event)" >
     <mx:ArrayCollection>
        <mx:Object label="AL" data="Montgomery"/>
        <mx:Object label="AK" data="Juneau"/>
        <mx:Object label="AR" data="Little Rock"/>
     </mx:ArrayCollection>
  </mx:ComboBox>
  <mx:TextArea id="forChange" width="150" height="100%"/>
</mx:Application>
```

ストリングの配列を使用して ComboBox コントロールを作成した場合、currentTarget.selectedItem フィールドにはストリングが設定されます。オブジェクトの配列を使用して作成した場合は、currentTarget.selectedItem フィールドには選択されたアイテムに対応するオブジェクトが設定されます。そのためこの場合、currentTarget.selectedItem.label は選択された Item オブジェクトのラベルフィールドを参照します。

この例では、ComboBox コントロールの selectedItem プロパティおよび selectedIndex プロパティをイベントハンドラで使用しています。change イベントが発生すると、選択されたアイテムのラベルとアイテムのインデックスでコントロール内の TextArea コントロールが更新され、open イベントまたは close イベントが発生すると、イベントタイプが追加されます。

ComboBox のラベルの指定

ComboBox のデータソースがストリングの配列である場合、コントロールには各アイテムのストリングが表示されます。データソースにオブジェクトが含まれる場合、デフォルトでは、ComboBox コントロールのアイテムとして表示されるテキストを定義する label という名前のプロパティが各オブジェクトに設定されている必要があります。オブジェクトに label プロパティがない場合、次の例のように、ComboBox コントロールの labelField プロパティを使用してプロパティ名を指定できます。

```
<mx:ComboBox open="dropEvt(event)" close="dropEvt(event)"
     change="changeEvt(event)" labelField="state">
   <mx:ArrayCollection>
     <mx:Object state="AL" capital="Montgomery"/>
     <mx:Object state="AK" capital="Juneau"/>
     <mx:Object state="AR" capital="Little Rock"/>
   </mx:ArrayCollection>
</mx:ComboBox>
```

セクション 432 ページの「ComboBox コントロールでのイベントの使用」のイベントハンドラで州 ID および州都を表示するには、次の例のように、change イベントハンドラが state というプロパティを使用するように修正します。

```
private function changeEvt(event) {
   forChange.text=event.currentTarget.selectedItem.state + " " +
   event.currentTarget.selectedItem.capital + " " +
   event.currenttarget.selectedIndex;
}
```

また、414 ページの「ラベル関数の使用」の説明にあるように、ラベル関数を使用して ComboBox のラベルを指定することもできます。

変数とモデルを使用した ComboBox コントロールの作成

Flex では、ActionScript の変数定義または Flex のデータモデルから ComboBox のデータプロバイダを作成できます。データプロバイダの各エレメントは、ストリングのラベルを含む必要があり、追加データのフィールドも含むことができます。次の例では、2 つの ComboBox コントロールを作成しています。1 つは ArrayCollection 変数からで、これは配列から直接作成され、もう 1 つは ArrayCollection からで、これは <MX:Model> タグのアイテムの配列から作成されます。

```
<?xml version="1.0"?>
<!-- dpcontrols/ComboBoxVariables.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   initialize="initData();">

   <mx:Script>
      <![CDATA[
         import mx.collections.*
         private var COLOR_ARRAY:Array=
            [{label:"Red", data:"#FF0000"},
            {label:"Green", data:"#00FF00"},
            {label:"Blue", data:"#0000FF"}];
         // Declare an ArrayCollection variable for the colors.
         // Make it Bindable so it can be used in bind
         // expressions ({colorAC}).
         [Bindable]
         public var colorAC:ArrayCollection;

         // Initialize colorAC ArrayCollection variable from the Array.
         // Use an initialize event handler to initialize data variables
         // that do not rely on components, so that the initial values are
         // available when the controls that use them are constructed.
         //See the mx:ArrayCollection tag, below, for a second way to
         //initialize an ArrayCollection.
         private function initData():void {
            colorAC=new ArrayCollection(COLOR_ARRAY);
         }
      ]]>
```

```
    </mx:Script>
    <!-- This example shows two different ways to
         structure a Model. -->
    <mx:Model id="myDP">
       <obj>
          <item label="AL" data="Montgomery"/>
          <item>
             <label>AK</label>
             <data>Juneau</data>
          </item>
          <item>
             <label>AR</label>
             <data>Little Rock</data>
          </item>
       </obj>
    </mx:Model>

    <!-- Create a stateAC ArrayCollection that uses as its source an Array of
          the item elements from the myDP model.
          This technique and the declaration and initialization code used for
          the colorAC variable are alternative methods of creating and
          initializing the ArrayCollection. -->
    <mx:ArrayCollection id="stateAC" source="{myDP.item}"/>

    <mx:ComboBox dataProvider="{colorAC}"/>
    <mx:ComboBox dataProvider="{stateAC}"/>
</mx:Application>
```

この例では、簡単なモデルが使用されています。しかし、外部データソースからのモデルの作成や ActionScript を使用したカスタムデータモデルの定義も可能です。データモデルの使用の詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

リモートデータプロバイダを使用して ComboBox コントロールにデータを提供することもできます。たとえば、Web サービス処理でストリングの配列が返される場合には、次のフォーマットを使用して、ComboBox コントロールの各行にストリングを表示できます。

```
<mx:ArrayCollection id=”resultAC”
   source=”mx.utils.ArrayUtil.toArray(service.operation.lastResult);”
<mx:ComboBox dataProvider="{resultAC}" />
```

リモートデータプロバイダの詳細については、200 ページの「リモートデータプロバイダの使用」を参照してください。

ComboBox コントロールのユーザー操作

ArrayCollection コントロールは編集できる場合とできない場合があります。これはブール型の `editable` プロパティで指定します。編集不可能な ComboBox コントロールでは、ユーザーはドロップダウンリストからアイテムを 1 つ選択できます。編集可能な ComboBox では、コントロールのテキストフィールドが編集できるようになっており、ユーザーはここに直接テキストを入力するか、またはドロップダウンリストからアイテムを選択して内容を設定できます。ユーザーが ComboBox コントロールのリストからアイテムを選択すると、選択したアイテムのラベルが ComboBox コントロールの最上部にあるテキストフィールドにコピーされます。

編集可能な ComboBox コントロール (ドロップダウンボックスではなく) にフォーカスがあるときには、すべてのキーストロークがテキストフィールドに送られ、TextInput コントロールの規則に従って処理されます (367 ページの「TextInput コントロール」を参照)。ただし Ctrl + ↓ (下矢印) キーの組み合わせは例外で、この場合はドロップダウンリストが開きます。ドロップダウンリストが開いたら、上矢印キーおよび下矢印キーでリスト内の移動、Enter キーでリスト内のアイテムの選択ができます。

編集不可能な ComboBox コントロールにフォーカスがあるときには、英数字キーを使用すると、データプロバイダ内で入力キーと先頭文字が一致する次のアイテムを選択し、そのラベルをテキストフィールドに表示することができます。ドロップダウンリストが開いている場合は、選択したアイテムが選択表示されます。

ドロップダウンリストが開いていない場合は、以下のキーを使用して、編集不可能な ComboBox コントロールを制御することもできます。

キー	説明
Ctrl + ↓ (下矢印)	ドロップダウンリストを開き、そのリストにフォーカスを移動します。
↓ (下矢印)	1 つ下のアイテムを選択します。
End	コレクションの一番下のアイテムを選択します。
Home	コレクションの一番上のアイテムを選択します。
PageDown	ドロップダウンリストの一番下にあるアイテムを表示します。現在 `rowCount` 値の倍数番目のアイテムが選択されている場合は、`rowCount` -1 個下にあるアイテムか、または最後のアイテムが表示されます。現在データプロバイダの最後のアイテムが選択されている場合は、何も実行されません。
PageUp	ドロップダウンリストの一番上にあるアイテムを表示します。現在 `rowCount` 値の倍数番目のアイテムが選択されている場合は、`rowCount` -1 個上にあるアイテムか、または最初のアイテムが表示されます。現在データプロバイダの最初のアイテムが選択されている場合は、何も実行されません。
↑ (上矢印)	1 つ上のアイテムを選択します。

編集不可能な ComboBox コントロールのドロップダウンリストにフォーカスがあるときには、英数字キーを使用すると、ドロップダウンリスト内で入力キーと先頭文字が一致する次のアイテムを選択できます。また、ドロップダウンリストが開いているときには、次のキーを使用して制御できます。

キー	説明
Ctrl + ↑ (上矢印)	ドロップダウンリストを閉じて、フォーカスを ComboBox コントロールに戻します。
↓ (下矢印)	1 つ下のアイテムを選択します。
End	コレクションの一番下のアイテムを選択します。
Enter	ドロップダウンリストを閉じて、フォーカスを ComboBox コントロールに戻します。
Esc	ドロップダウンリストを閉じて、フォーカスを ComboBox コントロールに戻します。
Home	コレクションの一番上のアイテムを選択します。
PageDown	表示されるリストの一番下のアイテムに移動します。現在選択されているアイテムがリストの一番下にある場合は、表示されているリストの一番上に移動して、次の `rowCount`-1 番目のアイテム (ある場合) が表示されます。現在データプロバイダの最後のアイテムが選択されている場合は、何も実行されません。
PageUp	表示されるリストの一番上のアイテムに移動します。現在選択されているアイテムがリストの一番上にある場合は、表示されているリストの一番下に移動して、前の `rowCount`-1 番目のアイテム (ある場合) が表示されます。現在データプロバイダの最初のアイテムが選択されている場合は、何も実行されません。
Shift+Tab	ドロップダウンリストを閉じて、DisplayList 内の前のオブジェクトにフォーカスを移動します。
Tab	ドロップダウンリストを閉じて、DisplayList 内の次のオブジェクトにフォーカスを移動します。
↑ (上矢印)	1 つ上のアイテムを選択します。

DataGrid コントロール

DataGrid コントロールは、複数列のデータを表示できるリストです。フォーマットされたデータのテーブルで、編集可能なテーブルセルを設定できます。また、数多くのデータ駆動アプリケーションの基盤となります。

このトピックでは、DataGrid コントロールを作成および使用する方法を紹介します。データのソート方法についても説明します。次のような内容についてはここでは説明しませんが、高度なデータグリッドコントロールを作成する際にはこれらも重要になってきます。

- 各 DataGrid セルの情報をフォーマットする方法、およびユーザーによるセルへのデータ入力を制御する方法。これらのトピックについては、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。
- オブジェクトをデータグリッドの内外にドラッグする方法。このトピックについては、983 ページ、第 29 章の「Drag and Drop Manager の使用」を参照してください。

詳細については、『Adobe Flex 2 リファレンスガイド』の DataGrid を参照してください。

DataGrid コントロールについて

DataGrid コントロールには次の機能があります。

- サイズ変更可能、ソート可能、カスタマイズ可能な列レイアウト (非表示設定可能な列を含む)
- オプションのカスタマイズ可能な列ヘッダーおよび行ヘッダー (オプションで折り返すヘッダーテキストを含む)
- 実行時にユーザーがサイズおよび順序を変更できる列
- 選択イベント
- すべての列でカスタムアイテムレンダラーを使用可能
- データのページ表示をサポート
- スクロール不可のロックされた行および列

次の図は DataGrid コントロールです。

Artist	Album	Price
Pavement	Slanted and Enchanted	11.99
Pavement	Crooked Rain, Crooked Rain	10.99
Pavement	Wowee Zowee	12.99
Pavement	Brighten the Corners	11.99
Pavement	Terror Twilight	11.99
Other	Other	5.99

行には、アイテムが表示されます。各行は、前の行の下にレイアウトされます。列では、制御されている幅、色、サイズなど各表示列の状態が維持されます。

DataGrid コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	列が空の場合、デフォルトの幅は 300 ピクセルです。列に情報が格納されていて明示的に幅が定義されていない場合、デフォルトの幅は 100 ピクセルです。DataGrid の幅は、可能であれば、すべての列が収まる幅に調整されます。デフォルトの表示される行 (ヘッダーを含む) の数は 7、各行のデフォルトの高さは 20 ピクセルです。
最小サイズ	0
最大サイズ	5000 x 5000

DataGrid コントロールの作成

DataGrid コントロールを MXML で定義するには <mx:DataGrid> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

DataGrid コントロールではリストデータプロバイダを使用します。詳細については、152 ページの「データプロバイダについて」を参照してください。

DataGrid コントロールのデータを指定するには、dataProvider プロパティを使用します。データはさまざまな方法で指定できます。DataGrid コントロールを作成する最も簡単な方法では、<mx:ArrayCollection> とともに <mx:dataProvider> プロパティのサブタグを使用し、<mx:Object> タグも使用して、オブジェクトの ArrayCollection として項目を定義します。次の例のように、各オブジェクトで DataGrid コントロールの行を定義し、オブジェクトのプロパティで行内の列項目を定義します。

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:DataGrid>
      <mx:ArrayCollection>
         <mx:Object>
            <mx:Artist>Pavement</mx:Artist>
            <mx:Price>11.99</mx:Price>
            <mx:Album>Slanted and Enchanted</mx:Album>
         </mx:Object>
         <mx:Object>
            <mx:Artist>Pavement</mx:Artist>
            <mx:Album>Brighten the Corners</mx:Album>
            <mx:Price>11.99</mx:Price>
         </mx:Object>
      </mx:ArrayCollection>
   </mx:DataGrid>
</mx:Application>
```

この例は、MXML のデフォルトを活用する方法を示しています。dataProvider は DataGrid コントロールのデフォルトのプロパティなので、<mx:dataProvider> タグを使用する必要はありません。同様に、source は ArrayCollection クラスのデフォルトのプロパティなので、<mx:ArrayCollection> タグ内で <mx:source> タグを使用する必要はありません。最後に、ソース配列に <mx:Array> タグを指定する必要もありません。

しかし、次の例のように、オブジェクトのタグで直接プロパティを使用してオブジェクトを定義することもできます。

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridSimpleAttributes.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:DataGrid>
      <mx:ArrayCollection>
         <mx:Object Artist="Pavement"
            Album="Slanted and Enchanted" Price="11.99" />
         <mx:Object Artist="Pavement"
            Album="Brighten the Corners" Price="11.99" />
      </mx:ArrayCollection>
   </mx:DataGrid>
</mx:Application>
```

DataGrid コントロールに表示される列名は、配列オブジェクトのプロパティ名です。デフォルトでは、列はプロパティ名のアルファベット順に表示されます。異なるオブジェクトには異なる順序でプロパティを定義することができます。プロパティが省略されている配列オブジェクトがある場合、DataGrid コントロールでは、その配列オブジェクトに対応する行に空のセルが表示されます。

列の指定

DataGrid コントロール内の各列は、DataGridColumn オブジェクトで表されます。DataGrid コントロールの columns プロパティと <mx:DataGridColumn> タグを使用して、DataGrid の列の選択、表示列の順序の指定、および他のプロパティの追加設定が可能です。また、441 ページの「列の表示と非表示」の説明にあるように、DataGridColumn クラスの visible プロパティを使用すれば列を非表示にしたり再表示したりできます。

<mx:DataGridColumn> タグの詳細については、『Adobe Flex 2 リファレンスガイド』の DataGridColumn を参照してください。

次の例のように、<mx:DataGrid> タグの子タグ <mx:columns> で配列エレメントを指定します。

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridSpecifyColumns.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
   <mx:DataGrid>
      <mx:ArrayCollection>
         <mx:Object Artist="Pavement" Price="11.99"
            Album="Slanted and Enchanted" />
         <mx:Object Artist="Pavement"
```

```
            Album="Brighten the Corners" Price="11.99" />
      </mx:ArrayCollection>
      <mx:columns>
         <mx:DataGridColumn dataField="Album" />
         <mx:DataGridColumn dataField="Price" />
      </mx:columns>
   </mx:DataGrid>
</mx:Application>
```

この例で DataGrid コントロールに表示される列は、Album および Price だけです。次の例のように、列の順序を変更することもできます。

```
<mx:columns>
  <mx:DataGridColumn dataField="Price" />
  <mx:DataGridColumn dataField="Album" />
</mx:columns>
```

この例では、DataGrid コントロールの 1 列目として Price 列、2 列目として Album 列を指定しています。

<mx:DataGridColumn> タグを使用すると、この他のオプションを設定することもできます。次の例では、headerText プロパティを使用して列の名前をデフォルト名の Album とは異なる値に設定し、width プロパティを使用してアルバム名の列を完全なアルバム名が表示できる幅に設定します。

```
<mx:columns>
  <mx:DataGridColumn dataField="Album" width="200" />
  <mx:DataGridColumn dataField="Price" headerText="List Price" />
</mx:columns>
```

列の表示と非表示

列を場合によって表示したり非表示にしたりするには、DataGridColumn クラスの visible プロパティを指定します。次の例では、ボタンをクリックすることでアルバムの価格を表示したり非表示にしたりできます。

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridVisibleColumn.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
   <mx:DataGrid id="myDG" width="350">
      <mx:dataProvider>
         <mx:ArrayCollection>
            <mx:source>
               <mx:Object Artist="Pavement" Price="11.99"
                  Album="Slanted and Enchanted" />
               <mx:Object Artist="Pavement"
                  Album="Brighten the Corners" Price="11.99" />
            </mx:source>
         </mx:ArrayCollection>
      </mx:dataProvider>
      <mx:columns>
         <mx:DataGridColumn dataField="Artist" />
```

```
         <mx:DataGridColumn dataField="Album" />
         <mx:DataGridColumn id="price" dataField="Price" visible="false"/>
      </mx:columns>
   </mx:DataGrid>

   <!-- The column id property specifies the column to show.-->
   <mx:Button label="Toggle Price Column"
      click="price.visible = !price.visible;" />
</mx:Application>
```

DataGrid コンテナへのデータの受け渡し

Flex では、ActionScript の変数定義または Flex のデータモデルから DataGrid コントロールを作成できます。次の例では、変数を使用して DataGrid コントロールを作成しています。

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridPassData.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   initialize="initData()">
   <mx:Script>
   <![CDATA[
      import mx.collections.*;
      private var DGArray:Array = [
         {Artist:'Pavement', Album:'Slanted and Enchanted', Price:11.99},
         {Artist:'Pavement', Album:'Brighten the Corners', Price:11.99}];

      [Bindable]
      public var initDG:ArrayCollection;
      //Initialize initDG ArrayCollection variable from the Array.
      //You can use this technique to convert an HTTPService,
      //WebService, or RemoteObject result to ArrayCollection.
      public function initData():void {
         initDG=new ArrayCollection(DGArray);
      }
   ]]>
   </mx:Script>

   <mx:DataGrid id="myGrid" width="350" height="200"
      dataProvider="{initDG}" >
      <mx:columns>
         <mx:DataGridColumn dataField="Album" />
         <mx:DataGridColumn dataField="Price" />
      </mx:columns>
   </mx:DataGrid>
</mx:Application>
```

この例では、変数 initDG を <mx:dataProvider> プロパティにバインドしています。また、データバインディングを使用する場合は、列定義イベントを指定することもできます。モデルをデータプロバイダとして使用する方法については、434 ページの「変数とモデルを使用した ComboBox コントロールの作成」を参照してください。

DataGrid コントロールでのイベント処理

DataGrid コントロールおよび DataGridEvent クラスでは、ユーザーの操作に応答する数種類のイベントを定義できます。たとえば Flex では、ユーザーが DataGrid コントロールのアイテムをクリックすると、type プロパティの値が mx.events.ListEvent.ITEM_CLICK ("itemClick") に設定された mx.events.ListEvent クラスのイベントがブロードキャストされます。このイベントは次の例のように処理できます。

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridEvents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
   <![CDATA[
      import mx.events.ListEvent;
      private function itemClickEvent(event:ListEvent):void {
         clickColumn.text=String(event.columnIndex);
         clickRow.text=String(event.rowIndex);
         eventType.text=event.type;
      }
   ]]>
   </mx:Script>
   <mx:DataGrid id="myGrid" width="350" height="150"
         itemClick="itemClickEvent(event);">
      <mx:ArrayCollection>
         <mx:Object Artist="Pavement" Price="11.99"
            Album="Slanted and Enchanted" />
         <mx:Object Artist="Pavement" Album="Brighten the Corners"
            Price="11.99" />
      </mx:ArrayCollection>
   </mx:DataGrid>

   <mx:TextArea id="clickColumn" />
   <mx:TextArea id="clickRow" />
   <mx:TextArea id="eventType" />
</mx:Application>
```

この例では、イベントハンドラを使用して、列のインデックス、行のインデックス、およびイベントタイプを 3 つの TextArea コントロールに表示しています。

DataGrid コントロールの列のインデックスは 0 から始まります。つまり、値は 0、1、2、...、n - 1 となります。n は列の総数です。行のアイテムも 0 から始まるインデックスが付けられます。したがって、2 番目の行の 1 番目のアイテムを選択した場合、この例では列のインデックスを表す 1 つ目の TextArea コントロールに 0、列の中のアイテムインデックスを表す 2 つ目の TextArea コントロールには 1 が表示されます。

イベントハンドラで選択したアイテムにアクセスするには、次の例のコードのように、イベントオブジェクトの currentTarget プロパティ、および DataGrid コントロールの selectedItem プロパティを使用します。

```
var selectedArtist:String=event.currentTarget.selectedItem.Artist;
```

イベントハンドラに渡されるオブジェクトの currentTarget プロパティには、DataGrid コントロールへの参照が設定されています。currentTarget にピリオドとプロパティ名を続けて使用すれば、コントロールのあらゆるプロパティを参照できます。currentTarget.selectedItem フィールドには選択されたアイテムが設定されます。

DataGrid コントロール内でのデータのソート

DataGrid コントロールは、ソートされたデータの表示について 2 種類の方法をサポートしています。

- デフォルトでは、データは基になるデータプロバイダのコレクションにおけるソートされた順序で表示されます。そのため、コレクションの Sort クラスと SortField クラスを使用すれば、行の順序を制御できます。
- デフォルトでは、ユーザーは列ヘッダーをクリックすれば表示順序をソートできます。列ヘッダーをクリックすると、まず選択した列の項目の表示順序が降順にソートされ、ヘッダーをもう一度クリックするとソート順序が逆になります。ソート機能は、DataGrid コントロール全体について、または個々の列について無効にすることができます。

Sort クラスおよび SortField クラスの使用方法の詳細については、167 ページの「表示するデータのソートとフィルタ」を参照してください。

DataGrid の初期ソート順序の決定

DataGrid の最初のソート順序を指定するには、データプロバイダをソートします。さまざまな方法が可能ですが、Flex のコレクションのビルトイン機能を最大限活用できるのは次の方法です。

- DataGrid の dataProvider プロパティで、ArrayCollection のような、ICollectionView インターフェイスを実装するオブジェクトを使用します。データプロバイダオブジェクトの sort フィールドで Sort オブジェクトを指定します。
- Sort オブジェクトを使用して、dataProvider オブジェクト内の行の順序を制御します。

DataGrid 上での初期の複数列ソートを設定する例については、446 ページの「例 : DataGrid の複数列のソート」を参照してください。

ユーザーによる DataGrid の表示内容のソートの制御

ユーザーがデータの表示順序をソートする方法は、DataGrid および DataGridColumn の 3 つのプロパティで制御します。

- DataGrid の `sortableColumns` プロパティは、ユーザーが列ヘッダーをクリックして DataGrid の表示内容のソートすることを可能にするグローバルスイッチです。デフォルト値は `true` です。
- DataGridColumn クラスの `sortable` プロパティは、ユーザーが個々の列をソートできるかどうかを指定します。デフォルト値は `true` です。
- DataGridColumn クラスの `sortCompareFunction` プロパティでは、カスタム比較関数を指定できます。このプロパティは、ユーザーがヘッダーをクリックした際に DataGrid でグリッドのソートに使用される、デフォルトの SortField クラスオブジェクトの `compare` プロパティを設定します。このプロパティを使用すれば、データプロバイダで Sort オブジェクトを明示的に作成することなく、2 つのオブジェクトを比較してどちらがソート順で上になるかを決定する関数を指定できます。比較関数のシグネチャおよび動作の詳細については、『Adobe Flex 2 リファレンスガイド』の sortCompareFunction を参照してください。

デフォルトでは、DataGrid クラスは独自のソートコードを使用して、ユーザーが列をクリックした際のデータのソート方法を制御します。この動作をオーバーライドするには、`headerRelease` イベントハンドラを作成して、ユーザーが列ヘッダーをクリックした際に生成される DataGridEvent クラスのイベントを制御します。このイベントハンドラは以下の内容を実行する必要があります。

1. イベントオブジェクトの `columnIndex` プロパティを使用して、クリックされた列を判断する。
2. クリックされた列、およびソート順の制御が必要とされる他の規則に基づいて、SortField オブジェクトのセットを含む Sort オブジェクトを設定する。
3. Sort オブジェクトをデータプロバイダ ICollectionView に適用する。
4. DataGridEvent クラスのイベントオブジェクトの `preventDefault()` メソッドを呼び出して、DataGrid のデフォルトの列ソートが実行されるのを防止する。

メモ

`labelFunction` プロパティを指定した場合は、`sortCompareFunction` 関数も指定する必要があります。Flex Explorer の " 計算された列 " の例は、この使用方法を示したものです。

次の例は、ユーザーが DataGrid の列ヘッダーをクリックした際に、headerRelease イベントハンドラを使用して複数列のソートを実行する方法を示しています。

例 : DataGrid の複数列のソート

次の例は、コレクションを Sort オブジェクトとともに使用して最初の複数行ソートを決定し、ヘッダーをクリックした際の列のソート方法を制御する方法を示しています。データグリッドは最初、1 番目に在庫状況、2 番目にアーティスト、3 番目にアルバム名という形でソートされます。いずれかのヘッダーをクリックすると、その列が第 1 ソート基準となり、それまでの第 1 基準が第 2 基準に、第 2 基準が第 3 基準になります。

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridSort.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
      initialize="initDP();" width="550" height="400">

   <mx:Script>
      <![CDATA[
         import mx.events.DataGridEvent;
         import mx.collections.*;

         // Declare storage variables and initialize the simple variables.
         // The data provider collection.
         private var myDPColl:ArrayCollection;
         // The Sort object used to sort the collection.
         [Bindable]
         private var sortA:Sort;
         // The sort fields used to determine the sort.
          private var sortByInStock:SortField;
          private var sortByArtist:SortField;
          private var sortByAlbum:SortField;
          private var sortByPrice:SortField;
         // The data source that populates the collection.
         private var myDP:Array = [
            {Artist:'Pavement', Album:'Slanted and Enchanted',
               Price:11.99, InStock: true},
            {Artist:'Pavement', Album:'Crooked Rain, Crooked Rain',
               Price:10.99, InStock: false},
            {Artist:'Pavement', Album:'Wowee Zowee',
               Price:12.99, InStock: true},
            {Artist:'Asphalt', Album:'Brighten the Corners',
               Price:11.99, InStock: false},
            {Artist:'Asphalt', Album:'Terror Twilight',
               Price:11.99, InStock: true},
            {Artist:'Asphalt', Album:'Buildings Meet the Sky',
               Price:14.99, InStock: true},
            {Artist:'Other', Album:'Other', Price:5.99, InStock: true}
         ];

         //Initialize the DataGrid control with sorted data.
         private function initDP():void {
            //Create an ArrayCollection backed by the myDP array of data.
            myDPColl = new ArrayCollection(myDP);
```

```
    //Create a Sort object to sort the ArrrayCollection.
    sortA = new Sort();
    //Initialize SortField objects for all valid sort fields:
    // A true second parameter specifies a case-insensitive sort.
    // A true third parameter specifies descending sort order.
    // A true fourth parameter specifies a numeric sort.
     sortByInStock = new SortField("InStock", true, true);
     sortByArtist = new SortField("Artist", true);
     sortByAlbum = new SortField("Album", true);
     sortByPrice = new SortField("Price", true, false, true);
    // Sort the grid using the InStock, Artist, and Album fields.
    sortA.fields=[sortByInStock, sortByArtist, sortByAlbum];
    myDPColl.sort=sortA;
    // Refresh the collection view to show the sort.
    myDPColl.refresh();
    // Initial display of sort fields
    tSort0.text = "First Sort Field: InStock";
    tSort1.text = "Second Sort Field: Artist";
    tSort2.text = "Third Sort Field: Album";

    // Set the ArrayCollection as the DataGrid data provider.
    myGrid.dataProvider=myDPColl;
    // Set the DataGrid row count to the array length,
    // plus one for the header.
    myGrid.rowCount=myDPColl.length +1;
}

// Re-sort the DataGrid control when the user clicks a header.
private function headRelEvt(event:DataGridEvent):void {
    // The new third priority was the old second priority.
    sortA.fields[2] = sortA.fields[1];
    tSort2.text = "Third Sort Field: " + sortA.fields[2].name;
    // The new second priority was the old first priority.
    sortA.fields[1] = sortA.fields[0];
    tSort1.text = "Second Sort Field: " + sortA.fields[1].name;
    // The clicked column determines the new first priority.
    if (event.columnIndex==0) {
       sortA.fields[0] = sortByArtist;
    } else if (event.columnIndex==1) {
       sortA.fields[0] = sortByAlbum;
    } else if (event.columnIndex==2) {
       sortA.fields[0] = sortByPrice;
    } else {
       sortA.fields[0] = sortByInStock;}
    tSort0.text = "First Sort Field: " + sortA.fields[0].name;
    // Apply the updated sort fields and re-sort.
    myDPColl.sort=sortA;
    // Refresh the collection to show the sort in the grid.
    myDPColl.refresh();
    // Prevent the DataGrid from doing a default column sort.
```

```
            event.preventDefault();
        }
    ]]>
</mx:Script>

<!-- The Data Grid control.
      By default the grid and its columns can be sorted by clicking.
      The headerRelease event handler overrides the default sort
      behavior. -->
<mx:DataGrid id="myGrid" width="100%" headerRelease="headRelEvt(event);">
    <mx:columns>
        <mx:DataGridColumn minWidth="120" dataField="Artist" />
        <mx:DataGridColumn minWidth="200" dataField="Album" />
        <mx:DataGridColumn width="75" dataField="Price" />
        <mx:DataGridColumn width="75" dataField="InStock"
            headerText="In Stock"/>
    </mx:columns>
</mx:DataGrid>
<mx:VBox>
    <mx:Label id="tSort0" text="First Sort Field: "/>
    <mx:Label id="tSort1" text="Second Sort Field: "/>
    <mx:Label id="tSort2" text="Third Sort Field: "/>
</mx:VBox>
</mx:Application>
```

DataGrid コントロールのユーザー操作

DataGrid コントロールは、マウスおよびキーボードの操作に応答します。マウスクリックやキーの押し下げに対する応答は、セルが編集可能かどうかによって異なります。セルが含まれている DataGrid コントロールおよび DataGridColumn の `editable` プロパティが共に `true` に設定されている場合、そのセルは編集可能です。編集可能なセル内をクリックすると、そのセルにフォーカスが移動します。編集不可能なセルをクリックしても、フォーカスは変化しません。

ユーザーは次の方法で、DataGrid コントロールの外観を変更できます。

- `sortableColumns` プロパティの値がデフォルト値である `true` の場合、列ヘッダー内をクリックすると、その列のセルの値に基づいて DataGrid コントロールがソートされます。
- `draggableColumns` プロパティの値がデフォルトの `true` の場合、列ヘッダーの中でマウスをクリックし、マウスボタンを押したまま水平にドラッグしてからマウスボタンを離すと、列が新しい位置に移動します。
- `resizableColumns` プロパティの値がデフォルト値の `true` の場合、列間の領域をクリックすると列のサイズを変更できます。

キーボード操作

DataGrid コントロールでは次のキーボード操作ができます。

キー	アクション
Enter Return Shift+Enter	セルが編集可能な場合に、変更を確定して、同じ列の上下いずれかのセルに編集対象を変更します。上下どちらのセルに移動するかは、Shift が押されているかどうかによって決まります。
Tab	フォーカスを次の編集可能セルに移動します。編集可能かどうかは行方向に向かって調べていきます。最後の行の最後のセルからは、親コンテナ内でフォーカスを取得できる次のエレメントに移動します。
Shift+Tab	フォーカスを前の編集可能セルに移動します。行の先頭でこのキーを押した場合、前の行の末尾に移動します。最初の行の先頭からは、親コンテナ内でフォーカスを取得できる前のエレメントに移動します。
↑ (上矢印) Home PageUp	セルを編集中の場合には、カーソルがセルのテキストの先頭に移動します。セルが編集不可能な場合は、1 つ上のアイテムが選択されます。
↓ (下矢印) End PageDown	セルを編集中の場合には、カーソルがセルのテキストの末尾に移動します。セルが編集不可能な場合は、1 つ下のアイテムが選択されます。
Ctrl	トグルキーです。DataGrid コントロールの `allowMultipleSelection` プロパティが `true` に設定されている場合に、隣接しない複数のアイテムを選択または選択解除できます。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。
Shift	連続選択キーです。DataGrid コントロールの `allowMultipleSelection` プロパティが `true` に設定されている場合に、隣接する複数のアイテムを選択できます。キーの押し下げ、クリックによる選択、およびドラックによる選択の際に、同時に押すことで機能します。

Tree コントロール

Tree コントロールを使用すると、階層データを展開可能なツリー形式で表示できます。

詳細については、『Adobe Flex 2 リファレンスガイド』の Tree を参照してください。階層データプロバイダについては、185 ページの「階層データプロバイダの使用」を参照してください。

このトピックでは、Tree コントロールの作成方法および使用方法について説明します。次のような内容についてはここでは説明しませんが、高度な Tree コントロールを使用する際にはこれらも重要になってきます。

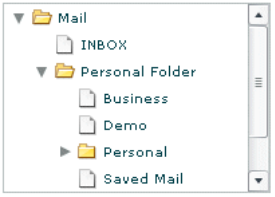
- 各 Tree ノードの情報をフォーマットする方法、およびユーザーによるノードへのデータ入力を制御する方法。これらのトピックについては、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。
- オブジェクトを Tree コントロールの内外にドラッグする方法。このトピックについては、983 ページ、第 29 章の「Drag and Drop Manager の使用」を参照してください。

Tree コントロールについて

Tree コントロールは、「ブランチ」ノードおよび「リーフ」ノードの階層構造になっています。ツリー内の各アイテムはノードと呼ばれ、ノードにはリーフノードとブランチノードがあります。ブランチノードとは、リーフノードまたはブランチノードを含んだノードです。また、空になる (子を持たない) 場合もあります。リーフノードとは、ツリー内の各末端にあるノードです。

デフォルトでは、リーフノードはファイルアイコンの横にテキストラベルが表示され、ブランチノードは展開用矢印とフォルダアイコンの横にテキストラベルが表示されます。展開用矢印をユーザーがクリックすると、ノードが展開されて子が表示されます。

次の図は Tree コントロールです。



Tree コントロールには、次のデフォルトのサイズ変更プロパティがあります。

プロパティ	デフォルト値
デフォルトサイズ	幅は最初に表示される 7 つの行 (折り畳みなし) の中で最も幅の広いノードのアイコン、ラベル、および展開用矢印 (ある場合) が収まるだけの広さ、高さは各行を 20 ピクセルとして 7 行分の高さ。スクロールバーが必要な場合、スクロールバーの幅はこの幅の計算に含まれません。
最小サイズ	0
最大サイズ	5000 x 5000

Tree コントロールの作成

MXML では Tree コントロールを <mx:Tree> タグで定義します。Tree コントロールは List コントロールから派生するコントロールで、List コントロールのすべてのプロパティとメソッドを継承します。List コントロールの使用については、412 ページの「List コントロール」を参照してください。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコントロールを参照する場合は、id 値を指定します。

Tree コントロールは通常、XML などの階層データプロバイダからデータを取得します。Tree が動的に変化するデータを表す場合は、ArrayCollection や XMLListCollection などのような、ICollectionView インターフェイスを実装するオブジェクトを使用します。

Tree コントロールでは、データ記述子を使用してデータプロバイダの内容の解析および操作が実行されます。デフォルトでは、Tree コントロールでは DefaultDataDescriptor クラスの記述子が使用されますが、独自のクラスを作成してそれをメニューコントロールの dataDescriptor プロパティで指定することもできます。

DefaultDataDescriptor クラスでは、以下の種類のデータがサポートされています。

XML　有効な XML テキストを含むストリング、あるいは有効な E4X 形式の XML データを含む <mx:XML> または <mx:XMLList> の各コンパイル時タグか XML オブジェクトまたは XMLList オブジェクトのいずれか。

その他のオブジェクト　アイテムの配列、またはアイテムの配列を含むオブジェクト。ノードの子は children というアイテムに含まれます。

コレクション　ICollectionView インターフェイスを実装するオブジェクト (ArrayCollection クラスや XMLListCollection クラスなど) で、データソースが上記の箇条書きで指定された構造体に準じるもの。DefaultDataDescriptor クラスには、コレクションを効率的に処理するためのコードが含まれます。メニューのデータが動的に変化する場合は、必ずコレクションをデータプロバイダとして使用します。そうでないと、Tree コントロールに古いデータが表示される場合があります。

DefaultDataDescriptor クラスは、メニューのデータプロバイダとして <mx:Model> タグを使用することもサポートしていますが、リーフノードはすべて必ず children という名前である必要があります。原則として、バインディングを使用する Tree データプロバイダが必要な場合は <mx:XML> タグか <mx:XMLList> タグを使用することをお勧めします。

DefaultDataDescriptor でサポートされる形式の詳細など、階層オブジェクトやデータ記述子の詳細については、186 ページの「データ記述子と階層データプロバイダ構造」を参照してください。

次のコードには、449 ページの「Tree コントロール」の図に示したツリーを定義する Tree コントロールが含まれています。ここでは、<mx:XMLList> タグの前後に XMLListCollection ラッパーを使用しています。XMLListCollection を使用すると、MailBox XMLListCollection の内容を変更することで、基になる XML データプロバイダを修正できます。Tree コントロールにはデータの変更が反映されます。この例では、dataProvider が Tree コントロールのデフォルトのプロパティなので、<mx:dataProvider> タグは使用していません。

```
<?xml version="1.0"?>
<!-- dpcontrols/TreeSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Tree id="tree1" labelField="@label" showRoot="true" width="160">
      <mx:XMLListCollection id="MailBox">
         <mx:XMLList>
            <folder label="Mail">
               <folder label="INBOX"/>
               <folder label="Personal Folder">
                  <Pfolder label="Business" />
                  <Pfolder label="Demo" />
                     <Pfolder label="Personal" isBranch="true" />
                     <Pfolder label="Saved Mail" />
                  </folder>
                  <folder label="Sent" />
                  <folder label="Trash" />
            </folder>
         </mx:XMLList>
      </mx:XMLListCollection>
   </mx:Tree>
</mx:Application>
```

XML データでツリーのノードを表すタグの名前は何でも構いません。Tree コントロールは XML を読み取って、ネストされたノード間の関係に応じて階層を表示します。有効な XML 構造体については、185 ページの「階層データプロバイダの使用」を参照してください。

一部のデータプロバイダには、単一の「ルート」ノードというトップレベルがあります。その他のデータプロバイダはノードのリストになっており、ルートノードはありません。場合によっては、ルートノードを Tree のルートとして表示しないほうがよいこともあります。ツリーでルートノードの表示を阻止するには、`showRoot` プロパティに `false;` を指定します。これはデータプロバイダの内容には影響を与えず、Tree の表示方法だけが変わります。`showRoot` プロパティを `false` に指定できるのは、ルートのあるデータプロバイダだけです。つまり、XML およびオブジェクトベースのデータプロバイダのみになります。

ブランチノードは複数の子ノードを持つことができ、デフォルトで展開用矢印とフォルダアイコンが表示されます。ユーザーは展開用矢印を使用して、フォルダを開閉します。リーフノードはデフォルトでファイルアイコンとして表示され、このノードには子ノードを含めることはできません。

Tree コントロールで XML 以外のデータプロバイダのノードを表示するとき、デフォルトでは、ノードの `label` プロパティをテキストラベルとして使用します。ただし E4X XML ベースのデータプロバイダを使用する場合は、ラベルが "label" という属性で指定されていても、ラベルフィールドを指定する必要があります。ラベルフィールドを指定するには、`labelField` プロパティを使用します。たとえばラベルフィールドが label 属性の場合は、`labelField="@label"` を指定します。

Tree コントロールのイベント処理

通常、Tree コントロールに対するユーザーの操作への応答はイベントで処理します。Tree コントロールは List コントロールから派生しているので、List コントロールに対して定義されているすべてのイベントを使用できます。Tree コントロールはまた、Event.change や TreeEvent.itemOpen などを含む、複数の Event および TreeEvent クラスのイベントを送出します。次の例では、change イベントおよび nodeOpen イベントのイベントハンドラを定義しています。

```
<?xml version="1.0"?>
<!-- dpcontrols/TreeEvents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
         import flash.events.*;
         import mx.events.*;
         import mx.controls.*;

         private function changeEvt(event:Event):void {
            var theData:String = ""
            if (event.currentTarget.selectedItem.@data) {
               theData = " Data: " + event.currentTarget.selectedItem.@data;
            }
            forChange.text = event.currentTarget.selectedItem.@label +
               theData;
         }

      private function itemOpenEvt(event:TreeEvent):void {
         forOpen.text = event.item.@label;
      }
   ]]>
   </mx:Script>

   <mx:Tree id="XMLtree1" width="150" height="170"
          labelField="@label" itemOpen="itemOpenEvt(event);"
          change= "changeEvt(event);">
      <mx:XMLListCollection id="MailBox">
         <mx:XMLList>
            <node label="Mail" data="100">
               <node label="Inbox" data="70"/>
               <node label="Personal Folder" data="10">
                  <node label="Business" data="2"/>
                  <node label="Demo" data="3"/>
                  <node label="Personal" data="0" isBranch="true" />
                  <node label="Saved Mail" data="5" />
               </node>
               <node label="Sent" data="15"/>
               <node label="Trash" data="5"/>
            </node>
         </mx:XMLList>
```

```
      </mx:XMLListCollection>
   </mx:Tree>

   <mx:Label text="Change Event:" />
   <mx:TextArea id="forChange" width="150" />
   <mx:Label text="Open Event:" />
   <mx:TextArea id="forOpen" width="150" />
</mx:Application>
```

この例では、change イベントおよび itemOpen イベントのイベントリスナーを定義しています。Tree コントロールでは、ユーザーがツリーアイテムを選択すると change イベントがブロードキャストされ、ユーザーがブランチノードを展開すると itemOpen イベントがブロードキャストされます。いずれのイベントでも、イベントハンドラによって TextArea コントロールにラベルが表示されます。設定されている場合には、データプロパティも表示されます。この例では、Business ノードおよび SavedMail ノードにだけデータ値が定義されています。

ツリーノードの展開

デフォルトでは、ツリーを最初に開いたときにはルートノードが表示されます。ツリーを開いたときにノードを展開した状態で表示したい場合は、Tree コントロールの expandItem() メソッドを使用します。453 ページの「Tree コントロールのイベント処理」の例を次のように変更すると、expandItem() メソッドが Tree コントロールの creationComplete イベントリスナーの一部として呼び出され、ツリーのルートノードが展開されます。

```
<mx:Script>
  <![CDATA[
.
.
.
     private function initTree():void {
        XMLTree1.expandItem(MailBox.getItemAt(0), true);
        forOpen.text=XMLTree1.openItems[0].@label;
     }
     ]]>
  </mx:Script>

<mx:Tree id="tree1" ... creationComplete="initTree();" >
   ...
</mx:Tree>
```

この例では、Tree コントロールの initialize イベントではなく、creationComplete イベントを使用する必要があります。creationComplete イベントが実行されるまでは、データプロバイダが完全に初期化されて使用できる状態にはならないからです。

Tree コントロールの openItems プロパティは、展開されたすべてのツリーノードを含む配列です。例のコードに含まれる次の行は、ツリー内で最初かつ唯一の開いているアイテムのラベルを表示します。

```
forOpen.text=XMLTree1.openItems[0].@label;
```

ただしこの例では、expandItem() メソッドで itemOpen イベントを送出するように設定すれば、openItems ボックスを取得して最初の開いているアイテムを示すこともできます。これは expandItem() メソッドのオプションの第 4 パラメータを true に指定すれば実行できます。第 4 パラメータが true の場合、アイテムが開かれるとツリーから open イベントが送出されます。次の例は、第 4 パラメータの使用方法を示したものです。

```
XMLTree1.expandItem(MailBox.getItemAt(0), true, false, true);
```

Tree コントロールのアイコンの指定

Tree コントロールでは、4 つの方法でノードアイコンを指定できます。

- folderOpenIcon、folderClosedIcon、および defaultLeafIcon の各プロパティ
- データプロバイダのノードアイコンのフィールド
- setItemItcon() メソッド
- iconFunction プロパティ

アイコンプロパティの使用

folderOpenIcon、folderClosedIcon、および defaultLeafIcon の各プロパティを使用して、Tree コントロールのアイコンを制御できます。たとえば、次のコードでは、デフォルトアイコン、ブランチノードを開いた状態および閉じた状態のアイコンを指定しています。

```
<mx:Tree folderOpenIcon="@Embed(source='open.jpg')"
  folderClosedIcon="@Embed(source='closed.jpg')"
  defaultLeafIcon="@Embed(source='def.jpg')">
```

アイコンフィールドの使用

XML を使用して Tree のデータを設定する場合は、次のようにして、各 Tree リーフの隣に表示するアイコンを指定できます。

```
<?xml version="1.0"?>
<!-- dpcontrols/TreeIconField.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

   <mx:Script>
      <![CDATA[
         [Bindable]
         [Embed(source="assets/radioIcon.jpg")]
         public var iconSymbol1:Class;
         [Bindable]
         [Embed(source="assets/topIcon.jpg")]
         public var iconSymbol2:Class;
      ]]>
   </mx:Script>
```

```
   <mx:Tree iconField="@icon" labelField="@label" showRoot="false"
      width="160">
      <mx:XMLList>
         <node label="New">
            <node label="HTML Document" icon="iconSymbol2"/>
            <node label="Text Document" icon="iconSymbol2"/>
         </node>
         <node label="Close" icon="iconSymbol1"/>
      </mx:XMLList>
   </mx:Tree>
</mx:Application>
```

この例では、iconField プロパティを使用して、アイコンを使用する各アイテムのフィールドを指定しています。Embed メタデータを使用してアイコンを読み込み、それを XML 定義内で参照しています。個々のブランチノードにアイコンを指定することはできません。代わりに、Tree コントロールの folderOpenIcon プロパティと folderClosedIcon プロパティを使用する必要があります。これらのプロパティは、すべての開いているおよび閉じているブランチに使用されるアイコンを指定します。

setItemIcon() メソッドの使用

setItemIcon() メソッドでは、ツリーのアイテムにアイコン、または開いたアイコンと閉じたアイコンの両方を指定できます。このメソッドを使用すれば、個々のブランチやノードにアイコンを動的に指定および変更できます。この関数の詳細については、『ActionScript 3.0 リファレンスガイド』の setItemIcon() を参照してください。次の例では、最初のブランチノードの開かれたおよび閉じたノードアイコンと、2 番目のブランチ (リーフを持たないもの) のアイコンを設定します。

```
<?xml version="1.0"?>
<!-- dpcontrols/TreeItemIcon.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

   <mx:Script>
      <![CDATA[
         [Bindable]
         [Embed(source="assets/radioIcon.jpg")]
         public var iconSymbol1:Class;
         [Bindable]
         [Embed(source="assets/topIcon.jpg")]
         public var iconSymbol2:Class;

         private function setIcons():void {
            myTree.setItemIcon(myTree.dataProvider.getItemAt(0),
               iconSymbol1, iconSymbol2);
            myTree.setItemIcon(myTree.dataProvider.getItemAt(1),
               iconSymbol2, null);
         }
      ]]>
   </mx:Script>
```

```
   <mx:Tree id="myTree" labelField="@label" showRoot="false"
      width="160" initialize="setIcons();">
      <mx:XMLList>
         <node label="New">
            <node label="HTML Document"/>
            <node label="Text Document"/>
         </node>
         <node label="Close"/>
      </mx:XMLList>
   </mx:Tree>
</mx:Application>
```

アイコン関数の使用

Tree コントロールの iconFunction プロパティを使用すれば、ツリーのすべてのアイコンを動的に設定する関数を指定できます。Flex のコントロールで iconFunction プロパティを使用する方法については、418 ページの「List コントロールへのアイコンの指定」を参照してください。

ユーザーによるツリーの操作

ツリーコントロールのラベルは、ユーザーによる編集を可能にできます。コントロールではさまざまなキーボード操作および編集キーもサポートされています。

実行時のノードラベル編集

Tree コントロールの editable プロパティを true に設定すれば、実行時にノードラベルを編集できます。ノードラベルを編集するには、ユーザーはノードを選択してから、新しいラベルを入力するか、既存のラベルテキストを編集します。

ラベルの編集のために、Tree コントロールの List スーパークラスでは次のイベントを使用しています。これらのイベントは ListEvent クラスに属しています。

イベント	説明
itemEditBegin	editedItemPosition プロパティが設定され、セルが編集可能になったときに送出されます。
itemEditEnd	何らかの理由でセル編集セッションが終了するときに送出されます。
itemFocusIn	ツリーノードがフォーカスを取得したとき、すなわちユーザーがツリーノードのラベルまたはタブを選択したときに送出されます。
itemFocusOut	ラベルがフォーカスを失ったときに送出されます。
itemClick	ユーザーがコントロール内のアイテムをクリックしたときに送出されます。

これらのイベントは通常、カスタムのアイテムエディタで使用されます。詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

キーボード入力によるラベルの編集

Tree の `editable` プロパティを `true` に設定した場合、以下のキーを使用してラベルを編集できます。

キー	説明
↓ (下矢印) PageDown End	キャレットをラベルの末尾に移動します。
↑ (上矢印) PageUp Home	キャレットをラベルの冒頭に移動します。
→ (右矢印)	キャレットを 1 文字前に移動します。
← (左矢印)	キャレットを 1 文字後ろに移動します。
Enter	編集を終了して、次の表示されているノードを選択し、そのノードの編集を開始します。最後のノードに来たら、そのラベルを選択します。
Shift Enter	編集を終了して、前の表示されているノードを選択し、そのノードの編集を開始します。最初のノードに来たら、そのラベルを選択します。
Esc	編集をキャンセルして、テキストを元に戻し、行の状態が編集中から選択中に変化します。
TAB	編集中の場合は、現在の変更を確定してから下の行が選択され、ラベルテキストが選択された状態でその行が編集対象になります。ツリーの末尾のエレメントを編集中の場合、またはいずれのノードも編集していない場合には、次のコントロールにフォーカスが移動します。
Shift+TAB	編集中の場合は、現在の変更を確定してから上の行が選択され、その行が編集対象になります。ツリーの先頭のエレメントを編集中の場合、またはいずれのノードも編集していない場合には、前のコントロールにフォーカスが移動します。

キーによるツリー操作

Tree コントロールが編集不可能な際、クリックまたは Tab キーによってフォーカスを Tree コントロールに移したら、次のキーを使用して Tree コントロールを制御できます。

キー	説明
↓ (下矢印)	1つ下のアイテムを選択します。Tree コントロールがフォーカスを取得した際、下矢印キーを使用すると最初のノードにフォーカスが移動します。
↑ (上矢印)	1つ上のアイテムを選択します。
→ (右矢印)	選択したブランチノードを開きます。ブランチが既に開いている場合は、最初の子ノードに移動します。
← (左矢印)	選択したブランチノードを閉じます。リーフノードまたは閉じたブランチノードを選択していた場合は、親ノードが選択されます。
スペースバー、または * (テンキーのアスタリスク)	選択したブランチノードを開くまたは閉じます (状態を切り替えます)。
+ (テンキーのプラス記号)	選択したブランチノードを開きます。
- (テンキーのマイナス記号)	選択したブランチノードを閉じます。
Ctrl + 矢印キー	フォーカスを移動しますが、ノードは選択しません。ノードを選択するには、スペースバーを使用してください。
End	リストの一番下のアイテムを選択します。
Home	リストの一番上のアイテムを選択します。
PageDown	1ページ下のアイテムを選択します。
PageUp	1ページ上のアイテムを選択します。
Ctrl	`allowMultipleSelection` プロパティが `true` の場合、複数のアイテムを非連続に選択できます。
Shift	`allowMultipleSelection` プロパティが `true` の場合、連続した複数のアイテムを選択できます。

キーを使用してツリーのラベルを編集する方法については、458 ページの「キーボード入力によるラベルの編集」を参照してください。

第 13 章

コンテナについて

13

コンテナは、子コンポーネントのレイアウト属性を制御できる階層構造を提供します。コンテナを使用すると、すべての子のサイズと位置の設定、または複数の子コンテナ間のナビゲーションを制御できます。

このトピックでは、レイアウトとナビゲータという 2 種類のコンテナについて説明します。コンテナの使用に関する概要 (たとえばレイアウト規則など) を示し、実際の使用方法と設定の例を示します。

目次

コンテナについて

" コンテナ " は、Adobe Flash Player の描画面の矩形領域を定義します。コンテナには、他のコントロールやコンテナなど、その中に表示するコンポーネントを定義します。コンテナ内で定義されたコンポーネントのことを " 子 " と呼びます。Adobe Flex には、単純なボックスから、パネルやフォーム、さらには子コンポーネント間の組み込みナビゲーション機能を提供するアコーディオンやタブ付きナビゲータなどのエレメントに至るまで、さまざまなコンテナが用意されています。

Flex アプリケーションのルートには、Application コンテナと呼ばれるコンテナが 1 つあります。これは、Flash Player の描画面全体を表します。この Application コンテナには、他のすべてのコンテナとコンポーネントが格納されます。

コンテナには、子のレイアウトを制御するサイズ設定や位置設定などの規則があらかじめ定義されています。Flex は、多様なアプリケーションを作成できる柔軟性を提供する一方で、レイアウト規則の定義により、リッチインターネットアプリケーションの設計と実装を簡素化します。

コンテナのレイアウトについて

コンテナにはナビゲーション規則とレイアウト規則があらかじめ定義されているので、開発者はこれらの定義に時間を費やす必要はありません。また、ユーザーの操作やアプリケーションの応答についての細かい部分の実装にわずらわされる必要もなく、どのような情報を配信し、どのようなオプションをユーザーに提供するかに意識を集中させることができます。このように、Flex では、豊富な機能と対話性を備えたアプリケーションを、短期間で容易に開発できる構造が採用されています。

レイアウト規則があらかじめ定義されていることは、ユーザーが短期間で操作を習得できるという利点ももたらします。操作規則を標準化することで、ユーザーはアプリケーションの操作について迷うことがなくなり、アプリケーションによって提供されるコンテンツに意識を集中できるようになります。

各コンテナがサポートするレイアウト規則は、コンテナの種類によって異なります。

- Canvas コンテナを除くすべてのコンテナは、" 自動 " レイアウトをサポートします。このレイアウト方式では、コンテナの子の位置は指定しません。その代わりに、コンテナの種類を選択し、コンテナの子の順序を設定し、子の間隔などのプロパティや Spacer などのコントロールを指定することによって、位置を制御します。たとえば、矩形領域の中で子を水平方向にレイアウトするには、HBox コンテナを使用します。
- Canvas コンテナと、オプション設定を指定した Application コンテナおよび Panel コンテナは、" 絶対 " レイアウトを使用します。このレイアウト方式では、子の x 座標と y 座標を明示的に指定します。別の方法として、" 制約ベースのレイアウト " を使用し、子の端または中心を親に対して相対的に固定することもできます。

絶対レイアウトでは、自動レイアウトよりも詳細にサイズと位置を制御できます。たとえば、あるコントロールを別のコントロールに重ねることも可能です。しかし絶対レイアウトでは、このようにレイアウトを細かくできる反面、その代償として位置を詳細に指定することが必要になります。

レイアウトの詳細については、207 ページ、第 8 章の「コンポーネントのサイズと位置の制御」を参照してください。

レイアウトコンテナとナビゲータコンテナについて

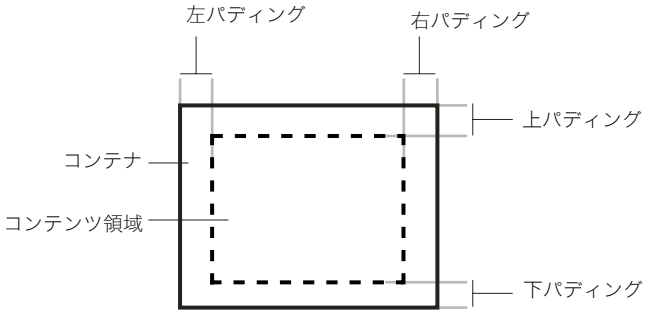
Flex には、次の 2 種類のコンテナが定義されています。

レイアウトコンテナは、子コントロールおよび子コンテナのサイズと位置を制御します。たとえば、Grid レイアウトコンテナでは、HTML テーブルのようなレイアウトで、子のサイズと位置が制御されます。レイアウトコンテナにはグラフィカルエレメントも含まれており、それによって特定のスタイルを適用したり、コンテナの機能を表したりできます。たとえば DividedBox コンテナには中央にバーがあり、ユーザーはそれをドラッグすることで、分割された 2 つの領域のサイズを相対的に変更できます。また、TitleWindow コントロールには、タイトルやステータス情報を格納できる初期バーがあります。これらのコンテナの詳細については、517 ページ、第 15 章の「レイアウトコンテナの使用」を参照してください。

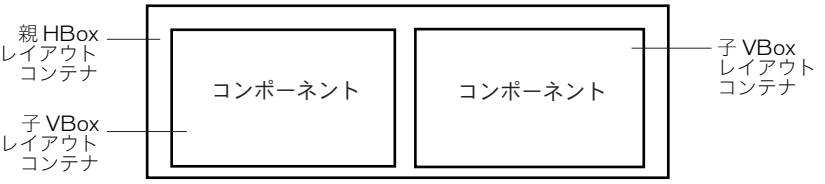
ナビゲータコンテナは、複数の子コンテナ間でのユーザーの移動 (ナビゲーション) を制御します。子のレイアウトと位置はナビゲータコンテナではなく、それぞれの子コンテナによって制御されます。たとえば、Accordion ナビゲータコンテナでは、複数の Form レイアウトコンテナにより、複数ページから成るフォームを作成できます。詳細については、583 ページ、第 16 章の「ナビゲータコンテナの使用」を参照してください。

コンテナの使用

コンテナの矩形領域内には、子コンポーネントを保持する領域として、" コンテンツ領域 " が存在します。コンテンツ領域の周囲にある領域のサイズは、コンテナのパディングと境界線の幅によって定義されます。コンテナの上下左右にはパディングがあり、その幅はピクセル数で設定できます。コンテナにはまた、境界線の種類とピクセル幅を指定できるプロパティもあります。次の図は、コンテナとそのコンテンツ領域、パディング、および境界線を示します。



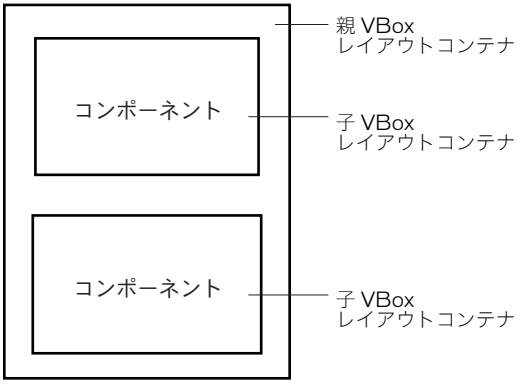
Flex アプリケーション全体を 1 つのコンテナで作成することもできますが、複数のコンテナを使用するのが一般的です。次の図は、3 つのレイアウトコンテナが使用されたアプリケーションの例です。



この例では、2 つの VBox (Vertical Box) レイアウトコンテナが、HBox (Horizontal Box) レイアウトコンテナ内に HBox コンテナの子としてネストされています。

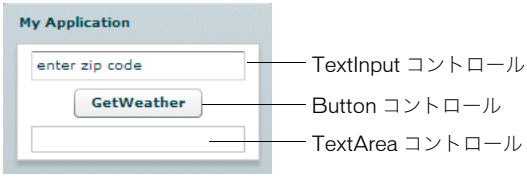
HBox コンテナは単一の行 (横方向) に子を配置し、VBox コンテナのサイズと位置の属性を管理します。たとえば、コンテナに存在する子と子の距離 (間隔) は、`horizontalGap` プロパティと `verticalGap` プロパティを使って制御できます。

VBox レイアウトコンテナは、単一の列 (縦方向) に子を配置し、自分の子のレイアウトを管理します。上の例で一番外側のコンテナを VBox レイアウトコンテナに変更すると、次の図のようになります。



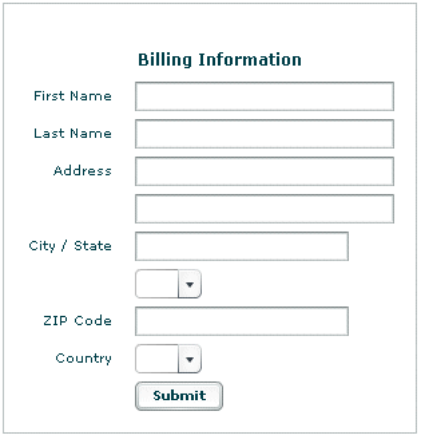
この例では、外側のコンテナが VBox コンテナであるため、子は縦方向の列として配置されます。

レイアウトコンテナは、主に子 (コントロールまたはその他のコンテナ) を整然と配置する目的で使用されます。次の図は、3 つの子コンポーネントを持つ、シンプルな VBox コンテナを示します。



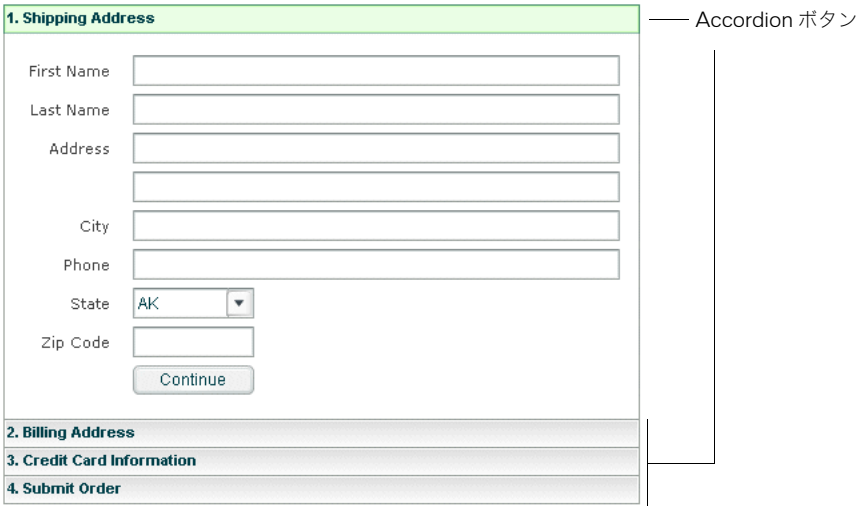
この例では、ユーザーが TextInput コントロールに郵便番号を入力し、Button コントロールをクリックすると、指定した郵便番号に該当する地域の現在の気温が表示されます。

Flex は Form レイアウトコンテナを使ったフォームベースのアプリケーションをサポートします。Form コンテナでは、ラベル位置を自動的に整列できるほか、TextInput コントロールのサイズを均等に設定したり、入力エラーを表示することもできます。次の図は Form コンテナを示します。



Form コンテナでは、Flex の検証メカニズムを利用して、ユーザーがフォームを送信する前に入力エラーを検出できます。入力エラーを検出し、フォームがサーバーに送信される前にユーザーにミスを修正させることによって、不要なサーバー接続を減らすことが可能です。Flex の検証メカニズムを使用したからといって、サーバー上で他の検証が実行できなくなることはありません。Form コンテナの詳細については、532 ページの「Form、FormHeading、および FormItem レイアウトコンテナ」を参照してください。バリデータの詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

TabNavigator や Accordion などのナビゲータコンテナはビルトインのナビゲーションコントロールを備えています。ナビゲータコンテナを使用して複数の子コンテナからの情報を体系化すると、ユーザーがその中を容易に移動できるようになります。次の図は Accordion コンテナを示します。



複数の子コンテナ間を移動するには、Accordion ボタンを使用します。Accordion コンテナでは子パネルを定義しますが、一度に表示されるパネルは 1 つだけです。コンテナを移動する場合、ユーザーはアクセスする子パネルに対応するナビゲーションボタンをクリックします。

Accordion コンテナを使用すると、段階的な手順を作成できます。上の図は、4 つのパネルから成る複合フォームを定義した Accordion コンテナです。フォームのすべての項目に記入するには、ユーザーは 4 つのパネルすべてにデータを入力しなければなりません。最初のパネルに情報を入力した後、Accordion ボタンをクリックして 2 番目のパネルに移動し、入力した情報を変更したい場合は再び最初のパネルに戻る、というような手順も Accordion であれば可能です。詳細については、594 ページの「Accordion ナビゲータコンテナ」を参照してください。

Flex コンテナ

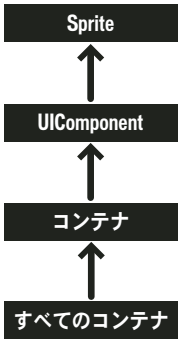
Flex の各種コンテナとその説明を次の表に示します。

コンテナ	タイプ	説明	詳細情報
Accordion	ナビゲータ	一連の子パネルの情報を体系化します。いずれか1つのパネルが常にアクティブになります。	594 ページの「Accordion ナビゲータコンテナ」
Application ControlBar	レイアウト	グローバルなナビゲーションコマンドとアプリケーションコマンドを提供するコンポーネントを格納します。Application コンテナの上部にドッキングできます。	526 ページの「ApplicationControlBar レイアウトコンテナ」
Box (HBox および VBox)	レイアウト	行方向または列方向に均等に割り当てられた領域にコンテンツを表示します。HBox コンテナでは水平方向に子が配置され、VBox コンテナでは垂直方向に子が配置されます。	523 ページの「Box、HBox、および VBox レイアウトコンテナ」
Canvas	レイアウト	子の位置を明示的に設定する必要があるコンテナを定義します。	518 ページの「Canvas レイアウトコンテナ」
ControlBar	レイアウト	Panel コンテナまたは TitleWindow コンテナの下部にコントロールを配置します。	525 ページの「ControlBar レイアウトコンテナ」
DividedBox (HDividedBox および VDividedBox)	レイアウト	子を水平方向または垂直方向にレイアウトします。Box コンテナとよく似ていますが、DividedBox の場合は、子と子の間に調整可能な境界線が挿入されます。	529 ページの「DividedBox、HDividedBox、および VDividedBox レイアウトコンテナ」
Form	レイアウト	子を標準のフォーム形式で配置します。	532 ページの「Form、FormHeading、および FormItem レイアウトコンテナ」
Grid	レイアウト	HTML テーブルのように、行と列から成るセルとして子を配置します。	553 ページの「Grid レイアウトコンテナ」
Panel	レイアウト	タイトルバー、キャプション、境界線、およびその子を表示します。	559 ページの「Panel レイアウトコンテナ」
TabNavigator	ナビゲータ	異なるコンテンツ領域を切り替えるためのタブの付いたコンテナを表示します。	590 ページの「TabNavigator コンテナ」

コンテナ	タイプ	説明	詳細情報
Tile	レイアウト	複数の行または列に子を配置するレイアウトを定義します。	563 ページの「Tile レイアウトコンテナ」
TitleWindow	レイアウト	タイトルバー、キャプション、境界線、閉じるボタン、およびその子を格納するポップアップウィンドウを表示します。移動およびサイズ変更が可能です。	566 ページの「TitleWindow レイアウトコンテナ」
ViewStack	ナビゲータ	積み重ねられたコンテナを定義します。一度に表示できるコンテナは1つだけです。	584 ページの「ViewStack ナビゲータコンテナ」

コンテナのクラス階層

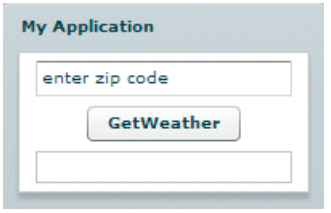
Flex コンテナは、次の図のように、ActionScript クラスライブラリの階層として実装されます。



コンテナはすべて ActionScript の各クラス (Sprite、UIComponent、Container) から派生しているため、スーパークラスのプロパティ、メソッド、スタイル、エフェクト、およびイベントが継承されます。一部のコンテナは他のコンテナのサブクラスです。たとえば、ApplicationControlBar は ControlBar コンテナのサブクラスです。詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

コンテナの例

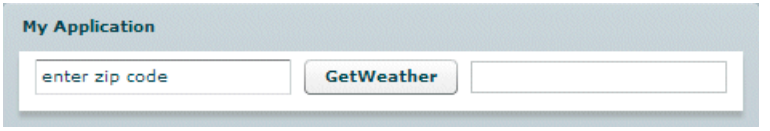
次の図は、3 つの子コントロールを含む Panel コンテナを使用した Flex アプリケーションを示します。ここでは、Panel コンテナの子は垂直にレイアウトされています。



この例を作成する MXML コードは次のとおりです。

```
<?xml version="1.0"?>
<!-- containers\intro\Panel3Children.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Panel title="My Application"
        layout="vertical" horizontalAlign="center"
        paddingLeft="10" paddingRight="10"
        paddingTop="10" paddingBottom="10">
            <mx:TextInput id="myinput" text="enter zip code"/>
            <mx:Button id="mybutton" label="GetWeather"/>
            <mx:TextArea id="mytext" height="20"/>
    </mx:Panel>
</mx:Application>
```

次の図は、上の例の Panel コンテナを水平レイアウトに変更した場合を示します。



この例のコードは次のとおりです。2 つの例の違いはコンテナの種類だけであり、水平方向のレイアウトに変更したことによって上の例よりも Application コンテナの幅が広くなっています。

```
<?xml version="1.0"?>
<!-- containers\intro\Panel3ChildrenHoriz.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Panel title="My Application"
        layout="horizontal"
        paddingLeft="10" paddingRight="10"
        paddingTop="10" paddingBottom="10">
            <mx:TextInput id="myinput" text="enter zip code"/>
            <mx:Button id="mybutton" label="GetWeather"/>
            <mx:TextArea id="mytext" height="20"/>
```

```
    </mx:Panel>
</mx:Application>
```

実際に気象情報を取得するには、Web サービスを設定し、そのサービスに click イベントから入力された ZIP コードを渡します。そして、返された情報を TextArea コントロールに格納します。

コンテナイベントの使用

コンテナとコンポーネントはすべてイベントをサポートしています。以降のセクションで、イベントがどのようにサポートされているかについて説明します。

イベントの概要

次のイベントは、コンテナのみによって送出されます。

- childAdd　子がコンテナに追加された後に送出されます。
- childRemove　子がコンテナから削除される前に送出されます。
- childIndexChange　コンテナ内の子のインデックスが変更された後に送出されます。
- scroll　ユーザーが手動でコンテナをスクロールしたときに送出されます。

最初の 3 つのイベントはコンテナの子ごとに送出され、最後のエベントはコンテナがスクロールしたときに送出されます。これらのイベントの詳細については、『Adobe Flex 2 リファレンスガイド』の「コンテナ」を参照してください。

次のイベントは、Application コンテナのみによって送出されます。

- applicationComplete　アプリケーションの初期化、LayoutManager による処理、および表示リストへの関連付けが完了した後に送出されます。これはアプリケーションの起動シーケンス中に送出される最後のイベントで、アプリケーションの creationComplete イベントよりも後に送出されます。creationComplete イベントは、プリローダーが削除されてアプリケーションが表示リストに関連付けられる前に送出されます。
- error　アプリケーションのどこかで不明なエラーが発生したときに送出されます。

次のイベントは、すべてのコンポーネントによって、コンポーネントがコンテナに追加された後またはコンテナから削除された後に送出されます。

- add　コンポーネントがそのコンテナに追加され、親と子が整合性の取れた状態になった後に、コンポーネントによって送出されます。このイベントは、コンテナが childAdd イベントを送出し、追加の結果として必要な変更がすべて発生した後に送出されます。
- remove　コンポーネントがその親コンテナから削除された後に、コンポーネントによって送出されます。このイベントは、コンテナが childRemove イベントを送出し、削除の結果として必要な変更がすべて発生した後に送出されます。

いくつかのイベントはすべてのコンポーネントで送出されますが、コンテナの作成時に一部の子が作成されないコンテナ (特に TabNavigator などのナビゲータコンテナ) では、これらのイベントについて特別な注意が必要です。次のイベントがこれに該当します。

- `preinitialize`　コンポーネントがその親コンテナに関連付けられた後、コンポーネントが初期化される前、つまりその子が作成される前に送出されます。ほとんどの場合、このイベントは、アプリケーションでコンポーネントの設定に使用するには送出されるタイミングが早すぎます。
- `initialize`　コンポーネントの構築と初期化プロパティの設定が完了したときに送出されます。この時点で、コンポーネントの直接の子はすべて作成されています (少なくともその `preinitialize` イベントは既に送出されています) が、レイアウトはまだされていません。initialize イベントが正確にいつ送出されるかは、コンテナの作成ポリシーによって異なります。これについては、このセクションの後で説明します。
- `creationComplete`　コンポーネントとそのすべての子孫の作成とレイアウトが完了し、それらが表示されたときに送出されます。

`initialize` イベントと `creationComplete` イベントの詳細については、472 ページの「initialize イベントと creationComplete イベントについて」を参照してください。残りのイベントの詳細については、『Adobe Flex 2 リファレンスガイド』の「コンテナ」を参照してください。

作成ポリシーについて

コンテナには、その子をいつ作成するかを指定する `creationPolicy` プロパティがあります。デフォルトでは、コンテナの作成ポリシーは `ContainerCreationPolicy.AUTO` です。これは、子孫が必要になるまでその作成を遅らせることを示します。この処理のことを " 遅延インスタンス化 " と呼びます。コンテナのデフォルト作成ポリシーが auto であるのは、Application コンテナのデフォルトポリシーが `ContainerCreationPolicy.AUTO` であり、それが子に継承されるためです。

auto 作成ポリシーを使用すると、最初に作成されるコンポーネントが少なくて済むため、起動時間が最短になります。ViewStack、TabNavigator、Accordion などのナビゲータコンテナで `ContainerCreationPolicy.AUTO` 作成ポリシーを使用すると、その直接の子はただちに作成されますが、各々の子の子孫は子が表示されるまで作成されません。したがって、初期化前処理段階を過ぎた後は、コンテナで最初に必要とされる子のみが処理されます。

作成ポリシーが `ContainerCreationPolicy.ALL` の場合は、コンテナのすべての子の作成と初期化が完了してから、コンテナが初期化されます。作成ポリシーの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 6 章の「起動時のパフォーマンスの向上」を参照してください。

initialize イベントと creationComplete イベントについて

コンテナの initialize イベントは、コンテナの直接の子コントロールがすべて関連付けられ、コンテナで最初に必要とされる子の preinitialize イベントが送出された後に送出されます。

コンテナまたはコントロールが initialize イベントを送出するとき、その初期プロパティは既に設定されていますが、幅と高さおよび位置はまだ計算されていません。initialize イベントは、コンテナの子の設定に役立ちます。たとえば、プログラムによる子の追加やコンテナのスクロールバースタイルの設定などにコンテナの initialize イベントを使用できます。また、コンテナまたはコンポーネントの initialize イベントを使用して、コントロールのデータプロバイダを初期化することもできます。

コンテナの creationComplete イベントは、最初に必要とされる子 (その子の必要な子孫をすべて含む) の処理がすべて完了し、画面に描画されたときに送出されます。たとえば、イベントハンドラで子のサイズや位置を使用する場合は、creationComplete イベントのリスナーを作成します。レイアウトプロパティを設定する処理のために creationComplete イベントを使用しないでください。そうすると、処理時間が余計にかかります。

イベントが送出される順序をわかりやすくするため、次の構造を持つアプリケーションを例にとります。

```
Application
  OuterVBox
    InnerVBox1
      InnerVBoxLabel1
    InnerVBox2
      InnerVBoxLabel2
```

コンテナとコントロールの preinitialize イベント、initialize イベント、および creationComplete イベントは、次の順序で送出されます。各インデントは、上のアウトライン構造のインデントに対応します。

```
OuterVBox  preinitialize
  InnerVBox1  preinitialize
    InnerVBox1Label  preinitialize
    InnerVBox1Label  initialize
  InnerVBox1  initialize
  InnerVBox2  preinitialize
    InnerVBox2Label  preinitialize
    InnerVBox2Label  initialize
  InnerVBox2  initialize
OuterVBox  initialize
    InnerBox1Label  creationComplete
    InnerVBox2Label  creationComplete
  InnerVBox1  creationComplete
  InnerVBox2  creationComplete
OuterVBox  creationComplete
```

この図を見ると、Label コントロールなどの末端コントロールは初期化前処理が終わってすぐに初期化されていることがわかります。コンテナでは、最も外側のコンテナから初期化前処理が始まり、最初のブランチの内側に進みます。続いて、初期化が同じブランチの外側に向かって進みます。この処理が、すべての初期化が完了するまで続きます。次に、creationComplete イベントの送出がリーフコンポーネントから始まり、続いてその親に進みます。この処理が、アプリケーションが creationComplete イベントを送出するまで続きます。

OuterVBox コンテナを、creationPolicy プロパティが auto に設定された ViewStack に変更すると、イベントの順序は次のようになります。

```
OuterViewStack  preinitialize
  InnerVBox1  preinitialize
  InnerVBox2  preinitialize
OuterViewStack  initialize
    InnerBox1Label  preinitialize
    InnerBox1Label  initialize
  InnerVBox1  initialize
    InnerBox1Label  creationComplete
  InnerVBox1  creationComplete
OuterViewStack  creationComplete
```

この場合、2 番目の VBox は表示されないため、初期化前処理が行われるだけです。ナビゲータコンテナが initialize イベントを送出するとき、その子は存在し、子の preinitialize イベントは既に送出されていますが、その子自体の子はまだ作成されていないため、子の initialize イベントは送出されていません。creationPolicy プロパティの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 6 章の「起動時のパフォーマンスの向上」を参照してください。

initialize イベントは、ContainerCreationPolicy.AUTO 作成ポリシーが設定されたナビゲータコンテナの直接の子であるコンテナで役立ちます。たとえば、デフォルトでは、ViewStack が初期化されると、最初に表示される子コンテナが initialize イベントを送出します。ユーザーがコンテナの別の子に移動すると、その子コンテナのイベントが送出されます。

次の例は、initialize イベントのイベントリスナーを定義します。このイベントは、ユーザーが Accordion コンテナのパネル 2 に初めて移動したときに送出されます。

```
<?xml version="1.0"?>
<!-- containers\intro\AccordionInitEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;

            public function pane2_initialize():void {
                Alert.show("Pane 2 has been created");
            }
        ]]>
    </mx:Script>
```

```
    <mx:Accordion width="200" height="100">
        <mx:VBox id="pane1" label="Pane 1">
            <mx:Label text="This is pane 1"/>
        </mx:VBox>
        <mx:VBox id="pane2"
            label="Pane 2"
            initialize="pane2_initialize();">
            <mx:Label text="This is pane 2"/>
        </mx:VBox>
    </mx:Accordion>
</mx:Application>
```

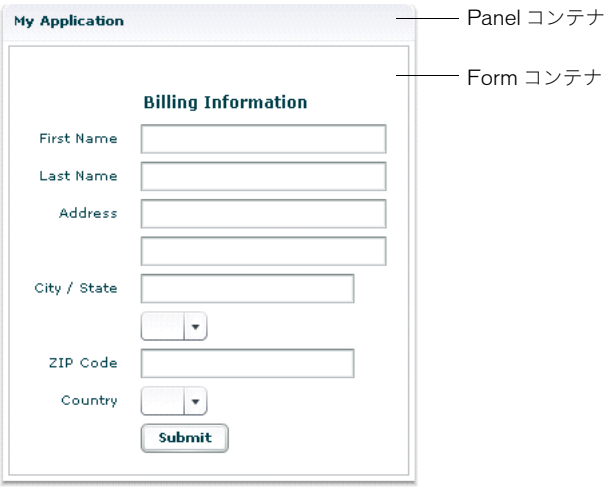
コンテナの無効化

すべてのコンテナは、enabled プロパティをサポートします。デフォルトでは、このプロパティは true に設定されており、ユーザーはコンテナとその子を操作することができます。あるコンテナに対して enabled を false に設定すると、そのコンテナとそのすべての子がグレー表示になり、それらに入力できなくなります。

Panel コンテナの使用

Flex アプリケーションでよく使用されるコンテナの 1 つに Panel コンテナがあります。Panel コンテナは、タイトルバー、キャプション、ステータスメッセージ、境界線、子のコンテンツ領域で構成されます。一般に、必要なアプリケーションモジュールをラップする場合に Panel コンテナを使用します。たとえば、アプリケーションに対して複数の Panel コンテナを定義し、それぞれにフォーム、ショッピングカート、ショッピングカタログを表示することが可能です。Flex RichTextEditor コントロールは、TextArea コントロールと、編集コントロールを持つ ControlBar コントロールを含む Panel コントロールです。

次の図は、Form コンテナを子として持つ Panel コンテナです。



Panel コンテナの詳細については、559 ページの「Panel レイアウトコンテナ」を参照してください。

また、Panel コンテナの一部として、ControlBar コントロールを定義することもできます。ControlBar コントロールは、Panel コンテナの (いずれの子よりも下に位置する) 下端の領域を定義します。

ControlBar コンテナを使用すると、Panel コンテナの子によって共有されるコンポーネントを一箇所にまとめることができます。また、Panel コンテナの内容を操作するコントロールのためにコンポーネントをまとめることも可能です。たとえば、ショッピングカートを定義した Panel コンテナで ControlBar コンテナを使用して、ショッピングカートの小計を表示できます。製品カタログでは、数量の指定やショッピングカートへの商品の追加を行うための Flex コントロールを ControlBar コンテナに配置できます。ControlBar コンテナの詳細については、525 ページの「ControlBar レイアウトコンテナ」を参照してください。

デフォルトボタンの定義

コンテナに対してデフォルトの Button コントロールを定義するには、対象のコンテナの defaultButton プロパティを使用します。いずれかのコントロールにフォーカスがあるときに Enter キーを押すと、あたかもそのボタンが明示的にクリックされたかのように、デフォルトの Button コントロールがアクティブになります。

たとえば、ユーザー名とパスワード入力用の TextInput コントロールと送信用の Button コントロールを備えたログインフォームを作成するとします。通常は、ユーザー名を入力し、Tab キーでパスワードフィールドに移動してパスワードを入力した後、Enter キーを押すと、ログイン情報が送信されます。Button コントロールを明示的にクリックする必要はありません。このようなユーザーインターフェイスを定義するには、次のように、Form コントロールの defaultButton プロパティを送信用 Button コントロールの id に設定します。

```
<?xml version="1.0"?>
<!-- containers\intro\ContainerDefaultB.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function submitLogin():void {
                text1.text="You just tried to log in";
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Default Button Example">

        <mx:Form defaultButton="{mySubmitBtn}">
            <mx:FormItem label="Username">
                <mx:TextInput id="username"
                    width="100"/>
            </mx:FormItem>
            <mx:FormItem label="Password">
                <mx:TextInput id="password"
                    width="100"
                    displayAsPassword="true"/>
            </mx:FormItem>
            <mx:FormItem>
                <mx:Button id="mySubmitBtn"
                    label="Login"
                    click="submitLogin();"/>
            </mx:FormItem>
        </mx:Form>
        <mx:Text id="text1" width="150"/>
    </mx:Panel>
</mx:Application>
```

メモ

ComboBox コントロールでは、Enter キーは特別な目的があります。ComboBox コントロールのドロップダウンリストが展開されているときに Enter キーを押すと、その ComboBox コントロールで現在ハイライト表示されているアイテムが選択されます。このとき、デフォルトボタンはアクティブになりません。また、カーソルが TextArea コントロールにあるときに Enter キーを押すと、改行が追加されます。このときもデフォルトボタンはアクティブになりません。

スクロールバーの使用

Flex コンテナはスクロールバーをサポートしており、これを使用することで、使用可能な画面領域より大きいオブジェクトやコンテナの現在のサイズに収まりきらないオブジェクトを表示できます。次にスクロールバーの例を示します。

フルサイズのイメー

HBox コンテナ内のイメージ

この例では、HBox コンテナを使用して、フルサイズの画像全体を表示する代わりに、ユーザーがイメージをスクロールできるようにしています。

```
<?xml version="1.0"?>
<!-- containers\intro\HBoxScroll.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox width="75" height="75">
        <mx:Image source="assets/logo.jpg"/>
    </mx:HBox>
</mx:Application>
```

この例では、HBox コンテナのサイズを明示的に 75x75 ピクセルに設定しています。これは読み込むイメージに比べてかなり小さいサイズです。HBox コンテナでサイズ制限を設定しなければ、イメージを格納できるだけのデフォルトサイズが使用されます。

デフォルトでは、スクロールバーは、コンテナの内容がコンテナより大きい場合にのみ描画されます。コンテナのスクロールバーを常に描画するには、`horizontalScrollPolicy` プロパティと `verticalScrollPolicy` プロパティを `on` に設定します。

次の例は、内部のイメージ全体をスクロールバーなしで表示できる場合でも、HBox コンテナをスクロールバー付きで作成しています。

```
<?xml version="1.0"?>
<!-- containers\intro\HBoxScrollOn.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HBox horizontalScrollPolicy="on" verticalScrollPolicy="on">
        <mx:Image source="assets/logo.jpg"/>
    </mx:HBox>
</mx:Application>
```

コンテナのスクロールプロパティの使用

コンテナの次のプロパティとスタイルは、スクロールバーの外観と動作を制御します。

- horizontalScrollPolicy プロパティと verticalScrollPolicy プロパティは、スクロールバーの表示を制御します。デフォルトでは、いずれのプロパティも auto に設定されており、必要なときにだけスクロールバーが表示されます。スクロールバーを常に表示するには、これらのプロパティを on に設定します。反対に、スクロールバーを一切表示したくない場合は、off に設定します。ActionScript では、ScrollPolicy クラスの定数 (ScrollPolicy.ON など) を使用してこれらの値を表すことができます。
- horizontalLineScrollSize プロパティと verticalLineScrollSize プロパティは、ユーザーがスクロールバーの矢印をクリックしたときにスクロールするピクセル数を決定します。デフォルト値は 5 ピクセルです。
- horizontalPageScrollSize プロパティと verticalPageScrollSize プロパティは、ユーザーがスクロールバーのトラックをクリックしたときにスクロールするピクセル数を決定します。デフォルト値は 20 ピクセルです。

メモ

clipContent プロパティが false の場合は、コンテナの境界線を越えて子が描画されます。したがって、スクロールバーは不要になり、horizontalScrollPolicy と verticalScrollPolicy が on に設定されていたとしても、スクロールバーは表示されません。

スクロールバーのレイアウトに関する注意事項

スクロールバーに対する設定が、アプリケーションのレイアウトに影響を及ぼす場合があります。たとえば、horizontalScrollPolicy プロパティと verticalScrollPolicy プロパティを on に設定すると、必要であるかどうかにかかわらず、コンテナのスクロールバーが常に表示されます。各スクロールバーの幅は 16 ピクセルです。したがって、不要なときにもスクロールバーを表示するように設定すると、コンテナの右と下のパディングのサイズをそれぞれ 16 ピクセル増やしたのと同じになります。

horizontalScrollPolicy プロパティと verticalScrollPolicy プロパティをデフォルト値の auto のままにした場合は、あたかもこれらのプロパティが off に設定されているかのようにアプリケーションがレイアウトされます。つまり、スクロールバーのサイズがレイアウトに影響することはありません。

この動作に留意しないと、アプリケーションの見た目が不適切になる場合があります。たとえば、30x100 ピクセルの HBox コンテナの中に 22x40 ピクセルのボタンが 2 つある場合、これらの子は HBox コンテナの中に完全に収まり、スクロールバーは表示されません。しかし、3 つ目のボタンを追加すると、子が HBox コンテナの幅を超えるため、コンテナの下部に水平スクロールバーが追加されます。スクロールバーの高さは 16 ピクセルなので、コンテンツ領域の高さが 30 ピクセルから 14 ピクセルに減少します。つまり、高さ 22 ピクセルの Button コントロールの高さ全体を表示するだけの領域が HBox に残っていないため、デフォルトでは、垂直スクロールバーが表示されます。

スクロールの遅延と時間間隔の制御

スクロールバーには、スクロールの方法に影響を与える 2 つのスタイルがあります。

- repeatDelay スタイルでは、ユーザーがスクロールボタンをクリックしてから、実際にスクロールが開始されるまでのミリ秒数を指定します。
- repeatInterval スタイルでは、ユーザーがスクロール矢印を押し続けている間、連続して発生するスクロールの間隔をミリ秒数で指定します。

これらの設定は、コンテナではなくスクロールバーサブコントロールのスタイルです。したがって、horizontalScrollPolicy などのプロパティとは異なる取り扱いが必要となります。次の例では、アプリケーション内のすべてのスクロールバーに対して一貫したスクロールポリシーを設定しています。

```
<?xml version="1.0"?>
<!-- containers\intro\HBoxScrollDelay.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        HScrollBar, VScrollBar {
            repeatDelay: 2000;
            repeatInterval:1000;
        }
```

```
    </mx:Style>

    <mx:HBox id="hb1" width="75" height="75">
        <mx:Image source="adobe_logo.gif"/>
    </mx:HBox>
</mx:Application>
```

この例を実行すると、477 ページの「スクロールバーの使用」と同じスクロール可能なロゴが表示されますが、スクロールバーの動作は異なります。ユーザーがスクロールバー矢印またはスクロールバートラックの上でマウスボタンをクリックしてそのまま押し続けると、イメージは最初1回スクロールし、2秒待ってから、1秒に1行または1ページの割合でスクロールします。

1つのスクロールバーに対してスタイルを設定するには、スクロールバーを持つアプリケーションまたはコントロールの initialize イベントのイベントリスナーで次のような行を使用します。

```
ScrollBar(hb1.horizontalScrollBar).setStyle("repeatDelay", 2000);
```

この例の hb1 は HBox コントロールです。どのコンテナにも、スクロールバーが存在していれば、ScrollBar サブコントロールを表す horizontalScrollBar プロパティと verticalScrollBar プロパティがあります。これらのプロパティの型は ScrollBar クラスではなく IScrollBar インターフェイスなので、ScrollBar クラスにキャストする必要があります。

Flex 座標の使用

Adobe Flash と Flex は、異なる目的のために次の 3 つの座標系をサポートしています。

- グローバル
- ローカル
- コンテンツ

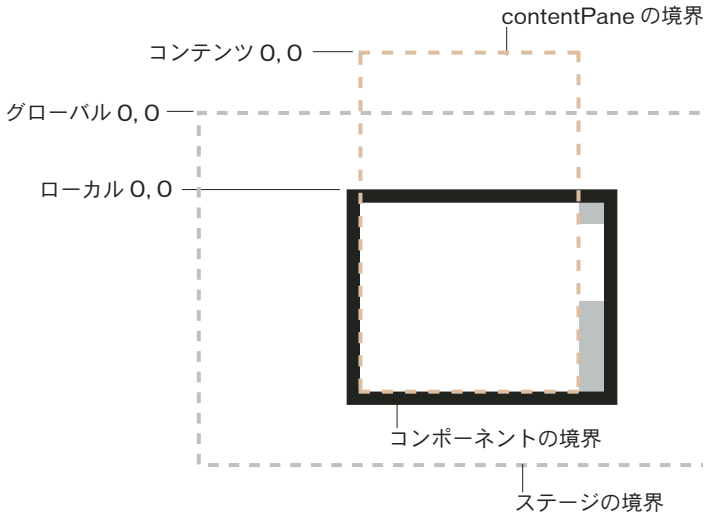
以降のセクションでは、これらの座標系について説明し、各座標系をどういう場合に使用すればよいかを示します。また、Flex プロパティおよびメソッドを使用して、どういう場合に、どのように座標系を変換すればよいかも示します。

座標系について

次の表に座標系の説明を示します。

座標系	説明
グローバル	この座標系は、Adobe Flash Player の Stage、つまりアプリケーションの最も外側の端の、左上隅を原点とします。 グローバル座標系は、コンポーネントのコンテキストから独立した、座標の全体集合を提供します。この座標系を使用する場面としては、オブジェクト間の距離を測定するときや、サブコントロールに対する相対座標から親コントロールに対する相対座標に座標を変換する際の中間点として、などが挙げられます。 MouseEvent クラスに、`stageX` および `stageY` というグローバル座標系のプロパティがあります。
ローカル	この座標系は、コンポーネントの左上隅を原点とします。 ローカル座標系は、マウスポインタの位置に使用されます。どのコンポーネントにも、ローカル座標系を使用する `mouseX` プロパティと `mouseY` プロパティが存在します。 MouseEvent クラスに、localX および localY というローカル座標系のプロパティがあります。また、Drag Manager はドラッグ & ドロップ操作にローカル座標を使用します。たとえば、`doDrag()` メソッドの `xOffset` プロパティと `yOffset` プロパティは、ローカル座標を基準とするオフセットです。
コンテンツ	この座標系は、コンポーネントの内容の左上隅を原点とします。ローカル座標やグローバル座標とは異なり、コンテンツ座標にはコンポーネントのコンテンツ領域がすべて含まれます。これには、現在クリッピングされていて、コンポーネントをスクロールしなければアクセスできない領域も含まれます。したがって、Canvas コンテナを下方向に 100 ピクセルスクロールした場合、表示内容の左上隅をコンテンツ座標で表すと (0, 100) になります。 コンテンツ座標系は、絶対配置を使用するコンテナの子の位置を取得または設定するときに使用します。絶対配置の詳細については、210 ページの「コンポーネントの位置設定について」を参照してください。 UIComponent の `contentMouseX` プロパティと `contentMouseY` プロパティは、マウスポインタの位置をコンテンツ座標系で返します。

次の図は、各座標系が互いにどのような関係にあるかを示します。



座標プロパティおよびメソッドの使用

場合によっては、位置の座標系を変換しなければならないことがあります。たとえば、次のような場合に座標の変換が必要になります。

- MouseEvent クラスには、イベントターゲットのマウス位置をグローバル座標系で提供するプロパティとローカル座標系で提供するプロパティがあります。Canvas コンテナ、または絶対配置を使用する Application コンテナまたは Panel コンテナでは、位置の指定にコンテンツ座標を使用します。マウスイベントの位置を、可視領域ではなく Canvas コンテナのコンテンツ領域内で特定するには、コンテンツ座標系で位置を特定する必要があります。
- カスタムドラッグ & ドロップハンドラでオブジェクト固有のドラッグアクションを判断する際、ローカル座標系とコンテンツ座標系間の座標変換が必要となる場合があります。たとえば、スクロールバーの付いたコントロールがあり、そのコンポーネントの内容にドラッグしたときのマウス位置を知りたいときなどがそれに該当します。484 ページの「例 : Canvas コンテナでのマウス位置の使用」の例は、この使用法を示しています。
- スクロールバーや仕切りなどのビジュアルエレメントとコンテンツエレメントをどちらも含むカスタムレイアウトコンテナ。たとえば、子の間に線を描画するカスタムコンテナがある場合は、線を描画するために、各々の子の位置をコンテナのコンテンツ座標で表す必要があります。

イベントハンドラでマウス座標を使用することがよくありますが、その場合は次の点に留意してください。

- マウスイベントを処理するときは、可能であれば MouseEvent オブジェクトの座標を使用するのが最適です。その理由は、MouseEvent オブジェクトの座標はイベントが生成されたときのマウス座標を表すためです。コンテナの `contentMouseX` プロパティと `contentMouseY` プロパティを使用してマウスポインタの位置をコンテンツ座標系で取得することもできますが、そうではなく、イベントオブジェクトからローカル座標値を取得し、それをコンテンツ座標系に変換するようにしてください。
- イベントオブジェクトに格納されたローカル座標 (MouseEvent の `localX` プロパティと `localY` プロパティなど) を使用するときは、イベントプロパティに格納されているマウスのローカル座標はイベントターゲットに対する相対値であることを覚えておく必要があります。ターゲットコンポーネントがコンポーネント自体ではなく、たとえば Button コンポーネントの中の UITextField のように、位置を特定するコンポーネントのサブコンポーネントである場合があります。そのような場合には、ローカル座標をいったんグローバル座標系に変換してから、そのグローバル座標をコンテンツ座標のコンテナに変換する必要があります。

座標変換プロパティおよびメソッド

すべての Flex コンポーネントには、座標系の使用とその変換に使用できる 2 つの読み取り専用プロパティと 6 つの関数があります。これらのプロパティと関数を次の表に示します。

プロパティまたは関数	説明
`contentMouseX`	マウスの x 座標をコンポーネントのコンテンツ座標で返します。
`contentMouseY`	マウスの y 座標をコンポーネントのコンテンツ座標で返します。
`contentToGlobal (point:Point):Point`	x 座標と y 座標を含む Point オブジェクトをコンテンツ座標系からグローバル座標系に変換します。
`contentToLocal (point:Point):Point`	Point オブジェクトをコンテンツ座標系からコンポーネントのローカル座標系に変換します。
`globalToContent (point:Point):Point`	Point オブジェクトをグローバル座標系からコンポーネントのコンテンツ座標系に変換します。
`globalToLocal (point:Point):Point`	Point オブジェクトをグローバル座標系からコンポーネントのローカル座標系に変換します。
`localToContent (point:Point):Point`	Point オブジェクトをローカル座標系からコンポーネントのコンテンツ座標系に変換します。
`localToGlobal (point:Point):Point`	Point オブジェクトをローカル座標系からグローバル座標系に変換します。

例 : Canvas コンテナでのマウス位置の使用

次の例は、localToGlobal() メソッドと globalToContent() メソッドを使用して、複数の子 Canvas コンテナを含む Canvas コンテナ内のマウスポインタの位置を特定する方法を示します。

この例は多少作為的であり、実際のコードではマウス位置をグローバル座標系で表す MouseEvent クラスの stageX プロパティと stageY プロパティを使うのが普通です。この例では代わりに localX プロパティと localY プロパティを使用して、ローカル座標とコンテンツ座標間の変換方法 (たとえば、最初の変換からグローバル座標を使用するまでの間、どのようにして正しい基準座標系を維持しているか) を示しています。

```
<?xml version="1.0"?>
<!-- containers\intro\MousePosition.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">

    <mx:Script>
        <![CDATA[
          import mx.controls.Alert;

          // Handle the mouseDown event generated
          // by clicking in the application.
          private function handleMouseDown(event:MouseEvent):void {

            // Convert the mouse position to global coordinates.
            // The localX and localY properties of the mouse event contain
            // the coordinates at which the event occurred relative to the
            // event target, typically one of the
            // colored internal Canvas controls.
            // A production version of this example could use the stageX
            // and stageY properties, which use the global coordinates,
            // and avoid this step.
            // This example uses the localX and localY properties only to
            // illustrate conversion between different frames of reference.
            var pt:Point = new Point(event.localX, event.localY);
            pt = event.target.localToGlobal(pt);

            // Convert the global coordinates to the content coordinates
            // inside the outer c1 Canvas control.
            pt = c1.globalToContent(pt);

            // Figure out which quadrant was clicked.
            var whichColor:String = "border area";

            if (pt.x < 150) {
                if (pt.y < 150)
                    whichColor = "red";
                else
                    whichColor = "blue";
```

```
            }
            else {
                if (pt.y < 150)
                    whichColor = "green";
                else
                    whichColor = "magenta";
            }

            Alert.show("You clicked on the " + whichColor);
          }
        ]]>
    </mx:Script>

    <!-- Canvas container with four child Canvas containers -->
    <mx:Canvas id="c1"
        borderStyle="none"
        width="300" height="300"
        mouseDown="handleMouseDown(event);">

        <mx:Canvas
            width="150" height="150"
            x="0" y="0"
            backgroundColor="red">
            <mx:Button label="I'm in Red"/>
        </mx:Canvas>
        <mx:Canvas
            width="150" height="150"
            x="150" y="0"
            backgroundColor="green">
            <mx:Button label="I'm in Green"/>
        </mx:Canvas>
        <mx:Canvas
            width="150" height="150"
            x="0" y="150"
            backgroundColor="blue">
            <mx:Button label="I'm in Blue"/>
        </mx:Canvas>
        <mx:Canvas
            width="150" height="150"
            x="150" y="150"
            backgroundColor="magenta">
            <mx:Button label="I'm in Magenta"/>
        </mx:Canvas>
    </mx:Canvas>
</mx:Application>
```

コンポーネントインスタンスの実行時の作成と管理

アプリケーションのユーザーインターフェイスをレイアウトする場合は MXML を、アプリケーションのイベント処理や実行時の制御を行う場合は ActionScript を使用するのが一般的です。また、ActionScript を使用して、コンポーネントのインスタンスを実行時に作成することも可能です。たとえば、MXML で空の Accordion コンテナを定義し、ActionScript を使用して、ユーザーアクションに応答する形で Accordion コンテナにパネルを追加できます。

表示リストとコンテナの子について

Flash Player は、アプリケーションを構成する可視オブジェクト (または可視になる可能性のあるオブジェクト) のツリーを維持します。ツリーのルートは Application オブジェクトで、子コンテナとコンポーネントがツリーのブランチとリーフノードになります。このツリーのことを " 表示リスト " と呼びます。子コンポーネントをコンテナに追加するとき、またはコンテナから削除するときは、表示リストに対して追加または削除操作を行うことになります。また、表示リスト内での子コンポーネントの位置を変更することで、子コンポーネントの相対的な位置を変更することもできます。

表示リストはアプリケーションの最上位をルートとするツリーですが、ActionScript でコンテナのメソッドとプロパティを使用してコンテナの子を操作するときは、コンテナの直接の子にアクセスするだけであり、それらをリスト内のインデックス付きアイテムとして扱います。このインデックスは 0 から始まり、表示リスト内のコンテナの最初の子が 0 になります。

Container クラスには、表示リスト内のコンテナの直接の子コンポーネントの数を保持する `numChildren` プロパティがあります。たとえば、次の HBox コンテナには 2 つの子コンポーネントが含まれているので、その `numChildren` プロパティの値は 2 になります。

```
<mx:HBox id="myContainer">
  <mx:Button click="clickHandler();"/>
  <mx:TextInput/>
</mx:HBox>
```

実行時にコンテナの子コンポーネントにアクセスして変更するには、Container クラスの `addChild()`、`addChildAt()`、`getChildren()`、`getChildAt()`、`getChildByName()`、`removeAllChildren()`、`removeChild()`、`removeChildAt()` の各メソッドを使用します。たとえば、次の例のように、コンテナのすべての子コンポーネントを反復処理できます。

```
private function clickHandler():void {
  var numChildren:Number = myContainer.numChildren;
  for (var i:int = 0; i < numChildren; i++) {
```

```
        trace(myContainer.getChildAt(i));
    }
}
```

Container クラスでは、コンテナのすべての子を含む完全な表示リストを保持する rawChildren プロパティも定義されています。このリストには、コンテナのすべての子に加えて、コンテナの境界線や背景イメージなどの " クロム " (表示エレメント) を実装する DisplayObject も含まれています。詳細については、488 ページの「表示専用の子へのアクセス」を参照してください。

コンテナまたはアプリケーション内の子コンポーネントの数の取得

コンテナに含まれる直接の子コンポーネントの数を取得するには、コンテナの numChildren プロパティの値を取得します。次のアプリケーションは、アプリケーションと VBox コンテナに含まれる子の数を取得します。VBox コントロールには 5 つのラベルコントロールが含まれているため、子の数は 5 になります。Application コンテナにはその子として VBox コントロールと Button コントロールが含まれているため、子の数は 2 になります。

```
<?xml version="1.0"?>
<!-- containers\intro\VBoxNumChildren.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
      <![CDATA[

            // Import the Alert class.
            import mx.controls.Alert;

            public function calculateChildren():void {
                var myText:String = new String();
                myText="The VBox container has " +
                    myVBox.numChildren + " children";
                myText+="\nThe Application has " +
                    numChildren + " children";
                Alert.show(myText);
            }
        ]]>
    </mx:Script>

    <mx:VBox id="myVBox" borderStyle="solid">
        <mx:Label text="This is label 1"/>
        <mx:Label text="This is label 2"/>
        <mx:Label text="This is label 3"/>
        <mx:Label text="This is label 4"/>
        <mx:Label text="This is label 5"/>
    </mx:VBox>

    <mx:Button label="Show Children" click="calculateChildren();"/>
</mx:Application>
```

<mx:Application> タグを含むメインの MXML アプリケーションファイルでは、現在のスコープは常に Application オブジェクトになります。したがって、オブジェクト接頭辞の付いていない numChildren プロパティへの参照は、Application オブジェクトの numChildren プロパティを参照します。ルートアプリケーションへのアクセスの詳細については、66 ページの「スコープについて」を参照してください。

表示専用の子へのアクセス

numChildren プロパティと getChildAt() メソッドでカウントまたはアクセスできるのは、子コンポーネントのみです。コンテナには、境界線や背景などのスタイルエレメントやスキンが含まれていることがあります。コンテナの rawChildren プロパティを使用すると、コンポーネントの " 内容を表す子 " とスキンおよびスタイルの " 表示されるだけの子 " を含む、コンテナのすべての子にアクセスできます。rawChildren プロパティによって返されるオブジェクトは IChildList インターフェイスを実装します。このインターフェイスのメソッドとプロパティ (getChildAt() など) を使用することで、コンテナのすべての子にアクセスし、操作できます。

コンポーネントの実行時の作成と削除

コンポーネントのインスタンスを実行時に作成するには、インスタンスを定義し、任意のプロパティを設定します。次に、親コンテナの addChild() メソッドを呼び出して、親コンテナの子としてそのインスタンスを追加します。このメソッドのシグネチャは次のとおりです。

```
addChild(child:DisplayObject):DisplayObject
```

child パラメータは、コンテナに追加するコンポーネントを指定します。

> **メモ** このメソッドの child パラメータは DisplayObject 型として指定されていますが、コンテナの子として追加するためには、このパラメータが IUIComponent インターフェイスを実装している必要があります。Flex コンポーネントはすべてこのインターフェイスを実装しています。

たとえば、次のアプリケーションは、myButton という名前の Button コントロールを持つ HBox コンテナを作成します。

```
<?xml version="1.0"?>
<!-- containers\intro\ContainerAddChild.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Button;

            public function addButton():void {
                var myButton:Button = new Button();
                myButton.label = "New Button";
                myHBox.addChild(myButton);
            }
```

```
        ]]>
    </mx:Script>

    <mx:HBox id="myHBox" initialize="addButton();"/>
</mx:Application>
```

この例では、ユーザーアクションに対する応答としてではなく、アプリケーションのロード時にコントロールを作成しています。ただし、次の例のように、ユーザーが既存のボタンを押したときに新しいコントロールを追加することもできます。

```
<?xml version="1.0"?>
<!-- containers\intro\ContainerAddChild2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="left">

    <mx:Script>
        <![CDATA[
            import mx.controls.Button;

            public function addButton():void {
                var myButton:Button = new Button();
                myButton.label = "New Button";
                myHBox.addChild(myButton);
            }
        ]]>
    </mx:Script>

    <mx:HBox id="myHBox">
        <mx:Button label="Add Button" click="addButton();"/>
    </mx:HBox>
</mx:Application>
```

次の図は、ユーザーが元 (一番左) のボタンを 3 回押した後のアプリケーションの状態を示しています。

コントロールをコンテナから削除するには、removeChild() メソッドを使用します。Flex が、削除した子の parent プロパティを設定します。removeChild() メソッドの呼び出し後に、削除した子がアプリケーションのどこからも参照されない場合、その子はガベージコレクションプロセスによって破棄されます。次の例は、ユーザーがボタンを押したときにそのボタンをアプリケーションから削除します。

```
<?xml version="1.0"?>
<!-- containers\intro\ContainerRemoveChild.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function removeButton():void {
                myHBox.removeChild(myButton);
```

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

    <mx:HBox id="myHBox">
        <mx:Button id="myButton"
            label="Remove Me"
            click="removeButton();"/>
    </mx:HBox>
</mx:Application>
```

コンテナの子に対して使用できるその他のメソッドについては、『Adobe Flex 2 リファレンスガイド』の Container クラスを参照してください。

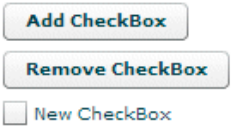
例 : VBox コンテナの子の作成と削除

次の例では、MXML を使用して、2 つの Button コントロールを含む VBox コンテナを定義しています。2 つの Button コントロールは、VBox コンテナに CheckBox コントロールを追加するためのボタンと、VBox コンテナから CheckBox コントロールを削除するためのボタンとして使用します。

```
<?xml version="1.0"?>
<!-- containers\intro\ContainerComponentsExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
      <![CDATA[

        // Import the CheckBox class.
        import mx.controls.CheckBox;

        // Define a variable to hold the new CheckBox control.
        private var myCheckBox:CheckBox;

        // Define a variable to track if the CheckBox control
        // is in the display list.
        private var checkBoxDisplayed:Boolean = false;

        public function addCB():void {
            // Make sure the check box isn't being displayed.
            if(checkBoxDisplayed==false){
                // Create the check box if it does not exist.
                if (!myCheckBox) {
                    myCheckBox = new CheckBox();
                }

                // Add the check box.
                myCheckBox.label = "New CheckBox";
                myVBox.addChild(myCheckBox);
                checkBoxDisplayed=true;
            }
```

```
        }

        public function delCB():void {
            // Make sure a CheckBox control exists.
            if(checkBoxDisplayed){
                myVBox.removeChild(myCheckBox);
                checkBoxDisplayed=false;
            }
        }
      ]]>
    </mx:Script>

    <mx:VBox id="myVBox">
        <mx:Button label="Add CheckBox"
            click="addCB();"/>
        <mx:Button label="Remove CheckBox"
            click="delCB();"/>
    </mx:VBox>
</mx:Application>
```

次の図は、ユーザーが [Add CheckBox] ボタンを押した後のアプリケーションの状態を示しています。



例 : Accordion コンテナの子の作成と削除

このセクションでは、Accordion コンテナにパネルを追加および削除する例を示します。初期状態では、Accordion コンテナにパネルが 1 つだけ存在します。[Add HBox] ボタンをクリックするたびに、Accordion コンテナに新しい HBox コンテナが追加されます。

```
<?xml version="1.0"?>
<!-- containers\intro\ContainerComponentsExample2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Import HBox class.
            import mx.containers.HBox;

            //Array of created containers
            private var hBoxes:Array = [];

            public function addHB():void {
                // Create new HBox container.
                var newHB:HBox = new HBox();
```

```
                newHB.label="my label: " + String(hBoxes.length);

                // Add it to the Accordion container, and to the
                // Array of HBox containers.
                hBoxes.push(myAcc.addChild(newHB));
            }

            public function delHB():void {
                // If there is at least one HBox container in the Array,
                // remove it.
                if (hBoxes.length>= 1) {
                    myAcc.removeChild(hBoxes.pop());
                }
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Accordion id="myAcc" height="150" width="150">
            <mx:HBox label="Initial HBox"/>
        </mx:Accordion>

        <mx:Button label="Add HBox" click="addHB();"/>
        <mx:Button label="Remove HBox" click="delHB();"/>
    </mx:VBox>
</mx:Application>
```

子の順序の制御

子の順序は、特定の順序で子を追加することによって制御できます。その他にも次のような方法があります。

- addChildAt() メソッドを使用して、コンポーネントの既存の子の中のどこに新しい子を追加するかを指定する
- setChildIndex() メソッドを使用して、表示リスト内のあるコンポーネントの子の中での特定の子の位置を指定する

メモ

addChild() メソッドと同様に、addChildAt() メソッドの child パラメータも DisplayObject 型として指定されていますが、コンテナの子として追加するためには、このパラメータが IUIComponent インターフェイスを実装している必要があります。Flex コンポーネントはすべてこのインターフェイスを実装しています。

次の例は、490 ページの「例 : VBox コンテナの子の作成と削除」の例を修正したものです。ここでは、addChildAt() メソッドを使用して、VBox の最初の子 (インデックス 0) として CheckBox コントロールを追加しています。また、setChildIndex() メソッドを使用する Reorder 子ボタンがあり、このボタンをクリックすると、CheckBox コントロールが VBox コンテナの最後の子になるまで表示リスト内を下方に移動します。ボールド体のテキストは、前の例に追加された行、または前の例から変更された行を示します。

```
<?xml version="1.0"?>
<!-- containers\intro\ContainerComponentsReorder.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
      <![CDATA[

        // Import the CheckBox and Alert classes.
        import mx.controls.CheckBox;
        import mx.controls.Alert;

        // Define a variable to hold the new CheckBox control.
        private var myCheckBox:CheckBox;

        // Define a variable to track if the CheckBox control
        // is in the display list.
        private var checkBoxDisplayed:Boolean = false;

        public function addCB():void {
            // Make sure the check box isn't being displayed.
            if(checkBoxDisplayed==false){
                // Create the check box if it does not exist.
                if (!myCheckBox) {
                    myCheckBox = new CheckBox();
                }
                // Add the check box as the first child of the VBox.
                myCheckBox.label = "New CheckBox";
                myVBox.addChildAt(myCheckBox, 0);
                checkBoxDisplayed=true;
            }
        }

        public function delCB():void {
            // Make sure a CheckBox control exists.
            if(checkBoxDisplayed){
                myVBox.removeChild(myCheckBox);
                checkBoxDisplayed=false;
            }
        }

        public function reorder():void {
        // Make sure a CheckBox control exists.
```

```
            if(checkBoxDisplayed==true){
                // Don't try to move the check box past the end
                // of the children. Because indexes are 0 based,
                // the last child index is one less
                // than the number of children.
                if (myVBox.getChildIndex(myCheckBox) < myVBox.numChildren-1)
                {
                    // Increment the checkBoxIndex variable and use it to
                    // set the index of the check box among the VBox children.
                    myVBox.setChildIndex(myCheckBox,
                        myVBox.getChildIndex(myCheckBox) + 1);
                }
            }
            else {
                Alert.show("Add the check box before you can move it");
            }
        }
      ]]>
    </mx:Script>

    <mx:VBox id="myVBox">
        <mx:Button label="Add CheckBox" click="addCB();"/>
        <mx:Button label="Remove CheckBox" click="delCB();"/>
        <mx:Button label="Reorder children" click="reorder();"/>
    </mx:VBox>
</mx:Application>
```

第 14 章

14 Application コンテナの使用

Adobe Flex には、別途コンテナを定義しなくても内容を追加するだけでアプリケーションの開発ができるデフォルトの Application コンテナが定義されています。このトピックでは、Application コンテナの使用方法について説明します。

Flex では、<mx:Application> タグが含まれている MXML ファイルをすべて Application オブジェクトとして定義します。詳細については、502 ページの「Application オブジェクトについて」を参照してください。

Application コンテナは SWF ファイルのダウンロード状況をプログレスバーで表示するアプリケーションプリローダーをサポートします。デフォルトのプログレスバーをオーバーライドすることによって、独自のプログレスバーを定義できます。詳細については、507 ページの「アプリケーションのダウンロード状況の表示」を参照してください。

目次

Application コンテナの使用

Flex では、アプリケーションに追加したすべての内容を格納するデフォルトコンテナとして、Application コンテナが定義されています。Application コンテナは <mx:Application> タグから作成されます。このタグは MXML アプリケーションファイルの最初のタグである必要があります。Application オブジェクトはファイル内のすべての ActionScript コードのデフォルトスコープであり、<mx:Application> タグはアプリケーションの初期サイズを定義します。

Application コンテナを、アプリケーション内で使用する唯一のコンテナにした方が好都合のようにも思われますが、通常は、アプリケーションにコントロールを追加する前に、少なくともう 1 つのコンテナを明示的に定義することになります。<mx:Application> タグに続く最初のコンテナとしては、Panel コンテナがよく使用されます。

Application コンテナには、次のデフォルトレイアウト属性があります。

プロパティ	デフォルト値
デフォルトサイズ	ブラウザウィンドウのサイズ
子の整列方法	垂直に整列
子の水平方向の整列方法	中央揃え
デフォルトパディング	top、bottom、left、right ともに 24 ピクセル

Application コンテナとその子のサイズ設定

Application コンテナでは子が縦1列に配置されます。Application コンテナの height と width は、ピクセル値で明示的に設定する方法の他に、ブラウザウィンドウのサイズに対する相対的なパーセント値を使用する方法もあります。デフォルトでは、Application コンテナの高さと幅は 100%、つまり、ブラウザのウィンドウ全体を占めるように設定されます。

次の例は、Application コンテナのサイズを、ブラウザウィンドウの幅と高さの 1/2 に設定します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   height="50%" width="50%">
   ...
</mx:Application>
```

サイズをパーセント値で指定することの利点は、ユーザーがブラウザウィンドウのサイズを変更したときに、それに応じてアプリケーションのサイズが自動的に調整される点です。ブラウザウィンドウのサイズが変更されたとしても、そのサイズに対する比率として Application コンテナのサイズを維持できます。

MXML タグで子コンポーネントの width プロパティと height プロパティをパーセント値に設定した場合、コンポーネントのサイズがアプリケーションのサイズ変更に合わせて拡大 / 縮小されます。次にその例を示します。

```
<?xml version="1.0"?>
<!-- containers\application\AppSizePercent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%">

    <mx:Panel title="Main Application" width="100%" height="100%">
        <mx:HDividedBox width="100%" height="100%">
            <mx:TextArea width="50%" height="100%"/>
            <mx:VDividedBox width="50%" height="100%">
                <mx:DataGrid width="100%" height="25%"/>
                <mx:TextArea width="100%" height="75%"/>
            </mx:VDividedBox>
        </mx:HDividedBox>
    </mx:Panel>
</mx:Application>
```

次のコードは、Application コンテナのサイズに明示的にピクセル値を設定した例です。

```
<?xml version="1.0"?>
<!-- containers\application\AppSizePixel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="100" width="150">

    <mx:Panel title="Main Application">
        <mx:TextInput id="mytext" text="Hello"/>
        <mx:Button id="mybutton" label="Get Weather"/>
    </mx:Panel>
</mx:Application>
```

前の例のように、一部またはすべてのコンポーネントが Application コンテナの可視領域の外側に位置するように Application コンテナの子のサイズと位置が設定された場合は、Application コンテナにスクロールバーが追加されます。

子コンテナが Application コンテナ全体を占めるようにする場合、最も簡単な方法は、MXML タグで子の `width` プロパティと `height` プロパティを 100% に設定し (ActionScript を使用する場合は、`percentWidth` プロパティと `percentHeight` プロパティを 100 に設定します)、Application コンテナのパディングを 0 に設定することです。Application コンテナの幅と高さに基づいて子コンテナの `width` プロパティと `height` プロパティを設定する場合は、Application コンテナのパディングを差し引かないと、子コンテナのサイズが使用可能な領域よりも大きくなり、アプリケーションにスクロールバーが表示されてしまいます。

次の例では、Application コンテナのパディングで定義されている領域を除く使用可能な領域全体に VBox コンテナが拡張されます。

```
<?xml version="1.0"?>
<!-- containers\application\AppVBoxSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100" height="100">

    <mx:VBox width="100%" height="100%" backgroundColor="#A9C0E7">
        <!-- ... -->
    </mx:VBox>
</mx:Application>
```

次の例では、VBox コンテナが Application コンテナの使用可能領域より大きいため、スクロールバーが表示されます。

```
<?xml version="1.0"?>
<!-- containers\application\AppVBoxSizeScroll.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100" height="100">

    <mx:VBox width="200" height="200" backgroundColor="#A9C0E7">
        <!-- ... -->
    </mx:VBox>
</mx:Application>
```

次の例では、Application コンテナのパディングがすべて 0 に設定されているため、その子である VBox コンテナがウィンドウ全体を占めます。

```
<?xml version="1.0"?>
<!-- containers\application\AppNoPadding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100" height="100"
    paddingTop="0" paddingBottom="0"
    paddingLeft="0" paddingRight="0">

    <mx:VBox width="100" height="100" backgroundColor="#A9C0E7">
        <!-- ... -->
    </mx:VBox>
</mx:Application>
```

Application コンテナのデフォルトスタイルのオーバーライド

デフォルトでは、Application コンテナには、Flex アプリケーションの次の視覚的要素を定義したデフォルトスタイルプロパティがあります。これらのプロパティはコンテナのデフォルト値とは異なります。

プロパティ	デフォルト値
backgroundColor	Adobe Flash Player のステージ領域の色。アプリケーションのロードおよび初期化中に表示されます。また、アプリケーションの背景が透明の場合にも、この色が表示されます。デフォルト値は 0x869CA7 です。
backgroundGradientAlphas	[1.0, 1.0]。完全に不透明な背景。
backgroundGradientColors	[0x9CB0BA, 0x68808C]。下部の方がわずかに暗いグレーの背景。
backgroundImage	backgroundGradientAlphas スタイルと backgroundGradientColors スタイルによって制御されるグラデーション。デフォルト値は mx.skins.halo.ApplicationBackground です。
backgroundSize	100%. このプロパティを 100% に設定すると、背景イメージが Application コンテナ全体を占めます。
horizontalAlign	中央揃え
paddingBottom	24 ピクセル
paddingLeft	24 ピクセル
paddingRight	24 ピクセル
paddingTop	24 ピクセル

これらのデフォルト値をオーバーライドして、独自のデフォルトスタイルプロパティを定義することもできます。

Application コンテナの背景の変更

Application コンテナの背景は、コンテナの backgroundGradientAlphas、backgroundGradientColors、backgroundImage の各スタイルによって制御されます。デフォルトでは、これらのプロパティは不透明なグレーのグラデーション背景を定義します。

アプリケーションの背景としてイメージを指定するには、backgroundImage プロパティを使用します。backgroundImage プロパティと backgroundGradientColors プロパティを両方とも設定すると、backgroundGradientColors が無視されます。

アプリケーションにグラデーション背景を指定する方法は 2 とおりあります。

- backgroundGradientColors プロパティに 2 つの値を設定する。次に例を示します。

  ```
  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundGradientColors="[0x0000FF, 0xCCCCCC]">
  ```

  指定した 2 つの値の間のグラデーションパターンが自動的に計算されます。

- backgroundColor プロパティを目的の値に設定する。次に例を示します。

  ```
  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="red">
  ```

  赤より少し暗い色と少し明るい色との間のグラデーションパターンが自動的に計算されます。

単色の背景をアプリケーションに設定するには、backgroundGradientColors プロパティに同じ値を 2 つ指定します。次に例を示します。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  backgroundGradientColors="[#FFFFFF, #FFFFFF]">
```

この例では、単色の白の背景を定義しています。

backgroundColor プロパティは、アプリケーションのロードおよび初期化中に表示される、Flash Player のステージ領域の背景色と、アプリケーション実行時の背景のグラデーションを指定します。デフォルトでは、backgroundColor プロパティは 0x869CA7 (ダークブルーグレー) に設定されます。

backgroundGradientColors プロパティを使用してアプリケーションの背景を設定する場合は、backgroundColor プロパティも一緒に設定して backgroundGradientColors プロパティを補完してください。次に例を示します。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  backgroundGradientColors="[0x0000FF, 0xCCCCCC]"
  backgroundColor="0x0000FF">
```

この例では、backgroundGradientColors プロパティを使用してダークブルーからグレーへのグラデーションパターンを設定し、backgroundColor プロパティを使用して Flash Player のステージ領域をダークブルーに設定しています。この色はアプリケーションのロードおよび初期化中に表示されます。

プレーンスタイルの使用

Flex のデフォルトスタイルシートにはプレーンスタイル名が定義されています。このスタイルは、すべてのパディングを 0 ピクセルに設定し、デフォルトの背景イメージを除去し、背景色を白に設定して、子を左揃えにします。次の例は、このスタイルの設定方法を示しています。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  styleName="plain">
```

次のように、プレーン設定の個々の値をオーバーライドすることもできます。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  styleName="plain" horizontalAlign="center"/>
```

Style タグを使用したスタイルのオーバーライド

また、アプリケーションで <mx:Style> タグを使用して、別のスタイル値を指定することもできます。次にその例を示します。

```
<?xml version="1.0"?>
<!-- containers\application\AppStyling.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Style definition for the entire application. -->
    <mx:Style>
        Application {
            paddingLeft: 10px;
            paddingRight: 10px;
            paddingTop: 10px;
            paddingBottom: 10px;
            horizontalAlign: "left";
            backgroundImage: "";
            backgroundColor: #AAAACC;
        }
    </mx:Style>

    <mx:Panel title="Main Application">
        <mx:TextInput id="mytext" text="Hello"/>
        <mx:Button id="mybutton" label="Get Weather"/>
    </mx:Panel>
</mx:Application>
```

この例では、背景イメージを除去し、すべてのパディングを 10 ピクセルに設定し、子を左揃えにして、背景色をライトブルーに設定しています。

スタイルの使用の詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

アプリケーションのソースコードの表示

Application コンテナの viewSourceURL プロパティを使用すると、アプリケーションのソースコードの URL を指定できます。このプロパティを設定すると、アプリケーションのコンテキストメニューに [ソースの表示] メニューアイテムが追加されます。このコンテキストメニューは、アプリケーションの任意の場所でマウスを右クリックすると開きます。[ソースの表示] メニューアイテムを選択すると、新しいブラウザウィンドウが開き、viewSourceURL プロパティで指定された URL が表示されます。

viewSourceURL プロパティは、ActionScript ではなく、次の例のように MXML を使用して設定します。

```
<?xml version="1.0"?>
<!-- containers\application\AppSourceURL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="http://localhost:8100/flex/assets/AppSourceURL.txt">

    <mx:Button/>
</mx:Application>
```

通常はソースコードを MXML ファイルとしてではなく、テキスト ファイルまたは HTML ファイルとしてデプロイします。この例では、ソースコードはファイル AppSourceURL.txt ファイル内にあります。HTML ファイルを使用してソースコードを表す場合、フォーマットと色付けを追加してソースコードを読みやすくすることができます。

Application コンテナのオプションの指定

<mx:Application> タグのオプションを指定して、アプリケーションを制御できます。それらのオプションを次の表に示します。

オプション	データ型	説明
frameRate	Number	アプリケーションのフレームレートを 1 秒あたりのフレーム数で指定します。デフォルト値は 24 です。
pageTitle	String	ブラウザのタイトルバーに表示するストリングを指定します。このプロパティは HTML の <title> タグと同じ機能を提供します。
preloader	Path	カスタムプログレスバーを定義する、SWC コンポーネントクラスまたは ActionScript コンポーネントクラスのパスを指定します。 SWC コンポーネントは、MXML ファイルと同じディレクトリ、または Flex Web アプリケーションの "WEB-INF\flex\user_classes" ディレクトリに存在する必要があります。 詳細については、507 ページの「アプリケーションのダウンロード状況の表示」を参照してください。

オプション	データ型	説明
scriptRecursionLimit	Number	Flash Player がスタックをコールする最大深度を指定します。この深度を超えると Flash Player が停止します。これは実質的にスタックオーバーフローの制限になります。 デフォルト値は 1000 です。
scriptTimeLimit	Number	ActionScript イベントリスナーの最長実行時間を秒単位で指定します。この時間が過ぎると、このイベントリスナーは処理が停止しているものと見なされ、強制終了されます。 デフォルト値は 60 秒で、この値は設定可能な最大値でもあります。
usePreloader	Boolean	アプリケーションプリローダーを無効にするかどうかを指定します。無効にする場合は false、無効にしない場合は true にします。デフォルト値は true です。デフォルトプリローダーを使用するには、アプリケーションの幅を 160 ピクセル以上にする必要があります。 詳細については、507 ページの「アプリケーションのダウンロード状況の表示」を参照してください。

Application オブジェクトについて

アプリケーションは、<mx:Application> タグによって定義された 1 つの Application オブジェクトを含む SWF ファイルにコンパイルされます。ほとんどの場合、Flex アプリケーションに含まれる Application オブジェクトは 1 つだけです。アプリケーションによっては、SWFLoader コントロールを使用して複数のアプリケーションを追加するものもあります。

Application オブジェクトには、次の特性があります。

- Application オブジェクトは、<mx:Application> タグを含む MXML ファイルです。
- ほとんどの Flex アプリケーションは、Application オブジェクトを 1 つ持ちます。
- Application ファイルは、最初にロードされるファイルです。
- Application オブジェクトは Document オブジェクトでもありますが、Document オブジェクトは必ずしも Application オブジェクトではありません。Document オブジェクトの詳細については、503 ページの「Document オブジェクトについて」を参照してください。
- Application オブジェクトは、Flex アプリケーションのどこからでも、mx.core.Application.application として参照できます。
- 複数のネストされたアプリケーションを SWFLoader コントロールを使用してロードした場合は、parentApplication や parentApplication.parentApplication などを使用して、ネスト階層の上位の各アプリケーションのスコープにアクセスできます。

Document オブジェクトについて

Document オブジェクトは、Flex アプリケーションで使用されるすべての MXML ファイルに対して作成されます。たとえば、Application オブジェクトでもあるドキュメントから、カスタムコントロールを定義する他の MXML ファイルを使用できます。

Document オブジェクトには、次の特性があります。

- Flex アプリケーションで使用される MXML ファイルは、Application オブジェクトのファイルを含めて、すべて Document オブジェクトです。
- カスタム ActionScript コンポーネントファイルは Document オブジェクトです。
- Flex コンパイラは、`<mx:Application>` タグが含まれていないファイルから SWF ファイルをコンパイルすることはできません。
- 通常、ドキュメントは、Flex アプリケーションで使用される MXML カスタムコントロールで構成されます。
- ドキュメントの親ドキュメントのスコープにアクセスするには、`parentDocument` や `parentDocument.parentDocument` などを使用します。
- Flex に用意されている `UIComponent.isDocument` プロパティを使用すると、指定したオブジェクトが Document オブジェクトであるかどうかを検出できます。

Document オブジェクトおよび Application オブジェクトのスコープへのアクセス

アプリケーションのメイン MXML ファイル、つまり `<mx:Application>` タグを含むファイルでは、`this` キーワードを使用して Application オブジェクトのメソッドとプロパティにアクセスできます。ただし、カスタム ActionScript および MXML コンポーネント、イベントリスナー、または外部 ActionScript クラスファイルでは、Flex はそれらのコンポーネントおよびクラスのコンテキスト内で実行され、`this` キーワードは Application オブジェクトではなく現在の Document オブジェクトを参照します。親ドキュメントの場所を指定せずにいずれかの子ドキュメントからアプリケーション内のコントロールまたはメソッドを参照することはできません。

親ドキュメントにアクセスするには、次のプロパティを使用します。

mx.core.Application.application　最上位の Application オブジェクト。ドキュメントツリーのどの場所でオブジェクトが実行されているかは関係ありません。

mx.core.UIComponent.parentDocument　現在のドキュメントの親ドキュメント。`parentDocument.parentDocument` を使用して、複数のドキュメントを含むツリーをさかのぼることができます。

mx.core.UIComponent.parentApplication　現在のオブジェクトが存在する Application オブジェクト。Flex アプリケーションはアプリケーションを他のアプリケーションにロードできるので、このプロパティを使用して直接の親アプリケーションにアクセスできます。`parentApplication.parentApplication` を使用して、複数のアプリケーションを含むツリーをさかのぼることができます。

以降のセクションでは、これらのプロパティの使用方法について説明します。

mx.core.Application.application プロパティの使用

最上位の Application オブジェクトのプロパティとメソッドに、アプリケーションの任意の場所からアクセスするには、Application クラスの `application` プロパティを使用します。たとえば、次のコードのように、doSomething() メソッドを含むアプリケーションを定義します。

```
<?xml version="1.0"?>
<!-- containers\application\AppDoSomething.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:MyComps="myComponents.*">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;

            // Open an Alert control.
            public function doSomething():void {
                Alert.show("doSomething() called.");
            }
        ]]>
    </mx:Script>

    <!-- Include the ButtonMXML.mxml component. -->
    <MyComps:ButtonMXML/>
</mx:Application>
```

次に、次の例に示すように、ButtonMXML.mxml コンポーネントで `Application.application` プロパティを使用すると、doSomething() メソッドを参照できます。

```
<?xml version="1.0"?>
<!-- containers\application\myComponents\ButtonMXML.mxml -->
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // To refer to the members of the Application class,
            // you must import mx.core.Application.
            import mx.core.Application;
        ]]>
    </mx:Script>
```

```
    <mx:Button  label="MXML Button"
        click="Application.application.doSomething();"/>
</mx:HBox>
```

application プロパティは、それぞれがデータの共有セットを使用する1つまたは複数のカスタム MXML または ActionScript コンポーネントを含むアプリケーションにおいて特に役立ちます。アプリケーションレベルでは通常、共有の情報を保持し、どのコンポーネントからもアクセスできるユーティリティ関数を提供します。

たとえば、アプリケーションレベルでユーザー名を保持し、"Hi, userName" というストリングを返すユーティリティ関数 getSalutation() を実装するとします。次の "MyApplication.mxml" サンプルファイルに、getSalutation() メソッドを定義するアプリケーションのソースを示します。

```
<?xml version="1.0"?>
<!-- containers\application\AppSalutation.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:MyComps="myComponents.*">

    <mx:Script>
        <![CDATA[
            public var userName:String="SMG";
            public function getSalutation():String {
                return "Hi, " + userName;
            }
        ]]>
    </mx:Script>

    <!-- Include the ButtonGetSalutation.mxml component. -->
    <MyComps:ButtonGetSalutation/>

</mx:Application>
```

MXML コンポーネントで userName にアクセスし、getSalutation() メソッドを呼び出すには、次のように application プロパティを使用できます。次の例は "MyComponent.mxml" コンポーネントに記述されたコードです。

```
<?xml version="1.0"?>
<!-- containers\application\myComponents\ButtonGetSalutation.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%" >

    <mx:Script>
        <![CDATA[
            // To refer to the members of the Application class,
            // you must import mx.core.Application.
            import mx.core.Application;
        ]]>
    </mx:Script>

    <mx:Label id="myL"/>
    <mx:Button
```

```
        click="myL.text=Application.application.getSalutation();"/>
</mx:VBox>
```

この例では、Button コントロールをクリックすると、getSalutation() 関数が実行され、Label コントロールにテキストが格納されます。

parentDocument プロパティの使用

オブジェクトの親ドキュメントにアクセスするには、parentDocument プロパティを使用できます。親ドキュメントは現在のオブジェクトを含むオブジェクトです。UIComponent クラスを継承するすべてのクラスは、parentDocument プロパティを持ちます。

次の例では、アプリケーションはカスタム AccChildObject.mxml コンポーネントを参照します。

```
<?xml version="1.0"?>
<!-- containers\application\AppParentDocument.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:MyComps="myComponents.*">

    <!-- Include the AccChildObject.mxml component. -->
    <MyComps:AccChildObject/>

</mx:Application>
```

この例では、アプリケーションは AccChildObject.mxml コンポーネントの親ドキュメントです。次のコードは AccChildObject.mxml コンポーネントのコードです。ここでは、parentDocument プロパティを使用して、Application コンテナより少し小さい Accordion コンテナを定義しています。

```
<?xml version="1.0"?>
<!-- containers\application\myComponents\AccChildObject.mxml -->
<mx:Accordion xmlns:mx="http://www.adobe.com/2006/mxml"
    width="{parentDocument.width*.80}"
    height="{parentDocument.height*.50}">

    <mx:HBox/>

</mx:Accordion>
```

親ドキュメントのチェーンをさかのぼるには、MXML スクリプトで parentDocument プロパティを使用します。次の例のように、複数の parentDocument プロパティを使用して、ドキュメントオブジェクトチェーン内を移動することもできます。

```
parentDocument.parentDocument.doSomething();
```

Application オブジェクトの parentDocument プロパティは、アプリケーションへの参照です。

parentDocument は Object 型なので、祖先 Document オブジェクトのプロパティとメソッドにキャストせずにアクセスできます。

すべての UIComponent クラスは、isDocument() プロパティを持ちます。このプロパティは、その UIComponent クラスが Document オブジェクトである場合に true に設定され、そうでない場合に false に設定されます。

UIComponent クラスが Document オブジェクトの場合は documentDescriptor プロパティを持ちます。このプロパティは、生成された Document クラス内に生成される記述子ツリーの最上位記述子を参照します。

たとえば、"AddressForm.mxml" コンポーネントで、住所フォームを定義する Form コンテナのサブクラスを作成し、"MyApp.mxml" コンポーネントでそのインスタンスを 2 つ (<AddressForm id="shipping"> と <AddressForm id="billing">) 作成するとします。

この例では、shipping オブジェクトは Document オブジェクトです。その documentDescriptor プロパティは、"AddressForm.mxml" ファイル (コンポーネントの定義) の最上位の <mx:Form> タグに対応します。一方、記述子は、"MyApp.mxml" ファイル (コンポーネントのインスタンス) の <AddressForm id="shipping"> タグに対応します。

parentDocument プロパティを使用してドキュメントチェーン内を移動する操作は、parentApplication プロパティを使用してアプリケーションチェーン内を移動する操作に似ています。

parentApplication プロパティの使用

アプリケーションは他のアプリケーションをロードすることができるので、各アプリケーション内にドキュメントの階層を持つのと同じように、アプリケーションを階層化できます。すべての UIComponent クラスは、parentApplication 読み取り専用プロパティを持ちます。このプロパティは、オブジェクトが存在する Application オブジェクトを参照します。Application オブジェクトの parentApplication プロパティはそれ自体の参照ではなく、ロード先の Application オブジェクトか、または null (Application オブジェクトの場合) となります。

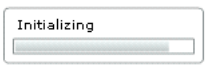
parentApplication プロパティを使用してアプリケーションチェーン内を移動する操作は、parentDocument プロパティを使用してドキュメントチェーン内を移動する操作に似ています。

アプリケーションのダウンロード状況の表示

Application クラスは、SWF ファイルのダウンロード状況をダウンロードプログレスバーで表示するアプリケーションプリローダーをサポートします。デフォルトでは、アプリケーションプリローダーは有効になっています。プリローダーはダウンロードされたバイト数を追跡し、プログレスバーを連続的に更新します。

ダウンロードプログレスバーには、アプリケーションの 2 つの段階 (ダウンロード段階と初期化段階) に関する情報が表示されます。プリローダーは Application.creationComplete イベントが発生した時点で画面から消えます。

次の例は、初期化段階のダウンロードプログレスバーを示します。



SWF ファイルがローカルホスト上にあるか、既にキャッシュされている場合は、ダウンロードプログレスバーは表示されません。SWF ファイルがローカルホスト上になく、かつキャッシュされていない場合は、ダウンロード開始から 700 ミリ秒後にアプリケーションの半分がダウンロード未了であれば、プログレスバーが表示されます。

ダウンロードプログレスバーの無効化

ダウンロードプログレスバーを無効にするには、次のように、Application コンテナの `usePreloader` プロパティを `false` に設定します。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  usePreloader="false">
```

カスタムプログレスバーの作成

デフォルトでは、アプリケーションプリローダーは、mx.preloaders パッケージの DownloadProgressBar クラスを使用してダウンロードプログレスバーを表示します。カスタムのダウンロードプログレスバーを作成するには、DownloadProgressBar クラスのサブクラスを作成するか、mx.preloaders.IPreloaderDsiplay インターフェイスを実装する flash.display.Sprite クラスのサブクラスを作成します。

ダウンロードプログレスバーコンポーネントは、SWC コンポーネントまたは ActionScript コンポーネントとして実装できます。Sprite クラスを拡張するカスタムのプログレスバーコンポーネントを作成する場合は、Flex の標準コンポーネントは一切使用しないでください。ロードに時間がかかりすぎて実用的ではありません。ダウンロードプログレスバーを MXML コンポーネントとして実装した場合も、ロードに著しく時間がかかるため、そのような実装は避けてください。

カスタムのダウンロードプログレスバークラスを使用するには、Application コンテナの `preloader` プロパティを、SWC コンポーネントクラスまたは ActionScript コンポーネントクラスのパスに設定します。SWC コンポーネントは、MXML ファイルと同じディレクトリ、または Flex アプリケーションのクラスパス上のディレクトリに存在する必要があります。ActionScript コンポーネントは、これらのいずれかのディレクトリまたはそのサブディレクトリに置くことができます。クラスがサブディレクトリに存在する場合、`preloader` 値にサブディレクトリの場所をパッケージ名として指定します。それ以外の場合は、クラス名を指定します。

次の例では、アプリケーションのルートディレクトリの下の "mycomponents/mybars" ディレクトリにある CustomBar というカスタムダウンロードプログレスバーを指定しています。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  preloader="mycomponents.mybars.CustomBar">
```

ダウンロードプログレスバーのイベント

ダウンロードプログレスバーの動作は、一連のイベントで定義されます。これらのイベントは、Preloader クラスによって送出されます。カスタムのダウンロードプログレスバーは、これらのイベントを処理する必要があります。

ダウンロードプログレスバーのイベントを次の表に示します。

イベント	説明
ProgressEvent.PROGRESS	アプリケーション SWF ファイルのダウンロード中に送出されます。最初の PROGRESS イベントは、ダウンロード処理の開始を示します。
Event.COMPLETE	SWF ファイルのダウンロードが完了すると送出されます。0 または 1 つの COMPLETE イベントが送出されます。
FlexEvent.INIT_COMPLETE	Flex アプリケーションの初期化が完了すると送出されます。このイベントは常に 1 回送出され、Preloader によって送出される最後のイベントです。 ダウンロードプログレスバーは、INIT_COMPLETE イベントを受け取った後に COMPLETE イベントを送出する必要があります。COMPLETE イベントは、ダウンロードプログレスバーの操作がすべて完了し、画面から消える準備ができたことを Preloader に通知します。 ダウンロードプログレスバーは、INIT_COMPLETE イベントを受け取ってから COMPLETE イベントを送出するまでの間に、アニメーションの再生などの追加タスクを実行できます。COMPLETE イベントを送出することが、ダウンロードプログレスバーの最後のアクションになります。
FlexEvent.INIT_PROGRESS	Flex アプリケーションの初期化段階が完了すると送出されます。初期化段階は、measure()、commitProperties()、updateDisplayList() のいずれかのメソッドの呼び出しによって定義されます。このイベントは、初期化段階でのアプリケーションの進捗状況を記述します。
RslEvent.RSL_ERROR	RSL (Runtime Shared Library : ランタイム共有ライブラリ) のロードが失敗すると送出されます。
RslEvent.RSL_LOADED	RSL のロードが完了すると送出されます。イベントオブジェクトには、総バイト数とロードされたバイト数が格納されています。このイベントは、ロードが成功したすべての RSL について送出されます。
RSLEvent.RSL_PROGRESS	RSL のダウンロード中に送出されます。最初の progress イベントは、RSL のダウンロードの開始を示します。 このイベントのイベントオブジェクトは RSLEvent 型です。

DownloadProgressBar クラスは、上記のイベントすべてに対してイベントリスナーを定義します。DownloadProgressBar クラスをオーバーライドする場合は、必要に応じてイベントリスナーのデフォルトの動作をオーバーライドできます。カスタムのダウンロードプログレスバーを Sprite クラスのサブクラスとして作成する場合は、上記の各イベントについてイベントリスナーを定義する必要があります。

DownloadProgressBar クラスの簡単なサブクラスの作成

独自のダウンロードプログレスバーを作成する最も簡単な方法は、mx.preloaders.DownloadProgressBar クラスのサブクラスを作成し、それをアプリケーションの要件に合わせて変更することです。

ダウンロードプログレスバーに対してカスタムストリングを定義し、プログレスバーが表示される最小時間を設定する例を次に示します。

```
package myComponents
{
    import mx.preloaders.*;
    import flash.events.ProgressEvent;

    public class DownloadProgressBarSubClassMin extends DownloadProgressBar
    {
        public function DownloadProgressBarSubClassMin()
        {
            super();
            // Set the download label.
            downloadingLabel="Downloading app..."
            // Set the initialization label.
            initializingLabel="Initializing app..."
            // Set the minimum display time to 2 seconds.
            MINIMUM_DISPLAY_TIME=2000;
        }

        // Override to return true so progress bar appears
        // during initialization.
        override protected function showDisplayForInit(elapsedTime:int,
            count:int):Boolean {
                return true;
        }

        // Override to return true so progress bar appears during download.
        override protected function showDisplayForDownloading(
            elapsedTime:int, event:ProgressEvent):Boolean {
                return true;
        }
    }
```

```
}
```

このカスタムクラスは、Flex アプリケーションで次のように使用します。

```
<?xml version="1.0"?>
<!-- containers\application\MainDPBMin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    preloader="myComponents.DownloadProgressBarSubClassMin">

    <mx:Button/>
    <mx:TextInput text="sub class min" />
</mx:Application>
```

DownloadProgressBar クラスのサブクラスの作成

次の例では、DownloadProgressBar クラスのサブクラスを作成し、アプリケーションのダウンロードと初期化のステータスを示すテキストメッセージを表示します。そのために、ダウンロードプログレスバーによって送出されるイベントのイベントリスナーを定義し、メッセージを flash.text.TextField オブジェクトに書き込んでいます。

```
package myComponents
{

    import flash.display.*;
    import flash.text.*;
    import flash.utils.*;
    import flash.events.*;
    import mx.preloaders.*;
    import mx.events.*;

    public class MyDownloadProgressBar extends DownloadProgressBar
    {
        // Define a TextField control for text messages
        // describing the download progress of the application.
        private var progressText:TextField;

        // Define a TextField control for the final text message.
        // after the application initializes.
        private var msgText:TextField;

        public function MyDownloadProgressBar()
        {
            super();

            // Configure the TextField for progress messages.
            progressText = new TextField();
            progressText.x = 10;
            progressText.y = 90;
            progressText.width = 400;
            progressText.height = 400;
```

```
        addChild(progressText);

        // Configure the TextField for the final message.
        msgText = new TextField();
        msgText.x = 10;
        msgText.y = 10;
        msgText.width = 400;
        msgText.height = 75;

        addChild(msgText);
    }

    // Define the event listeners for the preloader events.
    override public function set preloader(preloader:Sprite):void {
        // Listen for the relevant events
        preloader.addEventListener(
            ProgressEvent.PROGRESS, myHandleProgress);
        preloader.addEventListener(
            Event.COMPLETE, myHandleComplete);

        preloader.addEventListener(
            FlexEvent.INIT_PROGRESS, myHandleInitProgress);
        preloader.addEventListener(
            FlexEvent.INIT_COMPLETE, myHandleInitEnd);
    }

    // Event listeners for the ProgressEvent.PROGRESS event.
    private function myHandleProgress(event:ProgressEvent):void {
        progressText.appendText("\n" + "Progress l: " +
            event.bytesLoaded + " t: " + event.bytesTotal);
    }

    // Event listeners for the Event.COMPLETE event.
    private function myHandleComplete(event:Event):void {
        progressText.appendText("\n" + "Completed");
    }

    // Event listeners for the FlexEvent.INIT_PROGRESS event.
    private function myHandleInitProgress(event:Event):void {
        progressText.appendText("\n" + "App Init Start");
    }

    // Event listeners for the FlexEvent.INIT_COMPLETE event.
    private function myHandleInitEnd(event:Event):void {
        msgText.appendText("\n" + "App Init End");

        var timer:Timer = new Timer(2000,1);
        timer.addEventListener(TimerEvent.TIMER, dispatchComplete);
        timer.start();
```

```
        }

        // Event listener for the Timer to pause long enough to
        // read the text in the download progress bar.
        private function dispatchComplete(event:TimerEvent):void {
            dispatchEvent(new Event(Event.COMPLETE));
        }
    }
}
```

このカスタムクラスは、Flex アプリケーションで次のように使用します。

```
<?xml version="1.0"?>
<!-- containers\application\MainDPB.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    preloader="myComponents.MyDownloadProgressBar">

    <mx:Button/>
    <mx:TextInput/>
</mx:Application>
```

Sprite のサブクラスの作成

カスタムのダウンロードプログレスバーは、Sprite クラスのサブクラスとして定義できます。ダウンロードプログレスバーを Sprite のサブクラスとして実装すると、DownloadProgressBar クラスに組み込まれた動作をオーバーライドせずに、完全に独自の外観を備えたダウンロードプログレスバーを作成できます。

この種のダウンロードプログレスバーの一般的な使い方の 1 つとして、アプリケーションの初期化中に SWF ファイルを表示させることがあります。たとえば、動いている時計などのイメージを表す SWF ファイルを表示できます。

このセクションの例では、ダウンロードプログレスバーとして SWF ファイルを表示します。このクラスには IPreloaderDisplay インターフェイスを実装する必要があります。

```
package myComponents
{
    import flash.display.*;
    import flash.utils.*;
    import flash.events.*;
    import flash.net.*;
    import mx.preloaders.*;
    import mx.events.*;

    public class MyDownloadProgressBarSWF extends Sprite
        implements IPreloaderDisplay
    {
        // Define a Loader control to load the SWF file.
        private var dpbImageControl:flash.display.Loader;
```

```
public function MyDownloadProgressBarSWF() {
    super();
}

// Specify the event listeners.
public function set preloader(preloader:Sprite):void {
    // Listen for the relevant events
    preloader.addEventListener(
        ProgressEvent.PROGRESS, handleProgress);
    preloader.addEventListener(
        Event.COMPLETE, handleComplete);

    preloader.addEventListener(
        FlexEvent.INIT_PROGRESS, handleInitProgress);
    preloader.addEventListener(
        FlexEvent.INIT_COMPLETE, handleInitComplete);
}

// Initialize the Loader control in the override
// of IPreloaderDisplay.initialize().
public function initialize():void {
    dpbImageControl = new flash.display.Loader();
    dpbImageControl.contentLoaderInfo.addEventListener(
        Event.COMPLETE, loader_completeHandler);
    dpbImageControl.load(new URLRequest("assets/dpbSWF.swf"));
}

// After the SWF file loads, set the size of the Loader control.
private function loader_completeHandler(event:Event):void
{
    addChild(dpbImageControl);
    dpbImageControl.width = 50;
    dpbImageControl.height= 50;
    dpbImageControl.x = 100;
    dpbImageControl.y = 100;
}

// Define empty event listeners.
private function handleProgress(event:ProgressEvent):void {
}

private function handleComplete(event:Event):void {
}

private function handleInitProgress(event:Event):void {
}

private function handleInitComplete(event:Event):void {
    var timer:Timer = new Timer(2000,1);
    timer.addEventListener(TimerEvent.TIMER, dispatchComplete);
    timer.start();
```

```
        }

        private function dispatchComplete(event:TimerEvent):void {
            dispatchEvent(new Event(Event.COMPLETE));
        }

        // Implement IPreloaderDisplay interface

        public function get backgroundColor():uint {
            return 0;
        }

        public function set backgroundColor(value:uint):void {
        }

        public function get backgroundAlpha():Number {
            return 0;
        }

        public function set backgroundAlpha(value:Number):void {
        }

        public function get backgroundImage():Object {
            return undefined;
        }

        public function set backgroundImage(value:Object):void {
        }

        public function get backgroundSize():String {
            return "";
        }

        public function set backgroundSize(value:String):void {
        }

        public function get stageWidth():Number {
            return 200;
        }

        public function set stageWidth(value:Number):void {
        }

        public function get stageHeight():Number {
            return 200;
        }

        public function set stageHeight(value:Number):void {
        }
    }
}
```

第 15 章

レイアウトコンテナの使用

15

Adobe Flex のレイアウトコンテナは、Flex アプリケーションの Button コントロールや ComboBox コントロールなどのコンポーネントを配置、設定するための階層構造を提供します。

このトピックでは、レイアウトコンテナとその用法について説明します。ここではすべての Flex レイアウトコンテナに対し、説明と用例を示します。Flex によるコンテナと子の詳しいレイアウト方法については、207 ページ、第 8 章の「コンポーネントのサイズと位置の制御」を参照してください。

目次

レイアウトコンテナについて

レイアウトコンテナは、Adobe Flash Player の描画面上に矩形領域を定義し、この中に定義された子コントロールおよび子コンテナのサイズと位置を制御します。たとえば、Form レイアウトコンテナは、HTML フォームに似たレイアウトで子のサイズの変更や配置を行います。

レイアウトコンテナを使用するには、コンテナを作成してから、アプリケーションを定義するコンポーネントを追加します。

Flex には、次のレイアウトコンテナがあります。

- Canvas レイアウトコンテナ
- Box、HBox、および VBox レイアウトコンテナ
- ControlBar レイアウトコンテナ
- ApplicationControlBar レイアウトコンテナ
- DividedBox、HDividedBox、および VDividedBox レイアウトコンテナ
- Form、FormHeading、および FormItem レイアウトコンテナ
- Grid レイアウトコンテナ
- Panel レイアウトコンテナ
- Tile レイアウトコンテナ
- TitleWindow レイアウトコンテナ

以下のセクションでは、これらの各 Flex レイアウトコンテナを使用する方法について説明します。

Canvas レイアウトコンテナ

Canvas レイアウトコンテナは、子コンテナや子コントロールを配置する矩形領域を定義します。他のすべてのコンポーネントとは異なり、子コントロールは Flex によって自動配置することができません。子コンポーネントを配置するには、" 絶対レイアウト " または " 制約ベースのレイアウト " を使用します。絶対レイアウトを使用する場合は、子の x および y 座標を指定します。制約ベースのレイアウトの場合は、左右または中央のアンカーを指定します。それぞれのレイアウト方法の詳細については、207 ページ、第 8 章の「コンポーネントのサイズと位置の制御」を参照してください。

Canvas コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	すべての子をデフォルトサイズで保持するのに十分な大きさです。
デフォルトパディング	`top`、`bottom`、`left`、および `right` の各値が 0 ピクセルです。

詳細については、『Adobe Flex 2 リファレンスガイド』の Canvas を参照してください。

Canvas コントロールの作成と使用

MXML で Canvas コントロールを定義するには、<mx:Canvas> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

絶対配置を使用した Canvas コントロールの作成

それぞれの子の x および y プロパティを使用して、Canvas コンテナ内で子の位置を指定できます。これらのプロパティでは、Canvas コンテナの左上隅を基準とした子の x および y 座標を指定します。左上隅の座標は (0,0) です。x および y 座標の値には、正の整数または負の整数を指定できます。負の値を指定すると、コンテナの可視領域外に子を配置できます。この場合は ActionScript を使用することで、たとえばイベントへの応答として、可視領域内に子を移動できます。

次の例に示す Canvas コンテナには、LinkButton コントロール 3 つと Image コントロール 3 つが配置されています。

この Canvas コンテナは次の MXML コードで作成されます。

```
<?xml version="1.0"?>
<!-- containers\layouts\CanvasSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Canvas id="myCanvas"
      height="200" width="200"
      borderStyle="solid"
      backgroundColor="white">

      <mx:LinkButton label="Search"
        x="10" y="30"
        click="navigateToURL(new URLRequest('http://mycomp.com/search'))"/>
      <mx:Image
        height="50" width="50"
        x="100" y="10"
        source="@Embed(source='assets/search.jpg')"
        click="navigateToURL(new URLRequest('http://mycomp.com/search'))"/>

      <mx:LinkButton label="Help"
```

```
        x="10" y="100"
        click="navigateToURL(new URLRequest('http://mycomp.com/help'))"/>
      <mx:Image
        height="50" width="50"
        x="100" y="75"
        source="@Embed(source='assets/help.jpg')"
        click="navigateToURL(new URLRequest('http://mycomp.com/help'))"/>

      <mx:LinkButton label="Complaints"
        x="10" y="170"
        click="navigateToURL(
            new URLRequest('http://mycomp.com/complain'))"/>
      <mx:Image
        height="50" width="50"
        x="100" y="140"
        source="@Embed(source='assets/complaint.jpg')"
        click="navigateToURL(
            new URLRequest('http://mycomp.com/complaint'))"/>
    </mx:Canvas>
</mx:Application>
```

制約ベースのレイアウトを使用した Canvas コンテナの作成

制約ベースのレイアウトを使用すると、子の上端、下端、左端、右端の任意の組み合わせを、Canvas の端から指定の距離に固定できます。あるいは、子の水平または垂直方向の中心を、Canvas の中心から指定のピクセル数 (正数または負数) の位置に固定することもできます。制約ベースのレイアウトを指定するには、top、bottom、left、right、horizontalCenter、および verticalCenter スタイルを使用します。Canvas の端との相対位置に子コンテナの上下の端、または左右の端を固定した場合は、Canvas コントロールのサイズが変更されると、子のサイズも変更されます。次の例では、制約ベースのレイアウトを使用して HBox の水平位置を指定し、絶対値を使用して高さと垂直位置を指定します。

```
<?xml version="1.0"?>
<!-- containers\layouts\CanvasConstraint.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Canvas
        width="150" height="150"
        backgroundColor="#FFFFFF">

        <mx:HBox id="hBox2"
            left="30"
            right="30"
            y="50"
            height="50"
            backgroundColor="#A9C0E7">
        </mx:HBox>
    </mx:Canvas>
</mx:Application>
```

この例では、次のイメージが作成されます。

子の重複の防止

Canvas コンテナを使用していると、コンポーネント同士が重なり合ってしまうことがあります。Canvas コンテナが子を配置する際、子のサイズは無視されるためです。また、Canvas コンテナは子を考慮して座標系を調整することはしないため、子コンポーネントがいずれかの境界線やパディングと重なることがあります。

次の例では、コンポーネントが重ならないように、各コンポーネントのサイズと位置を慎重に計算しています。

```
<?xml version="1.0"?>
<!-- containers\layouts\CanvasOverlap.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100" height="100"
    backgroundGradientColors="[0xFFFFFF, 0xFFFFFF]">

    <mx:Canvas id="chboard" backgroundColor="#FFFFFF">
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="0" y="0"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="20" y="0"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="40" y="0"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="10" y="10"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="30" y="10"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="0" y="20"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="20" y="20"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="40" y="20"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="10" y="30"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="30" y="30"/>
        <mx:Image source="assets\BlackBox.jpg"
```

```
            width="10" height="10" x="0" y="40"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="20" y="40"/>
        <mx:Image source="assets\BlackBox.jpg"
            width="10" height="10" x="40" y="40"/>
    </mx:Canvas>
</mx:Application>
```

この例では、次のイメージが作成されます。

このいずれかのイメージの width および height プロパティを 20 ピクセルに変更した場合、それに応じた位置変更を行わないと、そのイメージが格子縞模様上の他のイメージと重なってしまいます。たとえば、前の例の 7 番目の <mx:Image> タグを次の行に置き換えると、この結果表示されるイメージは下図のようになります。

```
<mx:Image source="BlackBox.jpg" width="10" height="10" x="20" y="20"/>
```

実行時の子の再配置

アプリケーションにロジックを構築し、Canvas コンテナの子を実行時に再配置することができます。たとえば、次のコードでは、ボタンのクリックに応答して、id 値が text1 のテキスト入力ボックスを x=110、y=110 の位置に再配置します。

```
<?xml version="1.0"?>
<!-- containers\layouts\CanvasOverlap.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Canvas
        width="300" height="300"
        backgroundColor="#FFFFFF">

        <mx:TextInput id="text1"
            text="Move me"
            x="50" y="50"/>
        <mx:Button id="button1"
            label="Move text1"
            x="50" y="200"
            click="text1.x=110; text1.y=110;"/>
    </mx:Canvas>
</mx:Application>
```

Box、HBox、および VBox レイアウトコンテナ

Box レイアウトコンテナでは、子が単一の垂直列または水平行にレイアウトされます。Box コンテナの direction プロパティを使用し、垂直 (デフォルト) レイアウトか水平レイアウトかを指定します。HBox および VBox コンテナは、horizontal および vertical direction プロパティ値を持つ Box コンテナです。

メモ

子を複数の行または列にレイアウトする場合は、Tile コンテナまたは Grid コンテナを使用します。詳細については、563 ページの「Tile レイアウトコンテナ」および 553 ページの「Grid レイアウトコンテナ」を参照してください。

次の例は、水平レイアウトの Box コンテナおよび垂直レイアウトの Box コンテナを示しています。

水平レイアウトの Box コンテナ

垂直レイアウトの Box コンテナ (デフォルト

Box コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト
デフォルトサイズ	**垂直ボックス**　height は、デフォルトの高さまたは明示的に指定した高さを持つすべての子を収容し、さらに垂直方向の子同士の間隔、コンテナの上下のパディングを収容できる十分な大きさです。 width は、最も広い幅を持つ子のデフォルトの幅または明示的に指定した幅に、コンテナの左右のパディングを加えた値です。 **水平ボックス**　height は、最も高い子のデフォルトの高さまたは明示的に指定した高さに、コンテナの上下のパディングを加えた値です。 width は、デフォルトの幅を持つすべての子、水平方向の子同士の間隔、およびコンテナの左右のパディングをすべて収容できる十分な大きさです。
デフォルトパディング	top、bottom、left、および right の各値が 0 ピクセルです。

詳細については、『Adobe Flex 2 リファレンスガイド』の Box、HBox、および VBox を参照してください。

Box、HBox、または VBox コンテナの作成

Box コンテナを定義するには、<mx:Box>、<mx:VBox>、および <mx:HBox> タグを使用します。VBox (垂直ボックス) および HBox (水平ボックス) コンテナをショートカットとして使用するので、Box コンテナ内で direction プロパティを指定する必要はありません。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

次の例では、Box コンテナを垂直レイアウトで作成します。

```
<?xml version="1.0"?>
<!-- containers\layouts\BoxSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Box direction="vertical"
        borderStyle="solid"
        paddingTop="10"
        paddingBottom="10"
        paddingLeft="10"
        paddingRight="10">

        <mx:Button id="fname" label="Button 1"/>
        <mx:Button id="lname" label="Button 2"/>
        <mx:Button id="addr1" label="Button 3"/>
        <mx:ComboBox id="state"/>
    </mx:Box>
</mx:Application>
```

次のコード例は前の例とほぼ同じですが、<mx:VBox> タグを使って垂直ボックスコンテナを定義している点で異なります。

```
<?xml version="1.0"?>
<!-- containers\layouts\VBoxSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox borderStyle="solid"
        paddingTop="10"
        paddingBottom="10"
        paddingLeft="10"
        paddingRight="10">

        <mx:Button id="fname" label="Button 1"/>
        <mx:Button id="lname" label="Button 2"/>
        <mx:Button id="addr1" label="Button 3"/>
        <mx:ComboBox id="state"/>
    </mx:VBox>
</mx:Application>
```

ControlBar レイアウトコンテナ

ControlBar コンテナを Panel コンテナや TitleWindow コンテナと共に使用すると、Panel コンテナや TitleWindow コンテナの他の子と共有可能なコンポーネントを配置できます。次の例に示すように、製品カタログの場合は、数量の指定やショッピングカートへの商品の追加を行う Flex コントロールを ControlBar コンテナに配置できます。

ControlBar コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	height は、最も高い子のデフォルトの高さまたは明示的に指定した高さに、コンテナの上下のパディングを加えた値です。 width は、デフォルトの幅または明示的に指定した幅を持つすべての子、水平方向の子同士の間隔、およびコンテナの左右のパディングをすべて収容できる十分な大きさです。
デフォルトパディング	`top`、`bottom`、`left`、および `right` の各値が 10 ピクセルです。

詳細については、『Adobe Flex 2 リファレンスガイド』の ControlBar を参照してください。

ControlBar コンテナの作成

`<mx:ControlBar>` タグを使用して、MXML で ControlBar コントロールを定義します。MXML コードの他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、`id` 値を指定します。`<mx:ControlBar>` タグは、次の例に示すように、`<mx:Panel>` タグの最後の子タグとして指定します。

```
<?xml version="1.0"?>
<!-- containers\layouts\CBarSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
```

```
            private function addToCart():void {
                // Handle event.
            }
        ]]>
    </mx:Script>

    <mx:Panel title="My Application"
        paddingTop="10" paddingBottom="10"
        paddingLeft="10" paddingRight="10">

        <mx:HBox width="250" height="200">
            <!-- Area for your catalog. -->
        </mx:HBox>

        <mx:ControlBar width="250">
            <mx:Label text="Quantity"/>
            <mx:NumericStepper/>
            <!-- Use Spacer to push Button control to the right. -->
            <mx:Spacer width="100%"/>
            <mx:Button label="Add to Cart"
                click="addToCart();"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

ApplicationControlBar レイアウトコンテナ

アプリケーションのナビゲーションエレメントおよびコマンドへのアクセスを提供するコンポーネントを収容するために、ApplicationControlBar コンテナを使用します。たとえば、エディタの ApplicationControlBar コンテナには、フォントの太さを設定するための Button コントロール、フォントを選択するための ComboBox、および編集モードを選択するための MenuBar コントロールを含めることができます。ApplicationControlBar は ControlBar クラスのサブクラスですが、ControlBar とは外観も操作性も異なります。

一般的に ApplicationControlBar コンテナは、次の例に示すように、アプリケーションの一番上に配置します。

ApplicationControlBar コンテナをアプリケーションの一番上にドッキングすると、アプリケーションの内容と共にスクロールされなくなります。

ApplicationControlBar コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	height は、最も高い子のデフォルトの高さまたは明示的に指定した高さに、コンテナの上下のパディングを加えた値です。 通常モードでの width は、デフォルトの幅または明示的に指定した幅を持つすべての子、水平方向の子同士の間隔、およびコンテナの左右のパディングをすべて収容できる十分な大きさです。ドッキングモードでは、この幅はアプリケーションの幅に一致します。 アプリケーションの幅が、ApplicationControlBar コンテナ内のすべてのコントロールを収容できる十分な長さではない場合は、ApplicationControlBar はアプリケーション幅に合わせてクリッピングされます。
デフォルトパディング	`top` の値は 5 ピクセルです。 `bottom` の値は 4 ピクセルです。 `left` および `right` の値は 8 ピクセルです。

詳細については、『Adobe Flex 2 リファレンスガイド』の ApplicationControlBar を参照してください。

ApplicationControlBar コンテナの作成

`<mx:ApplicationControlBar>` タグを使用して、MXML で ControlBar コントロールを定義します。MXML コードの他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、`id` 値を指定します。

ApplicationControlBar コンテナは、次のいずれかのモードに設定できます。

ドッキングモード　バーは常に、アプリケーションの描画領域の最上部に表示されます。アプリケーションレベルのスクロールバーはいずれも、このコンテナには適用されません。したがって、このバーは常に表示可能領域の最上部に固定され、アプリケーションの幅一杯に拡張して表示されます。ドッキングした ApplicationControlBar コンテナを作成するには、コンテナの `dock` プロパティを `true` に設定します。

通常モード　バーはアプリケーション上のどこにでも配置できます。他のすべてのコンポーネントと同様にサイズ設定および配置でき、アプリケーションと共にスクロールされます。`dock` プロパティを `false` に設定すると、ApplicationControlBar は固定されず、フローティングします。デフォルト値は `false` です。

メモ

ControlBar コンテナとは異なり、ApplicationControlBar のインスタンスには `backgroundColor` スタイルを設定できます。ApplicationControlBar コンテナは、ControlBar コンテナによってサポートされない `fillColors` および `fillAlpha` という 2 つのスタイルを備えます。

次の例は、MenuBar を含む、ドッキング型の単純な ApplicationControlBar を持つアプリケーションを示しています。このアプリケーションには、アプリケーションサイズより大きな HBox コントロールも含まれます。アプリケーションをスクロールして HBox コントロールを下まで表示しても、ApplicationControlBar コントロールはスクロールされません。

```
<?xml version="1.0"?>
<!-- containers\layouts\AppCBarSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
        ]]>
    </mx:Script>

    <mx:XMLList id="menuXML">
        <menuitem label="File">
            <menuitem label="New" data="New"/>
            <menuitem label="Open" data="Open"/>
            <menuitem label="Save" data="Save"/>
            <menuitem label="Exit" data="Exit"/>
        </menuitem>
        <menuitem label="Edit">
            <menuitem label="Cut" data="Cut"/>
            <menuitem label="Copy" data="Copy"/>
            <menuitem label="Paste" data="Paste"/>
        </menuitem>
        <menuitem label="View"/>
    </mx:XMLList>

    <mx:Array id="cmbDP">
        <mx:String>Item 1</mx:String>
        <mx:String>Item 2</mx:String>
        <mx:String>Item 3</mx:String>
    </mx:Array>

    <mx:ApplicationControlBar id="dockedBar"
        dock="true">
        <mx:MenuBar height="100%"
            dataProvider="{menuXML}"
            labelField="@label"
            showRoot="true"/>
        <mx:HBox paddingBottom="5"
            paddingTop="5">
            <mx:ComboBox dataProvider="{cmbDP}"/>
            <mx:Spacer width="100%"/>
            <mx:TextInput id="myTI" text=""/>
            <mx:Button id="srch1"
                label="Search"
                click="Alert.show('Searching')"/>
```

```
        </mx:HBox>
    </mx:ApplicationControlBar>

    <mx:TextArea width="300" height="200"/>
</mx:Application>
```

DividedBox、HDividedBox、および VDividedBox レイアウトコンテナ

DividedBox レイアウトコンテナは、子を水平方向または垂直方向にレイアウトします。Box コンテナとよく似ていますが、DividedBox には子と子の間に分割線が挿入される点で異なります。マウスポインタを使って分割線を動かし、それぞれの子に割り当てられたコンテナ領域をサイズ変更することができます。DividedBox コンテナの `direction` プロパティを使用し、垂直レイアウト (デフォルト) か水平レイアウトかを決定します。HDividedBox および VDividedBox コンテナは、`horizontal` および `vertical direction` プロパティ値を持つ DividedBox コンテナです。

次に、DividedBox コンテナの例を示します。

この例では、最も外側のコンテナが水平レイアウトの DividedBox コンテナです。水平分割線は、Tree コントロールと vertical DividedBox コンテナの間に境界を設定します。

vertical DividedBox コンテナは、DataGrid コントロール (上) と TextArea コントロール (下) を保持します。垂直分割線は、これら 2 つのコントロールの間に境界を設定します。

DividedBox、HDividedBox、または VDividedBox コンテナには、次のデフォルトサイズ設定特定があります。

プロパティ	デフォルト値
デフォルトサイズ	**Vertical DividedBox**　height は、デフォルトの高さまたは明示的に指定した高さを持つすべての子を収容し、さらに垂直方向の子同士の間隔、コンテナの上下のパディングを収容できる十分な大きさです。 width は、最も広い幅を持つ子のデフォルトの幅または明示的に指定した幅に、コンテナの左右のパディングを加えた値です。 **Horizontal DividedBox**　height は、最も高い子のデフォルトの高さまたは明示的に指定した高さに、コンテナの上下のパディングを加えた値です。 width は、デフォルトの幅または明示的に指定した幅を持つすべての子、水平方向の子同士の間隔、およびコンテナの左右のパディングをすべて収容できる十分な大きさです。
デフォルトパディング	top、bottom、left、right ともに 0 ピクセル。
デフォルトの間隔	水平方向および垂直方向の間隔は 10 ピクセルです。

詳細については、『Adobe Flex 2 リファレンスガイド』の DividedBox、HDividedBox、および VDividedBox を参照してください。

DividedBox、HDividedBox、または VDividedBox コンテナの作成

<mx:DividedBox>、<mx:VDividedBox>、および <mx:HDividedBox> タグを使用して、DividedBox コンテナを定義します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。通常は、VDividedBox (vertical DividedBox) および HDividedBox (horizontal DividedBox) コンテナをショートカットとして使用するので、direction プロパティを指定する必要はありません。

次のコード例では、529 ページの「DividedBox、HDividedBox、および VDividedBox レイアウトコンテナ」で示したイメージを作成します。

```
<?xml version="1.0"?>
<!-- containers\layouts\HDivBoxSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="white">

    <mx:Script>
      <![CDATA[
        private function myGrid_initialize():void {
          myGrid.dataProvider = [
            {Artist:'Pavement', Album:'Slanted and Enchanted',
                Price:11.99, Comment:'One of their best. 4 Stars.'},
            {Artist:'Pavement', Album:'Brighten the Corners',
```

```
                Price:11.99, Comment:'My favorite.'}
          ];
        }
      ]]>
    </mx:Script>

    <mx:HDividedBox width="100%" height="100%">
        <mx:Tree id="tree1"
            width="30%" height="100%"
            labelField="@label"
            showRoot="true">
            <mx:XMLList>
                <menuitem label="Products">
                    <menuitem label="Posters" isBranch="true"/>
                    <menuitem label="CDs">
                        <menuitem label="Pavement"/>
                        <menuitem label="Pavarotti"/>
                        <menuitem label="Phish"/>
                    </menuitem>
                    <menuitem label="T-shirts" isBranch="true"/>
                    <menuitem label="Tickets" isBranch="true"/>
                </menuitem>
            </mx:XMLList>
        </mx:Tree>

        <mx:VDividedBox width="70%" height="100%">
            <mx:DataGrid id="myGrid"
                width="100%" height="100%"
                initialize="myGrid_initialize();"
                change="currentMessage.text=
                    event.currentTarget.selectedItem.Comment;"/>
            <mx:TextArea id="currentMessage"
                width="100%"
                height="60"
                text="One of their best. 4 Stars."/>
        </mx:VDividedBox>

    </mx:HDividedBox>
</mx:Application>
```

この例では、Tree コントロールでノードが選択されたときに VDividedBox コンテナの上部領域を変更するロジックが実装されていません。

分割線の使用

DividedBox コンテナの分割線により、子に割り当てられたコンテナの領域をサイズ変更できます。ただし、分割線が適切に機能するのは、子がサイズ変更可能な場合のみです。つまり、子のサイズをパーセント値ベースで指定する必要があります。明示的な、またはデフォルトの高さまたは幅を持つ子は、分割線を使って水平方向または垂直方向にサイズ変更することはできません。したがって、DividedBox コンテナを使用する場合は通常、パーセント値のサイズを指定することで子をサイズ変更可能にします。

子の `height` または `width` プロパティにパーセント値を指定してサイズ変更可能にすると、子は最初、可能であれば指定のパーセント値のサイズで表示されます。以降は、子は有効なスペースの範囲内で自動的にサイズ変更されます。

パーセント値でサイズ指定された子は、分割線を使用して、最大サイズまたは最小サイズまで変更できます。DividedBox の領域の最小サイズまたは最大サイズを制限するには、その領域内の子の `minWidth` および `minHeight` プロパティ、または `maxWidth` および `maxHeight` プロパティに明示的な値を設定します。

ライブドラッグの使用

DividedBox コンテナでは、デフォルトでライブドラッグが無効になっています。したがって、DividedBox コンテナで子のレイアウトが更新されるのは、ユーザーが分割線のドラッグを終えた後で、選択した分割線上でマウスボタンを離したときです。

DividedBox コンテナでライブドラッグを使用するには、`liveDragging` プロパティを `true` に設定します。ライブドラッグが有効になっている場合は、ユーザーが分割線を移動すると、その動きに応じてレイアウトが更新されます。ただし、ライブドラッグを有効にしたことによってパフォーマンスが低下する場合があります。

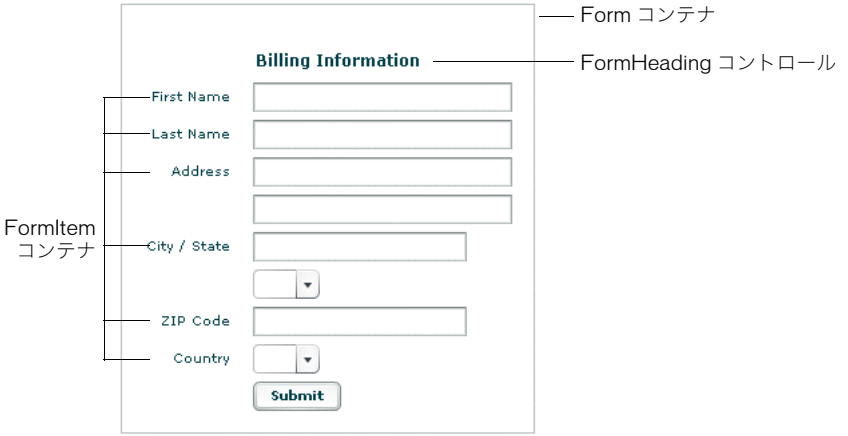
Form、FormHeading、および FormItem レイアウトコンテナ

フォームは、Web アプリケーションでユーザーから情報を収集する場合に最もよく使われる方法の1つです。フォームは、登録、購買、課金情報など、さまざまなデータ収集タスクに使用されます。

フォームについて

Flex は、Form レイアウトコンテナとその子コンポーネントを使用したフォームの開発をサポートします。Form コンテナでは、フォームのレイアウトを制御し、フォームフィールドが必須かオプションであるかを明示し、エラーメッセージを処理し、フォームデータを Flex データモデルにバインドしてデータのチェックと検証を実行することができます。また、スタイルシートを適用することによってフォームの外観を設定することもできます。

次の例に示すように、3 種類のコンポーネントを使用してフォームを作成します。



次の表に、Flex でフォーム作成に使用するコンポーネントタイプの説明を示します。

コンテナ	タグ	説明
Form	`<mx:Form>`	フォーム全体に対するコンテナ、およびフォーム全体のレイアウトを定義します。フォームの内容は、FormHeading コントロールおよび FormItem コンテナを使用して定義します。別のタイプのコンポーネントを Form コンテナに挿入することもできます。
FormHeading	`<mx:FormHeading>`	フォーム内の見出しを定義します。1 つの Form コンテナ内に複数の FormHeading コントロールを配置できます。
FormItem	`<mx:FormItem>`	フォームに収容する子を、水平方向または垂直方向に揃えて配置します。フォームの子には、コントロールやその他のコンテナなどがあります。1 つの Form コンテナで複数の FormItem コンテナを保持できます。

詳細については、『Adobe Flex 2 リファレンスガイド』の Form、FormHeading、および FormItem を参照してください。

フォームの作成

フォームを作成する場合は、通常は以下のエレメントを定義します。

- Form コントロール
- FormHeading コンポーネント (Form コントロール内にネスト)
- FormItem コンテナ (Form コントロール内にネスト)
- ComboBox、TextInput コントロールなどの Form フィールド (FormItem コンテナ内にネスト)

フォーム内には必要に応じて、HRule コントロールなどの他のコンポーネントを含めることもできます。

以降のセクションでは、Form、FormHeading、および FormItem コンポーネントの作成方法について説明し、完全なフォームの作成例を示します。

Form コンテナの作成

Form コンテナは、Flex フォームの一番外側のコンテナです。Form コンテナの主な用途は、ラベルのサイズやアイテム間の間隔の設定など、フォームの内容のサイズおよびレイアウトの制御です。Form コンテナでは、フォーム内の子が常に垂直方向および左揃えで整列するよう配置されます。Form コンテナには、1つまたは複数の FormHeading および FormItem コンテナを含めることができます。

<mx:Form> タグを使用して、Form コンテナを定義します。フォーム全体を MXML の他の場所 (他のタグまたは ActionScript ブロック) から参照する場合は、id の値を指定します。

次のコード例は、533 ページの「フォームについて」の図で使用したフォームの Form コンテナ定義を示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form id="myForm" width="400" height="100">

        <!-- Define FormHeading and FormItem components here -->

    </mx:Form>
</mx:Application>
```

詳細については、『Adobe Flex 2 リファレンスガイド』の Form を参照してください。

FormHeading コントロールの作成

FormHeading コントロールは、FormItem コンテナのグループにオプションのラベルを指定します。ラベルの左端は、フォーム内部のコントロールの左端に揃えられます。フォームで複数の FormHeading コントロールを保持して、複数のコンテンツ領域を指定することができます。また、FormHeading コントロールに空白の label を使用して、フォーム内に垂直方向の領域を作成することもできます。

<mx:FormHeading> タグを使用して、FormHeading コンテナを定義します。FormHeading を MXML の他の場所 (他のタグまたは ActionScript ブロック) から参照する場合は、id の値を指定します。

次のコード例では、533 ページの「フォームについて」の図の FormHeading コントロールを定義します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormHeadingSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form id="myForm" width="400" height="100">

        <mx:FormHeading label="Billing Information"/>

        <!--Define FormItem containers here. -->

    </mx:Form>
</mx:Application>
```

詳細については、『Adobe Flex 2 リファレンスガイド』の FormHeading を参照してください。

FormItem コンテナの作成

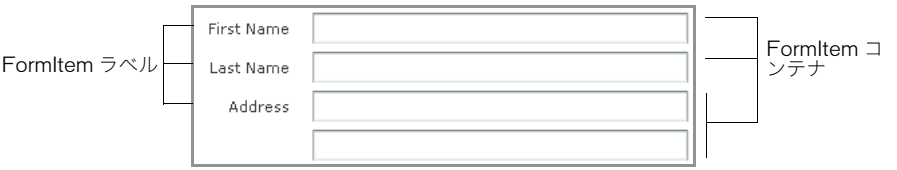
FormItem コンテナの定義するフォームエレメントは、次の部分から構成されます。

- 単一のラベル
- 1つまたは複数の子コントロールまたは子コンテナ (入力コントロールなど)

ラベルは、コンテナ内の最初の子と垂直方向に揃えられます。また、コンテナの左側で右揃えに表示されます。

FormHeading コンテナを定義するには、<mx:FormItem> タグを使用します。このアイテムを MXML の他の場所 (他のタグまたは ActionScript ブロック) から参照する場合は、id の値を指定します。

次の例に示すように、通常は複数の FormItem コンテナが Form コンテナに含まれています。



この例では、3 つの FormItem コンテナを定義します。それぞれに対し、First Name ラベル、Last Name ラベル、Address ラベルが定義されています。Address FormItem コンテナは、ユーザーがアドレス情報を 2 行で入力できるように、2 つのコントロールを保持します。他の 2 つの FormItem コンテナには、それぞれ 1 つのコントロールが含まれています。

詳細については、『Adobe Flex 2 リファレンスガイド』の FormItem を参照してください。

フォームアイテムの方向の指定

FormItem コンテナを作成する場合、`direction` プロパティに値 `vertical` (デフォルト) または `horizontal` を使用して、方向を指定します。

vertical　Flex は、FormItem ラベルの右側に垂直方向に子を配置します。

horizontal　Flex は、FormItem ラベルの右側に水平方向に子を配置します。すべての子を 1 行に収めることができない場合は、これらの子は同じ列サイズで複数行に分割されます。すべての子が 1 行に収まるようにするには、幅をパーセント値で指定するか、すべてのコンポーネントが収まる十分な幅を明示的に指定します。

フォームアイテムのラベルスタイルの制御

FormItem コントロールのラベルは、FormItemLabel オブジェクトです。したがって、FormItemLabel 型のスタイルセレクタを設定してラベルのスタイルを指定することで、FormItemLabel のスタイルを制御できます。次の例は、FormItem ラベルの色をダークブルーに設定します。

```
<mx:Style>
  FormItemLabel {
    color: #333399;
  }
</mx:Style>
```

例 : 単純なフォーム

次の例では、フォームの例の FormItem コンテナ定義が示されています。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormComplete.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function setValues():void {
                // Handle value setting.
            }
        ]]>
    </mx:Script>

    <mx:Form id="myForm" width="400">

        <mx:FormHeading label="Billing Information"/>

        <mx:FormItem label="First Name">
            <mx:TextInput id="fname" width="100%"/>
        </mx:FormItem>

        <mx:FormItem label="Last Name">
            <mx:TextInput id="lname" width="100%"/>
        </mx:FormItem>

        <mx:FormItem label="Address">
            <mx:TextInput id="addr1" width="100%"/>
            <mx:TextInput id="addr2" width="100%"/>
        </mx:FormItem>

        <mx:FormItem label="City / State" direction="vertical">
            <mx:TextInput id="city"/>
            <mx:ComboBox id="st" width="50"/>
        </mx:FormItem>

        <mx:FormItem label="Zip Code">
            <mx:TextInput id="zip" width="100"/>
        </mx:FormItem>

        <mx:FormItem label="Country">
            <mx:ComboBox id="cntry"/>
        </mx:FormItem>

        <mx:FormItem>
            <mx:Button label="Submit"
                click="setValues();"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

フォームのレイアウト

Flex では、次の方法によってフォームのデフォルトサイズが決定されます。

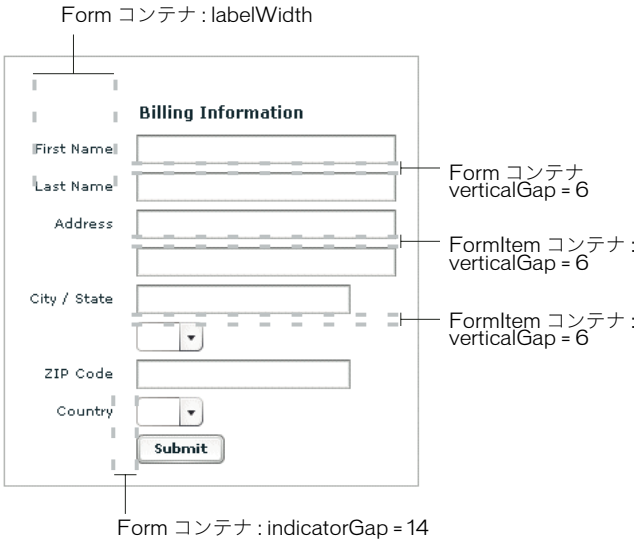
- デフォルトの高さは、コンテナ内のすべての子のデフォルトの高さまたは明示的に指定した高さを収容し、さらにコンテナ上下のパディングと、子同士の間隔を含めた値です。
- デフォルトの幅は、最大の FormItem ラベルを収容し、さらにラベルと子コントロール間の `indicatorGap` と、FormItem 内で最大幅の子コントロールのデフォルト幅または明示的な幅を含めた値です。

次のセクションでは、Form の子を配置およびレイアウトする方法の詳細について説明します。

Form コンテナの子の配置と間隔の制御

コンテナ内の Form コンテナラベルはすべて右揃えで、子はすべて左揃えです。この配置をオーバーライドすることはできません。

次の例は、制御可能な Form コンテナの子の間隔を示しています。



つぎの表では、間隔を制御するために使用する各スタイルプロパティについて説明し、それぞれのデフォルト値を示します。

コンポーネント	スタイル	内容	デフォルト値
Form	verticalGap	Form コンテナの子同士の垂直方向の間隔です。	6 ピクセル
	horizontalGap	Form コンテナの子同士の水平方向の間隔です。	8 ピクセル
	labelWidth	ラベルの幅です。	コンテナによって子ラベルに基づいて計算
	paddingTop paddingBottom paddingLeft paddingRight	子の周囲の境界線の間隔です。	上下左右すべて 16 ピクセル
	indicatorGap	ラベル用に予約されたフォーム領域の終端から、FormItem の子または FormHeading までの間隔です。	14 ピクセル
FormHeading	indicatorGap	<mx:Form> タグで設定されたインジケータの間隔をオーバーライドします。	14 ピクセル
	paddingTop	コンポーネントの上端からラベルテキストまでの間隔です。	16 ピクセル
FormItem	direction	FormItem の子の方向 : vertical または horizontal	vertical
	horizontalGap	FormItem コンテナ内の子同士の水平方向の間隔です。	8 ピクセル
	labelWidth	FormItem の見出しの幅です。	ラベルテキストの幅
	paddingTop paddingBottom paddingLeft paddingRight	FormItem 周囲の境界線の間隔です。	上下左右すべて 0 ピクセル
	verticalGap	FormItem コンテナ内の子同士の垂直方向の間隔です。	6 ピクセル
	indicatorGap	<mx:Form> タグで設定されたインジケータの間隔をオーバーライドします。	<mx:Form> タグで決定

Form コンテナの子のサイズと位置の制御

Form レイアウトコンテナでは、子は垂直列状に配置されます。子に対して指定されたコンテナの領域は、Form コンテナ全体ではありません。ラベルとして定義された領域の右側に、indicatorGap プロパティで定義された間隔を空けて配置されます。たとえば、Form コンテナの幅が 500 ピクセルであり、そのうちの 100 ピクセルがラベルと indicatorGap プロパティに割り当てられている場合、子の領域の幅は 400 ピクセルです。

デフォルトでは、Flex は Form レイアウトの子を垂直方向にサイズ設定し、デフォルトの高さにします。次に、Flex はそれぞれの子のデフォルト幅を決定し、個別の子の幅を伸縮して、子の領域全体の幅の 1/4、1/2、3/4、または全体の値のうちで、最も近い値に切り上げて設定します。

たとえば、コンテナに存在する 1 つの子の領域の幅が 400 ピクセルだとします。TextArea コントロールのデフォルト幅が 125 ピクセルである場合、Flex は TextArea コントロールを水平方向に伸縮し、子の領域全体の幅の 1/4、1/2、3/4、または全体の値のうちで、最も近い値に切り上げて設定します。この場合は、1/2 である 200 ピクセルの境界線が設定されます。このサイズ変更アルゴリズムは、幅が明示的に指定されていないコンポーネントにのみ適用されます。これにより、コントロールの幅が不揃いであるためにコンテナの右端が不揃いになる問題を防ぐことができます。

また、子の height および width プロパティを使用して、フォーム内の子の高さや幅に、ピクセル値または Form サイズに対するパーセント値を明示的に設定することもできます。

デフォルトボタンの定義

コンテナの defaultButton プロパティを使用して、デフォルトの Button コントロールを定義します。任意のフォームコントロールにフォーカスがあるときに Enter キーを押すと、Button コントロールを明示的に選択したときと同じように、デフォルトの Button コントロールのクリックイベントがトリガされます。

たとえば、ログインフォームは、ユーザー名とパスワードの入力および送信用の Button コントロールを表示します。ユーザー名を入力し、Tab キーでパスワードフィールドに移動してパスワードを入力した後、Enter キーを押すと、ログイン情報を送信できます。Button コントロールを明示的に選択する必要はありません。このような操作を定義するには、defaultButton に、送信用 Button コントロールの id を設定します。次の例では、送信用ボタンの click イベントに対するイベントリスナーによって、Alert コントロールが表示されます。これは、いずれかのフォームフィールドにフォーカスがあるときにユーザーが Enter キーを押すと、Flex によってこのイベントがトリガされることを示します。この例の中でコメントアウトされている行は、ユーザーのログインを行うための Web サービスを起動する、実際のアクションを実行する部分です。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDefButton.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
```

```
        <![CDATA[
            import flash.events.MouseEvent;
            import mx.controls.Alert;

            private function submitLogin(eventObj:MouseEvent):void {
                // Display an Alert to show the event happened.
                Alert.show("Login Requested");
                // Commented out to work without a web service.
                //myWebService.Login.send();
            }
        ]]>
    </mx:Script>

    <mx:Form defaultButton="{mySubmitButton}">
        <mx:FormItem label="Username">
            <mx:TextInput id="username"
                width="100"/>
        </mx:FormItem>
        <mx:FormItem label="Password">
            <mx:TextInput id="password"
                width="100"
                displayAsPassword="true"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button id="mySubmitButton"
                label="Login"
                click="submitLogin(event);"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

メモ

ComboBox コントロールのドロップダウンリストが展開されているときに Enter キーを押すと、その ComboBox コントロールで現在ハイライト表示されているアイテムが選択されます。このとき、デフォルトボタンのクリックイベントはトリガされません。

必須フィールドの指定

Flex は、フォームの必須入力フィールドの定義をサポートします。必須フィールドを定義するには、FormItem コンテナの `required` プロパティを指定します。このプロパティを指定すると、その FormItem コンテナのすべての子に必須のマークが付きます。

Flex は、FormItem のラベルと子の間に区切り記号として赤いアスタリスク (*) 文字を挿入し、必須フィールドであることを示します。たとえば、次の例では、オプションの ZIP コードフィールドと必須の ZIP コードフィールドが示されています。

次のコード例では、これらのフィールドを定義します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormReqField.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form>
        <mx:FormItem label="ZIP Code">
            <mx:TextInput id="zipOptional"
                width="100"/>
        </mx:FormItem>

        <mx:FormItem label="ZIP Code" required="true">
            <mx:TextInput id="zipRequired"
                width="100"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

FormItem の子に対する必須インジケータを、実行時に有効にすることができます。これは、特定のフォームフィールドが入力されると別のフォームフィールドが必須になるようにする場合に便利です。たとえば、ユーザーがニュースレターを購読する場合に選択する CheckBox コントロールが、フォームに配置されているとします。このチェックボックスをオンにすると、ユーザーの電子メールフィールドが必須になるようにします。次に例を示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormReqFieldRuntime.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form>
        <mx:FormItem label="Subscribe">
            <mx:CheckBox label="Subscribe?"
                click="emAddr.required=true"/>
        </mx:FormItem>

        <mx:FormItem id="emAddr" label="e-mail address">
            <mx:TextInput id="emailAddr"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

Flex は、必須フィールドを自動的に適用することはありません。フィールドに必須のマークを付けるだけです。必須フィールドを適用するには、検証ロジックをフォームに組み込む必要があります。適用ロジックの一部として、Flex バリデータを使用できます。すべての Flex バリデータは required プロパティを備え、これはデフォルトで true にされています。バリデータは、必須フィールドと検証機能をどのように適用するかによって、いくつかの方法で使用できます。詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。フォームでバリデータを使用する例については、546 ページの「Flex データモデルによるフォームデータの保存」を参照してください。

フォームデータの保存と検証

フォームの設計の一環として、フォームデータを保存する方法を考える必要があります。Flex では、次の方法から選択できます。

- データをフォームコントロール内に保存します。
- Flex データモデルを作成してデータを保存します。

データをどのように表現するかによっても、入力エラーの検出、つまり " データ検証 " の方法が異なります。これは、堅牢で安定したフォームを構築する上で最も重要な作業の 1 つです。一般に、ユーザー入力の検証はデータをサーバーに送信する前に行います。ユーザー入力の検証は、送信関数内で行うことも、ユーザーがフォームにデータを入力したときに行うことも可能です。

Flex は、フォームによって収集されるデータのうち、特に使用頻度の高い種類のデータに対するさまざまなデータバリデータを備えています。Flex バリデータは、次のタイプのデータに対して使用できます。

- クレジットカード情報
- 日付
- 電子メールアドレス
- 数値
- 電話番号
- 社会保障番号
- 文字列
- ZIP コード

データ検証は、フォーム構築プロセスの一環として、独自のカスタムロジックを使って実行するか、Flex のデータ検証メカニズムを利用します。または、カスタムロジックと Flex データ検証を組み合わせて実行します。

次のセクションでは、フォーム内で検証機能を起動する方法について説明します。Flex データ検証の詳細な使用方法については、1165 ページ、第 40 章の「データ検証」を参照してください。

Form コントロールによるフォームデータの保持

次の例では、Form コントロールを使用してフォームデータを保存します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormData.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function processValues(zip:String, pn:String):void {
                // Validate and process data.
            }
```

```
        ]]>
    </mx:Script>

    <mx:Form id="myForm" defaultButton="{mySubmitButton}">

        <mx:FormItem label="Zip Code">
            <mx:TextInput id="zipCode"/>
        </mx:FormItem>
        <mx:FormItem label="Phone Number">
            <mx:TextInput id="phoneNumber"/>
        </mx:FormItem>

        <mx:FormItem>
            <mx:Button label="Submit" id="mySubmitButton"
                click="processValues(zipCode.text, phoneNumber.text);"/>
        </mx:FormItem>

    </mx:Form>
</mx:Application>
```

この例では、フォームが 2 つのフォームコントロールを定義します。1 つは ZIP コード用、もう 1 つは電話番号用です。フォームを送信すると、各コントロールに格納されたデータに対応する 2 つのパラメータをとる関数が呼び出されます。次に、フォームデータの処理の前に、入力に対するデータ検証が送信関数によって実行されます。

データを送信関数に渡す必要はありません。次の例に示すように、送信関数はフォームコントロールデータに直接アクセスできます。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDataSubmitNoArg.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function processValues():void {
                var inputZip:String = zipCode.text;
                var inputPhone:String = phoneNumber.text;
                // Check to see if pn is a number.
                // Check to see if zip is less than 4 digits.
                // Process data.
            }
        ]]>
    </mx:Script>

    <mx:Form id="myForm" defaultButton="{mySubmitButton}">

        <mx:FormItem label="Zip Code">
            <mx:TextInput id="zipCode"/>
        </mx:FormItem>
        <mx:FormItem label="Phone Number">
```

```
            <mx:TextInput id="phoneNumber"/>
        </mx:FormItem>

        <mx:FormItem>
            <mx:Button label="Submit" id="mySubmitButton"
                click="processValues();"/>
        </mx:FormItem>

    </mx:Form>
</mx:Application>
```

しかし、このようにフォームフィールドを直接使用する方法には、関数がフォーム固有のものとなるため、他のフォームで容易に再利用できないという欠点があります。

ユーザー入力時のフォームコントロールデータの検証

ユーザー入力時にフォームデータを検証するために、アプリケーションに Flex データバリデータを追加できます。次の例では、ZipCodeValidator と PhoneNumbervalidator を使用して検証を行います。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDataValidate.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            private function processValues():void {
                var inputZip:String = zipCode.text;
                var inputPhone:String = phoneNumber.text;
                // Perform any additional validation.
                // Process data.
            }
        ]]>
    </mx:Script>

    <mx:ZipCodeValidator id="zcVal"
        source="{zipCode}" property="text"
        domain="US or Canada"/>

    <mx:PhoneNumberValidator id="pnVal"
        source="{phoneNumber}" property="text"/>

    <mx:Form id="myForm" defaultButton="{mySubmitButton}">

        <mx:FormItem label="Zip Code">
            <mx:TextInput id="zipCode"/>
        </mx:FormItem>
        <mx:FormItem label="Phone Number">
            <mx:TextInput id="phoneNumber"/>
        </mx:FormItem>
```

```
        <mx:FormItem>
            <mx:Button label="Submit" id="mySubmitButton"
                click="processValues();"/>
        </mx:FormItem>

    </mx:Form>
</mx:Application>
```

ユーザーが入力するたびに入力データを検証する場合は、送信関数内で再び検証を行わなくてもかまいません。ただし、2 つのフィールドを比較するときに両フィールドが有効であることを確認する場合など、送信関数内で再び検証を実行しなければならないこともあります。

たとえば、Flex バリデータを使用して ZIP コードフィールドと状態フィールドを個別に検証することができます。ここで、フォームデータを送信する前に、特定の状態に対して ZIP コードが有効であることを検証する必要があるとします。このような場合には、送信関数内で 2 回目の検証を実行します。

バリデータの使用の詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

Flex データモデルによるフォームデータの保存

Flex のデータモデルを使用すると、フォームデータを構造化および保存して、データ検証のフレームワークを提供できます。データモデルでは、特定のデータセットの各部分を表すフィールドにデータが格納されます。たとえば、person モデルは個人の氏名、年齢、電話番号などの情報を保存します。これにより、各モデルフィールドに保存されたデータ型に基づき、モデル内のデータを検証できます。

次の例では、2 つの値を含む Flex データモデルを定義しています。この 2 つの値は、フォームの次の 2 つの入力フィールドに対応します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDataModel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define the submit function that validates and
        processes the data. -->
    <mx:Script>
        <![CDATA[
            private function processValues():void {
                var inputZip:String = myFormModel.zipCodeModel;
                var inputPhone:String = myFormModel.phoneNumberModel;
                // Process data.
            }
        ]]>
    </mx:Script>

    <!-- Define data model. -->
    <mx:Model id="myFormModel">
        <info>
            <zipCodeModel>{zipCode.text}</zipCodeModel>
            <phoneNumberModel>{phoneNumber.text}</phoneNumberModel>
        </info>
```

```
    </mx:Model>

    <!-- Define the form. -->
    <mx:Form borderStyle="solid">
        <mx:FormItem label="Zip Code">
            <mx:TextInput id="zipCode"/>
        </mx:FormItem>
        <mx:FormItem label="Phone Number">
            <mx:TextInput id="phoneNumber"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button id="b1"
                label="Submit"
                click="processValues();"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

<mx:Model> タグを使用してデータモデルを定義します。データモデルのすべての子タグは、それぞれがモデルの 1 つのフィールドを定義します。モデル内のすべての子タグのボディ部分は、それぞれフォームコントロールへの " バインディング " を定義します。この例では、zipCodeModel モデルフィールドを zipCode TextInput コントロールのテキスト値にバインドし、phoneNumberModel フィールドを phoneNumber TextInput コントロールのテキスト値にバインドします。データモデルの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

コントロールをデータモデルにバインドすると、ユーザーからの入力があるたびに、コントロールのデータが自動的にモデルにコピーされます。この例では、送信関数によるデータへのアクセスが、フォームコントロールから直接行われるのではなく、モデルから行われています。

フォームモデルでの Flex バリデータの使用

次の例では、546 ページの「Flex データモデルによるフォームデータの保存」で示した例を修正して、2 つのデータバリデータ (郵便番号フィールドと電話番号フィールドのバリデータ) を挿入しています。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDataModel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define the submit function that validates and processes the data -->
    <mx:Script>
        <![CDATA[
            private function processValues():void {
                var inputZip:String = myFormModel.zipCodeModel;
                var inputPhone:String = myFormModel.phoneNumberModel;
                // Process data.
            }
        ]]>
    </mx:Script>
```

```
    <!-- Define data model. -->
    <mx:Model id="myFormModel">
        <info>
            <zipCodeModel>{zipCode.text}</zipCodeModel>
            <phoneNumberModel>{phoneNumber.text}</phoneNumberModel>
        </info>
    </mx:Model>

    <!-- Define validators. -->
    <mx:ZipCodeValidator
        source="{myFormModel}" property="zipCodeModel"
        trigger="{zipCode}"
        listener="{zipCode}"/>
    <mx:PhoneNumberValidator
        source="{myFormModel}" property="phoneNumberModel"
        trigger="{b1}"
        listener="{phoneNumber}"
        triggerEvent="click"/>

    <!-- Define the form. -->
    <mx:Form borderStyle="solid">
        <mx:FormItem label="Zip Code">
            <mx:TextInput id="zipCode"/>
        </mx:FormItem>
        <mx:FormItem label="Phone Number">
            <mx:TextInput id="phoneNumber"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button id="b1"
                label="Submit"
                click="processValues();"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

ユーザーがデータを zipCode フォームフィールドに入力すると、このデータはデータモデルに自動的にコピーされます。ZipCodeValidator バリデータは、ユーザーが zipCode フォームフィールドの入力を終えると起動します。この動作は、バリデータの trigger プロパティ、および triggerEvent プロパティのデフォルト値である valueCommit によって定義されたものです。次に、Flex は zipCode フィールドの周囲に赤い境界線を表示しますが、これは listener プロパティで定義された動作です。

ユーザーがデータを phoneNumber フォームフィールドに入力すると、データはデータモデルに自動的にコピーされます。ユーザーが Button コントロールをクリックすると、PhoneNumberValidator バリデータが起動します。これは、バリデータの trigger および triggerEvent プロパティによって指定された動作です。次に Flex は listener プロパティの指定に従い、phoneNumber フィールドの周囲に赤い境界線を表示します。

バリデータの使用の詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

データモデルから Form コントロールへの値の入力

データモデルの別の用途として、モデルにデータを格納してフォームフィールドに値を入力することがあります。次の例では、データモデルから静的データを読み取ってフォームフィールドの値を取得するフォームを示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDataFromModel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define data model. -->
    <mx:Model id="myFormModel">
        <info>
            <fName>{firstName.text}</fName>
            <lName>{lastName.text}</lName>
            <department>Accounting</department>
        </info>
    </mx:Model>

    <mx:Form>
        <mx:FormItem label="First and Last Names">
            <mx:TextInput id="firstName"/>
            <mx:TextInput id="lastName"/>
        </mx:FormItem>
        <mx:FormItem label="Department">
            <mx:TextInput id="dept" text="{myFormModel.department}"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

このフォームでは部署フィールドに対して常に同じ値が表示されるので、このデータは静的であると見なされます。また、Web サービスから部署フィールドの値を取得する動的データモデルや、ユーザー入力に基づいて計算する動的データモデルも作成できます。

データモデルの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

サーバーへのデータの送信

フォームデータは、通常、クライアントでローカルに処理されるのではなく、サーバーで処理されます。したがって、送信イベントリスナーには、サーバーへの送信用にフォームデータをパックし、サーバーから返された結果を処理するメカニズムが必要です。Flex では、一般的に、Web サービス、HTTP サービス、またはリモート Java オブジェクトを使用して、データをサーバーに渡します。

送信関数にロジックを組み込むことで、送信が成功した場合または失敗した場合のアプリケーションのナビゲーションを制御することもできます。送信が成功すると、多くの場合は、結果を表示するアプリケーション領域に移動します。送信が失敗したときには、ユーザーがエラーを修正できるように、フォームに制御を戻すことができます。

次の例では、フォーム入力データを処理する Web サービスを追加します。この例では、ユーザーは ZIP コードを入力してから [Submit] ボタンを押します。データ検証の実行後、送信イベントリスナーは ZIP コードに対応する町名、現在の気温、および天気予報を取得するための Web サービスを呼び出します。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDataSubmitServer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define the web service connection.
        The specified WSDL URI is not functional. -->
    <mx:WebService id="WeatherService"
        wsdl="/ws/WeatherService?wsdl">
        <mx:operation name="GetWeather">
            <mx:request>
                <ZipCode>{zipCode.text}</ZipCode>
            </mx:request>
        </mx:operation>
    </mx:WebService>

    <mx:Script>
        <![CDATA[
            private function processValues():void {
                // Check to see if ZIP code is valid.
                WeatherService.GetWeather.send();
            }
        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="Zip Code">
            <mx:TextInput id="zipCode"
                width="200"
                text="Zip code please?"/>
            <mx:Button
                width="60"
                label="Submit"
                click="processValues();"/>
        </mx:FormItem>
    </mx:Form>

    <mx:VBox>
        <mx:TextArea
            text=
              "{WeatherService.GetWeather.lastResult.CityShortName}"/>
        <mx:TextArea
            text=
              "{WeatherService.GetWeather.lastResult.CurrentTemp}"/>
        <mx:TextArea
            text=
```

```
            "{WeatherService.GetWeather.lastResult.DayForecast}"/>
    </mx:VBox>
</mx:Application>
```

この例では、フォームの入力フィールド zipCode を、Web サービス要求の ZipCode フィールドに直接バインドします。Web サービスから取得した結果を表示するには、結果を VBox コンテナのコントロールにバインドします。

Web サービスへのデータの受け渡しは、柔軟に行うことができます。たとえば、この例を修正して、入力フォームフィールドをデータモデルにバインドしてから、データモデルを Web サービス要求にバインドすることができます。Web サービスの使用の詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

Web サービス用のイベントリスナーを追加することもできます。これには、成功した呼び出しを処理するには result イベントを使用し、エラーの発生した呼び出しを処理するには fault イベントを使用します。エラーの状況によっては、エラーメッセージと共にエラーの内容の説明をユーザーに表示する必要があります。呼び出しが成功した場合は、アプリケーションの別のセクションに移動できます。

次の例では、load イベントと fault イベントをフォームに追加します。この例では、フォームが ViewStack コンテナの 1 つの子として定義され、フォーム結果は ViewStack コンテナの 2 番目の子として定義されています。

```
<?xml version="1.0"?>
<!-- containers\layouts\FormDataSubmitServerEvents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define the web service connection.
        The specified WSDL URI is not functional. -->
    <mx:WebService id="WeatherService"
        wsdl="/ws/WeatherService?wsdl"
        result="successfulCall();"
        fault="errorCall();">
        <mx:operation name="GetWeather">
            <mx:request>
                <ZipCode>{zipCode.text}</ZipCode>
            </mx:request>
        </mx:operation>
    </mx:WebService>

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;

            private function processValues():void {
                // Check to see if ZIP code is valid.
                WeatherService.GetWeather.send();
            }
```

```
            private function successfulCall():void {
                vs1.selectedIndex=1;
            }

            private function errorCall():void {
                Alert.show("Web service failed!", "Alert Box", Alert.OK);
            }
        ]]>
    </mx:Script>

    <mx:ViewStack id="vs1">
        <mx:Form>
            <mx:FormItem label="Zip Code">
                <mx:TextInput id="zipCode"
                    width="200"
                    text="Zip code please?"/>
                <mx:Button width="60"
                    label="Submit"
                    click="processValues();"/>
            </mx:FormItem>
        </mx:Form>

        <mx:VBox>
            <mx:TextArea
                text=
                    "{WeatherService.GetWeather.lastResult.CityShortName}"/>
            <mx:TextArea
                text=
                    "{WeatherService.GetWeather.lastResult.CurrentTemp}"/>
            <mx:TextArea
                text=
                    "{WeatherService.GetWeather.lastResult.DayForecast}"/>
        </mx:VBox>
    </mx:ViewStack>
</mx:Application>
```

Web サービスへの呼び出しが成功すると、successfulCall() 関数によって、現在の ViewStack の子が VBox コンテナに切り替わり、取得した結果が表示されます。Web サービスからエラーが返されると、[Alert] ボックスが表示されます。現在の ViewStack コンテナの子は変更されず、フォームは表示されたままなので、ユーザーは入力エラーを修正できます。

送信関数の結果に応じてアプリケーションでのナビゲーションを処理するオプションが多数あります。前の例では、ViewStack コンテナを使用してナビゲーションを処理します。TabNavigator コンテナや Accordion コンテナを使用して処理することもできます。

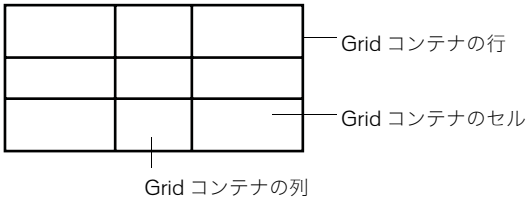
アプリケーションによっては、フォームを TitleWindow コンテナに埋め込む方法も選択できます。TitleWindow コンテナは、Adobe Flash Player の描画面に表示されるポップアップウィンドウです。このシナリオでは、ユーザーは TitleWindow コンテナからフォームデータを入力してフォームを送信します。送信が成功すると、TitleWindow コンテナは閉じられ、アプリケーションの別の領域に送信結果が表示されます。送信が失敗した場合はエラーメッセージが表示され、TitleWindow コンテナは表示されたままとなります。

別のタイプのアプリケーションでは、複数のパネルをダッシュボードで開くことができるダッシュボードレイアウトを使用する場合があります。フォームを送信すると、ダッシュボードの別の領域が更新され、そこに結果が表示されます。送信が失敗した場合はエラーメッセージが表示されます。

TabNavigator、Accordion、および TitleWindow コンテナの詳細については、583 ページ、第 16 章の「ナビゲータコンテナの使用」を参照してください。

Grid レイアウトコンテナ

Grid レイアウトコンテナを使用すると、セルで構成される行と列として子を配置できます。これは HTML テーブルとよく似ています。次の例に、9 つのセルが 3 x 3 のパターンで構成されている Grid コンテナを示します。



Grid コンテナの各セルには、0 または 1 つの子を配置できます。1 つのセル内に複数の子を配置する場合は、セル内にコンテナを配置してから、そのコンテナに子を追加します。1 つの行内のセルの高さはすべて同じですが、各行の高さは異なっていてもかまいません。1 つの列内のセルの幅はすべて同じですが、各列の幅は異なっていてもかまいません。

Grid コンテナの行または列ごとに、異なるセル数を定義できます。さらに、コンテナの複数の列や行にまたがる1つのセルを配置することもできます。

Grid、GridRow、および GridItem コンテナには、以下のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
グリッドの高さ	すべての行のデフォルトの高さ、または明示的な高さの合計に、行間の間隔を加えた高さです。
グリッドの幅	すべての列のデフォルトの幅、または明示的な幅の合計に、列間の間隔を加えた高さです。
各行および各セルの高さ	行内で最も高いアイテムのデフォルトの高さ、または明示的な高さです。GridItem コンテナに明示的なサイズが設定されていない場合は、このデフォルトの高さは、セル内の子のデフォルトまたは明示的な高さとなります。
各列および各セルの幅	列内で最も幅広いアイテムのデフォルトの幅、または明示的な幅です。GridItem コンテナに明示的な幅が設定されていない場合は、このデフォルトの幅は、セル内の子のデフォルトまたは明示的な幅となります。
行と列間の間隔	Grid クラスの `horizontalGap` および `verticalGap` プロパティによって決まります。どちらの間隔も、デフォルト値は 6 ピクセルです。
デフォルトパディング	3 つのコンテナクラスすべてにおいて、`top`、`bottom`、`left`、および `right` の各値が 0 ピクセルです。

明示的にサイズ設定したセルより、その子のデフォルトまたは明示的なサイズが大きい場合は、子はセルの境界線でクリッピングされます。

子の幅または高さのデフォルト値がセルよりも小さい場合、デフォルトでは、セル内の子の水平方向の整列は `left` であり、垂直方向の整列は `top` です。子の配置を制御するには、`<mx:GridItem>` タグの `horizontalAlign` および `verticalAlign` プロパティを使用します。

詳細については、『Adobe Flex 2 リファレンスガイド』の Grid、GridRow、および GridItem を参照してください。すべてのセルを同じサイズでレイアウトする機能を持つ Tile コンテナの詳細については、563 ページの「Tile レイアウトコンテナ」を参照してください。

Grid レイアウトコンテナの作成

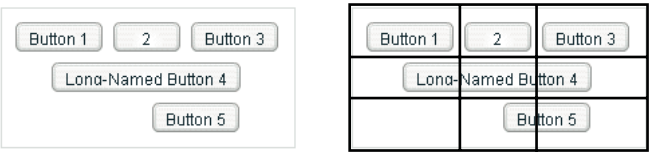
Grid レイアウトコンテナは、次のように作成します。

- Grid コンテナを定義するには、`<mx:Grid>` タグを使用します。このタグ内には、任意の数の `<mx:GridRow>` 子タグを含めることができます。
- 各行を定義するには、`<mx:GridRow>` タグを使用します。これは `<mx:Grid>` タグの子である必要があり、任意の数の `<mx:GridItem>` 子タグを含めることができます。
- `<mx:GridItem>` タグを使用して、行のセルを定義します。これは `<mx:GridRow>` タグの子である必要があります。

<mx:GridItem> タグには、アイテムの配置を定義する以下のオプションプロパティを指定できます。

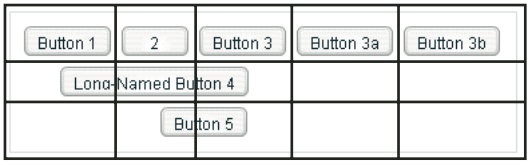
プロパティ	データ型	用途	説明
rowSpan	Number	プロパティ	Grid コンテナのセルの範囲となる行数を指定します。デフォルト値は 1 です。1 つのセルを、Grid コンテナ内の行数を超えて拡張することはできません。
colSpan	数値	プロパティ	Grid コンテナのセルの範囲となる列数を指定します。デフォルト値は 1 です。1 つのセルを、Grid 内の列数を超えて拡張することはできません。

次の図では、3 行と 3 列で構成される Grid コンテナを示しています。



左側の図は、Grid コンテナが Flash Player でどのように表示されるかを示します。右側の図は、行と列の構成を示す分割線が配置された Grid コンテナです。この例では、先頭 (最上) 行の各ボタンがそれぞれ 1 つのセルを占めています。2 番目の行のボタンは 3 つの列を占めており、3 番目の行のボタンは 2 列目と 3 列目にまたがっています。

Grid コンテナ内の各行に対し、同じ数のセルを定義する必要はありません。これを次の図で示します。この Grid コンテナの 1 行目では、5 つのセルが定義されています。2 行目では、3 つのセルにまたがる 1 つのアイテムが配置されています。3 行目では、1 つの空白セルに続き、2 つのセルにまたがる 1 つのアイテムが配置されています。



次の MXML コードでは、このセクションの最初の図に示した、3 つの行と 3 つの列からなる Grid コンテナを作成します。

```
<?xml version="1.0"?>
<!-- containers\layouts\GridSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Grid id="myGrid">

        <!-- Define Row 1. -->
        <mx:GridRow id="row1">
            <!-- Define the first cell of Row 1. -->
```

```
            <mx:GridItem>
                <mx:Button label="Button 1"/>
            </mx:GridItem>
            <!-- Define the second cell of Row 1. -->
            <mx:GridItem>
                <mx:Button label="2"/>
            </mx:GridItem>
            <!-- Define the third cell of Row 1. -->
            <mx:GridItem>
                <mx:Button label="Button 3"/>
            </mx:GridItem>
        </mx:GridRow>

        <!-- Define Row 2. -->
        <mx:GridRow id="row2">
            <!-- Define a single cell to span three columns of Row 2. -->
            <mx:GridItem colSpan="3" horizontalAlign="center">
                <mx:Button label="Long-Named Button 4"/>
            </mx:GridItem>
        </mx:GridRow>

        <!-- Define Row 3. -->
        <mx:GridRow id="row3">
            <!-- Define an empty first cell of Row 3. -->
            <mx:GridItem/>
            <!-- Define a cell to span columns 2 and 3 of Row 3. -->
            <mx:GridItem colSpan="2" horizontalAlign="center">
                <mx:Button label="Button 5"/>
            </mx:GridItem>
        </mx:GridRow>

    </mx:Grid>
</mx:Application>
```

この例を修正し、先頭行に 5 つのボタンを配置するには、最初の <mx:GridRow> タグを次のように変更します。

```
<?xml version="1.0"?>
<!-- containers\layouts\Grid5Button.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Grid id="myGrid">

        <!-- Define Row 1. -->
        <mx:GridRow id="row1">
            <!-- Define the first cell of Row 1. -->
            <mx:GridItem>
                <mx:Button label="Button 1"/>
            </mx:GridItem>
            <mx:GridItem>
                <mx:Button label="2"/>
```

```
            </mx:GridItem>
            <mx:GridItem>
                <mx:Button label="Button 3"/>
            </mx:GridItem>
            <mx:GridItem>
                <mx:Button label="Button 3a"/>
            </mx:GridItem>
            <mx:GridItem>
                <mx:Button label="Button 3b"/>
            </mx:GridItem>
        </mx:GridRow>

        <!-- Define Row 2. -->
        <mx:GridRow id="row2">
            <!-- Define a single cell to span three columns of Row 2. -->
            <mx:GridItem colSpan="3" horizontalAlign="center">
                <mx:Button label="Long-Named Button 4"/>
            </mx:GridItem>
        </mx:GridRow>

        <!-- Define Row 3. -->
        <mx:GridRow id="row3">
            <!-- Define an empty first cell of Row 3. -->
            <mx:GridItem/>
            <!-- Define a cell to span columns 2 and 3 of Row 3. -->
            <mx:GridItem colSpan="2" horizontalAlign="center">
                <mx:Button label="Button 5"/>
            </mx:GridItem>
        </mx:GridRow>

    </mx:Grid>
</mx:Application>
```

行と列の範囲の設定

GridItem コンテナの colSpan および rowSpan プロパティを使用すると、複数のグリッド行やグリッド列を占めるグリッドアイテムを作成できます。複数の行または列を占めるアイテムを作成する場合、その子コントロールまたは子コンテナは必ずしも拡大されません。領域に合わせて、子のサイズを適切に調整する必要があります。この例を次に示します。

次に、「Grid レイアウトコンテナの作成」の例を変更した図を示します。ここでは新たに、Button 3a は 2 つの行、Button 3b は 3 つの行、Button 5 は 3 つの列をそれぞれ占めています。

次のコードは、この結果を得るために加えた変更内容を示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\GridRowSpan.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Grid id="myGrid">

        <!-- Define Row 1. -->
        <mx:GridRow id="row1" height="33%">
            <!-- Define the first cell of Row 1. -->
            <mx:GridItem>
                <mx:Button label="Button 1"/>
            </mx:GridItem>
            <mx:GridItem>
                <mx:Button label="2"/>
            </mx:GridItem>
            <mx:GridItem>
                <mx:Button label="Button 3"/>
            </mx:GridItem>
            <mx:GridItem rowSpan="2">
                <mx:Button label="Button 3a" height="100%"/>
            </mx:GridItem>
            <mx:GridItem rowSpan="3">
                <mx:Button label="Button 3b" height="100%"/>
            </mx:GridItem>
        </mx:GridRow>

        <!-- Define Row 2. -->
        <mx:GridRow id="row2" height="33%">
            <!-- Define a single cell to span three columns of Row 2. -->
            <mx:GridItem colSpan="3" horizontalAlign="center">
                <mx:Button label="Long-Named Button 4"/>
            </mx:GridItem>
        </mx:GridRow>

        <!-- Define Row 3. -->
        <mx:GridRow id="row3" height="33%">
            <!-- Define an empty first cell of Row 3. -->
            <mx:GridItem/>
            <!-- Define a cell to span columns 2 and 3 and 4 of Row 3. -->
            <mx:GridItem colSpan="3">
                <mx:Button
                    label="Button 5 expands across 3 columns"
                    width="75%"/>
            </mx:GridItem>
        </mx:GridRow>

    </mx:Grid>
</mx:Application>
```

この例で行った変更には、以下の効果があります。

- 各行の高さをグリッドの 33% に設定することで、すべての行を同じ高さにしています。
- Buttons 3a および 3b の `rowSpan` プロパティを設定し、各ボタンがそれぞれ 2 行、および 3 行にまたがるようにしています。
- Buttons 3a および 3b の `height` プロパティを 100% に設定し、これらのボタンが、それぞれがまたがる全行を占有するようにします。これらのボタンの height プロパティを設定しないと、各ボタンの高さはデフォルト値に設定されるため、複数行にまたがって表示されなくなります。
- Button 5 が 3 行にまたがるように設定し、幅をパーセント値で 75 % に設定します。この例では、指定のテキストを表示するために、このボタンが 3 列の有効幅すべてを使用する必要があります。したがって、このボタンは指定した 75 % ではなく、テキストがすべて収まるようにデフォルトサイズに設定されます。`width` プロパティの設定を省略した場合も、同じ結果となります。パーセント値による幅の指定が反映されるようにするには、指定内容はそのままにして、ラベルテキストの長さを短くします。これで、ボタンは 3 列の 3/4 を占め、中央揃えで中央の列に表示されるようになります。

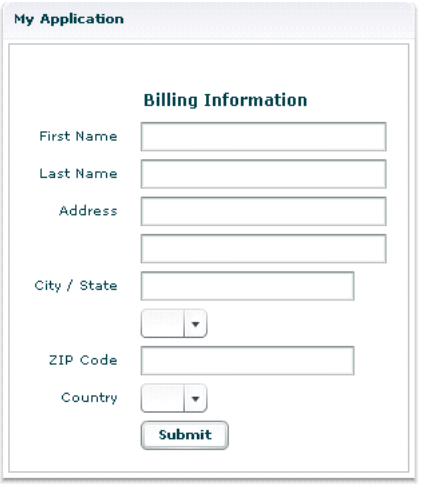
この結果作成されるグリッドには、いくつかの特徴が追加されます。2 行目では、3 列にまたがる 1 つのセルを定義する `<mx:GridItem>` タグが 1 つしか定義されていませんが、Buttons 3a および 3b がこの行に拡張表示されるように、さらに 2 つのセルが自動的に作成されます。3 行目も、5 つのセルからなります。先頭セルは、空白の `<mx:gridItem/>` タグによって定義されます。2 ～ 4 番目のセルは、3 列にまたがる Button 5 を保持する `GridItem` によって定義されます。5 列目は、1 行目の最後のアイテムが、3 列すべてにまたがるように指定されているために作成されます。

Panel レイアウトコンテナ

Panel レイアウトコンテナは、タイトルバー、タイトル、ステータスメッセージ領域 (タイトルバー内)、境界線、および子のコンテンツ領域で構成されます。一般に、Panel コンテナは、単体で使用可能なアプリケーションモジュールをラップする場合に使用します。たとえば、アプリケーションに対して 3 つの Panel コンテナを定義し、それぞれに、フォーム、ショッピングカート、カタログを表示させることが可能になります。

Panel コンテナの `layout` プロパティを使用すると、水平、垂直 (デフォルト)、または絶対レイアウトのいずれかを指定できます。水平および垂直レイアウトでは、Flex の自動レイアウト規則に基づき、子が水平または垂直に配置されます。絶対レイアウトを使用する場合は、パネルコンテンツ領域に対する各子の位置を x および y 座標で指定する必要があります。または、制約ベースのレイアウトスタイルを使用して、パネルコンテンツ領域に対し、1 つ以上の端、あるいはコンテナの水平または垂直方向の中心を固定します。コンテナで絶対レイアウトおよび制約ベースのレイアウトを使用する例については、519 ページの「Canvas コントロールの作成と使用」を参照してください。それぞれのレイアウト方法の詳細については、207 ページ、第 8 章の「コンポーネントのサイズと位置の制御」を参照してください。

次の例は、Form コンテナを子として持つ Panel コンテナです。



Panel コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	height は、デフォルトの高さでのすべての子、垂直方向の子同士の間隔、上下のパディング、上下の境界線、およびタイトルバーを含めることのできる値です。width は、最も幅の広い子のデフォルト幅にコンテナの左右のパディングを加えた値、またはタイトルテキストの幅に境界線を加えた値のうち、いずれか大きい方の値です。
パディング	top、bottom、left、および right の各値が 4 ピクセルです。

詳細については、『Adobe Flex 2 リファレンスガイド』の Panel を参照してください。

Panel レイアウトコンテナの作成

MXML で Panel コンテナを定義するには、<mx:Panel> タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、id 値を指定します。

次の例では、アプリケーションの最上位コンテナとしてフォームを保持する Panel コンテナを定義します。この例では、標準の GUI ウィンドウと同じようにタイトルバーを配置する Panel コンテナのメカニズムを示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\PanelSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Panel id="myPanel" title="My Application">
```

```
        <mx:Form width="300">
            <mx:FormHeading label="Billing Information"/>
            <!-- Form contents goes here -->
        </mx:Form>

    </mx:Panel>
</mx:Application>
```

Panel コンテナへの ControlBar コンテナの追加

ControlBar コンテナを Panel コンテナと共に使用し、Panel コンテナ内の他の子が共有できるコンポーネントを保持できます。たとえば RichTextEditor コントロールは、TextArea コントロールを備えた Panel コントロールと、テキストの書式コントロールを保持するカスタム ControlBar によって構成されます。次の例に示すように、製品カタログの場合は、数量の指定やショッピングカートへの商品の追加を行う Flex コントロールを ControlBar コンテナに配置できます。

次の例に示すように、<mx:Panel> タグの最後の子タグとして <mx:ControlBar> タグを指定します。

```
<?xml version="1.0"?>
<!-- containers\layouts\PanelCBar.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Panel title="My Application"
        paddingTop="10" paddingBottom="10"
        paddingLeft="10" paddingRight="10"
        width="300">

        <mx:Script>
            <![CDATA[
                private function addToCart():void {
                    // Handle event.
                }
            ]]>
        </mx:Script>
```

```
        <mx:HBox width="100%">
            <!-- Area for your catalog. -->
        </mx:HBox>

        <mx:ControlBar width="100%">
            <mx:Label text="Quantity"/>
            <mx:NumericStepper id="myNS"/>
            <!-- Use Spacer to push Button control to the right. -->
            <mx:Spacer width="100%"/>
            <mx:Button label="Add to Cart" click="addToCart();"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

ControlBar コンテナの詳細については、525 ページの「ControlBar レイアウトコンテナ」を参照してください。

次の例に示すように、ControlBar コンテナを Panel コンテナに動的に追加することもできます。

```
<?xml version="1.0"?>
<!-- containers\layouts\PanelCBarDynamicAdd.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.containers.ControlBar;
            import mx.controls.*;
            import flash.events.MouseEvent;

            private var myCB:ControlBar=new ControlBar();
            private var myLabel:Label=new Label();
            private var myNS:NumericStepper=new NumericStepper();
            private var mySpacer:Spacer=new Spacer();
            private var myButton:Button=new Button();

            private function addCBHandler():void {
                // Create Controlbar control.
                myLabel.text="Quantity";
                mySpacer.percentWidth=100;
                myButton.label="Add to Cart";
                myButton.addEventListener('click', addToCart);


                myCB.percentWidth=100;
                myCB.addChild(myLabel);
                myCB.addChild(myNS);
                myCB.addChild(mySpacer);
                myCB.addChild(myButton);
```

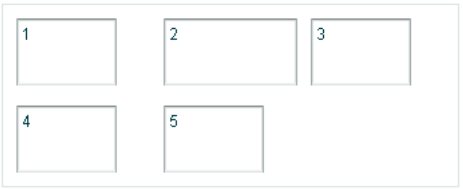
```
                // Add the ControlBar as the last child of the
                // Panel container.
                // The ControlBar appears in the normal content area
                // of the Panel container.
                myPanel.addChildAt(myCB, myPanel.numChildren);
                // Call createComponentsFromDescriptors() to make the
                // ControlBar appear in the bottom border area
                // of the Panel container. The ControlBar must be the
                // last child in the Panel container.
                myPanel.createComponentsFromDescriptors();
            }

            private function addToCart(event:MouseEvent):void {
                Alert.show("ControlBar Button clicked.");
            }
        ]]>
    </mx:Script>

    <mx:Panel id="myPanel"
        title="My Application"
        paddingTop="10" paddingBottom="10"
        paddingLeft="10" paddingRight="10"
        width="300">

        <mx:HBox width="100%">
            <!-- Area for your catalog. -->
        </mx:HBox>

        <mx:Button label="Add ControlBar" click="addCBHandler();"/>

    </mx:Panel>
</mx:Application>
```

Tile レイアウトコンテナ

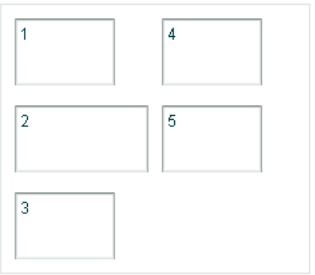
Tile レイアウトコンテナは、子を 1 つまたは複数の垂直列または水平行に配置し、必要に応じて新しい行や列を開始します。direction プロパティによってレイアウトが決まります。direction プロパティの有効値は、列レイアウトの場合は vertical、行レイアウトの場合は horizontal (デフォルト) です。

Grid レイアウトコンテナのセルと異なり (553 ページの「Grid レイアウトコンテナ」を参照)、Tile コンテナのすべてのセルは同一サイズです。Flex は Tile コンテナのセルを四角形のグリッド内に配置します。各セルには、1 つの子コンポーネントが保持されます。たとえば、16 個の子を Tile レイアウトコンテナに定義すると、Flex は、4 つのセルの幅と 4 つのセルの高さでレイアウトします。13 個の子を定義した場合は、4 つのセルの幅と 4 つのセルの高さでレイアウトされますが、4 番目の列内の最後の 3 つのセルは空のままになります。

次の図に、水平および垂直方向の Tile コンテナの例を示します。



水平方向 (デフォルト)



垂直方向

Tile コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
方向	水平方向
全セルのデフォルトサイズ	height は、最も高い子のデフォルトまたは明示的な高さです。width は、最も幅の広い子のデフォルトまたは明示的な幅です。すべてのセルのデフォルトサイズは同一です。
Tile コンテナのデフォルトサイズ	Flex は、子の数の平方根を計算し、最も近い整数に切り上げます。たとえば、子の数が 26 である場合の平方根は 5.1 ですが、6 に切り上げられます。したがって Flex は、Tile コンテナを 6 x 6 グリッドでレイアウトします。 Tile コンテナのデフォルトの高さは、(タイルセルのデフォルトの高さ) * (子の数の切り上げられた平方根) で求めた値に、子同士の間隔とパディングをすべて加算した値となります。デフォルトの幅は、(タイルセルのデフォルトの幅) * (子の数の切り上げられた平方根) で求めた値に、子同士の間隔とパディングをすべて加算した値となります。
Tile コンテナの最小サイズ	1 つのセルのデフォルトサイズです。Flex は常に、少なくとも 1 つのセルを表示するための十分な領域を割り当てます。
デフォルトパディング	`top`、`bottom`、`left`、`right` ともに 0 ピクセル。

詳細については、『Adobe Flex 2 リファレンスガイド』の Tile を参照してください。

Tile レイアウトコンテナの作成

MXML で Tile コンテナを定義するには、`<mx:Tile>` タグを使用します。MXML の他の場所 (別のタグまたは ActionScript ブロック) のコンポーネントを参照する場合は、`id` 値を指定します。`tileHeight` および `tileWidth` プロパティを使用すると、タイルのサイズを明示的に指定できます。

次の例では、563 ページの「Tile レイアウトコンテナ」の図で示した水平方向の Tile コンテナを作成します。すべてのセルには、最大の子の高さと幅が適用されます (高さ 50 ピクセル、幅 100 ピクセル)。

```
<?xml version="1.0"?>
<!-- containers\layouts\TileSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Tile id="myFlow"
        direction="horizontal"
        borderStyle="solid"
        paddingTop="10" paddingBottom="10"
        paddingRight="10" paddingLeft="10"
        verticalGap="15" horizontalGap="10">

        <mx:TextInput id="text1" text="1" height="50" width="75"/>
        <mx:TextInput id="text2" text="2" height="50" width="100"/>
        <mx:TextInput id="text3" text="3" height="50" width="75"/>
        <mx:TextInput id="text4" text="4" height="50" width="75"/>
        <mx:TextInput id="text5" text="5" height="50" width="75"/>
    </mx:Tile>
</mx:Application>
```

Tile コンテナの子のサイズと位置の制御

Flex では、各 Tile セルのデフォルトサイズとして、最も高い子の高さと最も幅の広い子の幅が適用されます。すべてのセルのデフォルトサイズは同一です。tileHeight および tileWidth プロパティを使用して明示的にセルのサイズを設定した場合など、子のデフォルトサイズがセルよりも大きくなるときは、子のサイズはセルの境界線内に収まるように自動的に制御されます。その結果、コントロールの内容が切り取られる場合があります。たとえば、ボタン自体はセル内に収まっても、ボタンのラベルが切り取られることがあります。子のサイズを tileHeight または tileWidth プロパティ値より大きな値に明示的に指定すると、子は自動的に切り取られます。

子の幅または高さのデフォルト値がセルよりも小さい場合、デフォルトでは、セル内の子の水平方向の整列は left で、垂直方向の整列は top です。子の配置を制御するには、<mx:Tile> タグの horizontalAlign および verticalAlign プロパティを使用します。

子のサイズをパーセントベースで指定すると、子はセルの指定のパーセントに従って拡大または縮小されます。564 ページの「Tile レイアウトコンテナの作成」の例では、text2 という名前の TextInput コントロールの幅が 100 ピクセルでした。したがって、すべての Tile セルのデフォルト幅は 100 ピクセルとなり、ほとんどの子のサイズはセルサイズより小さくなります。すべての子コントロールのサイズをセル幅に合わせて拡大するには、次の例に示すように、各子の width プロパティを 100 % に設定します。この例では、Tile コントロールの tileWidth プロパティを使用してタイルの幅を指定する方法も示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\TileSizing.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Tile id="myFlow"
```

```
        direction="horizontal"
        borderStyle="solid"
        paddingTop="10" paddingBottom="10"
        paddingRight="10" paddingLeft="10"
        verticalGap="15" horizontalGap="10"
        tileWidth="100">

        <mx:TextInput id="fname" text="1" height="50" width="100%"/>
        <mx:TextInput id="lname" text="2" height="50" width="100%"/>
        <mx:TextInput id="addr1" text="3" height="50" width="100%"/>
        <mx:TextInput id="addr2" text="4" height="50" width="100%"/>
        <mx:TextInput id="addr3" text="5" height="50" width="100%"/>
    </mx:Tile>
</mx:Application>
```

メモ

子の width および height プロパティにパーセント値を設定する場合は、パーセント値は Tile コンテナ自身のサイズではなく、タイルセルのサイズに基づき設定されます。明示的に定義されたコンテナでなくても、セルは Tile コンテナ内のコンポーネントの親として機能します。

TitleWindow レイアウトコンテナ

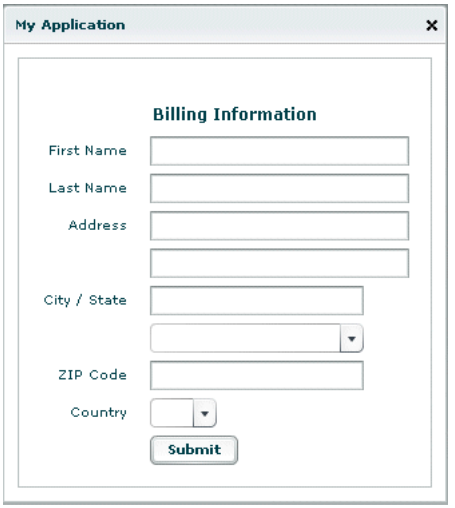
TitleWindow レイアウトコンテナは、ポップアップウィンドウとして使用するために最適化された Panel コンテナです。コンテナは、タイトルバー、タイトルバー内のキャプションとステータス領域、境界線、および子のコンテンツ領域で構成されます。Panel コンテナと異なり、TitleWindow コンテナには Close ボタンを表示できます。また、このコンテナは、ユーザーがアプリケーションウィンドウの画面内でドラッグ可能なポップアップウィンドウとして機能するように設計されています。

ポップアップ TitleWindow コンテナは、モーダルまたは非モーダルにすることができます。モーダルの場合は、ポップアップウィンドウが閉じられるまで、ポップアップウィンドウがすべてのキーボード入力とマウス入力を受け付けます。非モーダルの場合は、ポップアップウィンドウが開いた状態で、他のウィンドウも入力を受け付けることができます。

TitleWindow コンテナの典型的な用法の 1 つに、フォームの表示があります。ユーザーがフォームへの入力を終えた時点で、TitleWindow コンテナをプログラムによって閉じることができます。または、ユーザーがウィンドウ右上隅の閉じるアイコン (内部に “x” の表示されたボックス) を押して、アプリケーションに終了要求を送ることによっても閉じることができます。

Title ウィンドウはポップアップ表示させるため、他のほとんどのコントロールのように MXML 内で直接作成するのではありません。代わりに、PopUpManager を使用します。

次の例は、Form コンテナを子として持つ TitleWindow コンテナです。



TitleWindow コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	height は、デフォルトの高さまたは明示的に指定した高さを持つ、コンテンツ内のすべての子を収容し、さらにタイトルバーと境界線、垂直方向の子同士の間隔、コンテナの上下のパディングをすべて収容できる十分な大きさです。 width は、最も幅の広い子のデフォルト幅または明示的な幅にコンテナの左右のパディングを加えた値、またはタイトルテキストの幅のうち、いずれか大きい方の値です。
境界線	`left` および `right` の値は 10 ピクセルです。 `top` の値は 2 ピクセルです。 `bottom` の値は 0 ピクセルです。
パディング	`top`、`bottom`、`left`、`right` ともに 4 ピクセル。

詳細については、『Adobe Flex 2 リファレンスガイド』の TitleWindow を参照してください。

PopUpManager を使用した TitleWindow コンテナの作成

ポップアップ TitleWindow コンテナを作成および削除するには、PopUpManager の各メソッドを使用します。PopUpManager は、mx.managers package 内にあります。

ポップアップウィンドウの作成

ポップアップウィンドウを作成するには、PopUpManager の createPopUp() メソッドを使用します。createPopUp() メソッドのシグネチャを次に示します。

```
public static createPopUp(parent:DisplayObject, class:Class,
  modal:Boolean = false) : IFlexDisplayObject
```

このメソッドには、次のパラメータを指定できます。

パラメータ	内容
parent	ポップアップウィンドウの親ウィンドウへの参照です。
class	作成するオブジェクトのクラスへの参照です。通常は、TitleWindow コンテナを実装するカスタム MXML コンポーネントです。
modal	(オプション) ウィンドウをモーダル型にするかどうかを指定するブール値です。モーダルウィンドウの場合は (true)、ウィンドウが閉じられるまで、このウィンドウがすべてのマウス入力を受け付けます。モーダルでない場合は (false)、ウィンドウが表示された状態のまま、他のコントロールでも入力操作が許可されます。デフォルト値は false です。

メモ

Flex は、モーダルポップアップウィンドウを作成した後であっても、親内でコードの実行を継続します。

また、TitleWindow クラスまたはカスタムコンポーネントのインスタンスを PopUpManager' の addPopUp() メソッドに渡すことによっても、ポップアップウィンドウを作成できます。詳細については、579 ページの「addPopUp() メソッドの使用」を参照してください。

カスタム TitleWindow コンポーネントの定義

TitleWindow コンテナを作成する場合に最もよく使われる方法の 1 つは、カスタム MXML コンポーネントとして定義する方法です。

- この場合は、TitleWindow コンテナ、そのイベントハンドラ、およびカスタムコンポーネント内のすべての子を定義します。
- TitleWindow コンテナを作成および削除するには、PopUpManager の createPopUp() および removePopUp() メソッドを使用します。

次のコード例では、ポップアップウィンドウとして使用するカスタム MyLoginForm TitleWindow コンポーネントを定義しています。

```
<?xml version="1.0"?>
<!-- containers\layouts\myComponents\MyLoginForm.mxml -->
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
```

```
            import mx.managers.PopUpManager;

            private function processLogin():void {
                // Check credentials (not shown) then remove pop up.
                PopUpManager.removePopUp(this);
            }
        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="User Name">
            <mx:TextInput id="username" width="100%"/>
        </mx:FormItem>
        <mx:FormItem label="Password">
            <mx:TextInput id="password"
                displayAsPassword="true"
                width="100%"/>
        </mx:FormItem>
    </mx:Form>
    <mx:HBox>
        <mx:Button click="processLogin();" label="OK"/>
        <mx:Button
            label="Cancel"
            click="PopUpManager.removePopUp(this);"/>
    </mx:HBox>
</mx:TitleWindow>
```

この "MyLoginForm.mxml" というファイルは、`<mx:TitleWindow>` タグを使用して TitleWindow コンテナを定義します。TitleWindow コンテナは、ユーザー名およびパスワード用の 2 つの TextInput コントロールと、フォームを送信したり TitleWindow コンテナを閉じるときに使用する 2 つの Button コントロールを定義します。この例には、`submitForm()` イベントリスナー内でユーザー名とパスワードを確認するためのコードが含まれていません。

ここでは、MyLoginForm.mxml コンポーネントのイベントリスナー内でフォームデータを処理します。イベントリスナーをメインアプリケーション内で定義すると、このコンポーネントを再利用しやすくなります。こうすることで、フォームを使用するアプリケーションがデータ処理を受け持つことになり、汎用的なフォームを作成できます。メインアプリケーション内でイベントリスナーを定義する例については、572 ページの「TitleWindow および PopUpManager イベントの使用」を参照してください。

PopUpManager を使用したポップアップ TitleWindow の作成

ポップアップウィンドウを作成するには、PopUpManager の `createPopUp()` メソッドを呼び出します。このメソッドに、親、ポップアップを作成するクラスの名前、およびモーダル型かどうかを示すブール値を渡します。次のメインアプリケーションコードでは、「カスタム TitleWindow コンポーネントの定義」で定義した TitleWindow コンテナを作成します。

```
<?xml version="1.0"?>
<!-- containers\layouts\MainMyLoginForm.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.core.IFlexDisplayObject;
            import myComponents.MyLoginForm;

            private function showLogin():void {
                // Create a non-modal TitleWindow container.
                var helpWindow:IFlexDisplayObject =
                    PopUpManager.createPopUp(this, MyLoginForm, false);
            }
        ]]>
    </mx:Script>

    <mx:VBox width="300" height="300">
        <mx:Button click="showLogin();" label="Login"/>
    </mx:VBox>
</mx:Application>
```

この例では、ユーザーが [Login] ボタンを選択すると、click イベントのイベントリスナーが createPopUp() メソッドを使用して TitleWindow コンテナを作成し、"MyLoginForm.mxml" ファイルという名前をクラス名として渡します。

ポップアップ TitleWindow コンテナのプロパティを操作できるように、createPopUp() メソッドの戻り値を TitleWindow にキャストすることがよくあります。この例として、前の例の showLogin() メソッドを次のように修正します。

```
<?xml version="1.0"?>
<!-- containers\layouts\MainMyLoginFormCast.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
          import mx.managers.PopUpManager;
          import mx.core.IFlexDisplayObject;
          import myComponents.MyLoginForm;

          // Additional import statement to use the TitleWindow container.
          import mx.containers.TitleWindow;

          private function showLogin():void {
            // Create the TitleWindow container.
            var helpWindow:TitleWindow =
          TitleWindow(PopUpManager.createPopUp(this, MyLoginForm, false));

            // Add title to the title bar.
```

```
            helpWindow.title="Enter Login Information";

            // Make title bar slightly transparent.
            helpWindow.setStyle("borderAlpha", 0.9);

            // Add a close button.
            // To close the container, your must also handle the close event.
            helpWindow.showCloseButton=true;
          }
        ]]>
    </mx:Script>

    <mx:VBox width="300" height="300">
        <mx:Button click="showLogin();" label="Login"/>
    </mx:VBox>
</mx:Application>
```

ポップアップウィンドウの削除

ポップアップ TitleWindow を削除するには、PopUpManager の `removePopUp()` メソッドを使用します。`PopUpManager.createPopUp()` メソッドで作成したオブジェクトを `removePopUp()` メソッドに渡します。次の例は、569 ページの「PopUpManager を使用したポップアップ TitleWindow の作成」に示した例を修正して、ユーザーが [Done] ボタンをクリックするとポップアップウィンドウが削除されるようにしたものです。

```
<?xml version="1.0"?>
<!-- containers\layouts\MainMyLoginFormRemove.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
          import mx.managers.PopUpManager;
          import myComponents.MyLoginForm;
          import mx.core.IFlexDisplayObject;

          private var helpWindow:IFlexDisplayObject;

          private function showLogin():void {
            // Create the TitleWindow container.
            helpWindow = PopUpManager.createPopUp(this, MyLoginForm, false);
          }

          private function removeForm():void {
            PopUpManager.removePopUp(helpWindow);
          }
        ]]>
    </mx:Script>

    <mx:VBox width="300" height="300">
```

```
        <mx:Button click="showLogin();" label="Login"/>
        <mx:Button id="b2" label="Remove Form" click="removeForm();"/>
    </mx:VBox>
</mx:Application>
```

removePopUp() メソッドは、一般に TitleWindow の close イベント、および PopUpManager mouseDownOutside イベントから呼び出します。詳細については 577 ページの「イベントを使用したデータの受け渡し」を参照してください。

TitleWindow および PopUpManager イベントの使用

GUI 環境のダイアログボックスと同様に、TitleWindow のタイトルバーの右上隅に、閉じるアイコン (小さな x 記号) を追加できます。これには、次の例に示すように、showCloseButton プロパティを true に設定します。

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
  showCloseButton="true">
```

showCloseButton のデフォルト値は false です。

ユーザーが閉じるアイコンをクリックすると、TitleWindow が close イベントをブロードキャストします。このイベントのハンドラを作成し、そのイベントハンドラ内部から TitleWindow を閉じる必要があります。Flex はウィンドウを自動的に閉じません。

最も簡単な方法は、TitleWindow の close イベントプロパティを指定する際、PopUpManager の removePopUp() を直接呼び出します。この例を次に示します。

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" showCloseButton="true"
  close="PopUpManager.removePopUp(this);">
```

クリーンアップが必要な場合は、TitleWindow コントロールを閉じる前に close イベントのイベントリスナー関数を作成して、このハンドラ内から TitleWindow を閉じます。この例を次に示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\myComponents\MyLoginFormRemoveMe.mxml -->
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    showCloseButton="true"
    close="removeMe();">

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;

            private function removeMe():void {
                // Put any clean-up code here.
                PopUpManager.removePopUp(this);
            }
        ]]>
    </mx:Script>

</mx:TitleWindow>
```

また、PopUpManager の mouseDownOutside イベントを使用すると、ユーザーが TitleWindow の外部でマウスをクリックしたときにポップアップウィンドウを閉じることができます。これには、TitleWindow のインスタンスに、mouseDownOutside イベントのイベントリスナーを追加します。この例を次に示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\MainMyLoginFormRemove.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.containers.TitleWindow;
            import myComponents.MyLoginForm;
            import mx.events.FlexMouseEvent;

            private var helpWindow:TitleWindow;

            private function showLogin():void {
                // Create the TitleWindow container.
                helpWindow = TitleWindow(PopUpManager.createPopUp(this,
                    MyLoginForm, false));
                helpWindow.addEventListener("mouseDownOutside", removeForm);
            }

            private function removeForm(event:FlexMouseEvent):void {
                PopUpManager.removePopUp(helpWindow);
            }
        ]]>
    </mx:Script>

    <mx:VBox width="300" height="300">
        <mx:Button click="showLogin();" label="Login"/>
    </mx:VBox>
</mx:Application>
```

イベントリスナーはメインアプリケーション内で定義しておき、ポップアップウィンドウを作成するときに、ウィンドウに割り当てます。この方法では、MyLoginForm.mxml コンポーネントで定義された汎用的なポップアップウィンドウを使用してから、メインアプリケーションからイベントリスナーを割り当てることで、コンポーネントの動作を変更できます。

ポップアップウィンドウの中央への配置

PopUpManager の centerPopUp() メソッドを呼び出すと、別のコンテナ内でポップアップを中央に配置できます。次のカスタム MXML コンポーネントは、親コンテナ内で自身を中央に配置します。

```
<?xml version="1.0"?>
<!-- containers\layouts\myComponents\MyLoginFormCenter.mxml -->
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="handleCreationComplete();">

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;

            private function handleCreationComplete():void {
                // Center the TitleWindow container
                // over the control that created it.
                PopUpManager.centerPopUp(this);
            }

            private function processLogin():void {
                // Check credentials (not shown) then remove pop up.
                PopUpManager.removePopUp(this);
            }
        ]]>
    </mx:Script>
    <mx:Form>
        <mx:FormItem label="User Name">
            <mx:TextInput id="username" width="100%"/>
        </mx:FormItem>
        <mx:FormItem label="Password">
            <mx:TextInput id="password"
                width="100%"
                displayAsPassword="true"/>
        </mx:FormItem>
    </mx:Form>
    <mx:HBox>
        <mx:Button click="processLogin();" label="OK"/>
        <mx:Button
            label="Cancel"
            click="PopUpManager.removePopUp(this);"/>
    </mx:HBox>
</mx:TitleWindow>
```

モーダルポップアップウィンドウの作成

createPopUp() メソッドには、オプションの modal パラメータを指定できます。このパラメータを true に設定すると、ウィンドウはモーダル型になります。TitleWindow がモーダルの場合、ウィンドウが開いている間は他のコンポーネントを選択できません。modal のデフォルト値は false です。

次の例では、モーダルポップアップウィンドウを作成しています。

```
var pop1:IFlexDisplayObject = PopUpManager.createPopUp(this, MyLoginForm,
  true);
```

ポップアップウィンドウに対するデータの受け渡し

ポップアップウィンドウの柔軟性を高めるために、ポップアップコンテナに対してデータを受け渡すことができます。このためには、次のガイドラインに従います。

- ポップアップするカスタムコンポーネントを作成します。大半の場合、このコンポーネントは TitleWindow コンテナです。
- ポップアップを作成するアプリケーション内で設定する変数をポップアップ内で宣言します。
- カスタムコンポーネントと同じ型にポップアップをキャストします。
- ポップアップウィンドウがアプリケーションまたはいずれかのアプリケーションコンポーネントで値を設定する場合は、そのコンポーネントへの参照をポップアップウィンドウに渡します。

たとえば次のアプリケーションは、メインアプリケーションで定義した Array を、ポップアップウィンドウ内の ComboBox に格納します。

アプリケーションはポップアップの作成時に、ポップアップウィンドウを定義するカスタムコンポーネントの名前である ArrayEntryForm 型に、ポップアップをキャストします。この処理を行わないと、作成した各プロパティにアプリケーションがアクセスできなくなります。

このアプリケーションは、Application コンテナ内の TextInput コンポーネントへの参照をポップアップウィンドウに渡し、ポップアップが結果をコンテナに書き戻すことができるようにします。また、ポップアップ ComboBox コントロールのデータプロバイダに対してファイル拡張子の配列を渡し、ポップアップウィンドウのタイトルを設定します。これらの処理をアプリケーション内で行うことで、アプリケーション内の他の部分でも、このポップアップウィンドウを修正せずに再利用できます。アプリケーションは、データを書き戻すコンポーネントの名前、または表示するデータを認識する必要がなく、データが Array 内にあり、書き込み先が TextArea であることを認識するだけで済みます。

```
<?xml version="1.0"?>
<!-- containers\layouts\MainArrayEntryForm.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import myComponents.ArrayEntryForm;
```

```
            public var helpWindow:Object;

            public function displayForm():void {
                // Array with data for the custom control ComboBox control.
                var doctypes:Array = ["*.as", "*.mxml", "*.swc"]

                // Create the pop-up and cast the
                // return value of the createPopUp()
                // method to the ArrayEntryForm custom component.
                var pop1:ArrayEntryForm = ArrayEntryForm(
                    PopUpManager.createPopUp(this, ArrayEntryForm, true));
                // Set TitleWindow properties.
                pop1.title="Select File Type";
                pop1.showCloseButton=true;

                // Set properties of the ArrayEntryForm custom component.
                pop1.targetComponent = ti1;
                pop1.myArray = doctypes;
                PopUpManager.centerPopUp(pop1);
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:TextInput id="ti1" text=""/>
    </mx:VBox>
    <mx:Button id="b1" label="Select File Type" click="displayForm();"/>
</mx:Application>
```

次のカスタムコンポーネント ArrayEntryForm.mxml は、2 つの変数を宣言しています。第 1 の変数は、親アプリケーションがポップアップウィンドウに渡す Array のためのものです。第 2 の変数は、親アプリケーション TextInput コントロールへの参照を保持します。コンポーネントは、この参照を使用して親アプリケーションを更新します。

```
<?xml version="1.0"?>
<!-- containers\layouts\myComponents\ArrayEntryForm.mxml -->
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
        showCloseButton="true"
        width="200" borderAlpha="1"
        close="removeMe();">

    <mx:Script>
        <![CDATA[
            import mx.controls.TextInput;
            import mx.managers.PopUpManager;

            // Variables whose values are set by the main application.
            // Data provider array for the component's ComboBox control.
            [Bindable]
            public var myArray:Array;
```

```
            // A reference to the TextInput control
            // in which to put the result.
            public var targetComponent:TextInput;

            // OK button click event listener.
            // Sets the target component in the application to the
            // selected ComboBox item value.
            private function submitData():void {
                targetComponent.text = String(cb1.selectedItem);
                removeMe();
            }

            // Cancel button click event listener.
            private function removeMe():void {
                PopUpManager.removePopUp(this);
            }
        ]]>
    </mx:Script>

    <mx:ComboBox id="cb1" dataProvider="{myArray}"/>
    <mx:HBox>
        <mx:Button label="OK" click="submitData();"/>
        <mx:Button label="Cancel" click="removeMe();"/>
    </mx:HBox>
</mx:TitleWindow>
```

また、parentApplication プロパティを使用すると、ポップアップカスタムコンポーネントの内部から親アプリケーションのプロパティにアクセスできます。たとえば、アプリケーションが b1 という名前の Button コントロールを持つ場合、次の例に示すように Button コントロールのラベルを取得できます。

```
myLabel = parentApplication.b1.label;
```

ただしこの方法では、親のターゲットコンポーネント ID およびコンポーネントのプロパティに対し、ポップアップコンポーネント内でハードコード値を使用します。

イベントを使用したデータの受け渡し

次の例は、前セクションの例を一部変更したものです。メインアプリケーション内で定義されたイベントリスナーを使用して、ポップアップウィンドウから返されたデータを、メインアプリケーションに受け渡す処理を行います。この例では、イベントリスナーが定義されていない "ArrayEntryFormEvents.mxml" ファイルを示します。

```
<?xml version="1.0"?>
<!-- containers\layouts\myComponents\ArrayEntryFormEvents.mxml -->
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    showCloseButton="true"
    width="200"
    borderAlpha="1">
```

```
    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;

            // Variables whose values are set by the main application.
            // Data provider array for the component's ComboBox control.
            [Bindable]
            public var myArray:Array;

        ]]>
    </mx:Script>

    <mx:ComboBox id="cb1" dataProvider="{myArray}"/>
    <mx:HBox>
        <mx:Button id="okButton" label="OK"/>
        <mx:Button id="cancelButton" label="Cancel"/>
    </mx:HBox>
</mx:TitleWindow>
```

メインアプリケーションは、各イベントリスナーを定義し、ポップアップウィンドウ内で定義されたコントロールにこれらを登録します。

```
<?xml version="1.0"?>
<!-- containers\layouts\MainArrayEntryFormEvents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import flash.events.Event;
            import myComponents.ArrayEntryFormEvents;

            public var pop1:ArrayEntryFormEvents;

            public function displayForm():void {
                // Array with data for the custom control ComboBox control.
                var doctypes:Array = ["*.as", "*.mxml", "*.swc"]

                // Create the pop-up and cast the return value
                // of the createPopUp()
                // method to the ArrayEntryFormEvents custom component.
                pop1 = ArrayEntryFormEvents(
              PopUpManager.createPopUp(this, ArrayEntryFormEvents, true));
                // Set TitleWindow properties.
                pop1.title="Select File Type";
                pop1.showCloseButton=true;

                // Set the event listeners for
                // the ArrayEntryFormEvents component.
                pop1.addEventListener("close", removeMe);
                pop1["cancelButton"].addEventListener("click", removeMe);
```

```
                pop1["okButton"].addEventListener("click", submitData);

                // Set properties of the ArrayEntryFormEvents custom control.
                pop1.myArray = doctypes;
                PopUpManager.centerPopUp(pop1);
            }

            // OK button click event listener.
            // Sets the target component in the application to the
            // selected ComboBox item value.
            private function submitData(event:Event):void {
                ti1.text = String(pop1.cb1.selectedItem);
                removeMe(event);
            }

            // Cancel button click event listener.
            private function removeMe(event:Event):void {
                PopUpManager.removePopUp(pop1);
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:TextInput id="ti1" text=""/>
    </mx:VBox>
    <mx:Button id="b1" label="Select File Type" click="displayForm();"/>
</mx:Application>
```

addPopUp() メソッドの使用

PopUpManager の addPopUp() メソッドを使用すると、カスタムコンポーネントを定義せずにポップアップを作成できます。このメソッドは、IFlexDisplayObject を実装する任意のクラスのインスタンスを受け取ります。このメソッドはクラス自体ではなく、クラスのインスタンスを受け取ります。したがって、<mx:Script> ブロック内で ActionScript コードを使用して、個別のカスタムコンポーネントとしてではなく、コンポーネントインスタンスを作成してポップアップ表示できます。

別の場所で再利用しない単純なダイアログボックスをポップアップ表示する場合は、createPopUp() メソッドではなく addPopUp() メソッドを使用するほうが望ましい場合があります。しかし、ポップアップが複雑な場合や、別の場所で再利用できない場合、これは最適なコーディング方法ではありません。

次の例では、addPopUp() メソッドによりポップアップを作成し、クリック時にウィンドウを閉じる Button コントロールをウィンドウに追加します。

```
<?xml version="1.0"?>
<!-- containers\layouts\MyPopUpButton.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="600" width="600" >
```

```
    <mx:Script>
        <![CDATA[
            import mx.containers.TitleWindow;
            import flash.events.*;
            import mx.managers.PopUpManager;
            import mx.controls.Button;
            import mx.core.IFlexDisplayObject;

            // The variable for the TitleWindow container
            public var myTitleWindow:TitleWindow = new TitleWindow();

            // Method to instantiate and display a TitleWindow container.
            // This is the initial Button control's click event handler.
            public function openWindow(event:MouseEvent):void {
                // Set the TitleWindow container properties.
                myTitleWindow = new TitleWindow();
                myTitleWindow.title = "My Window Title";
                myTitleWindow.width= 220;
                myTitleWindow.height= 150;
                // Call the method to add the Button control to the
                // TitleWindow container.
                populateWindow();
                // Use the PopUpManager to display the TitleWindow container.
                PopUpManager.addPopUp(myTitleWindow, this, true);
            }

            // The method to create and add the Button child control to the
            // TitleWindow container.
            public function populateWindow():void {
                var btn1:Button = new Button();
                btn1.label="close";
                btn1.addEventListener(MouseEvent.CLICK, closeTitleWindow);
                myTitleWindow.addChild(btn1);
            }

            // The method to close the TitleWindow container.
            public function closeTitleWindow(event:MouseEvent):void {
                PopUpManager.removePopUp(event.currentTarget.parent);
            }
        ]]>
    </mx:Script>
    <mx:Button label="Open Window" click="openWindow(event);"/>
</mx:Application>
```

すべてのコンポーネントは、ポップアップさせることが可能です。次に、TextArea コントロールをポップアップする例を示します。この例では、まずコントロールのサイズを変更します。次に、TextArea を閉じるタイミングを判断するために、Shift キーを押しながらのマウスクリックをリッスンします。TextArea に入力したテキストはすべて、ポップアップの削除時にアプリケーションの Label コントロールに保存されます。

```
<?xml version="1.0"?>
<!-- containers\layouts\MyPopUpTextArea.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.controls.TextArea;
            import mx.core.IFlexDisplayObject;

            public var myPopUp:TextArea

            public function openWindow(event:MouseEvent):void {
                myPopUp = new TextArea();
                myPopUp.width= 220;
                myPopUp.height= 150;
                myPopUp.text = "Hold down the Shift key, and " +
                    "click in the TextArea to close it.";
                myPopUp.addEventListener(MouseEvent.CLICK, closeWindow);
                PopUpManager.addPopUp(myPopUp, this, true);
                PopUpManager.centerPopUp(myPopUp);
            }

            public function closeWindow(event:MouseEvent):void {
                if (event.shiftKey) {
                    label1.text = myPopUp.text;
        PopUpManager.removePopUp(IFlexDisplayObject(event.currentTarget));
                }
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Button id="b1" label="Create TextArea Popup"
            click="openWindow(event);"/>
        <mx:Label id="label1"/>
    </mx:VBox>
</mx:Application>
```

第 16 章

16 ナビゲータコンテナの使用

ナビゲータコンテナは、それ自体が別のコンテナである複数の子の間でのユーザーの移動 (ナビゲーション) を制御します。ナビゲータコンテナの子コンテナの子のレイアウトと位置の制御は個々の子コンテナで行われ、ナビゲータコンテナでは制御されません。

このトピックでは、ナビゲータコンテナとそのシンタックスについて説明し、ナビゲータコンテナの使用例を示します。

目次

ナビゲータコンテナについて

ナビゲータコンテナは、子コンテナのグループ内でのユーザーの移動を制御します。たとえば、TabNavigator コンテナでは、一連のタブを使用して、表示する子コンテナを選択できます。

メモ

ナビゲータコンテナの直下の子は、レイアウトコンテナまたはナビゲータコンテナである必要があります。ナビゲータ内にコントロールを直接ネストすることはできません。コントロールは、ナビゲータコンテナの子コンテナの子として作成してください。

Flex には、次のナビゲータコンテナが用意されています。

- ViewStack
- TabNavigator
- Accordion

以降のセクションでは、Flex の各ナビゲータコンテナの使用方法について説明します。

ViewStack ナビゲータコンテナ

ViewStack ナビゲータコンテナは、相互に積み重ねられた子コンテナのコレクションによって構成され、一度に1つのコンテナだけが表示されます (これを " アクティブになる " と言います)。ViewStack コンテナには、現在アクティブなコンテナをユーザーが切り替えるメカニズムは組み込まれていません。ユーザーが現在アクティブな子を変更できるようにするためには、LinkBar、TabBar、ButtonBar、ToggleButtonBar のいずれかのコントロールを使用するか、または ActionScript を使用して独自のロジックを作成する必要があります。たとえば、子コンテナ間の切り替えを制御する Button コントロールを定義できます。

次の例は、ViewStack コンテナ内の積み重ねられた子コンテナを示します。

子コンテナ 0 がアクティブ

子コンテナ 1 がアクティブ

左の図は、1つ目の子がアクティブな状態の ViewStack コンテナを表しています。ViewStack コンテナの子のインデックスは、0 ～ n - 1 の範囲の整数です (n は子コンテナの総数)。右の図は、2 つ目の子コンテナがアクティブな状態の ViewStack コンテナを表しています。

ViewStack コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	最初にアクティブになる子の幅と高さ。
コンテナのサイズ変更規則	ViewStack コンテナのサイズは、デフォルトでは、最初の子コンテナのサイズに合わせて一度だけ設定されます。デフォルトでは、他のコンテナに切り替えても、サイズは変更されません。別の子コンテナに切り替えたときに ViewStack コンテナのサイズを変更するには、`resizeToContent` プロパティを `true` に設定します。
子のサイズ設定規則	子のサイズはデフォルトサイズに設定されます。ただし、子のサイズが ViewStack コンテナより大きい場合は、一部がクリッピングされます。子コンテナが ViewStack コンテナより小さい場合は、ViewStack コンテナの左上隅に位置合わせされます。
デフォルトパディング	`top`、`bottom`、`left`、`right` ともに 0 ピクセル。

詳細については、『Adobe Flex 2 リファレンスガイド』の ViewStack を参照してください。

ViewStack コンテナの作成

ViewStack コンテナは <mx:ViewStack> タグを使用して定義し、タグ本体で子コンテナを定義します。アクティブにする子コンテナを制御するには、ViewStack コンテナの次のプロパティを使用します。

- selectedIndex　子コンテナが定義されている場合、現在アクティブなコンテナのインデックスを表します。子コンテナが定義されていない場合、このプロパティの値は -1 になります。インデックスは 0 ～ numChildren - 1 の範囲の整数です (numChildren は ViewStack コンテナ内の子コンテナの総数)。このプロパティをアクティブにするコンテナのインデックスに設定します。

  <mx:ViewStack> タグの selectedIndex プロパティを使用して、アプリケーションの起動時にデフォルトでアクティブにするコンテナを設定できます。次の例では、デフォルトでアクティブにするコンテナのインデックスを 1 に設定しています。

  ```
  <mx:ViewStack id="myViewStack" selectedIndex="1">
  ```

  次の例では、スタック内の 2 番目の子コンテナをアクティブにするため、ActionScript を使用して selectedIndex プロパティを設定しています。

  ```
  myViewStack.selectedIndex=1;
  ```

- selectedChild　子コンテナが定義されている場合、現在アクティブなコンテナを表します。子コンテナが定義されていない場合、このプロパティの値は null になります。このプロパティは ActionScript 内で、アクティブにするコンテナの識別子に設定します。

  このプロパティは ActionScript ステートメントでのみ設定できます。MXML では設定できません。

  次の例は、ActionScript を使用して selectedChild プロパティを設定し、search という識別子を持つ子コンテナをアクティブにします。

  ```
  myViewStack.selectedChild=search;
  ```

- numChildren　ViewStack コンテナ内の子コンテナの数を表します。

  次の例は、ActionScript ステートメントで numChildren プロパティを使用して、スタック内の最後の子コンテナをアクティブに設定します。

  ```
  myViewStack.selectedIndex=myViewStack.numChildren-1;
  ```

メモ

Application コンテナを除くすべてのコンテナでは、親コンテナの作成ポリシーがデフォルトのポリシーになります。Application コンテナのデフォルトポリシーは auto です。したがって、ほとんどの場合、View Stack コントロールの子は選択されるまで作成されません。selectedChild プロパティをまだ作成されていない子に設定することはできません。

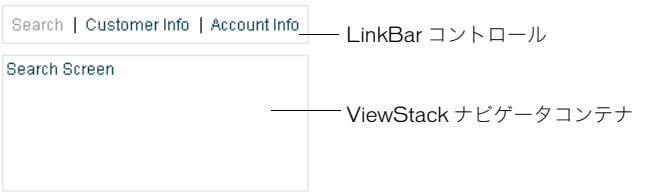
次の例では、3 つの子コンテナを持つ ViewStack コンテナを作成します。また、子コンテナを制御する 3 つの Button コントロールを定義し、ボタンをクリックしてアクティブにする子コンテナを選択します。

```
<?xml version="1.0"?>
<!-- containers\navigators\VSSimple.mxml  -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <!-- Create a VBox container so the container for
        the buttons appears above the ViewStack container. -->
    <mx:VBox>
        <!-- Create an HBox container to hold the three buttons. -->
        <mx:HBox borderStyle="solid">

            <!-- Define the three buttons.
                Each uses the child container identifier
                to set the active child container. -->
            <mx:Button id="searchButton"
                label="Search Screen"
                click="myViewStack.selectedChild=search;"/>

            <mx:Button id="cInfoButton"
                label="Customer Info Screen"
                click="myViewStack.selectedChild=custInfo;"/>

            <mx:Button id="aInfoButton"
                label="Account Info Screen"
                click="myViewStack.selectedChild=accountInfo;"/>
        </mx:HBox>

        <!-- Define the ViewStack and the three child containers and have it
            resize up to the size of the container for the buttons. -->
        <mx:ViewStack id="myViewStack"
            borderStyle="solid" width="100%">

            <mx:Canvas id="search" label="Search">
                <mx:Label text="Search Screen"/>
            </mx:Canvas>

            <mx:Canvas id="custInfo" label="Customer Info">
                <mx:Label text="Customer Info"/>
            </mx:Canvas>

            <mx:Canvas id="accountInfo" label="Account Info">
                <mx:Label text="Account Info"/>
            </mx:Canvas>
        </mx:ViewStack>
    </mx:VBox>
</mx:Application>
```

この例をロードすると、3 つの Button コントロールが表示され、ViewStack コンテナで最初に定義されている子コンテナがアクティブになります。Button コントロールをクリックすると、アクティブコンテナが変わります。

LinkBar、TabBar、ButtonBar、ToggleButtonBar のいずれかのコントロールを使用して、ViewStack コンテナのアクティブな子を設定することもできます。これらのクラスは NavBar クラスのサブクラスなので、総称して " ナビゲータバーコントロール " と呼びます。ナビゲータバーコントロールは ViewStack コンテナに含まれる子コンテナの数を調べ、次の図のように水平または垂直に並んだリンク、タブ、またはボタンを作成します。ユーザーはこれらのエレメントを使用して、アクティブにする子コンテナを選択できます。



LinkBar コントロール内のアイテムは、次の例のように、ViewStack コンテナの各子コンテナの label プロパティの値に対応します。

```
<?xml version="1.0"?>
<!-- containers\navigators\VSLinkBar.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox>
        <!-- Create a LinkBar control to navigate
            the ViewStack container. -->
        <mx:LinkBar dataProvider="{myViewStack}" borderStyle="solid"/>

        <!-- Define the ViewStack and the three child containers. -->
        <mx:ViewStack id="myViewStack"
            borderStyle="solid"
            width="100%">

            <mx:Canvas id="search" label="Search">
                <mx:Label text="Search Screen"/>
            </mx:Canvas>

            <mx:Canvas id="custInfo" label="Customer Info">
                <mx:Label text="Customer Info"/>
            </mx:Canvas>

            <mx:Canvas id="accountInfo" label="Account Info">
                <mx:Label text="Account Info"/>
            </mx:Canvas>
        </mx:ViewStack>
    </mx:VBox>
</mx:Application>
```

ナビゲータバーコントロールで設定するプロパティは dataProvider だけです。このプロパティでは、ナビゲータバーコントロールに関連付ける ViewStack の名前を指定します。LinkBar コントロールの詳細については、262 ページの「LinkBar コントロール」を参照してください。TabBar コントロールの詳細については、264 ページの「TabBar コントロール」を参照してください。ButtonBar コントロールと ToggleButtonBar コントロールの詳細については、259 ページの「ButtonBar コントロールと ToggleButtonBar コントロール」を参照してください。

ViewStack コンテナの子のサイズ設定

ViewStack コンテナのデフォルトの幅と高さは、最初の子の幅と高さになります。ViewStack コンテナのサイズは、アクティブな子が変更されても変わりません。

次のテクニックを使用すると、ViewStack コンテナのサイズを制御して、ViewStack コンテナの子の中にすべてのコンポーネントを表示できます。

- すべての子の width プロパティと height プロパティを明示的に同じ固定値に設定します。
- すべての子のパーセント値ベースの width プロパティと height プロパティを明示的に同じ固定値に設定します。
- ViewStack コンテナの width プロパティと height プロパティを固定値またはパーセント値ベースの値に設定します。

どのテクニックを使用するかは、作成するアプリケーションと ViewStack コンテナの内容に応じて決定してください。

ViewStack コンテナへのビヘイビアの適用

ViewStack コンテナまたは ViewStack コンテナの子に、エフェクトを割り当てることができます。たとえば、ViewStack コントロールの creationCompleteEffect プロパティに WipeRight エフェクトを割り当てると、ViewStack が初めて表示されたときに WipeRight エフェクトが一度だけ再生されます。

ViewStack の子が変更されたときに実行するエフェクトを指定するには、子の hideEffect プロパティと showEffect プロパティを使用します。hideEffect プロパティで指定したエフェクトはコンテナが非表示になると再生され、showEffect プロパティで指定したエフェクトは新しい子が表示されると再生されます。ViewStack コンテナは、非表示になるコンテナの hideEffect プロパティに指定されたエフェクトが完了するまで待機し、その後で新しい子コンテナを表示します。

次の例では、ViewStack コンテナが初めて作成されたときに WipeRight エフェクトを実行し、各々の子が非表示になるときに WipeDown エフェクトを、新しい子が表示されるときに WipeUp エフェクトをそれぞれ実行します。

```
<?xml version="1.0"?>
<!-- containers\navigators\VSLinkEffects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:WipeUp id="myWU" duration="300"/>
    <mx:WipeDown id="myWD" duration="300"/>
    <mx:WipeRight id="myWR" duration="300"/>

    <mx:VBox>
        <mx:LinkBar dataProvider="{myViewStack}"
            borderStyle="solid"
            backgroundColor="#EEEEFF"/>

        <mx:ViewStack id="myViewStack"
            borderStyle="solid"
            width="100%"
            creationCompleteEffect="{myWR}">

            <mx:Canvas id="search"
                label="Search"
                hideEffect="{myWD}"
                showEffect="{myWU}">
                <mx:Label text="Search Screen"/>
            </mx:Canvas>

            <mx:Canvas id="custInfo"
                label="Customer Info"
                hideEffect="{myWD}"
                showEffect="{myWU}">
                <mx:Label text="Customer Info"/>
            </mx:Canvas>

            <mx:Canvas id="accountInfo"
                label="Account Info"
                hideEffect="{myWD}"
                showEffect="{myWU}">
                <mx:Label text="Account Info"/>
            </mx:Canvas>
        </mx:ViewStack>
    </mx:VBox>
</mx:Application>
```

コンテナの子の showEffect プロパティは、コンテナの可視性が false から true に変わったときだけトリガされます。したがって、コンポーネントが初めて作成されたときにエフェクトをトリガする場合は、creationCompleteEffect プロパティを使用します。

TabNavigator コンテナ

TabNavigator コンテナは一連のタブを作成し、管理します。ユーザーはそれらのタブを使用して、TabNavigator コンテナの子から子へ移動できます。TabNavigator コンテナの子は別のコンテナです。TabNavigator コンテナは、子コンテナごとにタブを 1 つ作成します。ユーザーがタブを選択すると、次の図のように、関連付けられた子が TabNavigator コンテナに表示されます。

TabNavigator コンテナは ViewStack コンテナの子クラスで、ViewStack コンテナの機能の多くを継承します。TabNavigator コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	最初にアクティブになる子のデフォルトまたは明示的な幅と高さに、タブのデフォルトまたは明示的な高さと幅を加えたサイズ。デフォルトのタブの高さは、TabNavigator コンテナに適用されるフォント、スタイル、およびスキンによって決まります。
コンテナのサイズ変更規則	TabNavigator コンテナのサイズは、デフォルトでは、最初の子コンテナのサイズに合わせて一度だけ設定されます。デフォルトでは、他のコンテナに切り替えても、サイズは変更されません。別の子コンテナに切り替えたときに TabNavigator コンテナのサイズを変更するには、`resizeToContent` プロパティを `true` に設定します。
子のレイアウト規則	子のサイズが TabNavigator コンテナより大きい場合は、一部がクリッピングされます。子コンテナが TabNavigator コンテナより小さい場合は、TabNavigator コンテナの左上隅に位置合わせされます。
デフォルトパディング	`top`、`bottom`、`left`、`right` ともに 0 ピクセル。

詳細については、『Adobe Flex 2 リファレンスガイド』の TabNavigator を参照してください。

TabNavigator コンテナの作成

TabNavigator コンテナは、<mx:TabNavigator> タグを使用して定義します。一度に表示される TabNavigator コンテナの子は 1 つだけです。ユーザーは、タブの選択またはキーボード操作によって子を選択できます。ユーザーが現在アクティブな子を変更するたびに、TabNavigator コンテナから change イベントがブロードキャストされます。

TabNavigator コンテナは、その子に対応するタブを自動的に作成し、子の label プロパティからタブテキストを、子の icon プロパティからタブアイコンを、それぞれ決定します。タブは、子のインデックスの番号順に左から右に並べられます。TabNavigator コンテナの幅にすべてのタブが収まる場合には、タブはすべて表示されます。

TabNavigator コンテナの子の enabled プロパティを false に設定して無効にすると、関連するタブも無効になります。

次のコードは、590 ページの「TabNavigator コンテナ」の図にある TabNavigator コンテナを作成します。

```
<?xml version="1.0"?>
<!-- containers\navigators\TNSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:TabNavigator borderStyle="solid" >

        <mx:VBox label="Accounts"
            width="300"
            height="150">
            <!-- Accounts view goes here. -->
        </mx:VBox>

        <mx:VBox label="Stocks"
            width="300"
            height="150">
            <!-- Stocks view goes here. -->
        </mx:VBox>

        <mx:VBox label="Futures"
            width="300"
            height="150">
            <!-- Futures view goes here. -->
        </mx:VBox>

    </mx:TabNavigator>
</mx:Application>
```

次のように、ViewStack コンテナから継承される selectedChild プロパティと selectedIndex プロパティを使用して、現在アクティブな子を設定することもできます。

- selectedIndex　子コンテナが定義されている場合、現在アクティブなコンテナのインデックスを表します。インデックスは 0 ～ numChildren - 1 の範囲の整数です (numChildren は TabNavigator コンテナ内の子コンテナの総数)。このプロパティをアクティブにするコンテナのインデックスに設定します。
- selectedChild　子コンテナが定義されている場合、現在アクティブなコンテナを表します。子コンテナが定義されていない場合、このプロパティの値は null になります。このプロパティは、アクティブにするコンテナの識別子に設定します。このプロパティは ActionScript ステートメントでのみ設定できます。MXML では設定できません。

selectedChild プロパティと selectedIndex プロパティの詳細と例については、584 ページの「ViewStack ナビゲータコンテナ」を参照してください。

ユーザーが現在アクティブな子を変更したときに再生するエフェクトを指定するには、TabNavigator コントロールの子の showEffect プロパティと hideEffect プロパティを使用します。次の例は、選択されているタブが変更されるたびに WipeLeft エフェクトを再生します。

```
<?xml version="1.0"?>
<!-- containers\navigators\TNEffect.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:WipeLeft id="myWL"/>

    <mx:TabNavigator>

        <mx:VBox label="Accounts"
            width="300"
            height="150"
            showEffect="{myWL}">
            <!-- Accounts view goes here. -->
        </mx:VBox>

        <mx:VBox label="Stocks"
            width="300"
            height="150"
            showEffect="{myWL}">
            <!-- Stocks view goes here. -->
        </mx:VBox>

        <mx:VBox label="Futures"
            width="300"
            height="150"
            showEffect="{myWL}">
            <!-- Futures view goes here. -->
        </mx:VBox>

    </mx:TabNavigator>
</mx:Application>
```

TabNavigator コンテナの子のサイズ設定

TabNavigator コンテナのデフォルトの幅と高さは、最初の子の幅と高さになります。TabNavigator コンテナのサイズは、アクティブな子が変更されても変わりません。

次のテクニックを使用すると、TabNavigator コンテナのサイズを制御して、TabNavigator コンテナの子の中にすべてのコンポーネントを表示できます。

- すべての子の `width` プロパティと `height` プロパティを明示的に同じ固定値に設定します。
- すべての子のパーセント値ベースの `width` プロパティと `height` プロパティを明示的に同じ固定値に設定します。
- TabNavigator コンテナの `width` プロパティと `height` プロパティを明示的またはパーセント値ベースで設定します。

どの方法を使用するかは、作成するアプリケーションと TabNavigator コンテナの内容に応じて決定してください。

TabNavigator コンテナのキーボード操作

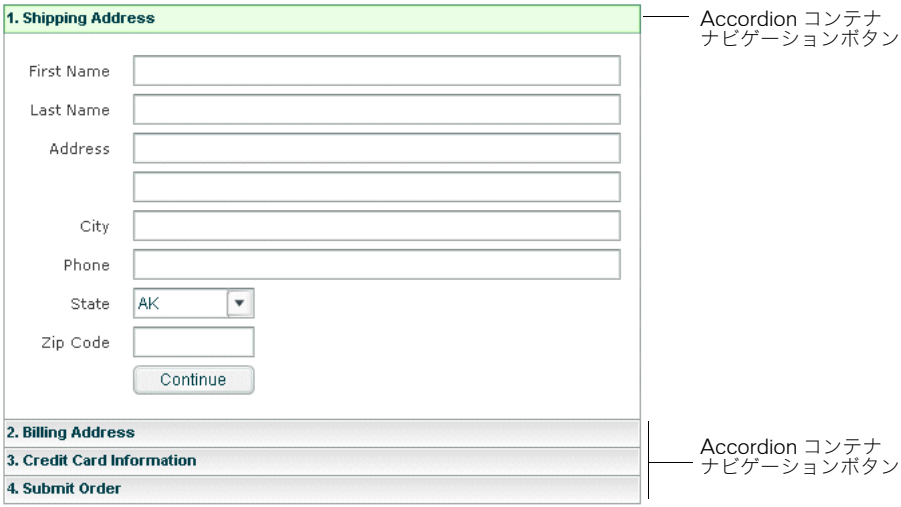
TabNavigator コンテナにフォーカスがあるときは、次の表のようにキーストロークが処理されます。

キー	アクション
↓ (下矢印) → (右矢印)	次のタブにフォーカスを移動します。最後のタブにフォーカスがあるときにこのキーを押すと、最初のタブにフォーカスが移動します。ただし、選択されている子は変化しません。
↑ (上矢印) ← (左矢印)	前のタブにフォーカスを移動します。最初のタブにフォーカスがあるときにこのキーを押すと、最後のタブにフォーカスが移動します。ただし、選択されている子は変化しません。
PageDown	次の子を選択します。最後の子を選択しているときにこのキーを押すと、最初の子が選択されます。
PageUp	前の子を選択します。最初の子を選択しているときにこのキーを押すと、最後の子が選択されます。
Home	最初の子を選択します。
End	最後の子を選択します。
Enter スペースバー	フォーカスが表示されているタブに関連付けられている子を選択します。

Accordion ナビゲータコンテナ

フォームは多くのアプリケーションの基本的なコンポーネントです。しかし、複雑なフォームの操作や複数ページにわたるフォームの移動は、ユーザーにわかりやすいとは言えません。また、フォームが大きすぎて画面内に収まらない場合もあります。

Flex では Accordion ナビゲータコンテナを使用して、フォームの外観と移動の操作性を向上させることができます。Accordion コンテナでは一連の子パネルを定義しますが、一度に表示されるパネルは 1 つだけです。次の図は、Accordion コンテナの例です。



コンテナを移動する場合、ユーザーはアクセスする子パネルに対応するナビゲーションボタンをクリックします。Accordion コンテナを使用すると、ユーザーはどのような順序でもフォーム内を移動できます。たとえば、ユーザーが Credit Card Information パネルを使用しているときに、Billing Address パネルの情報を変更したいと考えたとします。そのためには、Billing Address パネルに移動して情報を編集してから、Credit Card Information パネルに戻らなければなりません。

HTML の場合、配送先住所、請求先住所、およびクレジットカード情報を入力するフォームは 3 ページで作成されることが多く、ユーザーは次のページに移動する前に各ページをサーバーに送信する必要があります。Accordion コンテナを使用すると、3 つの子パネルの情報をまとめて、1 つの送信ボタンで送信できます。このアーキテクチャにより、サーバーのトラフィックが最小限に抑えられるとともに、ユーザーが自分が今どこにいてどのような操作をしているのかを把握しやすくなります。

メモ 子パネルのない空の Accordion コンテナはフォーカスを取得できません。

Accordion コンテナは特にフォームや Form コンテナの操作に便利ですが、Flex のどのコンポーネントも Accordion コンテナの子パネル内で使用できます。たとえば、Accordion コンテナを使用して製品のカタログを作成し、各パネルで類似製品のグループを紹介する使い方が考えられます。

Accordion コンテナには、次のデフォルトサイズ設定属性があります。

プロパティ	デフォルト値
デフォルトサイズ	現在アクティブな子の幅と高さ。
コンテナのサイズ変更規則	Accordion コンテナのサイズは、デフォルトでは、最初の子コンテナのサイズに合わせて一度だけ設定されます。デフォルトでは、他のコンテナに切り替えても、サイズは変更されません。別の子コンテナに切り替えたときに Accordion コンテナのサイズを変更するには、`resizeToContent` プロパティを `true` に設定します。
子のサイズ設定規則	子のサイズはデフォルトサイズに設定されます。子のサイズが Accordion コンテナより大きい場合は、一部がクリッピングされます。子コンテナが Accordion コンテナより小さい場合は、Accordion コンテナの左上隅に位置合わせされます。
デフォルトパディング	`top`、`bottom`、`left`、`right` ともに -1 ピクセル。

詳細については、『Adobe Flex 2 リファレンスガイド』の Accordion を参照してください。

Accordion コンテナの作成

Accordion コンテナは、<mx:Accordion> タグを使用して定義します。Accordion コンテナでは、子パネル1つに対して1つのコンテナを定義します。たとえば、Accordion コンテナに子パネルが 4 つあり、それぞれがフォームの 4 つの部分に対応する場合は、次の例のように、Form コンテナを使用して各子パネルを定義します。

```
<?xml version="1.0"?>
<!-- containers\navigators\AccordionSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Accordion id="accordion1" height="450">

        <mx:Form id="shippingAddress" label="1. Shipping Address">

            <mx:FormItem id="sfirstNameItem" label="First Name">
                <mx:TextInput id="sfirstName"/>
            </mx:FormItem>

            <!-- Additional contents goes here. -->

        </mx:Form>
```

```
        <mx:Form id="billingAddress" label="2. Billing Address">
            <!-- Form contents goes here. -->
        </mx:Form>

        <mx:Form id="creditCardInfo" label="3. Credit Card Information">
            <!-- Form contents goes here. -->
        </mx:Form>

        <mx:Form id="submitOrder"   label="4. Submit Order">
            <!-- Form contents goes here. -->
        </mx:Form>

    </mx:Accordion>
</mx:Application>
```

この例では、Form コンテナを使用して各子パネルを定義します。子パネルの定義には、他のコンテナも使用できます。

メモ

子パネルの定義には任意のコンテナを使用できます。ただし、子コンポーネントを持たない TabNavigator などのコンテナや、Accordion コンテナなど、子パネルとして適していないコンテナもあります。

Accordion コンテナのキーボード操作

Accordion コンテナにフォーカスがあるときは、次の表のようにキーストロークが処理されます。

キー	アクション
↓ (下矢印) → (右矢印)	次のボタンにフォーカスを移動します。最後のボタンにフォーカスがあるときにこのキーを押すと、最初のボタンにフォーカスが移動します。ただし、選択されている子は変化しません。
↑ (上矢印) ← (左矢印)	前のボタンにフォーカスを移動します。最初のボタンにフォーカスがあるときにこのキーを押すと、最後のボタンにフォーカスが移動します。ただし、選択されている子は変化しません。
PageUp	前の子パネルに移動します。最初の子パネルを選択しているときにこのキーを押すと、最後の子パネルが選択されます。
PageDown	次の子パネルに移動します。最後の子パネルを選択しているときにこのキーを押すと、最初の子パネルが選択されます。
Home	最初の子パネルに移動します。
End	最後の子パネルに移動します。
Enter スペースバー	フォーカスが表示されているタブに関連付けられている子を選択します。

Button コントロールを使用した Accordion コンテナの移動

ユーザーが Accordion コンテナのパネルを移動する最も簡単な方法は、移動先のパネルのナビゲーションボタンをクリックすることです。ただし、Back や Next などの移動用 Button コントロールを Accordion コンテナに追加して、ユーザーが移動しやすいようにすることもできます。

移動用の Button コントロールでは、Accordion コンテナの次のプロパティを使用して子パネル間を移動します。

プロパティ	説明
`numChildren`	Accordion コンテナ内に定義されている子パネルの総数を表します。
`selectedIndex`	現在アクティブな子パネルのインデックスを表します。子パネルには 0 ～ `numChildren - 1` までの番号が付けられます。`selectedIndex` プロパティを変更すると、現在アクティブなパネルが変更されます。
`selectedChild`	子コンテナが定義されている場合、現在アクティブなコンテナを表します。子コンテナが定義されていない場合、このプロパティの値は `null` になります。このプロパティは、アクティブにするコンテナの識別子に設定します。このプロパティは ActionScript ステートメントでのみ設定できます。MXML では設定できません。

これらのプロパティの詳細については、584 ページの「ViewStack ナビゲータコンテナ」を参照してください。

たとえば、Accordion コンテナの 2 つ目のパネル (パネル番号 1) で次の 2 つの Button コントロールを使用して、パネル番号 0 に戻ったり、パネル番号 2 に進んだりできます。

```
<?xml version="1.0"?>
<!-- containers\navigators\AccordionButtonNav.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Accordion id="accordion1" height="450">

        <mx:Form id="shippingAddress" label="1. Shipping Address">

            <mx:FormItem id="sfirstNameItem" label="First Name">
                <mx:TextInput id="sfirstName"/>
            </mx:FormItem>

        </mx:Form>

        <mx:Form id="billingAddress" label="2. Billing Address">
            <mx:Button id="backButton"
                label="Back"
                click="accordion1.selectedIndex=0;"/>
            <mx:Button id="nextButton"
```

```
                label="Next"
                click="accordion1.selectedIndex=2;"/>
        </mx:Form>

        <mx:Form id="creditCardInfo" label="3. Credit Card Information">
        </mx:Form>

    </mx:Accordion>
</mx:Application>
```

ナビゲーションボタンでは、相対位置も使用できます。次の Button コントロールを使用すると、Accordion コンテナの現在のパネル番号を基準として、前のパネルや次のパネルに移動できます。

```
<mx:Button id="backButton" label="Back" click="accordion1.selectedIndex =
  accordion1.selectedIndex - 1;"/>

<mx:Button id="nextButton" label="Next" click="accordion1.selectedIndex =
  accordion1.selectedIndex + 1;"/>
```

[Next] Button コントロールでは、次のように selectedChild プロパティを使用し、パネルのコンテナの id プロパティの値に基づいて次のパネルに移動できます。

```
<mx:Button id="nextButton" label="Next"
  click="accordion1.selectedChild=creditCardInfo;"/>
```

次の Button コントロールは Accordion コンテナの最後のパネルを開きます。

```
<mx:Button id="lastButton" label="Last"  click="accordion1.selectedIndex =
  accordion1.numChildren - 1;"/>
```

子のボタンイベントの処理

Accordion コンテナは、ユーザーがパネルを変更したときにイベントを認識します。ボタンのクリックや PageDown などのキー入力によってユーザーが子パネルを変更すると、Accordion コンテナから change イベントがブロードキャストされます。

メモ

change イベントは、子パネルがプログラムによって変更された場合には送出されません。たとえば、「Button コントロールを使用した Accordion コンテナの移動」で示したようにボタンを使用してパネルを変更するときは、change イベントは送出されません。ただし、valueCommit イベントは送出されます。

change イベントのイベントハンドラは、<mx:Accordion> タグの change プロパティを使用して登録します。ActionScript でハンドラを登録することもできます。次の例では、ユーザーがパネルを変更するたびに change イベントを "flashlog.txt" にログ出力しています。

```
<mx:Accordion id="accordion1" height="450" change="trace('change');">
```

アコーディオンボタンの外観の制御

Accordion コンテナのボタンは AccordionHeader クラスによってレンダリングされます。このクラスは Button のサブクラスで、Button クラスと同じスタイルプロパティがあります。

Accordion ボタンのスタイルを変更するには、Accordion クラスの `getHeaderAt()` メソッドを呼び出して子コンテナのボタンへの参照を取得してから、ボタンの `setStyle()` メソッドを呼び出してスタイルを設定します。次の例ではこの手法を使用して、それぞれの Accordion ボタンのテキストを異なる色に設定しています。

```
<?xml version="1.0"?>
<!-- containers\navigators\AccordionStyling.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="600"
    height="600"
    creationComplete="setButtonStyles();">

    <mx:Script>
        <![CDATA[
            public function setButtonStyles():void {
                comp.getHeaderAt(0).setStyle('color', 0xAA0000);
                comp.getHeaderAt(1).setStyle('color', 0x00AA00);
            }
        ]]>
    </mx:Script>

    <mx:Accordion id="comp">
        <mx:VBox label="first box">
            <mx:TextInput/>
            <mx:Button label="Button 1"/>
        </mx:VBox>
        <mx:VBox label="second box">
            <mx:TextInput/>
            <mx:Button label="Button 2"/>
        </mx:VBox>
    </mx:Accordion>
</mx:Application>
```

また、Accordion クラスの `headerStyleName` スタイルプロパティを使用して、ボタンの外観を制御することもできます。詳細については、『Adobe Flex 2 リファレンスガイド』の Accordion を参照してください。

第 3 部

ユーザーインターフェイスのカスタマイズ

3

第III部では、アプリケーションに追加の機能を付加してユーザーの操作性を向上させる方法について説明します。また、アプリケーションのローカライズに関する情報も示します。

次のトピックが含まれます。

第 17 章

ビヘイビアの使用

17

ビヘイビアを使用すると、ユーザーまたはプログラムの操作に応じて、アニメーションやモーションをアプリケーションに追加できます。たとえば、フォーカスが移ったときにダイアログボックスがわずかに飛び跳ねるような効果を出したり、ダイアログボックスがゆっくり表示状態に推移するフェードイン効果を実装することもできます。

このトピックでは、MXML および ActionScript を使用してビヘイビアをアプリケーションに実装する方法の他、ビヘイビアを構成するトリガとエフェクトという 2 つの要素について説明します。

目次

ビヘイビアについて

" ビヘイビア " とは、トリガにエフェクトを組み合わせたものです。" トリガ " は、マウスのクリック、フォーカスの移動、不可視状態から可視状態への変更など、コンポーネントに対して行われる操作を特に指します。" エフェクト " は、ターゲットコンポーネント上で一定時間 (ミリ秒単位) に発生する視覚的または聴覚的変化です。エフェクトには、コンポーネントのフェード、サイズ変更、移動などがあります。

1 つのトリガに対して再生されるエフェクトを複数定義することもできます。たとえば、ペットショップで使われるアプリケーションの場合、動物のカテゴリを選択するための Button コントロールに実装できます。ユーザーが Button コントロールをクリックすると、対応する動物の種名を表すウィンドウが表示されるような仕組みが考えられます。さらに、表示されたウィンドウを画面の左隅下に移動し、100 x 100 ピクセルから 300 x 300 ピクセルにサイズ変更することも可能です (エフェクト)。

Adobe Flex のコンポーネントでは、特に指定しない限りトリガが発生してもエフェクトは再生されません。コンポーネントでエフェクトを使用するには、トリガにエフェクトを関連付ける必要があります。

メモ

トリガとイベントは同じものではありません。たとえば、Button コントロールには、mouseDown イベントと mouseDownEffect トリガの両方が存在します。イベントは、ユーザーがコンポーネントをクリックしたときに、対応するエフェクトトリガを開始します。mouseDown イベントプロパティは、ユーザーがコンポーネントをクリックしたときに実行されるイベントリスナーを指定する際に使用します。mouseDownEffect トリガプロパティは、エフェクトとトリガを関連付ける際に使用します。

ビヘイビアの適用について

エフェクトの作成、設定、および Flex コンポーネントへの適用には、MXML と ActionScript の両方を使用します。MXML を使用すると、次の例に示すように、アプリケーションの基本ビヘイビアを定義する際に、エフェクトをトリガに関連付けることができます。

```
<?xml version="1.0"?>
<!-- behaviors\ButtonWL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define effect. -->
    <mx:WipeLeft id="myWL" duration="1000"/>

    <!-- Assign effect to targets. -->
    <mx:Button id="myButton" mouseDownEffect="{myWL}"/>
    <mx:Button id="myOtherButton" mouseDownEffect="{myWL}"/>

</mx:Application>
```

この例のエフェクトは WipeLeft です。このエフェクトの継続時間は 1000 ミリ秒 (ms) です。つまり、エフェクトを最初から最後まで再生するには 1000 ms かかります。

エフェクトを各 Button コントロールの mouseDownEffect プロパティに割り当てるには、データバインディングを使用します。mouseDownEffect プロパティは、ユーザーがマウスポインタを使用してコントロールをクリックしたときにエフェクトの再生が行われるように指定するエフェクトトリガです。前例のエフェクトでは、画面の右から左に Button コントロールが徐々に表示されます。

ActionScript を使用して、エフェクトを作成、変更、または再生することができます。ActionScript では、エフェクトがエフェクトトリガに応じて再生されるように設定することができます。また、エフェクトのクラスの play() メソッドを呼び出して、エフェクトを明示的に呼び出すこともできます。ActionScript を使用するとエフェクトを制御することができるので、エフェクトを環境設定の一部として設定したり、ユーザーの操作に基づいて変更することができます。次の例では、ActionScript を使用して WipeLeft エフェクトを作成します。

```
<?xml version="1.0"?>
<!-- behaviors\AsEffect.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="createEffect(event);" >

    <!-- Define effect. -->
```

```
    <mx:Script>
        <![CDATA[

            // Import the effect class.
            import mx.effects.*;

            // Define effect variable.
            private var myWL:WipeLeft;

            private function createEffect(eventObj:Event):void {
                // Create the WipeLeft effect object.
                myWL=new WipeLeft();

                // Set the effect duration.
                myWL.duration=1000;

                // Assign the effects to the targets.
                myButton.setStyle('mouseDownEffect', myWL);
                myOtherButton.setStyle('mouseDownEffect', myWL);
            }
        ]]>
    </mx:Script>

    <mx:Button id="myButton"  />
    <mx:Button id="myOtherButton" />

</mx:Application>
```

この例でも、引き続きイベントを使用してエフェクトを呼び出します。エフェクトをプログラムで再生するには、エフェクトの `play()` メソッドを呼び出します。ActionScript を使用してエフェクトを設定および呼び出す方法、および MXML の使用方法については、613 ページの「ビヘイビアの適用」を参照してください。

ファクトリクラスとインスタンスクラスについて

Flex は 1 つのアーキテクチャを使用してエフェクトを実装します。このアーキテクチャでは、各エフェクトは、ファクトリクラスとインスタンスクラスの 2 つのクラスによって表現されます。

ファクトリクラス　ファクトリクラスは、インスタンスクラスのオブジェクトを作成し、そのターゲットオブジェクト上でエフェクトを実行します。アプリケーション内でファクトリクラスインスタンスを作成し、ズームサイズやエフェクトの継続時間など、エフェクトを制御するために必要なプロパティをそのインスタンスに設定します。その後、次の例に示すように、ファクトリクラスインスタンスを、対象のコンポーネントのエフェクトトリガに割り当てます。

```
<!-- ファクトリクラスを定義します -->
<mx:WipeDown id="myWD" duration="1000"/>
<!-- ファクトリクラスをエフェクトターゲットに割り当てます -->
```

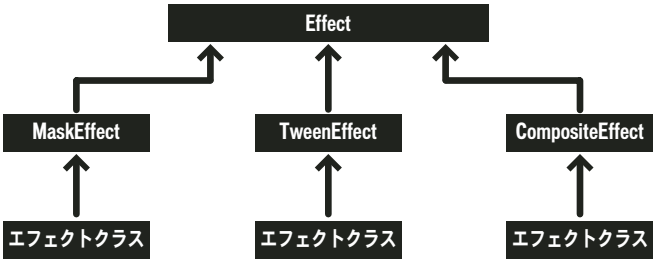
```
<mx:Button id="myButton" mouseDownEffect="{myWD}"/>
<mx:Button id="myOtherButton" mouseDownEffect="{myWD}"/>
```

慣例では、ファクトリクラスの名前は Zoom または Fade などのエフェクトの名前になります。

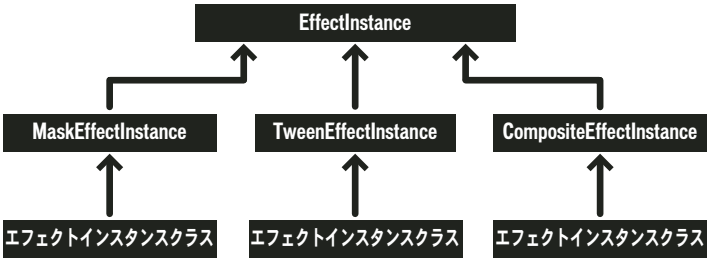
インスタンスクラス　インスタンスクラスは、エフェクトロジックを実装します。エフェクトトリガが実行されるか、ユーザーが `play()` メソッドを呼び出してエフェクトを起動すると、ファクトリクラスによりインスタンスクラスのオブジェクトが作成され、そのターゲット上でエフェクトが実行されます。エフェクトが終了すると、Flex によりインスタンスオブジェクトが破棄されます。エフェクトの対象となるコンポーネントが複数ある場合には、ファクトリクラスは対象ごとに 1 つずつ複数のインスタンスオブジェクトを作成します。

慣例として、インスタンスクラスの名前は、ZoomInstance や FadeInstance のように、エフェクト名に接尾辞 Instance を付加した名前になります。

次の図は、Flex のエフェクトクラスのクラス階層を示しています。



ファクトリクラス



インスタンスクラス

この図のとおり、各ファクトリクラスには対応するインスタンスクラスがあります。

エフェクトを使用するには、次の手順を実行します。

1. ファクトリクラスオブジェクトを作成します。
2. ファクトリクラスオブジェクトを設定します。

Flex では、エフェクトの再生時に、次のアクションが実行されます。

1. エフェクトの各ターゲットコンポーネントにインスタンスクラスのオブジェクトが 1 つ作成されます。
2. ファクトリオブジェクトからインスタンスオブジェクトに構成情報がコピーされます。
3. インスタンスオブジェクトを使用して、ターゲットでエフェクトが再生されます。
4. エフェクトが完了すると、インスタンスオブジェクトが削除されます。

ファクトリオブジェクトに加えた変更は、現在再生中のインスタンスオブジェクトには反映されませんが、次にエフェクトが再生される際には、インスタンスオブジェクトでは新しい設定が使用されます。

アプリケーションでエフェクトを使用する場合は、ファクトリクラスにのみ注意します。これは、インスタンスクラスが実装の詳細を示しているためです。ただし、カスタムエフェクトクラスを作成する場合は、ファクトリクラスとインスタンスクラスを実装する必要があります。詳細については、『Flex 2 コンポーネントの作成と拡張』の第 15 章の「エフェクトの作成」を参照してください。

使用可能なエフェクト

次の表は、Flex がサポートするエフェクトの一覧です。

エフェクト	説明
AnimateProperty	`height`、`width`、`scaleX`、`scaleY` など、コンポーネントの数値プロパティをアニメーション化します。ユーザーは、アニメーション化するプロパティのプロパティ名、開始値、および終了値を指定します。このエフェクトはまずプロパティを開始値に設定し、エフェクトの継続中はプロパティの値が終了値に達するまで値を更新し続けます。 たとえば、Button コントロールの幅を変更するには、アニメーション化するプロパティとして幅を指定し、開始時と終了時の幅の値をエフェクトに指定します。
Blur	コンポーネントにぼかし効果を適用します。Blur エフェクトは画像の詳細部分の表示をやわらかくします。このとき、ソフトで焦点の定まらないぼかし効果からガウスぼかし効果までを生成することができます。ガウスぼかしは、半透明のガラスを通して見たような、かすんだ効果をイメージに与えます。 Blur エフェクトでは、実装の一部として Flash BlurFilter クラスが使用されます。詳細については、『Adobe Flex 2 リファレンスガイド』の flash.filters.BlurFilter を参照してください。 コンポーネントに Blur エフェクトを適用する場合、BlurFilter を適用したり、同じコンポーネントに他の Blur エフェクトを適用することはできません。

エフェクト	説明
Dissolve	ターゲットコンポーネントが徐々に表示または非表示になるようにするには、オーバーレイの alpha プロパティを変更します。 Dissolve エフェクトには、次のプロパティがあります。 • alphaFrom　初期アルファレベルを指定します (0.0 = 透明、1.0 = 不透明)。省略した場合、コンポーネントの現在のアルファレベルが使用されます。 • alphaTo　最終的なアルファレベルを指定します。 • color　エフェクトによってターゲットオブジェクト上に表示されるオーバーレイ矩形の色を表す値です。デフォルト値は、ターゲットコンポーネントの backgroundColor スタイルプロパティで指定された色です。backgroundColor が設定されていない場合は 0xFFFFFF となります。 ターゲットオブジェクトがコンテナである場合、コンテナの子だけがディゾルブします。コンテナの境界線はディゾルブしません。 メモ : DataGrid コントロールの creationCompleteEffect トリガで Dissolve エフェクトを使用するには、DataGrid コントロールの子タグまたはデータバインディングを使用して、コントロールのデータプロバイダをインラインで定義する必要があります。この問題の原因は、creationComplete イベントが送出されるまでデータプロバイダが設定されないことにあります。このため、エフェクトの再生が開始されるときには、Flex では DataGrid コントロールのサイズ設定が完了していません。 メモ : Dissolve エフェクトをテキストに対して使用する場合は、埋め込みフォントを使用する必要があります。デバイスフォントは使用できません。詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。
Fade	コンポーネントが透明から不透明に、または不透明から透明に推移するようなアニメーション効果を設定します。 Fade エフェクトには、次のプロパティがあります。 • alphaFrom　初期アルファレベルを指定します (0.0 = 透明、1.0 = 不透明)。省略した場合、コンポーネントの現在のアルファレベルが使用されます。 • alphaTo　最終的なアルファレベルを指定します。 showEffect トリガや hideEffect トリガに対して Fade エフェクトを指定し、さらに alphaFrom プロパティと alphaTo プロパティを省略した場合、エフェクトには自動的に 0.0 からターゲットの現在の alpha 値 (show トリガの場合)、またはターゲットの現在の alpha 値から 0.0 (hide トリガの場合) に推移するトランジションが適用されます。 メモ : Fade エフェクトをテキストに対して使用する場合は、埋め込みフォントを使用する必要があります。デバイスフォントは使用できません。詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

エフェクト	説明
Glow	コンポーネントに発光効果を適用します。Glow エフェクトでは、実装の一部として Flash GlowFilter クラスが使用されます。詳細については、『Adobe Flex 2 リファレンスガイド』の flash.filters.GlowFilter クラスを参照してください。 コンポーネントに Glow エフェクトを適用する場合、GlowFilter を適用したり、同じコンポーネントに他の Glow エフェクトを適用することはできません。
Iris	ターゲットの中央に配置された矩形マスクを拡張または縮小して、エフェクトターゲットをアニメーション化します。このエフェクトでは、ターゲットの中央からマスクが拡張してターゲットが公開されるか、ターゲットの中央に向かってマスクが縮小し、ターゲットが覆い隠されます。 詳細については、632 ページの「マスクエフェクトの使用」を参照してください。
Move	指定された時間をかけてコンポーネントの位置を変更します。通常、このエフェクトは Canvas コンテナ、"`layout=absolute`" が指定された Application コンテナや Panel コンテナなど、絶対配置を使用するコンテナのターゲットに適用します。VBox コンテナや Grid コンテナなど、自動レイアウトを実行するコンテナのターゲットに適用した場合、移動は実行されますが、コンテナのレイアウトが次回更新されるとターゲットが元の位置に戻ってしまいます。コンテナの `autoLayout` プロパティを `false` に設定すると元の位置への移動を無効にできますが、コンテナのすべてのコントロールのレイアウトが無効になります。 Move エフェクトには、次のプロパティがあります。 • `xFrom` および `yFrom`　コンポーネントの初期位置を指定します。省略した場合、現在の位置が使用されます。 • `xTo` および `yTo`　移動先の位置を指定します。 • `xBy` および `yBy`　コンポーネントの移動量 (x 方向と y 方向のピクセル数) を指定します。正数または負数を指定できます。 `xFrom`、`xTo`、`xBy` の 3 つのプロパティのうち、指定するのはいずれか 2 つで構いません。残りの 1 つは Flex によって自動的に計算されます。3 つすべてを指定した場合、`xBy` プロパティは無視されます。`yFrom`、`yTo`、`yBy` の 3 つのプロパティについても同様です。 トリガに Move エフェクトを指定し、From、To、By の 6 つのプロパティ値をどれも設定しない場合、オブジェクトが現在の位置から新しい位置にスムーズに移動するように、これらのプロパティの値が設定されます。 Move エフェクトの実行時、移動しているコンポーネントの周囲のレイアウトは再調整されません。コンテナの `autoLayout` プロパティを `true` に設定しても、この動作は変わりません。
Pause	指定された時間が経過するまで、すべての処理を中断します。エフェクトを合成する場合などに使用します。詳細については、626 ページの「組み合わせたエフェクトの作成」を参照してください。

エフェクト	説明
Resize	指定された時間をかけてコンポーネントの幅と高さを変更します。 Resize エフェクトには、次のプロパティがあります。 • `widthFrom` および `heightFrom`　初期状態の幅と高さを指定します。省略した場合、現在のサイズが使用されます。 • `widthTo` および `heightTo`　最終的な幅と高さを指定します。 • `widthBy` および `heightBy`　サイズの変更量を初期状態の幅と高さに対する相対的なピクセル数 (正数または負数) で指定します。 `widthFrom`、`widthTo`、`widthBy` の 3 つのプロパティのうち、指定するのはいずれか 2 つで構いません。残りの 1 つは Flex によって自動的に計算されます。3 つのプロパティをすべて指定した場合、`widthBy` プロパティは無視されます。`heightFrom`、`heightTo`、`heightBy` の 3 つのプロパティについても同様です。 Resize トリガに対して Resize エフェクトを指定した場合で、From、To、By のプロパティ (合計 6 つ) をまったく設定しなかった場合、オブジェクトが変更前のサイズから変更後のサイズへとスムーズにトランジションするような値が Flex によって自動的に設定されます。 Resize エフェクトを適用すると、レイアウトマネージャは、対象コンポーネントのサイズの変化に基づいて、隣接するコンポーネントのサイズを変更します。他のコンポーネントのサイズを変えずにエフェクトを実行するには、対象のコンポーネントを Canvas コンテナに配置してください。 Panel コンテナで Resize エフェクトを使用する場合、Panel の子を非表示にしてパフォーマンスを向上させることができます。詳細については、643 ページの「Panel コンテナのサイズ変更におけるパフォーマンスの向上」を参照してください。
Rotate	指定された点を中心にコンポーネントを回転させます。回転の中心点を座標で指定したり、回転の開始時と終了時の角度を指定することができます。角度には正の値または負の値を指定できます。 Resize エフェクトには、次のプロパティがあります。 • `angleFrom` および `angleTo`　回転の初期角度と最終角度を指定します。 • `originX` および `originY`　回転の中心点を座標で指定します。 メモ : Rotate エフェクトをテキストに対して使用する場合は、埋め込みフォントを使用する必要があります。デバイスフォントは使用できません。詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

エフェクト	説明
SoundEffect	MP3 オーディオファイルを再生します。たとえば、ユーザーが Button コントロールをクリックしたときに、サウンドを再生することができます。このエフェクトでは、サウンドの繰り返し再生、ソースファイルの選択、およびボリュームとパンの制御が可能です。 MP3 ファイルを指定するには、source プロパティを使用します。Embed キーワードにより MP3 ファイルを既に埋め込んでいる場合は、MP3 ファイルの Class オブジェクトを source プロパティに渡すことができます。それ以外の場合は、MP3 ファイルへの完全な URL を指定してください。 詳細については、631 ページの「サウンドエフェクトの使用」を参照してください。
WipeLeft WipeRight WipeUp WipeDown	Wipe エフェクトを定義します。エフェクト適用前または適用後のコンポーネントの状態は不可視にする必要があります。 これらのエフェクトには、次のプロパティがあります。 • showTarget　true にするとコンポーネントが表示されます。false にすると、コンポーネントが非表示になります。デフォルト値は true です。 showEffect トリガまたは hideEffect トリガに対して Wipe エフェクトを指定した場合、デフォルトでは、コンポーネントが非表示状態のときに showTarget プロパティが true に設定され、表示状態のときに false に設定されます。 詳細については、632 ページの「マスクエフェクトの使用」を参照してください。
Zoom	コンポーネントをその中心点からズームインまたはズームアウトします。 Zoom エフェクトには、次のプロパティがあります。 • zoomHeightFrom および zoomWidthFrom　ズームの開始位置を表す数値を指定します。デフォルト値は 0.0 です。 • zoomHeightTo および zoomWidthTo　ズームの終了位置を表す数値を指定します。デフォルト値は 1.00 です。ターゲットのサイズを 2 倍にするには、値を 2.0 に指定します。 • originX および originY　ズーム原点の x 位置および y 位置 (基準点)。デフォルト値は、エフェクトターゲットの中心の座標です。 メモ : システムフォントを使用してレンダリングしたテキストに Zoom エフェクトを適用すると、そのテキストはすべてのポイントサイズの範囲で拡大および縮小されます。テキストに Zoom エフェクトを適用する際に埋め込みフォントを使用する必要はありません。しかし、埋め込みフォントを使用すれば、滑らかな Zoom エフェクトが得られます。詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

使用可能なトリガ

エフェクトをターゲットコンポーネントに割り当てるには、トリガ名を使用します。<mx:Style> タグ、または ActionScript の setStyle() 関数および getStyle() 関数を使用して、トリガ名を MXML タグのプロパティとして参照することができます。トリガ名では、以下の命名規則が適用されます。

*triggerEvent*Effect

triggerEvent は、エフェクトを呼び出すイベントです。たとえば、コンポーネントがフォーカスを取得すると、focusIn イベントが発生します。focusIn イベントに対して呼び出されるエフェクトを指定するには、focusInEffect トリガプロパティを使用します。コンポーネントがフォーカスを失ったときに発生する focusOut イベントに対応するトリガプロパティは focusOutEffect です。

次の表に、各トリガに対応するエフェクト名を示します。

トリガ名	トリガ
addedEffect	コンテナにコンポーネントが子として追加された。
creationCompleteEffect	コンポーネントが作成された。
focusInEffect	コンポーネントがキーボードのフォーカスを取得した。
focusOutEffect	コンポーネントがキーボードのフォーカスを失った。
hideEffect	コンポーネントの visible プロパティが true から false に変化し、非表示状態に推移した。
mouseDownEffect	ユーザーがコンポーネント上にマウスポインタを置いた状態でマウスボタンを押した。
mouseUpEffect	ユーザーがマウスボタンを離した。
moveEffect	コンポーネントが移動された。
removedEffect	コンポーネントがコンテナから削除された。
resizeEffect	コンポーネントのサイズが変更された。
rollOutEffect	ユーザーがマウスポインタをコンポーネント外に移動した。
rollOverEffect	ユーザーがマウスポインタをコンポーネント上に移動した。
showEffect	コンポーネントの visible プロパティが false から true に変化し、表示状態に推移した。

ビヘイビアの適用

このセクションでは、MXML と ActionScript の両方を使用してビヘイビアを適用する方法について説明します。

MXML でのビヘイビアの適用

MXML では、トリガ名をコンポーネントの MXML タグのプロパティとして使用し、そのコンポーネントのエフェクトを設定します。たとえば、ユーザーが Button コントロールをクリックしたときに WipeLeft エフェクトが適用されるようにそのコントロールを設定するには、次の例に示すように、<mx:Button> タグで mouseDownEffect トリガプロパティを使用します。

```
<?xml version="1.0"?>
<!-- behaviors\ButtonWL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define effect. -->
    <mx:WipeLeft id="myWL" duration="1000"/>

    <!-- Assign effect to targets. -->
    <mx:Button id="myButton" mouseDownEffect="{myWL}"/>
    <mx:Button id="myOtherButton" mouseDownEffect="{myWL}"/>

</mx:Application>
```

次の例では、Button コントロールに 2 つの Resize エフェクトを作成します。最初の Resize エフェクトでは、ユーザーがボタンをクリックすると、そのサイズが 10 ピクセル拡大します。もう片方の Resize エフェクトでは、ユーザーがマウスボタンを離すと、そのサイズが元に戻ります。各エフェクトの継続時間は 200 ms です。

```
<?xml version="1.0"?>
<!-- behaviors\ButtonResize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Resize id="myResizeUp"
        widthBy="50" heightBy="50"
        duration="200"/>
    <mx:Resize id="myResizeDown"
        widthBy="-50" heightBy="-50"
        duration="200"/>

    <mx:Button id="myButton"
        mouseDownEffect="{myResizeUp}"
        mouseUpEffect="{myResizeDown}"/>

</mx:Application>
```

データバインディングを使用した MXML でのビヘイビアの適用

MXML でデータバインディングを使用して、エフェクトのプロパティを設定できます。たとえば、次の例では、TextInput コントロールを使用して Zoom エフェクトの zoomHeightTo プロパティおよび zoomWidthTo プロパティを設定します。zoomHeightTo プロパティおよび zoomWidthTo プロパティでは、ズームの終了位置を表す数値を 0.0 ～ 1.0 の範囲で指定します。デフォルト値は 1.0 です。これは、オブジェクトの標準のサイズです。

```
<?xml version="1.0"?>
<!-- behaviors\DatabindingEffects.mxml  -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Zoom id="mZoom"
        zoomHeightTo="{Number(zoomHeightInput.text)}"
        zoomWidthTo="{Number(zoomWidthInput.text)}"/>

    <mx:TextInput id="zoomHeightInput" text="1.0"/>
    <mx:TextInput id="zoomWidthInput" text="1.0"/>

    <mx:Button rollOverEffect="{mZoom}"/>

</mx:Application>
```

デフォルトでは、TextInput コントロールの text プロパティは 1.0 に設定されています。TextInput コントロールを編集することにより、異なるズーム値を指定することができます。

スタイルを使用した MXML でのビヘイビアの適用

エフェクトトリガに対応するすべての MXML プロパティは、CSS スタイルとして実装されています。このため、<mx:Style> タグを使用してエフェクトを適用することもできます。たとえば、アプリケーションのすべての TextArea コントロールの mouseDownEffect プロパティを設定するには、次の例に示すように、CSS タイプセレクタを使用します。

```
<?xml version="1.0"?>
<!-- behaviors\MxmlTypeSel.mxml-->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        TextArea { mouseDownEffect: slowWipe; }
    </mx:Style>

    <mx:WipeLeft id="slowWipe" duration="5000"/>
    <mx:TextArea id="myTA"/>

    <mx:TextArea id="myTA2" mouseDownEffect="none"/>

</mx:Application>
```

コンポーネントタグで mouseDownEffect プロパティを設定すると、<mx:Style> タグでの設定はすべてオーバーライドされます。タイプセレクタで定義された関連エフェクトを削除するには、次の例に示すように、トリガの値を明示的に none に設定します。

```
<mx:TextArea id="myTA" mouseDownEffect="none"/>
```

また次の例に示すように、クラスセレクタを使用してエフェクトを適用することもできます。

```
<?xml version="1.0"?>
<!-- behaviors\ButtonWLClassSel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Style>
        <!-- Define a class selector for a TextArea control -->
        .textAreaStyle{ mouseDownEffect: WipeLeft; }
    </mx:Style>

    <mx:WipeLeft id="slowWipe" duration="5000"/>
    <mx:TextArea id="myTA" styleName="textAreaStyle" />

</mx:Application>
```

MXML で定義されたビヘイビアでの setStyle() と getStyle() の使用

トリガプロパティはスタイルとして実装されています。このため、setStyle() メソッドおよび getStyle() メソッドを使用して、トリガとその関連エフェクトを操作できます。setStyle() メソッドのシグネチャを次に示します。

```
setStyle("trigger_name", effect)
```

trigger_name　mouseDownEffect、focusInEffect などのトリガプロパティの名前を示すストリングです。

effect　トリガに関連付けられているエフェクトです。effect のデータ型は Effect オブジェクト、または Effect クラスのサブクラスのオブジェクトです。

getStyle() メソッドのシグネチャを次に示します。

```
getStyle("trigger_name"):return_type
```

trigger_name　トリガプロパティの名前を表すストリングです。

return_type　Effect オブジェクト、または Effect クラスのサブクラスのオブジェクトです。

次に、MXML で定義されたビヘイビアで getStyle() を使用する方法について説明します。

MXML タグプロパティまたは <mx:Style> タグを使用してエフェクトをターゲットに適用すると、getStyle() によって Effect オブジェクトが返されます。次の例に示すように、オブジェクトのタイプは、指定したエフェクトのタイプにより異なります。

```
<?xml version="1.0"?>
<!-- behaviors\ButtonWLGetStyleMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <!-- Set the behavior in MXML. -->
    <mx:WipeLeft id="slowWipe" duration="5000"/>
    <mx:Button id="myB" mouseDownEffect="{slowWipe}" />

    <!-- Call getStyle() to return an object of type WipeLeft. -->
    <mx:Button label="get style"
        click="myTA.text=String(myB.getStyle('mouseDownEffect').duration);"
        />
    <mx:TextArea id="myTA" />

</mx:Application>
```

スタイルの使用方法の詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

ActionScript でのビヘイビアの適用

ActionScript で、エフェクトをイベントリスナーの一部として宣言および再生できます。エフェクトを呼び出すには、エフェクトの `play()` メソッドを呼び出します。

この方法は、1 つのコントロールを使用して、他のコントロールでエフェクトを呼び出す場合に役立ちます。次の例では、Button コントロールの `click` イベントのイベントリスナーを使用して、TextArea コントロールで Resize エフェクトを呼び出します。

```
<?xml version="1.0"?>
<!-- behaviors\ASplay.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="createEffect(event);" >

    <mx:Script>
        <![CDATA[

            // Import effect class.
            import mx.effects.Resize;

            // Create a Resize effect
            private var resizeLarge:Resize = new Resize();

            private function createEffect(eventObj:Event):void {
                // Set the TextArea as the effect target.
                resizeLarge.target=myTA;

                // Set resized width and height, and effect duration.
                resizeLarge.widthTo=150;
                resizeLarge.heightTo=150;
```

```
                resizeLarge.duration=750;
            }
        ]]>
    </mx:Script>

    <mx:VBox borderStyle="solid">
        <mx:Button label="Start"
            click="resizeLarge.end();resizeLarge.play();"/>
        <mx:TextArea id="myTA" text="Here is some text."/>
    </mx:VBox>
</mx:Application>
```

この例では、アプリケーションの creationComplete イベントを使用してエフェクトを設定してから、ユーザーによる Button コントロールのクリックに応じて play() メソッドを呼び出し、そのエフェクトを呼び出します。プログラムでエフェクトを呼び出すため、イベントトリガは必要ありません。

この例では、まず Effect.end() メソッドを呼び出した後、play() メソッドを呼び出しています。end() メソッドを呼び出した後で play() メソッドを呼び出すことで、それまで実行されていたエフェクトのインスタンスをすべて終了してから新しいインスタンスを開始できます。

メモ

end() メソッドの呼び出しが必要になるのは、play() メソッドを使用してエフェクトを呼び出す場合のみです。mouseDownEffect トリガなどのエフェクトトリガを使用してエフェクトを呼び出した場合は、それまで実行されていたすべてのエフェクトが自動的に終了し、指定したエフェクトが呼び出されます。

また、この例では ActionScript を使用してエフェクトを定義します。ただし、ActionScript でエフェクトを定義する必要はありません。次の例に示すように、MXML を使用してこのコードを修正してエフェクトを定義することができます。

```
<?xml version="1.0"?>
<!-- behaviors\MxmlPlay.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Resize id="resizeLarge" target="{myTA}"
        widthTo="150" heightTo="150" duration="750" />

    <mx:Canvas height="300" width="400" borderStyle="solid">
        <mx:Button label="Start" x="50" y="50"
            click="resizeLarge.end();resizeLarge.play();" />
        <mx:TextArea x="100" y="100" id="myTA" text="Here is some text."/>
    </mx:Canvas>
</mx:Application>
```

オプションで、次の例に示すように、ターゲットの配列を play() メソッドに渡し、配列で指定されたすべてのコンポーネントでエフェクトを呼び出すこともできます。

```
resizeLarge.play([comp1, comp2, comp3]);
```

この例では、3 つのコンポーネントで Zoom エフェクトを呼び出します。end() メソッドでは、エフェクトターゲットを引数ではなく、エフェクトのインスタンスとして扱います。したがって、このエフェクトを終了するには、次の例に示すように、エフェクトそのものに対して end() メソッドを呼び出します。

```
resizeLarge.end();
```

play() メソッドを呼び出す場合は、原則的にエフェクトトリガをメソッド呼び出しに置き換えます。play() メソッドを使用して Effect.target プロパティまたは Effect.targets プロパティを呼び出す場合、それらのプロパティを使用してエフェクトのターゲットコンポーネントを指定します。この例では、エフェクトの target プロパティを使用して、1 つのターゲットコンポーネントを指定します。複数のコンポーネントでエフェクトを再生する場合は、Effects.targets プロパティを使用してターゲットコンポーネントの配列を指定します。詳細については、622 ページの「Effect.target プロパティと Effect.targets プロパティを使用したビヘイビアの適用」を参照してください。

target プロパティを使用してエフェクトのターゲットコンポーネントを指定する代わりに、次の例に示すように、ターゲットコンポーネントをコンストラクタに渡すこともできます。

```
// Resize エフェクトを作成します。
var resizeLarge = new mx.effects.Resize(myTA);
```

エフェクトの逆再生

次の例に示すように、オプションの引数を play() メソッドに渡して、エフェクトを逆再生することができます。

```
resizeLarge.play([comp1, comp2, comp3], true);
```

この例では、true を 2 つ目の引数として指定し、エフェクトの逆再生を指定します。デフォルト値は false です。

また、エフェクトを一時停止するには Effect.pause() メソッドを、一時停止しているエフェクトを再開するには Effect.resume() メソッドを、エフェクトを逆再生するには Effect.reverse() メソッドを使用します。

エフェクトの終了

以下の例に示すように、end() メソッドを使用すれば、いつでもエフェクトを終了できます。

```
<?xml version="1.0"?>
<!-- behaviors\ASend.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="createEffect(event);" >

    <mx:Script>
        <![CDATA[

            // Import effect class.
            import mx.effects.Resize;
```

```
            // Create a Resize effect
            private var resizeLarge:Resize = new Resize();

            private function createEffect(eventObj:Event):void {
                // Set the TextArea as the effect target.
                resizeLarge.target=myTA;

                // Set resized width and height, and effect duration.
                resizeLarge.widthTo=150;
                resizeLarge.heightTo=150;
                // Set a long duration.
                resizeLarge.duration=10000;
            }
        ]]>
    </mx:Script>

    <mx:Canvas height="300" width="500" borderStyle="solid">
        <mx:Button label="Start"
            x="10" y="10"
            click="resizeLarge.end();resizeLarge.play();"/>
        <mx:Button label="End"
            x="10" y="50"
            click="resizeLarge.end();"/>
        <mx:TextArea id="myTA"
            x="100" y="100"
            text="Here is some text."/>
    </mx:Canvas>
</mx:Application>
```

この例では、Resize エフェクトの duration プロパティを 10 秒に設定し、end() メソッドを使用する新しい Button コントロールを追加して、ユーザーがボタンをクリックしたときにエフェクトが終了するように設定します。

end() メソッドを呼び出すと、エフェクトはその終了状態までジャンプして終了します。Resize エフェクトの場合、それが終了する前に、拡張される TextArea コントロールの最終サイズが設定されます。つまり、エフェクトの再生が終了したときと同じ状態になります。Move エフェクトの場合、ターゲットコンポーネントがその最終位置まで移動してから、エフェクトが終了します。

effectEnd イベントのリスナーを定義している場合、そのリスナーは end() メソッドによって呼び出されます。つまり、エフェクトの再生が終了したときと同じ状態になります。エフェクトイベントの操作方法の詳細については、635 ページの「エフェクトイベントの処理」を参照してください。

再利用可能なエフェクトの作成

次の例では、Move エフェクトのターゲットに相当する 3 つの引数と、移動の座標を取る再利用可能な関数を作成します。次に、この関数によって Move エフェクトが作成され、それがターゲット上で再生されます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- behaviors\PlayEffectPassParams.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import flash.events.Event;
            import mx.effects.Effect;
            import mx.effects.Move;

            private var myMove:Move = new Move();

            // Click event listener that passes the target component
            // and the coordinates of the center of the parent container
            // to the function that creates the effect.
            private function playMove(target:Object,
                    newX:Number, newY:Number):void {
                myMove.end();
                myMove.target=target;
                myMove.duration = 1000;
                myMove.xTo = newX - target.width/2;
                myMove.yTo = newY - target.height/2;
                myMove.play();
            }

            // Create the Move effect and play it on the target
            // component passed to the function.
            private function handleClick(eventObj:Event):void {
                var targetComponent:Object = eventObj.currentTarget;
                var parentCont:Object = targetComponent.parent;
                playMove(eventObj.target, parentCont.width/2,
                    parentCont.height/2);
            }
        ]]>
    </mx:Script>

    <mx:Canvas width="200" height="200">
        <mx:Button id="myButton"
            label="Center me"
            click="handleClick(event);"/>
    </mx:Canvas>
</mx:Application>
```

スタイルを使用した ActionScript でのビヘイビアの適用

Flex ではエフェクトトリガに対応するプロパティがスタイルとして実装されているため、スタイルシート、`setStyle()` メソッド、および `getStyle()` メソッドを使用してエフェクトを適用できます。このため、次の例に示すように、ActionScript でエフェクトを作成し、`setStyle()` メソッドを使用してエフェクトをトリガに関連付けることができます。

```
<?xml version="1.0"?>
<!-- behaviors\ASStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="createEffect(event);">

    <mx:Script>
        <![CDATA[

            import flash.events.Event;
            import mx.effects.Zoom;

            private function createEffect(eventObj:Event):void {
                // Define a new Zoom effect.
                var zEffect:Zoom = new Zoom();
                zEffect.duration = 2000;
                zEffect.zoomHeightTo = 1.50;
                zEffect.zoomWidthTo = 1.50;

                // Apply the Zoom effect to the Button control.
                myB.setStyle("mouseDownEffect", zEffect);
            }
        ]]>
    </mx:Script>

    <mx:Button id="myB"/>
</mx:Application>
```

また、次の例に示すように、MXML でエフェクトを定義し、ActionScript を使用してそのエフェクトを適用することもできます。

1
```
<?xml version="1.0"?>
<!-- behaviors\ASStylesMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initializeEffect(event);">

    <mx:Script>
        <![CDATA[

            import flash.events.Event;

            private function initializeEffect(eventObj:Event):void {
                myB.setStyle("mouseDownEffect", myWL);
```

1

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

    <mx:WipeLeft id="myWL" duration="1000" />

    <mx:Button id="myB"/>
</mx:Application>
```

次のコードでは、Button コントロールの mouseDownEffect スタイルに対して WipeRight エフェクトと WipeLeft エフェクトが交互に適用されます。

```
<?xml version="1.0"?>
<!-- behaviors\ASStyleGetStyleMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            private function changeEffect():void {
                if (myButton.getStyle("mouseUpEffect") == myWR) {
                    myButton.setStyle("mouseUpEffect", myWL);
                }
                else if (myButton.getStyle("mouseUpEffect") == myWL) {
                    myButton.setStyle("mouseUpEffect", myWR);
                }
            }
        ]]>
    </mx:Script>

    <mx:WipeRight id="myWR" duration="1000"/>
    <mx:WipeLeft id="myWL" duration="1000"/>

    <mx:Button id="myButton"
        label="My Button"
        click="changeEffect();"
        mouseUpEffect="{myWL}"/>
</mx:Application>
```

Effect.target プロパティと Effect.targets プロパティを使用したビヘイビアの適用

通常、トリガではなく play() メソッドを使用してエフェクトを呼び出す場合、Effect.target プロパティまたは Effect.targets プロパティを使用してエフェクトターゲットを指定できます。次の例に示すように、1 つのターゲットを指定する場合は MXML で Effect.target プロパティを使用し、ターゲットの配列を指定する場合は Effect.targets プロパティを使用します。

```
<?xml version="1.0"?>
<!-- behaviors\TargetProp.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Zoom id="myZoom"
        zoomHeightFrom="0.10" zoomWidthFrom="0.10"
        zoomHeightTo="1.00" zoomWidthTo="1.00"
        target="{myButton}"/>

    <mx:Button id="myButton"
        label="Zoom target"
        click="myZoom.end();myZoom.play();"/>
</mx:Application>
```

この例では、target プロパティに対してデータバインディングを使用し、Button コントロールが Zoom エフェクトのターゲットになるように指定します。ただし、エフェクトをトリガに関連付けていないので、エフェクトを呼び出すにはその play() メソッドを呼び出す必要があります。

次の例では、エフェクトの targets プロパティでデータバインディングを使用して、1つの Zoom エフェクトを複数の Button コントロールに適用します。

```
<?xml version="1.0"?>
<!-- behaviors\TargetProp3Buttons.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Zoom id="myZoom"
        zoomHeightFrom="0.10" zoomWidthFrom="0.10"
        zoomHeightTo="1.00" zoomWidthTo="1.00"
        targets="{[myButton1, myButton2, myButton3]}"/>

    <mx:Button id="myButton1"/>
    <mx:Button id="myButton2"/>
    <mx:Button id="myButton3"/>

    <mx:Button label="Zoom targets" id="myButton4"
        click="myZoom.end();myZoom.play();"/>
</mx:Application>
```

エフェクトを呼び出すためのトリガを定義していないので、エフェクトを呼び出すにはその play() メソッドを呼び出す必要があります。エフェクトに対して 3 つのターゲットを指定しているので、play() メソッドを指定すると、これらすべての Button コントロールでエフェクトが呼び出されます。

次の例に示すように、ActionScript で target プロパティと targets プロパティを設定することもできます。

```
<?xml version="1.0"?>
<!-- behaviors\TargetPropAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="createEffect(event);">

    <mx:Script>
        <![CDATA[

            import mx.effects.Fade;
```

```
            import flash.events.Event;

            private var myFade:Fade = new Fade();

            private function createEffect(eventObj:Event):void {
                myFade.duration=5000;

                // Pass Array of targets to play().
                myFade.play([myPanel1, myPanel2]);

                // Alternatively, set targets to an array of components.
                // myFade.targets = [myPanel1, myPanel2];
                // myFade.play();
            }

            private function playZoom(eventObj:Event):void {
                myZoom.end();

                // Alternatively, pass Array of targets to play().
                // myZoom.play([myTA]);

                // Set target to a single component.
                myZoom.target = myTA;
                myZoom.play();
            }
    ]]>
    </mx:Script>

    <mx:Zoom id="myZoom" duration="2000"
        zoomHeightFrom="0.10" zoomWidthFrom="0.10"
        zoomHeightTo="1.00" zoomWidthTo="1.00"/>

    <mx:Panel id="myPanel1" >
        <mx:TextArea id="myTA"/>
    </mx:Panel>

    <mx:Panel id="myPanel2" >
        <mx:Button id="myButton" click="playZoom(event);"/>
    </mx:Panel>
</mx:Application>
```

この例では、Fade エフェクトの targets プロパティを使用して、2 つの Panel コンテナをエフェクトターゲットとして指定し、Zoom エフェクトの target プロパティを使用して、Button コントロールを単独のターゲットとして指定します。

targets プロパティを使用して複数のイベントターゲットを定義した場合、end() メソッドを使用してエフェクトを終了するときには、次の例に示すように、play() メソッドの戻り値を保存し、それを引数として end() メソッドに渡す必要があります。

```
var myFadeArray:Array = myFade.play();
```

次の例に示すように、すべてのターゲット上のエフェクトを終了するには、配列を end() メソッドに渡します。

```
myFade.end(myFadeArray);
```

エフェクトの操作

このセクションでは、エフェクトイベントの操作方法、エフェクトのカスタマイズ方法、およびエフェクトを使用する際の高度な手法について説明します。

エフェクトの継続時間の設定

すべてのエフェクトは duration プロパティを備えています。このプロパティを使用して、エフェクトが発生する時間の長さをミリ秒単位で指定できます。次の例では、WipeLeft エフェクトをカスタマイズし、新たに 2 つのエフェクトを作成しています。SlowWipe エフェクトには 2 秒の継続時間が、ReallySlowWipe エフェクトには 8 秒の継続時間が設定されています。

```
<?xml version="1.0"?>
<!-- behaviors\WipeDuration.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:WipeLeft id="SlowWipe" duration="2000"/>
    <mx:WipeLeft id="ReallySlowWipe" duration="8000"/>

    <mx:Button label="Slow Wipe"
        mouseDownEffect="{SlowWipe}"/>
    <mx:Button label="Really Slow Wipe"
        mouseDownEffect="{ReallySlowWipe}"/>
</mx:Application>
```

エフェクトでの埋め込みフォントの使用

Dissolve、Fade、および Rotate の各エフェクトは、埋め込みフォントを使用してレンダリングしたテキストでのみ機能します。システムフォントを使用しているコントロールにこれらのエフェクトを適用しても、そのテキストは変化しません。

システムフォントを使用してレンダリングしたテキストに Zoom エフェクトを適用すると、そのテキストはすべてのポイントサイズの範囲で拡大および縮小されます。テキストに Zoom エフェクトを適用する際に埋め込みフォントを使用する必要はありません。しかし、埋め込みフォントを使用すれば、滑らかな Zoom エフェクトが得られます。

次に示す例では、2 つの Label コントロールを使用しています。一方には埋め込みフォントを使用し、他方には使用していません。そのため、システムフォントを使用している Label コントロールに Fade エフェクトを適用しても何も起きません。

```
<?xml version="1.0"?>
<!-- behaviors\EmbedFont.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        @font-face {
            src:url("../assets/MyriadWebPro.ttf");
            font-family: myMyriadWebPro;
        }
    </mx:Style>

    <mx:Fade id="fadeOut" alphaFrom="1.0" alphaTo="0.5"/>
    <mx:Fade id="fadeIn" alphaFrom="0.5" alphaTo="1.0"/>

    <mx:VBox>
        <mx:Label fontFamily="myMyriadWebPro"
            mouseDownEffect="{fadeOut}" mouseUpEffect="{fadeIn}"
            text="This Label control uses the MyriadWebPro embedded font
 and the text will fade."/>

        <mx:Label
            mouseDownEffect="{fadeOut}" mouseUpEffect="{fadeIn}"
            text="This Label control uses the system font
and the text will not fade."/>
    </mx:VBox>
</mx:Application>
```

組み合わせたエフェクトの作成

Flex では、エフェクトを組み合わせる方法として、次の 2 種類の方法がサポートされています。

パラレル処理　エフェクトは同時に再生されます。

シーケンス処理　1つのエフェクトが完了してから、次のエフェクトが開始されます。

Parallel エフェクトまたは Sequence エフェクトを定義するには、<mx:Parallel> タグまたは <mx:Sequence> タグを使用します。次の例では、Zoom エフェクトと Rotate エフェクトをパラレル処理で組み合わせる Parallel エフェクト ZoomRotateShow、および Zoom エフェクトと Rotate エフェクトをシーケンス処理で組み合わせる ZoomRotateHide を定義します。

```
<?xml version="1.0"?>
<!-- behaviors\CompositeEffects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  >

    <mx:Parallel id="ZoomRotateShow">
            <mx:Zoom id="myZoomShow"
                zoomHeightFrom="0.0" zoomWidthFrom="0.0"
                zoomHeightTo="1.0" zoomWidthTo="1.0"/>
            <mx:Rotate id="myRotateShow" />
    </mx:Parallel>
```

```
    <mx:Sequence id="ZoomRotateHide">
            <mx:Rotate id="myRotateHide"/>
            <mx:Zoom id="myZoomHide"
                zoomHeightFrom="1.0" zoomWidthFrom="1.0"
                zoomHeightTo="0.0" zoomWidthTo="0.0"/>
    </mx:Sequence>

    <mx:VBox id="myBox" height="100" width="200">
        <mx:TextArea id="aTextArea"
            text="hello"
            hideEffect="{ZoomRotateHide}"
            showEffect="{ZoomRotateShow}"/>
    </mx:VBox>

    <mx:Button id="myButton1"
        label="Show!"
        click="aTextArea.visible=true;"/>
    <mx:Button id="myButton2"
        label="Hide!"
        click="aTextArea.visible=false;"/>
</mx:Application>
```

Button コントロールの `click` イベントのイベントリスナーにより、VBox コンテナの表示と非表示が切り替えられます。VBox コンテナが不可視状態になるとき、その hide エフェクトとして ZoomRotateShow エフェクトが使用されます。また、不可視状態であるとき、ZoomRotateHide エフェクトが使用されます。

VBox コンテナでは、`autoLayout` プロパティが `false` に設定されます。この設定により、エフェクトの再生中にコンテナのレイアウトが更新されなくなります。詳細については、641 ページの「エフェクトによって生じるコンテナレイアウトの無効化」を参照してください。

`<mx:Parallel>` タグと `<mx:Sequence>` タグは相互にネストできます。たとえば、2 つのエフェクトをパラレルに実行し、続けて 3 つ目のエフェクトをシーケンシャルに実行することも可能です。

Parallel エフェクトまたは Sequence エフェクトでは、`duration` プロパティによって各エフェクトの継続時間が設定されます。たとえば、Sequence エフェクトの `duration` プロパティが 3000 に設定されている場合、シーケンスの各エフェクトの再生には 3000 ms かかります。

複数のエフェクトを組み合わせたエフェクトとして合成し、さらにそれを再生するイベントリスナーを作成することもできます。次にその例を示します。

```
<?xml version="1.0"?>
<!-- behaviors\CompositeEffectsAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="createEffect(event);">

    <mx:Script>
        <![CDATA[
```

```
            // Import effect classes and create effect variables.
            import mx.effects.*;
            public var myZoomShow:Zoom;
            public var myRotateShow:Rotate;
            public var ZRShow:Parallel;

            private function createEffect(eventObj:Event):void {
                // Create a Zoom effect.
                myZoomShow=new Zoom(aTextArea);
                myZoomShow.zoomHeightFrom=0.0;
                myZoomShow.zoomWidthFrom=0.0;
                myZoomShow.zoomHeightTo=1.0;
                myZoomShow.zoomWidthTo=1.0;

                // Initialize a Rotate effect.
                myRotateShow=new Rotate(aTextArea);

                // Initialize a Parallel effect.
                ZRShow=new Parallel();
                ZRShow.addChild(myZoomShow);
                ZRShow.addChild(myRotateShow);
            }
        ]]>
    </mx:Script>

    <mx:VBox id="myBox" height="100" width="200">
        <mx:TextArea id="aTextArea" text="hello" visible="false"/>
    </mx:VBox>

    <mx:Button id="myButton1"
        label="Show!"
        click="aTextArea.visible=true; ZRShow.end(); ZRShow.play();"/>
</mx:Application>
```

この例では、Parallel.addChild() メソッドを使用して各エフェクトを Parallel エフェクトに追加し、その後 Effect.play() メソッドを使用してエフェクトを呼び出します。

AnimateProperty エフェクトの使用

AnimateProperty エフェクトを使用して、コンポーネントの数値プロパティをアニメーション化します。たとえば、次の例に示すように、このエフェクトを使用してコントロールの scaleX プロパティをアニメーション化することができます。

```
<?xml version="1.0"?>
<!-- behaviors\AnimateHScrollPos.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Sequence id="animateScaleXUpDown" >
        <mx:AnimateProperty
```

```
            property="scaleX"
            fromValue="1.0"
            toValue="1.5"/>
        <mx:AnimateProperty
            property="scaleX"
            fromValue="1.5"
            toValue="1.0"/>
    </mx:Sequence>

    <mx:Button label="Scale Button"
        mouseDownEffect="{animateScaleXUpDown}"/>
</mx:Application>
```

この例では、Button コントロールをクリックすると、2 つの AnimateProperty エフェクトで構成された Sequence エフェクトの再生が始まります。最初の AnimateProperty エフェクトではコントロールの幅が 150% に拡大され、次の AnimateProperty エフェクトではコントロールがスクロールされて元の幅に戻ります。

エフェクト開始の遅延

`Effect.startDelay` プロパティには、値をミリ秒で指定します。このプロパティを指定すると、エフェクトはトリガされてから指定した時間が経過した後に開始されます。このプロパティには 0 以上の整数値を指定できます。`Effect.repeatCount` プロパティを使用してエフェクトの繰り返し回数を指定している場合、`startDelay` プロパティはエフェクトの初回再生時にのみ適用され、その後の再生時には適用されません。

Parallel エフェクトの `startDelay` プロパティを設定すると、各エフェクトの再生後に遅延が挿入されます。

エフェクトの繰り返し

すべてのエフェクトで、`Effect.repeatCount` プロパティと `Effect.repeatDelay` プロパティがサポートされています。これらを使用して、エフェクトを繰り返すかどうかを設定できます。

- `repeatCount`　エフェクトの再生回数を指定します。値が 0 の場合、`end()` メソッドの呼び出しによって停止されるまでエフェクトが無期限に再生されます。デフォルト値は 1 です。エフェクトを繰り返す場合、`duration` プロパティにより、エフェクトの 1 つのインスタンスの継続時間を指定します。このため、エフェクトの `duration` プロパティが 2000 に、また `repeatCount` プロパティが 3 に設定されている場合、エフェクトの再生には合計で 6000 ms (6 秒) かかります。
- `repeatDelay`　エフェクトを繰り返す間隔をミリ秒で指定します。デフォルト値は 0 です。

たとえば、次の例では、ユーザーが Button コントロールをクリックするまで Rotate エフェクトが繰り返されます。

```
<?xml version="1.0"?>
<!-- behaviors\repeatEff.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Rotate id="myRotate"
        repeatCount="0"/>
    <mx:Image
        source="@Embed(source='../assets/myImage.jpg')"
        mouseDownEffect="{myRotate}"/>

    <mx:Button id="myButton" label="Stop Rotation"
        click="myRotate.end();"/>

</mx:Application>
```

各エフェクトが完了すると、それぞれ effectEnd イベントを送出します。エフェクトを繰り返す場合、最後の再生が完了した後に、エフェクトから effectEnd イベントが送出されます。

Fade や Move などのトゥイーンエフェクトの場合、エフェクトが完了すると、tweenEnd イベントと endEffect イベントの両方が送出されます。トゥイーンエフェクトを繰り返すよう指定した場合、エフェクトが再生されるたびに tweenEnd イベントが発生し、最後の繰り返しが完了すると endEffect イベントが発生します。

ViewStack コンテナおよび TabNavigator コンテナでのエフェクトの使用

ViewStack コンテナおよび TabNavigator コンテナはそれぞれ子コンテナのコレクションで構成され、これらの子コンテナを使用することで、現在表示されている子コンテナを選択できます。現在表示されている子コンテナを変更するときには、非表示にするコンテナの hideEffect プロパティおよび新たに表示する子コンテナの showEffect プロパティを使用して、子コンテナにまとめてエフェクトを適用できます。

ViewStack コンテナまたは TabNavigator コンテナは、非表示にする子コンテナの hideEffect が完了するまで待機し、その後で新しい子コンテナを表示します。エフェクトの再生中に ViewStack コンテナまたは TabNavigator コンテナの selectedIndex プロパティを変更する場合には、現在再生中のエフェクトを中断できます。

ViewStack コンテナおよび TabNavigator コンテナの詳細については、583 ページ、第 16 章の「ナビゲータコンテナの使用」を参照してください。

サウンドエフェクトの使用

SoundEffect クラスを使用して、MP3 ファイルとして表されたサウンドを再生できます。MP3 ファイルを指定するには、source プロパティを使用します。Embed キーワードにより MP3 ファイルを既に埋め込んでいる場合は、MP3 ファイルの Class オブジェクトを source プロパティに渡すことができます。それ以外の場合は、MP3 ファイルへの完全な URL を指定してください。

次の例では、両方の方法で MP3 ファイルを指定します。

```
<?xml version="1.0"?>
<!-- behaviors\Sound.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Embed MP3 file.
            [Bindable]
            [Embed(source="../assets/sound1.mp3")]
            public var soundClass:Class;
        ]]>
    </mx:Script>

    <mx:SoundEffect id="soundEmbed"
        useDuration="false"
        loops="0"
        source="{soundClass}"/>

    <mx:Button id="myButton2"
        label="Sound Embed"
        mouseDownEffect="{soundEmbed}"/>
</mx:Application>
```

この例では、アプリケーションに "sound1.mp3" ファイルを埋め込みます。つまり、ファイルは SWF ファイルにコンパイルされます。

SoundEffect クラスには useDuration や loops などの複数のプロパティがあり、これらを使用して MP3 ファイルの再生を制御できます。useDuration プロパティでは、duration プロパティを使用して MP3 ファイルの再生時間を制御するかどうかを指定します。useDuration プロパティが true の場合、MP3 ファイルは duration プロパティによって指定された時間の間再生されます (デフォルト値は 500 ms)。useDuration を false に設定すると、MP3 ファイルは最後まで再生されます。

loops プロパティでは、MP3 ファイルを繰り返す回数を指定します。値 0 を指定するとエフェクトが 1 回再生され、値 1 を指定すると 2 回再生されます。MP3 ファイルを繰り返す場合でも、再生時間は useDuration プロパティの設定で決まります。

duration プロパティは loops プロパティよりも優先されます。エフェクトの継続時間がサウンドファイルの 1 回の再生時間に満たない場合、サウンドはループされません。

また SoundEffect クラスにより、次のイベントも定義されます。

complete　サウンドファイルのロードが完了すると送出されます。

id3　MP3 サウンドファイルで ID3 データを使用できる場合に送出されます。

ioError　サウンドファイルのロード中にエラーが発生した場合に送出されます。

progress　サウンドファイルのロード時に、定期的に送出されます。イベントオブジェクトでは、現在ロードされているバイト数とロード対象の合計バイト数にアクセスできます。このイベントは常に送出されるとは限りません。progress イベントが送出されなくても complete イベントが生成されることがあります。

詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

マスクエフェクトの使用

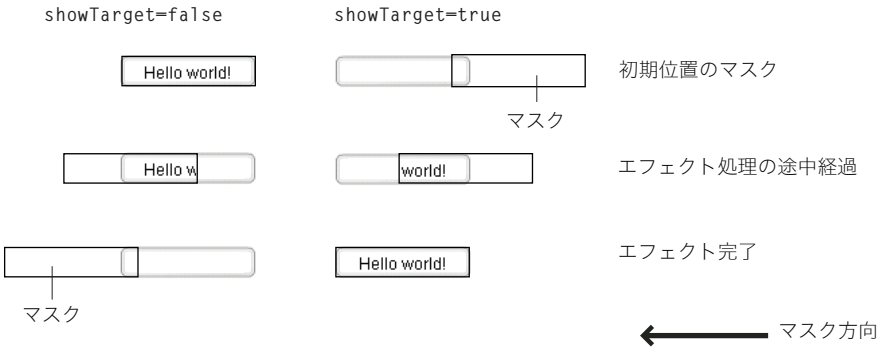
マスクエフェクトは、MaskEffect クラスのサブクラスであるエフェクトです。たとえば、ワイプエフェクトや Iris エフェクトなどがあります。マスクエフェクトでは、マスクと呼ばれるオーバーレイによってエフェクトが実行されます。デフォルトでは、ワイプエフェクトのマスクは、ターゲットコンポーネントと同じ大きさの長方形をしています。Iris エフェクトでは、デフォルトのマスクは長方形であり、コンポーネントの中央に配置されます。

マスクエフェクトでは、エフェクト適用前または適用後のターゲットコンポーネントの状態を不可視にする必要があります。つまり、マスクエフェクトを適用すると、ターゲットコンポーネントが常に画面上に表示されるか、非表示になります。

マスクエフェクトを制御するには、MaskEffect.showTarget プロパティをコンポーネントのアクションに対応するように設定します。ターゲットコンポーネントが見えるように変化させる場合、showTarget を true に設定します。ターゲットが見えなくなるように変化させる場合、showTarget を false に設定します。デフォルト値は true です。

通常、これらのエフェクトは showEffect トリガおよび hideEffect トリガと共に使用します。showEffect トリガは、コンポーネントの visible プロパティが false から true に変更され、そのコンポーネントが見えるように変化した場合に発生します。hideEffect トリガは、コンポーネントの visible プロパティが true から false に変更され、そのコンポーネントが見えなくなるように変化した場合に発生します。マスクエフェクトを showEffect トリガまたは hideEffect トリガと共に使用する場合、showTarget プロパティは自動的に設定されるため無視することができます。

マスクエフェクトを実行すると、showTarget プロパティの設定に基づいて、ターゲットコンポーネントが見えなくなる、または見えるように変化します。次の図に、showTarget プロパティの 2 つの設定における WipeLeft エフェクトのアクションを示します。



次に示すような MaskEffect クラスの複数のプロパティを使用して、マスクの位置とサイズを制御できます。

scaleXFrom、scaleYFrom、scaleXTo、および scaleX　マスクの初期状態および最後の状態の倍率を指定します。値 1.0 を指定すると、マスクのサイズはターゲットコンポーネントと同じになります。2.0 を指定すると、マスクのサイズはコンポーネントの 2 倍になります。また、0.5 を指定すると、マスクのサイズはコンポーネントの半分になります。4 つのプロパティすべてを指定すると、どのプロパティも使用できるようになります。

xFrom、yFrom、xTo、および yTo　ターゲットコンポーネントを基準にマスクの初期位置および最終位置の座標を指定します。たとえば、(0, 0) はターゲットの左上隅を表します。4 つのプロパティすべてを指定すると、どのプロパティも使用できるようになります。

マスクの初期位置および最終位置の座標は、エフェクトのタイプと showTarget プロパティの設定 (true または false) によって異なります。たとえば、WipeLeft エフェクトの showTarget 値が false の場合、マスクの初期位置の座標は (0, 0)、つまりターゲットの左上隅になります。また最終位置の座標は (-width, 0)、つまりターゲットの右上隅になります。このとき、width はターゲットの幅です。

WipeLeft エフェクトの showTarget 値が true の場合、マスクの初期位置の座標は (width, 0)、最終位置の座標は (0, 0) になります。

カスタムマスク関数の作成

マスクエフェクトにカスタムマスク関数を指定するには、createMaskFunction プロパティを使用します。カスタムマスク関数を使用すると、アプリケーション要件に応じてマスクのシェイプ、色、および他の属性を指定できます。

カスタムマスク関数には、次のシグネチャがあります。

```
public function funcName(targ:Object, bounds:Rectangle):Shape
  var myMask:Shape = new Shape();
  // マスクを作成します。

  return myMask;
}
```

カスタムマスク関数は、エフェクトのターゲットコンポーネントに対応する引数、およびターゲットのサイズを定義する 2 番目の引数を取ります。これにより、マスクのサイズを正しく指定できます。関数により、マスクを定義する 1 つの Shape オブジェクトが返されます。

次の例では、WipeLeft エフェクトでカスタムマスクを使用します。

```
<?xml version="1.0"?>
<!-- behaviors\CustomMaskSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[

            // Import the effect class.
            import mx.effects.*;

            public function createLargeMask(targ:Object,
                    bounds:Rectangle):Shape {
                // Create the Shape object.
                var largeMask:Shape = new Shape();

                // Access the Graphics object of the
                // Shape object to draw the mask.
                largeMask.graphics.beginFill(0x00FFFF, 0.5);
                largeMask.graphics.drawRoundRect(0, 0, bounds.width + 10,
                    bounds.height - 10, 3);
                largeMask.graphics.endFill();

                // Return the mask.
                return largeMask;
            }
        ]]>
    </mx:Script>

    <mx:WipeLeft id="customWL"
        createMaskFunction="createLargeMask"
```

```
        showTarget="false"/>

    <mx:WipeLeft id="standardWL"
        showTarget="false"/>

    <mx:HBox borderStyle="solid"
        paddingLeft="10" paddingRight="10"
        paddingTop="10" paddingBottom="10">

        <mx:Button label="custom mask"
            mouseDownEffect="{customWL}"
            height="100" width="100"/>

    </mx:HBox>
</mx:Application>
```

エフェクトイベントの処理

各エフェクトクラスは、次のイベントをサポートしています。

- `effectStart` エフェクトの再生開始時に送出されます。このイベントのイベントオブジェクトの type プロパティは、`EffectEvent.EFFECT_START` に設定されます。
- `effectEnd` エフェクトの再生が停止した後、つまりエフェクトの再生が終了した場合や、`end()` メソッドが呼び出されたことによってエフェクトが中断された場合に送出されます。このイベントのイベントオブジェクトの type プロパティは、`EffectEvent.EFFECT_END` に設定されます。

エフェクトの各ターゲットにつき 1 つのイベントが送出されます。このため、エフェクトに定義しているターゲットが 1 つのみの場合は、1 つの `effectStart` イベントと 1 つの `effectEnd` イベントが送出されます。エフェクトに 3 つのターゲットを定義している場合は、3 つの `effectStart` イベントと 3 つの `effectEnd` イベントが送出されます。

これらのイベントのイベントリスナーに渡されるイベントオブジェクトのタイプは EffectEvent です。EffectEvent クラスは Event クラスのサブクラスです。これには、`target` や `type` など、Event クラスから継承されたプロパティがすべて含まれます。また、このクラスは `effectInstance` という名前の新しいプロパティを定義します。

target イベントを送出した Effect オブジェクトへの参照が含まれます。これは、エフェクトのファクトリクラスです。

type `EffectEvent.EFFECT_END` または `EffectEvent.EFFECT_START` のいずれかです (イベントにより異なる)。

effectInstance EffectInstance オブジェクトへの参照が含まれます。これは、エフェクトのインスタンスクラスによって定義されるオブジェクトです。エフェクトの各ターゲットに、インスタンスクラスのオブジェクトが 1 つ作成されます。エフェクトのターゲットコンポーネントにアクセスするには、`effectInstance.target` プロパティを使用します。

次の例では、endEffect イベントのイベントリスナーを定義します。

```
<?xml version="1.0"?>
<!-- behaviors\EventEffects2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[

            import mx.effects.*;
            import mx.events.EffectEvent;
            import mx.core.UIComponent;

            private function endEffectListener(eventObj:EffectEvent):void {
                // Access the effect object.
                var effectObj:Effect = Effect(eventObj.target);

                // Access the target component of the effect.
                var effectTarget:UIComponent =
                    UIComponent(eventObj.effectInstance.target);

                myTA.text = effectTarget.id;
                myTA.text = myTA.text + " " + eventObj.type;
            }
        ]]>
    </mx:Script>

    <mx:Fade id="myFade" effectEnd="endEffectListener(event);"/>
    <mx:Button id="myButton" mouseUpEffect="{myFade}" />

    <mx:TextArea id="myTA" />
</mx:Application>
```

エフェクトに複数のターゲットが指定されている場合、次の例に示すように、ターゲットごとに effectStart イベントと effectEnd イベントが1回ずつ送出されます。

```
<?xml version="1.0"?>
<!-- behaviors\EventEffects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[

            import mx.effects.*;
            import mx.events.EffectEvent;
            import mx.core.UIComponent;

           private function endSlowFadeEffectListener(eventObj:EffectEvent):void
            {
                // Access the effect object.
```

```
                var effectObj:Effect = Effect(eventObj.target);

                // Access the target component of the effect.
                var effectTarget:UIComponent =
                    UIComponent(eventObj.effectInstance.target);

                myTA.text = myTA.text + effectTarget.id + '\n';
                myTA.text = myTA.text + " " + eventObj.type + '\n';
            }
        ]]>
    </mx:Script>

    <mx:Fade id="slowFade"
        duration="2000"
        effectEnd="endSlowFadeEffectListener(event);"/>

    <mx:Button id="myButton1" creationCompleteEffect="{slowFade}"/>
    <mx:Button id="myButton2" creationCompleteEffect="{slowFade}"/>
    <mx:Button id="myButton3" creationCompleteEffect="{slowFade}"/>
    <mx:Button id="myButton4" creationCompleteEffect="{slowFade}"/>

    <mx:TextArea id="myTA" height="200" width="100" />
</mx:Application>
```

1つのターゲットにつき effectEnd イベントが1回送出されるため、イベントリスナー endSlowFadeEffectListener() は1つの Button コントロールにつき1回ずつ、つまり合計 4 回呼び出されます。

トゥイーンエフェクトイベントの処理

TweenEffect クラスのサブクラスである、Fade エフェクトや Move エフェクトなどの各エフェクトクラスは次のイベントをサポートしています。

- tweenStart　トゥイーンエフェクトの開始時に送出されます。このイベントのイベントオブジェクトの type プロパティは TweenEvent.TWEEN_START に設定されます。tweenStart イベントが送出される前に、Effect.effectStart イベントが送出されます。
- tweenUpdate　TweenEffect クラスで新しい値が計算されるたびに送出されます。このイベントのイベントオブジェクトの type プロパティは TweenEvent.TWEEN_UPDATE に設定されます。
- tweenEnd　トゥイーンエフェクトの終了時に送出されます。このイベントのイベントオブジェクトの type プロパティは TweenEvent.TWEEN_END に設定されます。

これらのイベントのイベントリスナーに渡されるイベントオブジェクトのタイプは TweenEvent です。TweenEvent クラスは Event クラスのサブクラスです。これには、target や type など、Event クラスから継承されたプロパティがすべて含まれます。また、このクラスは次の新しいプロパティを定義します。

value　エフェクトで計算されたトゥイーン値を保存します。たとえば、Fade エフェクトの場合、value プロパティには Fade.alphaFrom プロパティの値から Fade.alphaTo プロパティの値の範囲にある数値が1つ保存されます。Move エフェクトの場合、value プロパティには 2 つのアイテムから成る配列が保存されます。その1番目の値はエフェクトターゲットの現在の x 値で、2 番目の値はエフェクトターゲットの現在の y 値です。value プロパティの詳細については、TweenEffect クラスのサブクラスである各エフェクトのインスタンスクラスを参照してください。

バックグラウンド処理の保留

エフェクトのパフォーマンスを向上させるには、エフェクトの継続時間中、アプリケーションのバックグラウンド処理を無効にします。そのためには、Effect.suspendBackgroundProcessing プロパティを true に設定します。無効にできるバックグランド処理には、コンポーネントの測定、レイアウト、およびエフェクトの継続時間中のデータサービスへの応答などがあります。

suspendBackgroundProcessing プロパティのデフォルト値は false です。ほとんどの場合、このプロパティを true に設定します。ただし、アプリケーションで次のいずれかの条件に該当する場合は、このプロパティを false に設定します。

- エフェクトの再生中にユーザー入力を受信する可能性があり、エフェクトの再生が完了する前にアプリケーションでユーザー入力に応答する必要がある場合
- エフェクトの再生中にサーバーから応答を受信する可能性があり、エフェクトをまだ再生している間にアプリケーションで応答を処理する必要がある場合

イージング関数の使用

エフェクトに対してイージング関数を定義することにより、アニメーションの速度を変更できます。イージングを適用すると、加速と減速をよりリアルに表現できます。また、イージング関数を使用することで、バウンス効果を作成したり、他のタイプのモーションを制御することが可能です。

mx.effects.easing パッケージには、定義済みのイージング関数が用意されています。このパッケージには、Bounce、Linear、Sine など、頻繁に使用するイージングのための関数が収められています。これらの関数の使用法の詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

メモ

イージング関数には、Robert Penner 氏による関数のシグネチャに基づいて、4 つのパラメータを指定します。詳細については、www.ericd.net/chapter7.pdf を参照してください。

次のコードは、イージング関数の基本的な記述形式を示したものです。

```
function myEasingFunction(t:Number, b:Number, c:Number, d:Number):Number {
    ...
}
```

イージング関数には、次のパラメータを指定します。

- t - 時刻を指定します。
- b - コンポーネントの初期位置を指定します。
- c - コンポーネント位置の変化量を指定します。
- d - エフェクトの継続時間をミリ秒単位で指定します。

Flex のイージング関数の使用

コンポーネントに対してイージング関数を指定するには、そのコンポーネントのプロパティに、イージング関数への参照を渡します。イージング関数の名前を渡せば、そのイージング関数で使用する引数の値は自動的に設定されます。

すべてのトゥイーンエフェクト、つまり TweenEffect クラスの子クラスであるエフェクトクラスは、easingFunction プロパティをサポートしています。このプロパティを使用して、目的のエフェクトに対するイージング関数を指定します。マスクエフェクト、つまり MaskEffect クラスの子クラスであるエフェクトクラスでも、イージング関数を使用できます。他にも、イージング関数をサポートしているコンポーネントがあります。たとえば、Accordion コンポーネントおよび Tree コンポーネントでは、openEasingFunction スタイルプロパティを使用してイージング関数を指定できます。また、ComboBox コンポーネントは closeEasingFunction スタイルプロパティをサポートしています。

たとえば、次のコードのように、openEasingFunction プロパティを使用し、Accordion コンテナに対して mx.effects.easing.Bounce.easeOut() メソッドを指定できます。

```
<?xml version="1.0"?>
<!-- behaviors\EasingFuncExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="550">

    <mx:Script>
        import mx.effects.easing.*;
    </mx:Script>

    <mx:Accordion
        openEasingFunction="{Bounce.easeOut}"
        openDuration="2000">
        <mx:VBox label="Pane 1" width="400" height="400"/>
        <mx:VBox label="Pane 2" width="400" height="400"/>
    </mx:Accordion>
</mx:Application>
```

カスタムイージング関数の作成

次の例では、Flex の Move エフェクトとの組み合わせでバウンスモーションを作成するカスタムイージング関数を作成しています。

```
<?xml version="1.0"?>
<!-- behaviors\Easing.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[
            private function myEasingFunction(t:Number, b:Number,
                    c:Number, d:Number):Number {
              if ((t /= d) < (1 / 2.75)) {
                  return c * (7.5625 * t * t) + b;
              }
              else if (t < (2 / 2.75)) {
                  return c * (7.5625 * (t-=(1.5/2.75)) * t + .75) + b;
              }
              else if (t < (2.5 / 2.75)) {
                  return c * (7.5625 * (t-=(2.25/2.75)) * t + .9375) + b;
              }
              else {
                  return c * (7.5625 * (t-=(2.625/2.75)) * t + .984375) + b;
              }
            };
        ]]>
    </mx:Script>

    <mx:Move id="moveLeftShow"
        xFrom="600" xTo="0" yTo="0"
        duration="3000"
        easingFunction="myEasingFunction"/>
    <mx:Move id="moveRightHide"
        xFrom="0" xTo="600"
        duration="3000"
        easingFunction="myEasingFunction"/>

    <mx:LinkBar dataProvider="myVS"/>
    <mx:ViewStack id="myVS" borderStyle="solid">
        <mx:Canvas id="Canvas0" label="Canvas0"
            creationCompleteEffect="{moveLeftShow}"
            showEffect="{moveLeftShow}"
            hideEffect="{moveRightHide}" >
                <mx:Box height="300" width="600" backgroundColor="#00FF00">
                    <mx:Label text="Screen 0" color="#FFFFFF" fontSize="40"/>
                </mx:Box>
        </mx:Canvas>
        <mx:Canvas id="Canvas1" label="Canvas1"
            showEffect="{moveLeftShow}" hideEffect="{moveRightHide}" >
```

```
                <mx:Box height="300" width="600" backgroundColor="#0033CC">
                    <mx:Label text="Screen 1" color="#FFFFFF" fontSize="40"/>
                </mx:Box>
            </mx:Canvas>
    </mx:ViewStack>
</mx:Application>
```

この例では、ViewStack コンテナの子の showEffect プロパティおよび hideEffect プロパティでカスタムエフェクトを使用しています。LinkBar　コントロールでラベルをクリックすると、ViewStack コンテナの対応する子が右から左へ移動し、ViewStack コンテナの左マージン部分に当たって停止します。最初に表示されていた ViewStack コンテナの子はその間に右側に移動します。

showEffect プロパティのカスタムエフェクトは、子の可視状態が false から true へと変化したときにだけトリガされます。そのため、ViewStack コンテナの最初の子には、creationCompleteEffect プロパティが指定されています。このプロパティは、Flex が最初にコンポーネントを作成する際にエフェクトをトリガするために必要です。creationCompleteEffect プロパティを省略した場合、アプリケーションを起動しても moveLeftShow エフェクトは表示されません。

エフェクトによって生じるコンテナレイアウトの無効化

デフォルトでは、子の追加、削除、サイズ変更、移動が生じた場合、コンテナの子のレイアウトが自動的に更新されます。子の位置が変更される Move エフェクト、および子のサイズと位置が変更される Zoom エフェクトでは、どちらもコンテナのレイアウト更新が伴います。

コンテナのレイアウトが更新されると、実質的にはエフェクトの結果が取り消されてしまいます。たとえば、Move エフェクトを使用して、コンテナの子の位置を変更したとします。その後、コンテナの他の子のサイズを変更し、コンテナのレイアウトを強制的に更新させると、先に移動した子の位置は、レイアウト更新時に元の位置に戻ってしまう可能性があります。

レイアウトが更新されないようにするには、コンテナの autoLayout プロパティを false に設定します。デフォルト値は true で、Flex によるレイアウト更新が有効にされています。目的のエフェクトを使用するコンポーネントの親コンテナに対して autoLayout プロパティを無効にしておく必要があります。たとえば、Grid コンテナの子のレイアウトを制御する場合は、その Grid コンテナに対してではなく、子の親にあたる GridItem コンテナの autoLayout プロパティを設定します。

Resize エフェクトや Zoom エフェクトと並行して Move エフェクトを使用するには、autoLayout プロパティを false に設定します。これは、Resize エフェクトや Zoom エフェクトにはコンテナのレイアウト更新が伴うので、子が元の位置に戻ってしまうからです。

Zoom エフェクトを単体で使用する場合、autoLayout プロパティは false に設定するか、デフォルトの true に設定します。たとえば、autoLayout プロパティを true に設定した状態で Zoom エフェクトを使用した場合は、子が拡大 / 縮小されるのに伴い、コンテナのレイアウトが自動的に更新され、子の位置がその変更後のサイズに基づいて確定されます。autoLayout プロパティを false に設定した状態で Zoom エフェクトを使用した場合は、子が中心点を基準にサイズ変更され、他の子の位置は変化しません。

次の例では、イメージの垂直方向と水平方向の配置に、HBox コンテナのデフォルト値 (それぞれ top と left) が使用されます。このイメージに Zoom エフェクトを適用すると、HBox コンテナはそのイメージ全体を表示できるようにサイズ変更され、イメージは HBox コンテナの左上隅に維持されます。

```
<mx:HBox>
  <mx:Image source="myImage.jpg"/>
</mx:HBox>
```

次の例では、イメージが HBox コンテナの中央に配置されます。このイメージに Zoom エフェクトを適用した場合、イメージはサイズが変化しても、HBox コンテナの中央に維持されます。

```
<mx:HBox horizontalAlign="center" verticalAlign="middle">
  <mx:Image source="myImage.jpg"/>
</mx:HBox>
```

デフォルトでは、HBox コンテナは、元のサイズのイメージ全体を表示できるだけの大きさになります。次の例のように、レイアウトの更新を無効にした上で、Zoom エフェクトでイメージを拡大するか、Move エフェクトでイメージを移動した場合、イメージは HBox コンテナの境界線を越えて拡大されます。

```
<mx:HBox autoLayout="false">
  <mx:Image source="myImage.jpg"/>
</mx:HBox>
```

autoLayout プロパティが false に設定されているため、イメージのサイズが変更されても HBox コンテナのサイズは変化しません。HBox コンテナの境界を越えるようなサイズにイメージを拡大した場合、コンテナにはスクロールバーが追加され、イメージはコンテナの境界線でクリッピングされます。

スクロールバーが表示されないようにするには、height プロパティと width プロパティを使用して、変更後のイメージ全体が表示されるように HBox コンテナのサイズを明示的に指定します。または、コンテナの clipContent プロパティを false に設定して、イメージがその境界を越えて描画されるようにします。

Panel コンテナのサイズ変更におけるパフォーマンスの向上

Panel コンテナに Resize エフェクトを適用すると、エフェクトの継続時間中、エフェクトについて計算とレイアウトのアルゴリズムが繰り返し実行されます。Panel コンテナに多数の子が存在する場合、Flex が十分な速度で画面を更新できず、アニメーションの描画がぎこちなくなる場合があります。また、特定の Panel コンテナのサイズを変更したときに、他の Panel コンテナのサイズも変更される場合があります。

この問題を解決するには、Resize エフェクトの hideChildrenTargets プロパティを使用して、Resize エフェクトの再生中は Panel コンテナの子を非表示にします。hideChildrenTargets プロパティの値は Panel コンテナの配列であり、これにアニメーションの再生中にサイズが変化する Panel コンテナを含めます。Resize エフェクトを再生する前に、Flex はこの配列を調べ、指定された各 Panel コンテナの子を非表示にします。

次の例では、Panel コンテナのサイズ変更中に、panelOne と panelTwo という Panel コンテナの子が非表示になります。

```
<?xml version="1.0"?>
<!-- behaviors\PanelResize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Resize id="myResize" heightTo="300"
        hideChildrenTargets="{[panelOne, panelTwo]}"/>

    <mx:HBox>
        <mx:Panel id="panelOne" mouseDownEffect="{myResize}">
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
        </mx:Panel>
        <mx:Panel id="panelTwo" mouseDownEffect="{myResize}">
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
        </mx:Panel>
        <mx:Panel id="panelThree" mouseDownEffect="{myResize}">
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
            <mx:Button/>
        </mx:Panel>
    </mx:HBox>
</mx:Application>
```

hideChildrenTargets 配列の各 Panel コンテナについて、次のエフェクトトリガが実行されます。

- resizeStartEffect　Resize エフェクトの再生前に実行されます。
- resizeEndEffect　Resize エフェクトの再生後に実行されます。

resizeStartEffect トリガによって、再生するエフェクトが指定されている場合、そのエフェクトの再生が完了するまで、Resize エフェクトは保留されます。

Panel コンテナの resizeStartEffect トリガおよび resizeEndEffect トリガのデフォルト値は、Dissolve エフェクトを再生する Dissolve になります。Dissolve エフェクトの詳細については、607 ページの「使用可能なエフェクト」を参照してください。

Dissolve エフェクトを無効にして、Panel コンテナの子を即座に非表示にさせるには、resizeStartEffect トリガと resizeStartEffect トリガの値を none に設定する必要があります。

エフェクトターゲットでの UIComponent.cachePolicy の設定

エフェクトでは、Flash Player のビットマップキャッシュ機能を使用して、アニメーションを高速にできます。通常、エフェクトは、エフェクトの再生中にターゲットコンポーネントの描画に変更がない場合、ビットマップキャッシュを使用します。

たとえば、Fade エフェクトは、ターゲットコンポーネントの alpha プロパティを変更することで動作します。alpha プロパティを変更しても、画面上におけるターゲットコンポーネントの描画方法は同じです。このため、ターゲットコンポーネントをビットマップとしてキャッシュに保存すると、エフェクトのパフォーマンスが向上します。Move エフェクトでは、ターゲットコンポーネントの x プロパティと y プロパティが変更されます。これらのプロパティの値を変更しても、ターゲットコンポーネントの描画方法は同じです。このため、ビットマップキャッシュを有効に活用することができます。

すべてのエフェクトでビットマップキャッシュを使用できるとは限りません。Zoom、Resize、ワイプなどのエフェクトでは、画面上におけるターゲットコンポーネントの描画方法が変更されます。Zoom エフェクトでは、コンポーネントの拡大 / 縮小のプロパティが変更されます。このため、コンポーネントのサイズが変更されます。このようなエフェクトでは、ターゲットコンポーネントをビットマップとしてキャッシュに保存しても、ビットマップがエフェクトの再生中に絶え間なく変更されるので意味がありません。

UIComponent.cachePolicy プロパティでは、エフェクト再生中のコンポーネントのキャッシュ処理を制御します。cachePolicy プロパティには次の値を設定できます。

CacheePolicy.ON　エフェクトターゲットを常にキャッシュに保存します。

CachePolicy.OFF　エフェクトターゲットをキャッシュに保存しません。

CachePolicy.AUTO　Flex により、エフェクトターゲットをキャッシュに保存するかどうかが決定されます。これはデフォルト値です。Flex では、次の規則に基づいて cacheAsBitmap プロパティが設定されます。

- ターゲット上で再生するエフェクトに、ビットマップキャッシュをサポートしていないものが 1 つでも含まれている場合は、そのターゲットの cacheAsBitmap プロパティを false に設定します。
- ターゲット上で再生するエフェクトに、ビットマップキャッシュをサポートしているものが 1 つ以上ある場合は、そのターゲットの cacheAsBitmap プロパティを true に設定します。

通常、cachePolicy プロパティは、デフォルト値である CachePolicy.AUTO のままにしておきます。ただし、ビットマップキャッシュがユーザーインターフェイスに干渉している場合や、ビットマップキャッシュを無効にした方が有効であるなど、アプリケーションの動作についてわかっている場合は、このプロパティを CachePolicy.OFF に設定することもできます。

第 18 章

スタイルとテーマの使用

18

スタイルは、Adobe Flex アプリケーションの外観を定義するのに役立ちます。スタイルを使用すると、単独のコンポーネントの外観を変更あるいは全コンポーネントに同じ外観を適用できます。このトピックでは、CSS (Cascading Style Sheet : カスケーディングスタイルシート) のシンタックスなど、アプリケーションでスタイルを使用する方法について説明します。また、テーマの使用方法についても述べます。

目次

スタイルについて

Flex コンポーネントの外観は、スタイルプロパティを使用して変更します。これらのプロパティでは、Label コントロールで使用するフォントのサイズや、Tree コントロールで使用する背景色を定義できます。Flex の一部のスタイルは親コンテナから子コンテナに継承され、さらにスタイルタイプ全体やクラス全体に継承されます。つまり、スタイルを一度定義するだけで、そのスタイルを特定のタイプの全コントロール、または一連のコントロールに適用することができます。さらに、各コントロールのプロパティをローカルレベル、コンポーネントレベル、グローバルレベルで個別にオーバーライドできるため、柔軟にアプリケーションの外観を制御することができます。

このセクションでは、コントロールにスタイルを適用する方法について説明します。また、CSS の初歩、スタイル値の形式 (Length、Color、および Time) の概要、およびスタイルの継承について説明します。以降のセクションでは、Flex でスタイルを適用するさまざまな方法について詳しく説明します。

Flex では、CSS を使用したコンポーネントレイアウトをすべての面で制御できるわけではありません。x、y、width、height などのプロパティは UIComponent クラスのプロパティであり、スタイルではありません。したがって、これらを CSS で設定することはできません。一方、left、right、top、bottom などはスタイルプロパティなので、コンテナでコンポーネントの場所を操作する処理に使用できます。

Flex でのスタイルの使用

Flex でスタイルを適用する方法は数多くあります。細かな制御を可能にし、プログラムで使用できる方法もあります。柔軟性が低い代わりに、計算処理が少なく済む方法もあります。Flex では、複数の方法を使用してコントロールにスタイルを適用できます。

スタイルを適用する際には、テーマでサポートされるプロパティに注意する必要があります。Flex のデフォルトテーマでは、すべてのスタイルプロパティがサポートされるわけではありません。詳細については、671 ページの「サポートされているスタイルについて」を参照してください。

外部スタイルシート

スタイルをドキュメントまたはアプリケーション全体に適用するには、CSS を使用します。ActionScript を呼び出さなくてもスタイルシートを指定することができます。これはスタイルを適用する上で最も簡潔な方法ですが、柔軟性は最も低くなります。スタイルシートでは、すべてのコントロールで継承されるグローバルスタイル、または特定のコントロールでのみ使用される個別のスタイルクラスを定義できます。

次の例では、外部スタイルシート "myStyle.css" を現在のドキュメントに適用します。

```
<mx:Style source="myStyle.css"/>
```

外部スタイルシート使用の詳細については、673 ページの「外部スタイルシートの使用」を参照してください。

Flex には "framework.swc" ファイル内にグローバルスタイルシート "defaults.css" があります。このファイルには、グローバルクラスセレクタ用のスタイル定義に加え、ほとんどの Flex コンポーネント用のタイプセレクタも含まれています。"defaults.css" の詳細については、673 ページの「デフォルトのスタイルシートについて」を参照してください。

Flex には、それぞれ独自の外観と操作性を持つ他のタイプのスタイルシートもいくつか用意されています。詳細については、700 ページの「付属のテーマファイルについて」を参照してください。

ローカルのスタイル定義

<mx:Style> タグを使用して、現在のドキュメントとその子に適用するスタイルを定義します。スタイルは、CSS のシンタックスを使用して <mx:Style> タグの中で定義します。コントロールのすべてのインスタンスに適用するスタイルを定義できる他、個別のコントロールに適用するスタイルも定義できます。次の例では、新しいスタイルを定義し、それを myButton コントロールのみに適用します。

```
<?xml version="1.0"?>
<!-- styles/ClassSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 15;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
</mx:Application>
```

次の例では、Button クラスのすべてのインスタンスに適用される新しいスタイルを定義します。

```
<?xml version="1.0"?>
<!-- styles/TypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        fontSize: 15;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

ローカルのスタイル定義を使用する方法の詳細については、674 ページの「ローカルのスタイル定義の使用」を参照してください。

StyleManager クラス

すべてのクラスまたは指定したクラスのすべてのインスタンスにスタイルを適用するには、StyleManager クラスを使用します。次の例では、すべての Button コントロールについて、fontSize スタイルを 15 に、color を 0x9933FF に設定します。

```
<?xml version="1.0"?>
<!-- styles/StyleManagerExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">

  <mx:Script><![CDATA[
     public function initApp():void {
        StyleManager.getStyleDeclaration("Button").setStyle("fontSize",15);
        StyleManager.getStyleDeclaration("Button").setStyle("color",0x9933FF);
```

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

  <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

CSSStyleDeclaration オブジェクトを使用して実行時スタイルシートを作成し、StyleManager の setStyleDeclaration() メソッドを使用して適用することもできます。

StyleManager の使用方法の詳細については、677 ページの「StyleManager クラスの使用」を参照してください。

getStyle() メソッドと setStyle() メソッド

コントロールのインスタンスのスタイルプロパティを操作するには、setStyle() メソッドと getStyle() メソッドを使用します。これらのメソッドを使用してスタイルを適用すると、スタイルシートを使用する場合よりも大きな処理能力がクライアントに要求されるものの、スタイルの適用方法をより詳細に制御できます。

次の例では、myButton インスタンスについてのみ、fontSize を 15 に、color を 0x9933FF に設定します。

```
<?xml version="1.0"?>
<!-- styles/SetStyleExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">

  <mx:Script><![CDATA[
     public function initApp():void {
        myButton.setStyle("fontSize",15);
        myButton.setStyle("color",0x9933FF);
     }
  ]]></mx:Script>

  <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

getStyle() メソッドおよび setStyle() メソッドの使用の詳細については、682 ページの「setStyle() メソッドと getStyle() メソッドの使用」を参照してください。

インラインスタイル

MXML タグの属性を使用して、スタイルプロパティを適用します。これらのプロパティは、コントロールのインスタンスにのみ適用されます。この方法は、ActionScript コードブロックやメソッド呼び出しが不要なので、インスタンスのプロパティを適用する最も効率的な方法です。

次の例では、myButton インスタンスについて、fontSize を 15 に、color を 0x9933FF に設定します。

```
<?xml version="1.0"?>
<!-- styles/InlineExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Button id="myButton" color="0x9933FF" fontSize="15" label="Click Here"/>
</mx:Application>
```

MXML タグでは、キャメルケースバージョンのスタイルプロパティを使用する必要があります。たとえば、前の例では、“font-size”(CSS 表記) ではなく “fontSize” を使用します。スタイルプロパティ名の詳細については、660 ページの「プロパティ名とセレクタ名について」を参照してください。

他のスタイルプロパティと同様、インラインスタイルプロパティも変数にバインドできます。

インラインスタイルの使用の詳細については、686 ページの「インラインスタイルの使用」を参照してください。

グローバルスタイルの設定

fontSize や color など、テキストスタイルおよびカラースタイルのほとんどは継承することができます。継承可能なスタイルをコンテナに適用すると、そのスタイルプロパティの値がコンテナのすべての子に継承されます。Panel コンテナの color を緑に設定すると、その Panel コンテナ内のボタンは、色設定がオーバーライドされない限り、すべて緑になります。

しかし、継承できないスタイルも数多くあります。このようなスタイルを親コンテナに適用すると、そのスタイルはそのコンテナにのみ使用されます。継承不可能なスタイルの値は、そのコンテナの子には適用されません。

グローバルスタイルを使用することで、継承不可能なスタイルを、スタイルを明示的にオーバーライドしないすべてのコントロールに適用することができます。Flex では、次の方法を使用してグローバルスタイルを適用できます。

- StyleManager global スタイル
- CSS global セレクタ

StyleManager では、global スタイルを使用して、すべてのコントロールにスタイルを適用できます。StyleManager クラスの使用の詳細については、677 ページの「StyleManager クラスの使用」を参照してください。

CSS スタイル定義で global セレクタを使用して、グローバルスタイルを適用することもできます。これは、外部 CSS スタイルシートまたは <mx:Style> タグの中に置かれます。詳細については、676 ページの「グローバルセレクタの使用」を参照してください。

スタイル値の形式について

スタイルプロパティは、String、Number のいずれかのデータ型になります。また、これらのデータ型の配列となる場合もあります。スタイルプロパティはデータ型だけでなく、プロパティの有効な値を記述する形式 (Length、Time、Color) も持ちます。プロパティによっては、Boolean 型 (true または false のいずれかの値を取る) のように見えるものもありますが、実際にはブール値として解釈される String 型です。ここでは、これらの形式について説明します。

Length 形式

Length 形式は、フォントのサイズ (fontSize) など、サイズの値を設定するすべてのスタイルプロパティに適用されます。Length は Number 型です。

Length 型のシンタックスは、次のとおりです。

length[*length_unit*]

次の例では、fontSize プロパティの値を 20 ピクセルに定義します。

```
<?xml version="1.0"?>
<!-- styles/LengthFormat.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 20px;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
</mx:Application>
```

単位を指定しない場合は、単位にピクセルが使用されます。次の例では、同じフォントサイズを持つ 2 つのスタイルを定義しています。

```
<?xml version="1.0"?>
<!-- styles/LengthFormat2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 20px;
        color: #9933FF;
     }
     .myOtherFontStyle {
        fontSize: 20;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
  <mx:Button id="myButton2" styleName="myOtherFontStyle" label="Click Here"/>
</mx:Application>
```

メモ	長さの値とその単位の間にはスペースを使用しません。

次の表は、Length 型で使用できる単位を示しています。

単位	尺度	説明
px	相対	ピクセル数です。
in	絶対	インチです。
cm	絶対	センチメートルです。
mm	絶対	ミリメートルです。
pt	絶対	ポイントです。
pc	絶対	パイカです。

Flex は em 単位および ex 単位をサポートしていません。次の係数を使用すると、これらの単位を px 単位に変換できます。

1em = 10.06667px

1ex = 6px

Flex では、長さはすべて表示前にピクセルに変換されます。この変換では、1 インチが 72 ピクセルに等しいと見なされます。他の長さについても、これに基づいて変換されます。たとえば、1 cm は 1/2.54 インチです。1 cm をピクセル数で表すには、1 に 72 を乗算して、2.54 で除算します。

インラインスタイルを使用する場合、Flex では単位は無視され、デフォルトのピクセルが使用されます。

fontSize スタイルプロパティでは、数値単位に加えて一連のキーワードも使用できます。fontSize スタイルプロパティを設定する際には、次のキーワードを使用できます。正確なサイズは、クライアントのブラウザで定義されます。

- xx-small
- x-small
- small
- medium
- large
- x-large
- xx-large

次に示すクラスセレクタの例では、fontSize を x-small として定義します。

```
<?xml version="1.0"?>
<!-- styles/SmallFont.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .smallFont {
        fontFamily: Arial, Helvetica, "_sans";
        fontSize: x-small;
        fontStyle: oblique;
```

```
    }
  </mx:Style>

  <mx:Button id="myButton" styleName="smallFont" label="Click Here"/>

</mx:Application>
```

Time 形式

Time 形式は、ComboBox コンポーネントにおけるドロップダウンやポップアップなどのビルトインエフェクトがあるコンポーネントプロパティや、移動するコンポーネントプロパティに使用します。Time 形式は Number 型です。ミリ秒単位で表されます。Time 形式で値を入力するときは、単位を指定しないでください。

次の例では、myTree コントロールの selectionDuration スタイルを 100 ミリ秒に設定します。

```
<?xml version="1.0"?>
<!-- styles/SetStyleExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">

  <mx:Script><![CDATA[
     public function initApp():void {
        myTree.setStyle("selectionDuration", 100);
     }
  ]]></mx:Script>

    <mx:XMLList id="treeData">
        <node label="Mail Box">
            <node label="Inbox">
                <node label="Marketing"/>
                <node label="Product Management"/>
                <node label="Personal"/>
            </node>
            <node label="Outbox">
                <node label="Professional"/>
                <node label="Personal"/>
            </node>
            <node label="Spam"/>
        <node label="Sent"/>
     </node>
  </mx:XMLList>

  <mx:Panel title="Tree Control Example" width="100%">
        <mx:Tree id="myTree" width="100%" labelField="@label"
dataProvider="{treeData}"/>
  </mx:Panel>
</mx:Application>
```

Color 形式

Color は複数の形式で定義します。ほとんどの形式は、CSS スタイル定義の中でのみ使用できます。次の表は、スタイルプロパティで認識される Color 形式を示しています。

形式	説明
hexadecimal	16 進数カラーは、6 桁のコードの前に 0 と小文字の x (0x) またはシャープ記号 (#) を付加して表します。有効な値の範囲は、0x000000 ～ 0xFFFFFF (または #000000 ～ #FFFFFF) です。 0x 接頭辞は、setStyle() メソッドの呼び出しでカラーを定義するとき、および MXML でカラーを定義するときに使用します。接頭辞 # は、CSS スタイルシートと <mx:Style> タグブロック内で使用します。
RGB	RGB カラーでは、赤、緑、および青の組み合わせを、それぞれのカラーが持つ彩度のパーセントで表します。RGB カラーを設定する形式は color:rgb(x%, y%, z%) です。各カラーに設定する彩度のパーセント値を、3 つの値に先頭から赤、緑、青の順で指定します。 RGB 形式は、スタイルシート定義でのみ使用できます。
VGA カラー名	VGA カラー名は、CSS をサポートするすべてのブラウザでサポートされる、16 色の基本カラーのセットです。使用できるカラー名は Aqua、Black、Blue、Fuchsia、Gray、Green、Lime、Maroon、Navy、Olive、Purple、Red、Silver、Teal、White、Yellow です。 VGA カラー名の形式は、スタイルシート定義およびインラインスタイル宣言で使用できます。 VGA カラー名では大文字と小文字が区別されません。

Color 形式は Number 型です。VGA カラー名などの形式を指定すると、String は Number に変換されます。

CSS スタイルの定義と <mx:Style> タグでは、次のカラー値形式がサポートされています。

```
<?xml version="1.0"?>
<!-- styles/ColorFormatCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myStyle {
        themeColor: #6666CC;    /* CSS hexadecimal format */
        color: Blue;        /* VGA color name */
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myStyle" label="Click Here"/>
</mx:Application>
```

スタイルをインラインで設定するときや setStyle() メソッドを使用して設定するときは、次のカラー値形式を使用できます。

```
<?xml version="1.0"?>
<!-- styles/ColorFormatStyleManager.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     public function initApp():void {
        StyleManager.getStyleDeclaration("Button").
            setStyle("themeColor",0x6666CC);
        StyleManager.getStyleDeclaration("Button").
            setStyle("color","Blue");
     }

     public function changeStyles(e:Event):void {
        e.currentTarget.setStyle("themeColor", 0xFF0099);
        e.currentTarget.setStyle("color", "Red");
     }
  ]]></mx:Script>
  <mx:Button id="myButton"
        label="Click Here"
        click="changeStyles(event)"
  />
</mx:Application>
```

スタイルプロパティに対する配列の使用

一部のコントロールは、色の配列を受け取ることができます。たとえば、Tree コントロールの depthColors スタイルプロパティでは、ツリーのレベルごとに異なる背景色を使用できます。配列のプロパティに色を割り当てるには、各項目をカンマで区切ったリストにして、プロパティ定義に追加します。リスト内の順序に従って、インデックスが各項目に割り当てられます。

次の例では、Tree タイプセレクタのプロパティに色の配列を定義します。

```
<?xml version="1.0"?>
<!-- styles/ArraysOfColors.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Tree {
        depthColors: #FFCC33, #FFCC99, #CC9900;
        alternatingItemColors: red, green;
     }
  </mx:Style>

  <mx:XMLList id="treeData">
     <node label="Mail Box">
        <node label="Inbox">
           <node label="Marketing"/>
           <node label="Product Management"/>
           <node label="Personal"/>
        </node>
        <node label="Outbox">
           <node label="Professional"/>
           <node label="Personal"/>
```

```
        </node>
        <node label="Spam"/>
        <node label="Sent"/>
     </node>
  </mx:XMLList>

  <mx:Panel title="Tree Control Example" width="100%">
        <mx:Tree id="myTree" width="100%" labelField="@label"
dataProvider="{treeData}"/>
  </mx:Panel>
</mx:Application>
```

この例では、depthColors のみが表示されます。depthColors が設定されていない場合にのみ、alternatingItemColors プロパティが表示されます。ここでは、説明のために両方が示されています。

次の例のように、括弧で囲まれたカンマ区切り値のリストを使用して、ActionScript 内に配列を定義することができます。

```
<?xml version="1.0"?>
<!-- styles/SetStyleArray.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     public function initApp():void {
        myTree.setStyle("depthColors",[0xFFCC33, 0xFFCC99, 0xCC9900]);
        myTree.setStyle("alternatingItemColors",["red", "green"]);
     }
  ]]></mx:Script>
  <mx:XMLList id="treeData">
     <node label="Mail Box">
        <node label="Inbox">
           <node label="Marketing"/>
           <node label="Product Management"/>
           <node label="Personal"/>
        </node>
        <node label="Outbox">
           <node label="Professional"/>
           <node label="Personal"/>
        </node>
        <node label="Spam"/>
        <node label="Sent"/>
     </node>
  </mx:XMLList>

  <mx:Panel title="Tree Control Example" width="100%">
        <mx:Tree id="myTree"
           width="100%"
           labelField="@label"
           dataProvider="{treeData}"
        />
```

```
        <mx:Tree id="myOtherTree"
           width="100%"
           labelField="@label"
           dataProvider="{treeData}"
           depthColors="[0xFFCC33, 0xFFCC99, 0xCC9900]"
           alternatingItemColors="['red', 'green']"
        />
  </mx:Panel>
</mx:Application>
```

またこの例は、インラインで配列を使用するプロパティを設定できることも示しています。

最後に、次の例に示すように、配列の値を MXML シンタックスで設定し、インラインで適用できます。

```
<?xml version="1.0"?>
<!-- styles/ArrayOfColorsMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Array id="myDepthColors">
     <mx:Object>0xFFCC33</mx:Object>
     <mx:Object>0xFFCC99</mx:Object>
     <mx:Object>0xCC9900</mx:Object>
  </mx:Array>

  <mx:Array id="myAlternatingRowColors">
     <mx:Object>red</mx:Object>
     <mx:Object>green</mx:Object>
  </mx:Array>

  <mx:XMLList id="treeData">
     <node label="Mail Box">
        <node label="Inbox">
           <node label="Marketing"/>
           <node label="Product Management"/>
           <node label="Personal"/>
        </node>
        <node label="Outbox">
           <node label="Professional"/>
           <node label="Personal"/>
        </node>
        <node label="Spam"/>
        <node label="Sent"/>
     </node>
  </mx:XMLList>

  <mx:Panel title="Tree Control Example" width="100%">
        <mx:Tree id="myTree"
           width="100%"
           labelField="@label"
           dataProvider="{treeData}"
           depthColors="{myDepthColors}"
           alternatingItemColors="{myAlternatingRowColors}"
```

```
        />
  </mx:Panel>
</mx:Application>
```

CSS (カスケーディングスタイルシート) の使用

カスケーディングスタイルシート (CSS) は、HTML を始めとするほとんどのスクリプト言語でテキストスタイルを宣言するために使用される標準メカニズムです。スタイルシートは、コンポーネントのタイプまたはコンポーネントのセットを含むクラスに対するフォーマット規則を集めたものです。Flex では、CSS シンタックスとスタイルを使用した、Flex コンポーネントへのスタイル適用がサポートされています。

CSS シンタックスでは、個々の宣言によってスタイル名 (セレクタ) が、スタイルプロパティおよびその値に関連付けられます。1 つのセレクタに複数のスタイルプロパティを定義するには、各プロパティをセミコロンで区切ります。たとえば、次のスタイルでは、myFontStyle という名前のセレクタを定義します。

```
<?xml version="1.0"?>
<!-- styles/ClassSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 15;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
</mx:Application>
```

この例では、myFontStyle はスタイルの新しいクラスを定義するので、" クラスセレクタ " と呼ばれます。マークアップでは、myFontStyle スタイルを明示的にコントロールやコントロールのクラスに適用できます。

" タイプセレクタ " は、特定タイプのすべてのコンポーネントおよびすべてのサブクラスに暗黙的に適用されます。

次の例では、Button という名前のタイプセレクタを定義します。

```
<?xml version="1.0"?>
<!-- styles/TypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        fontSize: 15;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

Flex では、このスタイルはすべての Button コントロール、および Button コントロールのすべてのサブクラスに適用されます。コンテナに対してタイプセレクタを定義すると、そのスタイルが継承スタイルであれば、そのコンテナのすべての子にスタイルが適用されます。

Flex では、新しいコンポーネントインスタンスのスタイルを決定する際、すべての親クラスを調べてタイプセレクタを探します。正確に一致したタイプセレクタのみではなく、すべてのタイプセレクタでの設定が適用されます。たとえば、クラス MyButton で Button を拡張するとします。まず、MyButton のインスタンスにつき、MyButton タイプセレクタがないか確認が行われます。MyButton タイプセレクタでのスタイルが適用され、次に Button タイプセレクタがないか確認が行われます。Button セレクタでスタイルが適用され、次に UIComponent タイプセレクタがないか確認が行われます。UIComponent で拡張処理は停止します。UIComponent から親チェーンをさらに上にたどることはありません。これは、Flex では Sprite (UIComponent の親) についても、Sprite のあらゆる親クラスについても、基本の Object クラスまでタイプセレクタをサポートしていないからです。

注	Flex では、クラスセレクタの名前にハイフンを使用できません。my-class-selector などのように、クラスセレクタ名にハイフンを使用した場合、このスタイルは無視されます。

StyleManager クラスを使用すると、新しいクラスセレクタとタイプセレクタをプログラムによって定義できます。詳細については、677 ページの「StyleManager クラスの使用」を参照してください。

プロパティ名とセレクタ名について

CSS を使用して <mx:Style> ブロックまたは外部スタイルシートでスタイルプロパティを適用する場合の最良の方法は、fontWeight および fontFamily の場合と同様に、そのスタイルプロパティのキャメルケースを使用することです。

開発作業を容易にするため、次の例に示すように、Flex ではキャメルケースによるシンタックスとハイフンを使用したシンタックスをスタイルシートでサポートしています。

```
<?xml version="1.0"?>
<!-- styles/CamelCase.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 15;
     }
     .myOtherFontStyle {
        font-size: 15;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
  <mx:Button id="myButton2" styleName="myOtherFontStyle" label="Click Here"/>
  <mx:Button id="myButton3" fontSize="15" label="Click Here"/>
</mx:Application>
```

ActionScript または MXML タグでは、スタイルプロパティ名にハイフンを使用できないので、キャメルケースによるスタイルプロパティを使用する必要があります。次の例に示すように、スタイルシートではスタイル名にハイフンを使用できません。

```
.myClass { ... } /* Valid style name */
.my-class { ... } /* Not a valid style name */
```

CSS での継承について

一部のスタイルプロパティは継承されます。継承可能なスタイルプロパティを親コンテナに設定した場合、そのスタイルプロパティは子に継承されます。たとえば、Panel コンテナで `fontFamily` を `Times` に定義した場合、プロパティがオーバーライドされない限り、そのコンテナのすべての子でも `fontFamily` に `Times` が使用されます。親コンテナに `textDecoration` などの継承不可能なスタイルを設定すると、そのスタイルは親コンテナにのみ使用され、子には使用されません。継承可能なスタイルプロパティの詳細については、668 ページの「スタイルの継承について」を参照してください。

一般的にカラーとテキストスタイルは、設定方法 (スタイルシートによる方法または `setStyle()` メソッドによる方法) に関わらず継承可能です。それ以外のスタイルは、特に説明がない限り継承できません。

継承の規則に対する例外を次に示します。

- CSS スタイル定義で `global` セレクタを使用する場合、スタイルプロパティが継承可能かどうかに関係なく、すべてのコントロールにこのプロパティが適用されます。`global` セレクタの詳細については、676 ページの「グローバルセレクタの使用」を参照してください。
- タイプセレクタに設定されている値は、スタイルプロパティが継承可能でなくても、ターゲットクラスとそのサブクラスに適用されます。たとえば、Flex では VBox コンテナに対して、VBox タイプセレクタのすべてのスタイルと Container タイプセレクタのスタイルが適用されます。

CSS の違い

Flex では、CSS のサポートと CSS の仕様に大きな違いが 2 つあります。

- Flex では、CSS で利用でき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。また、Flex のコントロールには CSS 仕様で定義されていない独自のスタイルプロパティもあります。Flex コントロールに適用できるスタイルの一覧については、666 ページの「サポートされている CSS プロパティ」を参照してください。
- Flex コントロールでは、現在のテーマで定義されているスタイルだけがサポートされます。テーマで特定のスタイルが使用されていない場合、そのスタイルをコントロールに適用しても、効果はありません。たとえば、デフォルトテーマの Halo Aeon は、`symbolColor` や `symbolBackgroundColor` などのスタイルをサポートしていません。テーマの詳細については、698 ページの「テーマについて」を参照してください。
- Flex スタイルシートでは、Embed キーワードでコントロールのスキンを定義できます。詳細については、741 ページ、第 20 章の「スキンの使用」を参照してください。

クラスセレクタについて

クラスセレクタは、任意のコンポーネントに適用できる一連のスタイル、またはクラスを定義します。スタイルクラスを定義した後、コンポーネントの MXML タグの styleName プロパティを使用してスタイルクラスにポイントします。UIComponent クラスのサブクラスに属するすべての Flex コンポーネントは、styleName プロパティをサポートしています。

次の例では、新しいスタイル myFontStyle を定義し、Button コンポーネントを myFontStyle スタイルクラスに割り当てることにより、このスタイルを Button コンポーネントに適用します。

```
<?xml version="1.0"?>
<!-- styles/ClassSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 15;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
</mx:Application>
```

getStyleDeclaration() メソッドを使用してクラスセレクタにアクセスする場合は、次の例に示すように、クラスセレクタ名の先頭にピリオドを付ける必要があります。

```
<?xml version="1.0"?>
<!-- styles/ClassSelectorStyleManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 15;
        color: #9933FF;
     }
  </mx:Style>

  <mx:Script><![CDATA[
     public function changeStyles(e:Event):void {

StyleManager.getStyleDeclaration('.myFontStyle').setStyle('color',0x3399CC);
     }
  ]]></mx:Script>

  <mx:Button id="myButton" label="Click Here" styleName="myFontStyle"
click="changeStyles(event)"/>

</mx:Application>
```

インラインスタイルで styleName プロパティを使用する場合は、クラスセレクタ名の前にピリオドを付けません。

タイプセレクタについて

タイプセレクタは、特定のタイプの全コンポーネントにスタイルを割り当てます。タイプセレクタを定義する場合、そのスタイルを明示的に適用する必要はありません。その代わり、スタイルは、Flex によってそのタイプのすべてのクラスに適用されます。また、タイプセレクタによって定義されているスタイルプロパティは、目的のタイプのすべてのサブクラスにも適用されます。

次の例は、Button コントロールのタイプセレクタを示しています。

```
<?xml version="1.0"?>
<!-- styles/TypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        fontSize: 15;
        color: #9933FF;
     }
  </mx:Style>
  <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

この例では、現在のドキュメント内のすべての Button コントロール、およびすべての子ドキュメント内のすべての Button コントロールに color スタイルを適用します。さらに、Button のすべてのサブクラスに color スタイルを適用します。

コンポーネントのカンマで区切ったリストを使用して、複数のコンポーネントタイプに同じスタイル宣言を設定することができます。次の例では、すべての Button コンポーネント、TextInput コンポーネント、および Label コンポーネントに対してスタイル情報を定義します。

```
<?xml version="1.0"?>
<!-- styles/MultipleTypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button, TextInput, Label {
        fontStyle: italic;
        fontSize: 24;
     }
  </mx:Style>

  <mx:Button id="myButton" label="Click Here"/>
  <mx:Label id="l1" text="My Label"/>
  <mx:TextInput id="ti1" text="Input text here"/>
</mx:Application>
```

同じ名前のタイプセレクタを異なるレベルで複数使用して、さまざまなスタイルプロパティを設定できます。次の例に示すように、外部 CSS ファイルでは、すべての Label コンポーネントのフォントカラーに青を使用するように設定できます。

```
/* assets/SimpleTypeSelector.css */
Button {
    fontStyle: italic;
```

```
    color: #99FF00;
}
```

さらにローカルのスタイル宣言では、次の例に示すように、すべてのラベルのフォントサイズを 10 ポイントに設定できます。

```
<?xml version="1.0"?>
<!-- styles/TypeSelectorWithExternalCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style source="../assets/SimpleTypeSelector.css"/>

  <mx:Style>
     Button {
        fontSize: 15;
     }
  </mx:Style>
  <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

ローカルのスタイル宣言は、外部のスタイル宣言には影響しません。Flex では指定したスタイルプロパティのみが適用されます。この例では、現在のドキュメントの子にあたる Label コントロールでフォントカラーに青が使用され、フォントサイズは 10 ポイントになります。

すべてのスタイルは、アプリケーション内のすべてのドキュメント、および同じアプリケーション内でロードされるすべてのアプリケーションで共有されます。たとえば、TabNavigator コンテナの別々のタブで 2 つの SWF ファイルをロードした場合、外部スタイル定義は両方のファイルで共有されます。

複合セレクタの使用

クラスセレクタとタイプセレクタを組み合わせて、複合スタイル宣言に基づくスタイルのコンポーネントを作成できます。たとえば、クラスセレクタで色を定義し、タイプセレクタでフォントサイズを定義して、その両方をコンポーネントに適用することができます。

```
<?xml version="1.0"?>
<!-- styles/CompoundSelectors.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
     Label {
        fontSize: 10pt;
     }
     .myLabel {
        color: Blue;
     }
  </mx:Style>

  <mx:Label styleName="myLabel" text="This label is 10pt Blue"/>

</mx:Application>
```

たとえば、タイプセレクタまたはクラスセレクタに対して StyleManager の clearStyleDeclaration() メソッドを呼び出すなどの方法で、いずれかのセレクタを後で削除しても、他方のセレクタのスタイル設定は削除されずに残ります。

セレクタの優先順位について

クラスセレクタは、タイプセレクタよりも優先されます。次の例では、1 番目のボタンのテキストはクラスセレクタによって赤になり、2 番目のボタンのテキストは暗黙的なタイプセレクタによって黄色になります。

```
<?xml version="1.0"?>
<!-- styles/SelectorPrecendence.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myclass {
        color: Red;
     }
     Button {
        fontSize: 10pt;
        color: Yellow;
     }
  </mx:Style>
  <mx:VBox width="500" height="200">
     <mx:Button styleName="myclass" label="I am red"/>
     <mx:Button label="I am yellow"/>
  </mx:VBox>
</mx:Application>
```

いずれのボタンもフォントサイズは 10 ポイントです。クラスセレクタによってタイプセレクタがオーバーライドされるとき、すべての値がオーバーライドされるのではなく、明示的に定義された値のみがオーバーライドされます。

タイプセレクタは、特定のクラスおよびそのサブクラスと子コンポーネントに適用されます。次の例では、VBox コントロールの color プロパティは blue です。つまり、VBox コントロールの直系の子にあたる、Button コントロールと Label コントロールの color プロパティが青になります。

```
<?xml version="1.0"?>
<!-- styles/BasicInheritance.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
     VBox {
        color:blue
     }
  </mx:Style>

  <mx:VBox width="500" height="200">
     <mx:Label text="This is a label"/>
     <mx:Button label="Click Me"/>
  </mx:VBox>
</mx:Application>
```

1つのクラスに適用する複数のタイプセレクタで、同じスタイルプロパティを適用すると、そのクラスに最も近いタイプが優先されます。たとえば、VBox クラスは、Container のサブクラスである Box のサブクラスです。VBox タイプセレクタがなく Box タイプセレクタと Container タイプセレクタがある場合、VBox コントロールの color プロパティの値は、次の例に示すように、Container タイプセレクタではなく Box タイプセレクタから取られます。

```
<?xml version="1.0"?>
<!-- styles/ContainerInheritance.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
     Container {
        color:red
     }
     Box {
        color:green
     }
  </mx:Style>

  <mx:VBox width="500" height="200">
     <mx:Label text="This is a green label"/>
     <mx:Button label="Click this green button"/>
  </mx:VBox>
</mx:Application>
```

すべてのスタイルプロパティが継承可能であるとは限りません。詳細については、668 ページの「スタイルの継承について」を参照してください。

サポートされている CSS プロパティ

Flex では、CSS 仕様で定義される CSS スタイルプロパティの次のサブセットをサポートしています。

- color
- fontFamily
- fontSize
- fontStyle
- fontWeight
- paddingBottom
- paddingLeft
- paddingRight
- paddingTop
- textAlign
- textDecoration
- textIndent

Flex では、CSS シンタックスおよび継承規則を使用して定義するプロパティもサポートされますが、これらは CSS プロパティライブラリには含まれていません。コントロールごとのスタイルプロパティの一覧については、『Adobe Flex 2 リファレンスガイド』で各コントロールに関するエントリを参照してください。

スタイルシートでのリソースの埋め込み

<mx:Style> ブロックで、埋め込みリソースを使用することができます。この機能は、イメージファイルなどの埋め込みリソースに適用できるスタイルプロパティ (たとえば backgroundImage) を使用する場合に便利です。次の例では、イメージを myImage として埋め込みます。このリソースを、スタイル宣言で参照しています。

```
<?xml version="1.0"?>
<!-- styles/EmbedInCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .style1 {
        backgroundImage: Embed("../assets/bird.gif");
     }
  </mx:Style>
  <mx:HBox>
     <mx:TextArea width="200" height="200" styleName="style1"/>
  </mx:HBox>
</mx:Application>
```

グラフィカルスキンでは、次の例に示すように、スタイルシート内で Embed ステートメントを直接使用します。

```
<?xml version="1.0"?>
<!-- skins/EmbedImagesCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        overSkin: Embed("../assets/orb_over_skin.gif");
        upSkin: Embed("../assets/orb_up_skin.gif");
        downSkin: Embed("../assets/orb_down_skin.gif");
     }
  </mx:Style>
  <mx:Button id="b1" label="Click Me"/>
</mx:Application>
```

プログラムスキンでは、次の例に示すように、ClassReference ステートメントを使用します。

```
<?xml version="1.0"?>
<!-- skins/ApplyButtonStatesSkin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        overSkin: ClassReference("ButtonStatesSkin");
        upSkin: ClassReference("ButtonStatesSkin");
```

```
        downSkin: ClassReference("ButtonStatesSkin");
    }
  </mx:Style>
  <mx:Button id="b1" label="Click Me"/>
</mx:Application>
```

Embed の使用の詳細については、1011 ページ、第 30 章の「アセットの埋め込み」を参照してください。スキニングの詳細については、741 ページ、第 20 章の「スキンの使用」を参照してください。

スタイルの継承について

ドキュメント内の一箇所でしかスタイルが定義されていなければ、Flex はその定義を使用してプロパティの値を設定します。しかし、アプリケーションでは、複数のスタイルシート、ローカルのスタイル定義、外部スタイルプロパティ、スタイルプロパティを直接コンポーネントのインスタンスに設定することができます。このため、Flex では特定の順序でこれらの定義を確認し、プロパティの値を決定します。

低いレベルのスタイルは、高いレベルのスタイルや外部スタイルよりも優先されます。インスタンスにスタイルを設定し、そのスタイルをグローバルに設定すると、ローカルスタイルを設定した後にグローバルスタイルを設定した場合でも、ローカルスタイルがグローバルスタイルによってオーバーライドされません。

スタイルの継承順序

Flex がスタイルを探す順序を理解することは、どのスタイルプロパティがどのコントロールに適用されるのかを把握するために重要です。

Flex は、コンポーネントインスタンスに対してインラインで設定されたスタイルプロパティを探します。インスタンスに対してインラインでスタイルが設定されていない場合、インスタンスの `setStyle()` メソッドを使用してスタイルが設定されているかどうかがチェックされます。インスタンスに対して直接スタイルが設定されていない場合、インスタンスの `styleName` プロパティにスタイル宣言が割り当てられているかどうかが調べられます。

スタイル宣言に `styleName` プロパティが割り当てられていない場合、Flex はタイプセレクタのスタイル宣言でプロパティを探します。タイプセレクタのスタイル宣言がない場合、`global` セレクタがチェックされます。以上のすべてのチェックが失敗した場合、プロパティは未定義となり、デフォルトスタイルが適用されます。

スタイルのチェックの初期段階では、コントロールの親コンテナのスタイル設定も調べられます。スタイルプロパティが定義されておらず、プロパティが継承不可能な場合、インスタンスの親コンテナのプロパティが調べられます。プロパティが親コンテナで定義されていなければ、親の親がチェックされ、以降同様にチェックされます。プロパティが継承不可能な場合、親コンテナのスタイル設定は無視されます。

スタイルプロパティの優先順位は、高い順番で次のとおりです。

- インライン
- クラスセレクタ
- タイプセレクタ (複数のセレクタで同じスタイルプロパティが適用されている場合は、最も近いクラスが優先)
- 親チェーン (スタイル継承のみ)
- global セレクタ

後でコンポーネントのインスタンスで setStyle() メソッドを呼び出すと、このメソッドは、インラインも含め、すべてのスタイル設定に優先します。

<mx:Style> タグ、外部スタイルシート、"defaults.css" スタイルシートのスタイル定義にも、優先順位があります。"defaults.css" 内の同一のスタイル定義は、<mx:Style source="*stylesheet*"/> タグで指定される外部スタイルシートでオーバーライドされます。これは <mx:Style> タグ内のスタイル定義によりオーバーライドされます。

次の例では、fontFamily プロパティを Times に設定し、fontSize プロパティを 24 に設定する Panel のタイプセレクタを定義します。この結果、Panel コンテナ内のすべてのコントロール、および Button や TextArea などのすべてのサブクラスがこのスタイルを継承します。ただし、button2 はインラインでスタイルを定義することにより、継承されたスタイルをオーバーライドします。アプリケーションがレンダリングされると、button2 にはフォント Arial、フォントサイズ 12 が使用されます。

```
<?xml version="1.0"?>
<!-- skins/MoreContainerInheritance.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
  Panel {
     fontFamily: Times, "_serif";
     fontSize: 24;
  }
  </mx:Style>
  <mx:Panel title="My Panel">
     <mx:Button id="button1" label="Button 1"/>
     <mx:Button id="button2" label="Button 2" fontFamily="Arial" fontSize="12"/>
     <mx:TextArea text="Flex has is own set of style properties which are
        extensible so you can add to that list when you create a custom
        component." width="425" height="400"/>
  </mx:Panel>
</mx:Application>
```

Flex コンポーネントにサブコンポーネントがある場合、タイプセレクタのスタイル継承が予期されない順序になることがあります。サブコンポーネントは、まるで独立したコンポーネントであるかのように、そのタイプセレクタに設定されたスタイルプロパティを継承します。

たとえば、TextInput クラスは、ComboBox コントロール、NumericStepper コントロール、および RichTextEditor コントロールでサブコンポーネントとして使用されます。したがって、TextInput タイプにタイプセレクタを定義すると、TextInput クラスを使用する ComboBox コントロール、NumericStepper コントロール、および RichTextEditor コントロールの領域にもこれらのスタイルが反映されます。

次の例では、TextInput タイプセレクタで fontSize スタイルを 15 に設定します。結果として、同様にフォントサイズが 15 になるテキストがあります。RichTextEditor ではフォントのドロップダウンリストのテキスト、NumericStepper ではテキスト領域、ComboBox ではドロップダウンリスト、フォントサイズのドロップダウンリスト、および URL フィールドのテキストです。

```
<?xml version="1.0"?>
<!-- skins/SubComponentInheritance.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     TextInput {
        fontSize:15;
     }
  </mx:Style>

  <mx:RichTextEditor/>

  <mx:ComboBox>
     <mx:dataProvider>
        <mx:String>2005</mx:String>
        <mx:String>2006</mx:String>
        <mx:String>2007</mx:String>
     </mx:dataProvider>

  </mx:ComboBox>
  <mx:NumericStepper/>

</mx:Application>
```

継承の例外

すべてのスタイルが継承可能というわけではなく、またすべてのコンポーネントとテーマでサポートされているわけではありません。一般的には、カラーとテキストのスタイルは、CSS またはスタイルプロパティを使用してどのように設定されているかに関わらず継承されます。それ以外のスタイルは、特に説明がない限り継承できません。

次の条件を満たす場合にのみ、スタイルは継承されます。

- スタイルがテーマでサポートされています。Flex のデフォルトのテーマでサポートされるスタイルの一覧については、671 ページの「サポートされているスタイルについて」を参照してください。

- スタイルが継承可能です。『Adobe Flex 2 リファレンスガイド』で各コントロールのエントリを参照すると、そのコントロールで継承されるスタイルをすべて知ることができます。また、静的メソッドである `isInheritingStyle()` や `isInheritingTextFormatStyle()` を StyleManager クラスで使用して、スタイルが継承可能であるかどうかも判断できます。
- スタイルがコントロールでサポートされています。各コントロールでサポートされるスタイルの情報については、『Adobe Flex 2 リファレンスガイド』で該当のコントロールの説明を参照してください。
- スタイルがコントロールの親コンテナ、またはコンテナの親に設定されています。他のクラスがコントロールの親コンテナ、またはコントロールの親コンテナの親コンテナでない限り、スタイルは他のクラスから継承されません。タイプセレクタを使用してスタイルプロパティを適用している場合は、この条件に対する例外となります。この場合は、そのクラスのタイプセレクタに設定されているプロパティ、および基本クラスのタイプセレクタに設定されているあらゆるプロパティが適用されます。
- スタイルが低いレベルでオーバーライドされません。たとえば、スタイルのタイプセレクタ (`Button { color:red }` など) を定義し、その後でコントロールにインスタンスプロパティ (`<mx:Button color="blue"/>` など) を設定した場合、タイプセレクタのスタイルが継承可能であっても、そのスタイルによってスタイルインスタンスプロパティがオーバーライドされることはありません。

`global` セレクタを使用すると、継承不可能なスタイルをすべてのコントロールに適用することができます。詳細については、676 ページの「グローバルセレクタの使用」を参照してください。

サポートされているスタイルについて

継承可能および継承不可能なテキストスタイルは、両方ともすべてのテーマでサポートされますが、すべてのスタイルが全テーマでサポートされるわけではありません。コントロールにスタイルプロパティを設定しようとしても、現在のテーマでそのスタイルがサポートされていなければ、スタイルは適用されません。

スタイルには、テーマのスキンでのみ使用されるものや、コンポーネントのコード自体によって使用されるものがあります。コンポーネントの表示テキストにはスキンを適用できないので、テキストスタイルのサポートはテーマに依存しません。

テーマ		サポートされているスタイル	

すべて	継承可	`color`
		`fontFamily`
		`fontSize`
		`fontStyle`
			`fontWeight`
			`textAlign`
			`textIndent`

すべて	継承不可	`paddingLeft`
		`paddingRight`
		`textDecoration`	

テーマ		サポートされているスタイル	
すべて	コンポーネント定義による	disabledColor headerHeight horizontalAlign horizontalGap leading paddingBottom paddingLeft paddingRight paddingTop	rollOverColor selectionColor tabHeight tabWidth textRollOverColor textSelectedColor verticalAlign verticalGap
Halo Aeon	コンポーネント依存およびすべてのテキストスタイル	backgroundGradientAlphas backgroundGradientColors borderAlpha borderColor borderSides borderThickness cornerRadius dateHeaderColor dateRollOverColor dropShadowEnabled fillAlphas fillColors	footerColors headerColors highlightAlphas highlightColor roundedBottomCorners selectedDateColor shadowDirection shadowDistance strokeWidth themeColor todayColor

テーマの詳細については、698 ページの「テーマについて」を参照してください。

themeColor プロパティについて

デフォルトの Halo テーマの多くのアセットで、themeColor というプロパティがサポートされています。このプロパティは Application タグで設定できます。設定したカラーは、Button コントロールの境界線、Accordion コントロールのヘッダ、ToolTip コントロールの背景など、コンポーネントアセット上の Flex アプリケーション全体に適用されます。

0xCCCCCC (銀) や 0x0066FF (青) などのカラー値のほか、themeColor プロパティの次の値も有効です。

- haloOrange
- haloBlue
- haloSilver
- haloGreen

デフォルト値は haloBlue です。次の例では、themeColor の値を haloOrange に設定します。

```
<?xml version="1.0"?>
<!-- styles/ThemeColorExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
themeColor="haloOrange">

  <mx:Button id="myButton" label="Click Me" toolTip="Click me"/>

</mx:Application>
```

外部スタイルシートの使用

Flex では外部 CSS スタイルシートをサポートしています。ローカルスタイルシートの場所を宣言するか、グローバルスタイルシートを使用して、すべてのアプリケーションで使用するスタイルを定義することができます。現在のドキュメントおよびその子ドキュメントにスタイルシートを適用するには、<mx:Style> タグの source プロパティを使用します。

メモ	1 つのアプリケーションで使用するスタイルシートの数をできるだけ少なくし、アプリケーションの最上位のドキュメント (<mx:Application> タグを含むドキュメント) にのみ設定してください。スタイルシートを子ドキュメントでのみ設定した場合、予期しない結果が生じることがあります。

次の例では、"flex_app_root/assets" ディレクトリにある "MyStyleSheet.css" ファイルを指定します。

```
<?xml version="1.0"?>
<!-- styles/ExternalCSSExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style source="../assets/SimpleTypeSelector.css"/>

  <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

source プロパティの値は、スタイル宣言を含むファイルの URL です。source プロパティを使用する場合、その <mx:Style> タグの内容を空にする必要があります。<mx:Style> タグを追加して、他のスタイルを定義することもできます。インクルードするファイルには <mx:Style> タグを追加しないでください。

外部スタイルシートファイルには、タイプセレクタとクラスセレクタの両方を含めることができます。

CSS ファイルを SWF ファイルにコンパイルして、実行時にロードすることもできます。詳細については、688 ページの「実行時のスタイルシートのロード」を参照してください。

デフォルトのスタイルシートについて

Flex には、すべてのアプリケーションで使用されるデフォルトのスタイルシートが用意されています。このスタイルシートを使用すると、Flex コンポーネントすべてに一貫したスタイルが適用されます。このファイルは "defaults.css" で、"/frameworks/libs" ディレクトリの "framework.swc file" ファイル内にあります。このファイルによって埋め込まれるプログラムスキンクラスは、mx.skins.halo パッケージに収められています。使用されるグラフィカルスキンも、"framework.swc" ファイルにあります。

デフォルトのスタイルシートで Halo テーマが構成されます。テーマの詳細については、698 ページの「テーマについて」を参照してください。

"defaults.css" ファイルは暗黙的にロードされ、コンパイルの段階で Flex アプリケーションに適用されます。`defaults-css-url` コンパイラオプションを使用すると、デフォルトのスタイルとして明示的に他のファイルを指定できます。"defaults.css" ファイルは、その名前を変更するか、"framework.swc" ファイルから削除することで無効にすることもできます。

"defaults.css" ファイルでは、すべての Flex コンポーネントの外観と操作性が定義されています。アプリケーションに追加のテーマや CSS ファイルを適用しても、カスタムスタイルでオーバーライドされないコンポーネントに対しては、引き続き "defaults.css" に記述されたスタイルが使用されます。Flex からデフォルトテーマを完全に除去するには、"defaults.css" で定義されているスタイルをすべて削除するか、オーバーライドする必要があります。

Flex では他のスタイルシートも扱うことができるので、テーマを短時間で容易に適用できます。詳細については、700 ページの「付属のテーマファイルについて」を参照してください。

ローカルのスタイル定義の使用

`<mx:Style>` タグには、CSS 2.0 のシンタックスに準拠するスタイルシート定義が含まれます。これらのスタイルシートは、現在のドキュメントとそのすべての子に適用されます。`<mx:Style>` タグは、次のシンタックスを使用してローカルスタイルを定義します。

```
<mx:Style>
  selector_name {
    style_property: value;
    [...]
  }
</mx:Style>
```

次の例では、`<mx:Style>` タグでクラスセレクタとタイプセレクタを定義します。

```
<?xml version="1.0"?>
<!-- styles/CompoundLocalStyle.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 15;
        color: #9933FF;
     }

     Button {
        fontStyle: italic;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
</mx:Application>
```

Application タイプセレクタの使用

Application コンテナは、Flex アプリケーションの最上位のコンテナです。Application タイプセレクタで定義された継承可能なスタイルは、コンテナのすべての子およびサブクラスに継承されます。継承不可能なスタイルは、Application コンテナ自体にのみ適用され、その子には適用されません。

Application タイプセレクタで適用されるスタイルは、そのスタイルが継承不可能な場合、Application オブジェクトの子には継承されません。CSS を使用して継承不可能なスタイルをグローバルに適用するには、global タイプセレクタを使用します。詳細については、676 ページの「グローバルセレクタの使用」を参照してください。

Application タイプセレクタでスタイルを定義する場合、コンポーネントはこれらのクラスの子であり、Application タイプセレクタのスタイルを継承するため、各コンポーネントに対してスタイルを宣言する必要はありません。

Application タイプセレクタのスタイルを定義するには、次のシンタックスを使用します。

```
<mx:Style>
  Application { style_definition }
</mx:Style>
```

Application タイプセレクタを使用して、背景イメージなどの表示設定を行い、Flex アプリケーションがブラウザでどのように表示されるかを定義できます。次の例の Application スタイル定義では、アプリケーションファイルを左揃えで配置し、余白を削除し、背景イメージを空に設定します。

```
<?xml version="1.0"?>
<!-- styles/ApplicationTypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Application {
        paddingLeft: 0px;
        paddingRight: 0px;
        paddingTop: 0px;
        paddingBottom: 0px;
        horizontalAlign: "left";
        backgroundImage: " "; /* The empty string sets the image to nothing. */
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
</mx:Application>
```

背景イメージに空のストリングを設定すると、Flex ではデフォルトの灰色のグラデーションは描画されません。

StyleManager クラスを使用すると、Application タイプセレクタの値をプログラムによって定義できます。詳細については、677 ページの「StyleManager クラスの使用」を参照してください。

グローバルセレクタの使用

Flex では、スタイルをすべてのコントロールに適用する global セレクタを使用できます。global タイプセレクタで定義されたプロパティは、各コントロールで明示的にオーバーライドしない限り、すべてのコントロールに適用されます。global セレクタはタイプセレクタと同じような性質を持っているので、CSS で定義するときにその先頭にはピリオドを付けません。

次の例では、継承可能なプロパティ fontSize と、継承不可能なプロパティ textDecoration を global タイプセレクタに定義します。

```
<?xml version="1.0"?>
<!-- styles/GlobalTypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     global {
        fontSize:22;
        textDecoration: underline;
     }
  </mx:Style>
  <mx:Button id="myButton" label="Click Here"/>
  <mx:Label id="myLabel" text="This is a label"/>
</mx:Application>
```

また次の例に示すように、getStyleDeclaration() メソッドを使用して、global セレクタによるスタイルを適用することもできます。

```
<?xml version="1.0"?>
<!-- styles/GlobalTypeSelectorAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp(event)">
  <mx:Script><![CDATA[
     public function initApp(e:Event):void {
        StyleManager.getStyleDeclaration("global").
            setStyle("fontSize", 22);
        StyleManager.getStyleDeclaration("global").
            setStyle("textDecoration", "underline");
     }
  ]]></mx:Script>

  <mx:Button id="myButton" label="Click Here"/>
  <mx:Label id="myLabel" text="This is a label"/>
</mx:Application>
```

クラスセレクタ、タイプセレクタ、およびインラインスタイルはすべて、global セレクタをオーバーライドします。

StyleManager クラスの使用

StyleManager クラスにより、ActionScript を使ってクラスセレクタおよびタイプセレクタにアクセスできます。また、継承可能なプロパティと継承不可能なプロパティをグローバルに適用することもできます。StyleManager を使用すると、新しい CSS スタイル宣言を定義し、それを Flex アプリケーションのコントロールに適用できます。

StyleManager を使用して値を設定には、次のシンタックスを使用します。

```
mx.styles.StyleManager.getStyleDeclaration(style_name).setStyle("property",
   value);
```

style_name に指定できるのは、リテラルな `global`、Button や TextArea などのタイプセレクタ、または `<mx:Style>` タグまたは外部スタイルシートで定義するクラスセレクタです。グローバルスタイルは、各オブジェクトで明示的にオーバーライドされない限り、すべてのオブジェクトに適用されます。

`getStyleDeclaration()` メソッドは、`textDecoration`, などの継承不可能なスタイルを 1 度に複数のクラスに適用する場合に役立ちます。このプロパティは、CSSStyleDeclaration タイプのオブジェクトを参照します。タイプセレクタや外部スタイルシートは、既に CSSStyleDeclaration タイプのものであると想定されます。Flex では、このタイプのオブジェクトに定義したクラスセレクタが内部で変換されます。

次の例では、スタイルプロパティを `Button`、`myStyle`、および `global` の各スタイル名に適用します。

```
<?xml version="1.0"?>
<!-- styles/USingStyleManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp(event)">
  <mx:Style>
     .myStyle {
        color: red;
     }
  </mx:Style>

  <mx:Script><![CDATA[
     import mx.styles.StyleManager;
     public function initApp(e:Event):void {
        // Type selector; applies to all Buttons and subclasses
        // of Button.
        StyleManager.getStyleDeclaration("Button").
            setStyle("fontSize",24);

        // Class selector; applies to controls using the style
        // named myStyle. Note that class selectors must be prefixed
        // with a period.
        StyleManager.getStyleDeclaration(".myStyle").
            setStyle("color",0xCC66CC);

        // Global style: applies to all controls.
```

```
        StyleManager.getStyleDeclaration("global").
            setStyle("fontStyle","italic");
    }
]]></mx:Script>

<mx:Button id="myButton"
      label="Click Here"
      styleName="myStyle"
/>
<mx:Label id="myLabel"
      text="This is a label"
      styleName="myStyle"
/>

</mx:Application>
```

メモ

継承可能なスタイルでも、継承不可能なスタイルでも、global スタイルに設定した場合は、階層内の位置とは無関係にすべてのコントロールに適用されます。

CSS スタイル宣言を作成するには、CSSStyleDeclaration クラスで ActionScript を使用します。これを利用すると、実行時にスタイルシートを作成して編集し、Flex アプリケーションのクラスに適用できます。実行時にスタイル定義を変更して適用するには、setStyle() メソッドを使用します。

StyleManager は、setStyleDeclaration() メソッドも備えています。このメソッドを使用すると CSSStyleDeclaration オブジェクトをセレクタとして適用できるので、あるタイプのコンポーネントすべてにスタイルシートを適用できます。このセレクタは、クラスセレクタまたはタイプセレクタとして機能します。

次の例では、新しい CSSStyleDeclaration オブジェクトを作成し、いくつかのスタイルプロパティをオブジェクトに適用します。次に、StyleManager の setStyleDeclaration() メソッドを使用してすべての Button コントロールに新しいスタイルを適用します。

```
<?xml version="1.0"?>
<!-- styles/StyleDeclarationTypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.styles.StyleManager;

     private var myDynStyle:CSSStyleDeclaration;

     private function initApp():void {
        myDynStyle = new CSSStyleDeclaration('myDynStyle');

        myDynStyle.setStyle('color', 'blue');
        myDynStyle.setStyle('fontFamily', 'georgia');
        myDynStyle.setStyle('themeColor', 'green');
        myDynStyle.setStyle('fontSize', 24);

        // Apply the new style to all Buttons. By using a type
```

```
        // selector, this CSSStyleDeclaration object will replace
        // all style properties, causing potentially unwanted
        // results.
        StyleManager.setStyleDeclaration("Button",myDynStyle, true);
    }
  ]]></mx:Script>

  <mx:Button id="myButton" label="Click Me"/>

</mx:Application>
```

タイプセレクタに新しい CSSStyleDeclaration を設定すると、既存のタイプセレクタ全体が新しいセレクタに置き換えられます。新しい CSSStyleDeclaration で明示的に値が定義されていないスタイルプロパティは、すべて null に設定されます。これによって、"defaults.css" ファイルまたは既に適用済みのスタイルシートで定義されているスキンやマージンなどのプロパティが削除されることがあります。

すべてのスタイルプロパティが無効にならないようにするには、クラスセレクタを使用して新しい CSSStyleDeclaration オブジェクトを適用します。既存のタイプセレクタのプロパティがクラスセレクタのスタイルプロパティに置き換えられることはないので、次の例に示すように、コンポーネントのデフォルト設定が維持されます。

```
<?xml version="1.0"?>
<!-- styles/StyleDeclarationClassSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.styles.StyleManager;

     private var myDynStyle:CSSStyleDeclaration;

     private function initApp():void {
        myDynStyle = new CSSStyleDeclaration('myDynStyle');

        myDynStyle.setStyle('color', 'blue');
        myDynStyle.setStyle('fontFamily', 'georgia');
        myDynStyle.setStyle('themeColor', 'green');
        myDynStyle.setStyle('fontSize', 24);

        // Apply the new style using a class selector.
        // This maintains the values of the existing style
        // properties that are not over-ridden by the new
        // CSSStyleDeclaration.
        StyleManager.setStyleDeclaration(".myButtonStyle",
            myDynStyle, true);

        // You can also apply the new style by setting the
        // value of the  styleName property in ActionScript.
        myOtherButton.styleName=myDynStyle;
```

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

  <mx:Button id="myButton"
        label="Click Me"
        styleName="myButtonStyle"
  />

  <mx:Button id="myOtherButton" label="Click Me"/>

</mx:Application>
```

CSSStyleDeclaration オブジェクトを削除するには、次の例に示すように、StyleManager の clearStyleDeclaration() メソッドを使用します。

```
<?xml version="1.0"?>
<!-- styles/ClearStyleDeclarationExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.styles.StyleManager;

     private var myDynStyle:CSSStyleDeclaration;

     private function initApp():void {
        myDynStyle = new CSSStyleDeclaration('myDynStyle');

        myDynStyle.setStyle('color', 'blue');
        myDynStyle.setStyle('fontFamily', 'georgia');
        myDynStyle.setStyle('themeColor', 'green');
        myDynStyle.setStyle('fontSize', 24);

        StyleManager.setStyleDeclaration(".myButtonStyle",
            myDynStyle, true);

     }

     private function resetStyles():void {
        StyleManager.clearStyleDeclaration(".myButtonStyle",
            true);
     }

  ]]></mx:Script>

  <mx:Button id="myButton"
        label="Click Me"
        styleName="myButtonStyle"
        click="resetStyles()"
  />

</mx:Application>
```

clearStyleDeclaration() メソッドを使用すると、指定したセレクタのスタイルのみが削除されます。コンポーネントにクラスセレクタを適用し、そのコンポーネントのクラスセレクタに対してこのメソッドを呼び出すと、このコンポーネントのタイプセレクタのスタイルは削除されずに残ります。

setStyleDeclaration() メソッドおよび clearStyleDeclaration() メソッドは多くの計算処理を行います。新しいスタイルを Adobe Flash Player がすぐに適用またはクリアしないようにするには、update パラメータを false に設定します。

次の例では、さまざまなターゲットに新しいクラスセレクタを設定しますが、最後のスタイル宣言の適用まで更新をトリガしません。

```
<?xml version="1.0"?>
<!-- styles/UpdateParameter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.styles.StyleManager;

     private var myButtonStyle:CSSStyleDeclaration =
        new CSSStyleDeclaration('myButtonStyle');
     private var myLabelStyle:CSSStyleDeclaration =
        new CSSStyleDeclaration('myLabelStyle');
     private var myTextAreaStyle:CSSStyleDeclaration =
        new CSSStyleDeclaration('myTextAreaStyle');

     private function initApp():void {
        myButtonStyle.setStyle('color', 'blue');
        myLabelStyle.setStyle('color', 'blue');
        myTextAreaStyle.setStyle('color', 'blue');
     }

     private function applyStyles():void {
        StyleManager.setStyleDeclaration("Button", myButtonStyle, false);
        StyleManager.setStyleDeclaration("Label", myLabelStyle, false);
        StyleManager.setStyleDeclaration("TextArea", myTextAreaStyle, true);
     }
  ]]></mx:Script>

  <mx:Button id="myButton" label="Click Me" click="applyStyles()"/>
  <mx:Label id="myLabel" text="This is a label"/>
  <mx:TextArea id="myTextArea" text="This is a TextArea"/>

</mx:Application>
```

update パラメータに false を渡した場合、Flash Player はセレクタを保存するだけでスタイルを適用しません。update パラメータとして true を渡すと、Flash Player はアプリケーションに含まれる各ビジュアルコンポーネントのスタイルを再計算します。

setStyle() メソッドと getStyle() メソッドの使用

他のプロパティと異なり、コンポーネントのスタイルプロパティを直接 get または set することはできません。その代わり、ActionScript メソッドである `getStyle()` および `setStyle()` を使用して、実行時にスタイルプロパティを設定します。`getStyle()` および `setStyle()` メソッドを使用すると、オブジェクトのインスタンスまたはスタイルシートのスタイルプロパティにアクセスできます。

これらのメソッドは、すべての Flex コンポーネントで公開されます。オブジェクトをインスタンス化し、初めてスタイルを設定する際は、コンピュータへの負荷が大きい `setStyle()` メソッドの使用より、スタイルシートの適用を試行してください。このメソッドは、実行時にオブジェクトのスタイルを変更する場合にのみ使用してください。`setStyle()` メソッドを使用する場合のパフォーマンス向上については、686ページの「setStyle() メソッドによるパフォーマンスの向上」を参照してください。

スタイルの設定

`getStyle()` メソッドのシグネチャを次に示します。

return_type componentInstance.getStyle(*property_name*)

return_type は、アクセスするスタイルによって異なります。スタイルは String、Number、Boolean などの型で、スキンの場合は Class です。property_name は、スタイルプロパティの名前 (`fontSize`、`borderStyle` など) を表す String です。

`setStyle()` メソッドのシグネチャを次に示します。

componentInstance.setStyle(*property_name*, *property_value*)

property_value は、指定したプロパティの新しい値を設定します。プロパティの有効な値については、『Adobe Flex 2 リファレンスガイド』を参照してください。

次の例では、`getStyle()` および `setStyle()` メソッドを使用して、ボタンの `fontSize` スタイルを変更し、新しいサイズを TextInput 内に表示します。

```
<?xml version="1.0"?>
<!-- styles/SetSizeGetSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        fontSize: 10pt;
        color: Blue;
     }
     .myClass {
        fontFamily: Arial, Helvetica, "_sans";
        color: Red;
        fontSize: 22;
        fontWeight: bold;
     }
  </mx:Style>
  <mx:Script><![CDATA[
```

```
    public function showStyles():void {
       lb1.text = String(ip1.getStyle("fontSize"));
    }
    public function setNewStyles(newSize:Number):void {
       ip1.setStyle("fontSize",newSize);
    }
  ]]></mx:Script>
  <mx:VBox id="vb">
    <mx:TextInput styleName="myClass" text="My attrs" id="ip1"
       width="400"/>
    <mx:Label id="lb1" text="" width="400"/>
    <mx:Button label="Get Style" click="showStyles();"/>
    <mx:Button label="Set Style" click="setNewStyles(Number(ip2.text));"/>
    <mx:TextInput text="" id="ip2" width="50"/>
  </mx:VBox>
</mx:Application>
```

getStyle() メソッドを使用すると、スタイルプロパティの設定方法に関係なくスタイルプロパティにアクセスできます。スタイルプロパティを <mx:Style> タグ内ではなく、インラインのタグプロパティとして定義した場合、このスタイルを get および set できます。<mx:Style> タグ内や外部スタイルシート内など、なんらかの方法で適用されたスタイルプロパティをオーバーライドできます。

次の例では、スタイルプロパティをインラインで設定した後、getStyle() メソッドを使用してそのプロパティを読み取ります。

```
<?xml version="1.0"?>
<!-- styles/GetStyleInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
    private function readStyle():void {
       myLabel.text = "Style: " + myLabel.getStyle("fontStyle");
    }
  ]]></mx:Script>
  <mx:VBox width="500"
    height="200">
    <mx:Button id="b1" click="readStyle()" label="Get Style"/>
    <mx:Label fontStyle="italic" id="myLabel"/>
  </mx:VBox>
</mx:Application>
```

setStyle() メソッドでカラーのスタイルプロパティを設定する場合、次の例に示すように、16 進形式または VGA カラー名のいずれかを使用できます。

```
<?xml version="1.0"?>
<!-- styles/ColorFormatStyleManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
    public function initApp():void {
       StyleManager.getStyleDeclaration("Button").
           setStyle("themeColor",0x6666CC);
```

```
        StyleManager.getStyleDeclaration("Button").
            setStyle("color","Blue");
    }

    public function changeStyles(e:Event):void {
        e.currentTarget.setStyle("themeColor", 0xFF0099);
        e.currentTarget.setStyle("color", "Red");
    }
  ]]></mx:Script>
  <mx:Button id="myButton"
        label="Click Here"
        click="changeStyles(event)"
  />
</mx:Application>
```

getStyle() メソッドを使用して color のスタイルプロパティを取得すると、スタイルプロパティの 16 進数値を表す整数が返されます。この値を 16 進形式に変換するには、カラー変数の toString() メソッドを使って、その基数 (または基底) に 16 を渡します。

```
<?xml version="1.0"?>
<!-- styles/ColorFormatNumericValue.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">

  <mx:Style>
    Button {
       color: #66CCFF;
    }
  </mx:Style>

  <mx:Script><![CDATA[
    [Bindable]
    private var n:Number;

    private function initApp():void {
       n = myButton.getStyle("color");
    }

    public function changeStyles(e:Event):void {
       if (myButton.getStyle("color").toString(16) == "ff0000") {
          myButton.setStyle("color", 0x66CCFF);
       } else {
          myButton.setStyle("color", "Red");
       }

       n = myButton.getStyle("color"); //// Returns 16711680
    }
  ]]></mx:Script>

  <mx:Button id="myButton" label="Click Here" click="changeStyles(event)"/>
```

```
  <mx:Label id="myLabel" text="0x{n.toString(16).toUpperCase()}"/>

</mx:Application>
```

setStyle() メソッドを使用して既存のスタイルを変更しても (たとえば、Button コントロールの color プロパティをデフォルトの 0x000000 以外に設定する場合など)、元のスタイル設定は上書きされません。元の設定に戻すには、スタイルプロパティを null に設定します。次の例ではこの手法を使用して、Button コントロールのカラーを青とデフォルト設定の間で切り替えます。

```
<?xml version="1.0"?>
<!-- styles/ResetStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        public function toggleStyle():void {
            if (cb1.selected == true) {
                b1.setStyle("color","blue");
                b1.setStyle("fontSize", 8);
            } else {
                b1.setStyle("color", null);
                b1.setStyle("fontSize", null);
            }
        }
        ]]>
    </mx:Script>

    <mx:Style>

        Button {
            color: red;
            fontSize: 25;
        }

    </mx:Style>

    <mx:Button id="b1" label="Click Me"/>

    <mx:CheckBox id="cb1"
        label="Set Style/Unset Style"
        click="toggleStyle()"
        selected="false"
    />
</mx:Application>
```

setStyle() メソッドによるパフォーマンスの向上

実行時のカスケーディングスタイルは非常に有用ですが、むやみに使用せずに、正しいコンテキストで使用する必要があります。オブジェクトのインスタンスに対して動的にスタイルを設定するということは、UIComponent の setStyle() メソッドにアクセスすることを意味します。setStyle() メソッドの呼び出しを使用すると、新しくスタイルを設定したオブジェクトのすべての子に対して、再度スタイル参照を実行するように通知する必要があります。したがって、このメソッドの呼び出しは、Flex フレームワークにおいて最もリソースを要する呼び出しの 1 つです。通知を必要とする子のツリーは、非常に大きくなる場合があります。

パフォーマンスに影響する失敗として頻繁にあるのは、setStyle() メソッドを使い過ぎることや、不必要な場合にも使用することです。一般的に、setStyle() メソッドを使用する必要があるのは、既存のオブジェクトのスタイルを変更する場合のみです。オブジェクトのスタイルを初めて設定する際には使用しないでください。代わりに <mx:Style> ブロック内で外部 CSS スタイルシートを通して、あるいはグローバルスタイルとしてスタイルを設定します。プログラムの実行中 (アプリケーション、ナビゲータコンテナ内の新しいビュー、動的に作成されたコンポーネントなど) にスタイルが変更される予定がない場合には、正しいスタイル情報を使用してオブジェクトを初期化することが重要です。

アプリケーションによっては、アプリケーションまたはオブジェクトのインスタンス化の際に setStyle() メソッドを呼び出す必要があります。その場合、初期化の早い段階で setStyle() メソッドを呼び出します。つまり creationComplete などのイベントではなく、コンポーネントまたはアプリケーションの preinitialize イベントからスタイルを設定するということです。初期化の際にできるだけ早くスタイルを設定することで、不要なスタイル通知や参照を避けます。コンポーネントの起動ライフサイクルの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 6 章の「起動時のパフォーマンスの向上」を参照してください。

インラインスタイルの使用

スタイルプロパティは、MXML タグでコンポーネントのプロパティとして設定できます。インラインスタイル定義は、他のスタイル定義よりも優先されます。次の例では、Button コンポーネントにタイプセレクタが定義されていますが、インライン定義を使用して color をオーバーライドします。

```
<?xml version="1.0"?>
<!-- styles/InlineOverride.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        fontSize: 10pt;
        fontStyle: italic;
        color: #FF0000;
     }
  </mx:Style>
```

```
  <mx:Button label="Button Type Selector Color"/>
  <mx:Button color="0x999942" label="Inline Color"/>
</mx:Application>
```

インラインでスタイルプロパティを設定する場合は、CSS の命名シンタックスではなく、ActionScript スタイルプロパティの命名シンタックスに従う必要があります。たとえば、Button コントロールの fontSize プロパティは、<mx:Style> 宣言では font-size または fontSize に設定できますが、タグ定義では fontSize に設定する必要があります。

```
<?xml version="1.0"?>
<!-- styles/CamelCase.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myFontStyle {
        fontSize: 15;
     }
     .myOtherFontStyle {
        font-size: 15;
     }
  </mx:Style>
  <mx:Button id="myButton" styleName="myFontStyle" label="Click Here"/>
  <mx:Button id="myButton2" styleName="myOtherFontStyle" label="Click Here"/>
  <mx:Button id="myButton3" fontSize="15" label="Click Here"/>
</mx:Application>
```

詳細については、660 ページの「プロパティ名とセレクタ名について」を参照してください。

インラインでカラースタイルプロパティを設定する場合、次の例を示すように、16 進数形式または VGA カラー名を使用できます。

```
<?xml version="1.0"?>
<!-- styles/ColorFormatInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Button id="myButton" themeColor="0x6666CC" color="Blue" label="Click
Here"/>

</mx:Application>
```

clearStyle() メソッドを使用すると、インラインスタイル定義を削除できます。

変数を [Bindable] としてタグ付けしてあれば、インラインスタイルプロパティをその変数にバインドできます。次の例では、HBox コントロールの backgroundColor プロパティの値を、colorValue 変数の値にバインドします。

```
<?xml version="1.0"?>
<!-- styles/PropertyBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     [Bindable]
     public var colorValue:int = 0x333999;
```

```
      public function changeHBoxStyle():void {
         colorValue = cp.selectedColor;
      }
   ]]></mx:Script>
   <mx:HBox width="100" height="100" backgroundColor="{colorValue}"/>
   <mx:ColorPicker id="cp" showTextField="true" change="changeHBoxStyle()"
selectedColor="0x333999"/>
</mx:Application>
```

スタイルプロパティのバインドでは、多くの計算処理が必要になることがあります。プロパティの適用にこのメソッドを使用するのは、それ以外に手段がない場合に限るようにします。

実行時のスタイルシートのロード

実行時にスタイルシートをロードするには、StyleManager を使用します。これらのスタイルシートは、Flex アプリケーションの実行中に動的にロードされる SWF ファイルの形式を取ります。

実行時にスタイルシートをロードすると、イメージ (グラフィカルスキン用)、フォント、タイプセレクタとクラスセレクタ、およびプログラムスキンを、コンパイル時に埋め込まずに Flex アプリケーションにロードできます。この方法では、スキンとフォントを、メインアプリケーションとは分離した別個の SWF ファイルに分割することができます。その結果、アプリケーションの SWF ファイルのサイズは縮小し、初期ダウンロードの時間は短縮されます。ただし、実行時スタイルシートを初めて使用するときは、必要な CSS ベースの SWF ファイルを Flex にダウンロードする必要があるため、スタイルおよびスキンの適用に時間がかかります。

実行時にスタイルシートをロードするには、次の 3 つの手順を実行します。

1. アプリケーションの CSS ファイルを作成します。
2. mxmlc コンパイラを使用して CSS ファイルを SWF ファイルにコンパイルします。
3. Flex アプリケーションで `StyleManager.loadStyleDeclarations()` メソッドを呼び出します。このメソッドは、CSS ベースの SWF ファイルをアプリケーションにロードします。このメソッドが実行されると、新しい CSSStyleDeclarations が StyleManager にロードされます。

同一のスタイルを定義する複数のスタイルシートをロードできます。スタイルを設定すると、それ以降のスタイルシートが、共通のセレクタを持つ以前のスタイルシートを上書きすることがあります。ただし、実行時スタイルシートでロードされたスタイルが、コンパイル時のスタイルに完全に取って代わるわけではありません。実行時スタイルシートがアンロードされるまでオーバーライドするだけです。アンロードが実行されると、コンパイル時のスタイル設定に戻ります。コンパイル時のスタイル設定には、コンパイル時にロードされたデフォルトのスタイルシート、`theme` コンパイルオプションを使用してロードされたテーマファイル、MXML ファイル内で `<mx:Style>` ブロックを使用して設定されたスタイルなどが含まれます。

実行時スタイルシートの作成

実行時にスタイルシートをロードするには、最初に、SWF ファイルにコンパイルされたスタイルシートを作成する必要があります。実行時のスタイルシートは、他のスタイルシートと同様です。次の例に示すように、基本的なスタイルプロパティを設定するだけの簡単なスタイルシートにすることもできます。

```
/* styles/runtime/assets/BasicStyles.css */
Button {
    fontSize:    24;
    color: #FF9933;
}

Label {
    fontSize:    24;
    color: #FF9933;
}
```

あるいは、次の例に示すように、プログラムスキンやグラフィックスキン、フォントなどのスタイルプロパティが埋め込まれ、タイプセレクタやクラスセレクタを使用する複雑なスタイルシートにすることもできます。

```
/* styles/runtime/assets/ComplexStyles.css */
Application {
    backgroundImage : "assets/greenBackground.gif";
    theme-color: #9DBAEB;
}
Button {
    fontFamily: Tahoma;
    color: #000000;
    fontSize: 11;
    fontWeight: normal;
    text-roll-over-color: #000000;
    upSkin: Embed(source="../../../assets/orb_up_skin.gif");
    overSkin: Embed(source="../../../assets/orb_over_skin.gif");
    downSkin: Embed(source="../../../assets/orb_down_skin.gif");
}

.noMargins {
    margin-right: 0;
    margin-left: 0;
    margin-top: 0;
    margin-bottom: 0;
    horizontal-gap: 0;
    vertical-gap: 0;
}
```

実行時に、コンパイルされていない CSS ファイルを Flex アプリケーションにロードすることはできません。CSS ファイルをロードする前に、必ず SWF ファイルにコンパイルしてください。

CSS ベースの SWF ファイルのコンパイル

実行時にスタイルシートをロードできるようにするには、事前にスタイルシートを SWF ファイルにコンパイルする必要があります。

CSS ファイルを SWF ファイルにコンパイルするには、mxmlc コマンドラインコンパイラを使用します。デフォルトでは、コンパイルの結果、CSS ファイルと同じ名前で拡張子が .swf の SWF ファイルが生成されます。次の例では、"BasicStyles.swf" ファイルが生成されます。

```
> mxmlc /skins/runtime/BasicStyles.css
```

SWF ファイルにコンパイルするスタイルシートには、.css のファイル名拡張子が付いている必要があります。

Flex アプリケーションをコンパイルするときに、コンパイルは、そのアプリケーションが使用する CSS ベースの SWF ファイルに対してコンパイル時のリンクチェックを実行しません。つまり、メインアプリケーションをコンパイルする前に SWF ファイルを作成する必要はありません。

実行時のスタイルシートのロード

実行時に CSS ベースの SWF ファイルをロードするには、StyleManager の `loadStyleDeclarations()` メソッドを使用します。このメソッドを使用するには、mx.core.StyleManager クラスを読み込む必要があります。

次の例では、ボタンをクリックしたときに、スタイルシートをロードします。

```
<?xml version="1.0"?>
<!-- styles/runtime/BasicApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
        import mx.styles.StyleManager;

        public function applyRuntimeStyleSheet():void {
            StyleManager.loadStyleDeclarations("assets/BasicStyles.swf")
        }
        ]]>
    </mx:Script>

    <mx:Label text="Click the button to load a new CSS-based SWF file"/>
    <mx:Button id="b1" label="Click Me" click="applyRuntimeStyleSheet()"/>

</mx:Application>
```

`loadStyleDeclarations()` メソッドの第 1 パラメータは、ロードするスタイルシートの場所を示します。ローカルまたはリモートの場所を指定できます。

次の例は、ローカルの SWF ファイルとリモートの SWF ファイルのロードを示しています。

```
// Load a locally-accessible style sheet:
StyleManager.loadStyleDeclarations("assets/LocalStyles.swf");
// Load a remote style sheet:
StyleManager.loadStyleDeclarations("http://www.domain2.com/styles/
  RemoteStyles.swf", true, true);
```

スタイルシートがローカル (ロードするアプリケーションと同じドメインから見て) の場合は、追加の設定なしでスタイルシートをロードできます。スタイルシートの場所がリモートの場合 (つまり、ロードするアプリケーションのドメインと完全に一致するドメイン以外)、loadStyleDeclarations() メソッドの第 3 パラメータ trustContent を true に設定する必要があります。リモートのスタイルシートをロードする場合、ロードするアプリケーションに SWF ファイルのロードを許可する "crossdomain.xml" ファイルは不要です。

また、リモートのスタイルシートを使用するには、ネットワークアクセスが可能なロードアプリケーション (use-network コンパイラプロパティがデフォルトの true に設定されている) をコンパイルする必要があります。アプリケーションをローカルファイルシステムでコンパイルして実行すると、リモートアクセスが可能な SWF ファイルをロードできなくなることがあります。

loadStyleDeclarations() メソッドは IEventDispatcher クラスのインスタンスを返します。このオブジェクトを使用すると、スタイルシートのロードが成功したかどうかに応じてイベントをトリガできます。ロードプロセスの StyleEvent.PROGRESS、StyleEvent.COMPLETE、StyleEvent.ERROR の各イベントにアクセスできます。

次のアプリケーションは、スタイルシートのロードが終了したときにメソッドを呼び出します。

```
<?xml version="1.0"?>
<!-- styles/runtime/StylesEventApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">

    <mx:Script>
        <![CDATA[
        import mx.styles.StyleManager;
        import mx.events.StyleEvent;

        public function init():void {
            var myEvent:IEventDispatcher =
                StyleManager.loadStyleDeclarations("assets/ACBStyles.swf");
            myEvent.addEventListener(StyleEvent.COMPLETE, getImage);
        }

        private function getImage(event:StyleEvent):void {
            map1.source = acb.getStyle("dottedMap");
        }
        ]]>
    </mx:Script>
```

```
    <mx:ApplicationControlBar id="acb" width="100%" styleName="homeMap">
        <mx:Image id="map1"/>
        <mx:Button label="Submit"/>
    </mx:ApplicationControlBar>

</mx:Application>
```

loadStyleDeclarations() メソッドの第 2 パラメータは、update です。update パラメータを true に設定すると、即時にスタイルが強制的に適用されます。アプリケーションのスタイルを強制的に即時更新しないようにするには、false に設定します。スタイルは、次回、このメソッドまたは、update プロパティが true 設定されている unloadStyleDeclarations() メソッドを呼び出したときに更新されます。

update パラメータが true 設定されている loadStyleDeclarations() メソッドを呼び出すたびに、Adobe Flash Player ではすべてのスタイルが表示リストに再適用され、これが原因でパフォーマンスが低下することがあります。同時に複数の CSS ベースの SWF ファイルをロードする場合は、最後の呼び出しを除くこのメソッドへのすべての呼び出しで、update パラメータを false に設定する必要があります。そのように設定すると、Flash Player では、新しいスタイル SWF ファイルごとに適用するのではなく、新しいすべての SWF ファイルに対して一度にスタイルを適用します。

次の例では、3 種類のスタイルの SWF ファイルをロードしますが、3 番目のファイルがロードされるまでスタイルは適用されません。

```
<?xml version="1.0"?>
<!-- styles/runtime/DelayUpdates.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">

    <mx:Script>
        <![CDATA[
        import mx.styles.StyleManager;
        import mx.events.StyleEvent;

        public function init():void {
            StyleManager.loadStyleDeclarations(
                "assets/ButtonStyles.swf", false);
            var myEvent:IEventDispatcher =
                StyleManager.loadStyleDeclarations(
                "assets/LabelStyles.swf", false);
            myEvent.addEventListener(StyleEvent.COMPLETE, doUpdate);
        }

        public function doUpdate(event:StyleEvent):void {
            StyleManager.loadStyleDeclarations(
                "assets/ACBStyles.swf", true);
        }
        ]]>
    </mx:Script>
```

```
    <mx:Label text="This is a label"/>

    <mx:ApplicationControlBar id="acb" width="100%">
        <mx:Button label="Submit"/>
    </mx:ApplicationControlBar>

</mx:Application>
```

実行時のスタイルシートのアンロード

実行時にロードしたスタイルシートをアンロードすることができます。そのためには、StyleManager の unloadStyleDeclarations() メソッドを使用します。このメソッドの結果、指定したスタイルの SWF ファイルで設定されたすべてのスタイルプロパティが、デフォルト値に戻されます。

次の例では、チェックボックスを切り替えたときに、スタイル SWF をロードまたはアンロードします。

```
<?xml version="1.0"?>
<!-- styles/runtime/UnloadStyleSheets.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import mx.styles.StyleManager;

        public function toggleStyleSheet():void {
            if (cb1.selected == true) {
                StyleManager.loadStyleDeclarations(
                    "assets/ButtonStyles.swf", true, false)
                StyleManager.loadStyleDeclarations(
                    "assets/LabelStyles.swf", true, false)
            } else {
                StyleManager.unloadStyleDeclarations(
                    "assets/ButtonStyles.swf", true);
                StyleManager.unloadStyleDeclarations(
                    "assets/LabelStyles.swf", true);
            }
        }
        ]]>
    </mx:Script>

    <mx:Button id="b1" label="Click Me"/>

    <mx:Label id="l1" text="Click the button"/>

    <mx:CheckBox id="cb1"
        label="Load style sheet"
        click="toggleStyleSheet()"
        selected="false"
    />
</mx:Application>
```

カスタムコンポーネントでの実行時スタイルシートの使用

実行時スタイルシートは、カスタムコンポーネントでも使用できます。そのためには、通常、コンポーネントが初期化された後に loadStyleDeclaration() メソッドを呼び出します。スタイルシートにクラスセレクタがある場合は、styleName プロパティを設定することにより適用します。

次の例では、specialStyle という名前のクラスセレクタにあるスタイルプロパティとスキンを定義します。

```
/* styles/runtime/assets/CustomComponentStyles.css */
.specialStyle {
    fontSize:    24;
    color: #FF9933;
    upSkin: Embed(source="SubmitButtonSkins.swf", symbol="MyUpSkin");
    overSkin: Embed(source="SubmitButtonSkins.swf", symbol="MyOverSkin");
    downSkin: Embed(source="SubmitButtonSkins.swf", symbol="MyDownSkin");
}
```

次の例では、カスタムコンポーネントがこのスタイルシートをロードし、初期化中に specialStyle クラスセレクタをそれ自体に適用します。

```
// styles/runtime/MyButton.as -->
package {

    import mx.controls.Button;
    import mx.events.*;
    import mx.styles.StyleManager;

    public class MyButton extends Button {

        public function MyButton() {
            addEventListener(FlexEvent.INITIALIZE,
                initializeHandler);
        }

         // Gets called when the component has been initialized
        private function initializeHandler(event:FlexEvent):void {
            StyleManager.loadStyleDeclarations(
                "assets/CustomComponentStyles.swf");
            this.styleName = "specialStyle";
        }
    }
}
```

実行時スタイルシートとしてのテーマ SWC ファイルの使用

既存のテーマ SWC ファイルがある場合は、そのファイルを実行時スタイルシートとして使用できます。そのためには、テーマ SWC ファイルから CSS ファイルを抽出する必要があります。その後に、残りの SWC ファイルを 1 つのライブラリとして渡すことにより、スタイル SWF ファイルをコンパイルします。

次の手順は、コマンドラインを使用してこのプロセスを実行する方法を示しています。

1. PKZip などのアーカイブユーティリティを使用して、SWC ファイルから CSS ファイルを抽出します。

   ```
   $ unzip haloclassic.swc defaults.css
   ```

2. (オプション) CSS ファイルの名前を、意味のある名前に変更します。この名前がスタイル SWF ファイルの名前になります。次の例では、ファイル名を "defaults.css" から "haloclassic.css" に変更します。

   ```
   $ mv defaults.css haloclassic.css
   ```

3. スタイル SWF ファイルをコンパイルします。テーマ SWC ファイルをスタイル SWF ファイルに追加するには、次の例に示すように、`include-libraries` オプションを使用します。

   ```
   $ mxmlc -include-libraries=haloclassic.swc haloclassic.css
   ```

テーマ SWC ファイル内に複数の CSS ファイルがある場合は、スタイル SWF ファイルをコンパイルする前に CSS ファイルをすべて抽出する必要があります。

Flex でのフィルタの使用

Adobe Flash フィルタを使用すると、Label や Text などの Flex コンポーネントにスタイルに似た効果を適用できます。UIComponent から派生した可視 Flex コンポーネントであれば、フィルタの適用が可能です。スタイルシートや `setStyle()` メソッドではフィルタを適用できないので、フィルタはスタイルではありません。しかし、フィルタを適用して得られた結果はスタイルの一種であると見なすことが普通です。

フィルタは flash.filters.* パッケージに収められ、DropShadowFilter、GlowFilter、BlurFilter などのクラスがあります。MXML でフィルタをコンポーネントに適用するには、flash.filters パッケージをローカル名前空間に追加します。次に、Flex コンポーネントの `filters` プロパティの値をフィルタクラスに設定します。`filters` プロパティは、DisplayObject クラスから継承されるプロパティです。

次の例では、拡張 MXML シンタックスとインラインシンタックスを使用して、Label コントロールにドロップシャドウを適用します。

```
<?xml version="1.0"?>
<!-- styles/ApplyFilterInline.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:flash="flash.filters.*">
  <!-- Apply filter using MXML syntax to set properties. -->
  <mx:Label text="DropShadowFilter" fontSize="20">
    <mx:filters>
     <flash:DropShadowFilter distance="10" angle="45"/>
    </mx:filters>
  </mx:Label>

  <!-- Apply filter and set properties inline. -->
  <mx:Label text="DropShadowFilter (inline)" fontSize="20"
     filters="{[new DropShadowFilter(10, 45)]}"/>
</mx:Application>
```

ActionScript でフィルタを適用できます。そのためには、flash.filters.* パッケージをインポートし、新しいフィルタを Flex コントロールのフィルタ配列に追加します次の例では、ユーザーがボタンをクリックしたときに Label コントロールにホワイトシャドウを適用します。

```
<?xml version="1.0"?>
<!-- styles/ApplyFilterAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
     import flash.filters.*;

     public function addFilter():void {
        // First four properties are distance, angle, color, and alpha.
        var f:DropShadowFilter = new DropShadowFilter(5,30,0xFFFFFF,.8);
        var myFilters:Array = new Array();
        myFilters.push(f);
        label1.filters = myFilters;
     }
  </mx:Script>
  <mx:Label id="label1" text="ActionScript-applied filter"/>
  <mx:Button id="b1" label="Add Filter" click="addFilter()"/>
</mx:Application>
```

フィルタプロパティを他の値にバインドすることはできません。

フィルタを変更した場合は、その効果が反映されるように、コンポーネントにフィルタを割り当て直す必要があります。次の例では、ボタンをクリックしたときにフィルタのカラーが変化します。

```
<?xml version="1.0"?>
<!-- styles/FilterChange.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="createFilters()">
  <mx:Script><![CDATA[
     import flash.filters.*;

     private var myBlurFilter:BlurFilter;
     private var myGlowFilter:GlowFilter;
     private var myBevelFilter:BevelFilter;
     private var myDropShadowFilter:DropShadowFilter;
```

```
    private var color:Number = 0xFF33FF;

    public function createFilters():void {

       myBlurFilter = new BlurFilter(4, 4, 1);

       myGlowFilter = new GlowFilter(color, .8, 6, 6, 2, 1,
           false, false);

       myDropShadowFilter = new DropShadowFilter(15, 45,
           color, 0.8, 8, 8, 0.65, 1, false, false);

       myBevelFilter = new BevelFilter(5, 45, color, 0.8,
           0x333333, 0.8, 5, 5, 1, BitmapFilterQuality.HIGH,
           BitmapFilterType.INNER, false);

       applyFilters();
    }

    public function applyFilters():void {
       rte1.filters = [myGlowFilter];
       b1.filters = [myDropShadowFilter];
       dc1.filters = [myBevelFilter];
       hs1.filters = [myBlurFilter];
    }

    public function changeFilters():void {
       color = 0x336633;
       createFilters();
    }
  ]]></mx:Script>

  <mx:RichTextEditor id="rte1"/>

  <mx:DateChooser id="dc1"/>

  <mx:HSlider id="hs1"/>

  <mx:Button id="b1" label="Click me" click="changeFilters()"/>


</mx:Application>
```

テーマについて

テーマでは、Flex アプリケーションの外観と雰囲気を定義します。テーマには、アプリケーションのカラースキーマや共通フォントなどの単純なものから、アプリケーションで使用する全コンポーネントの完全な再スキンのような複雑なものまで定義できます。

テーマは通常、SWC ファイルの形をとります。ただし、テーマを CSS ファイルにすることや、SWF ファイルのシンボルのような埋め込みグラフィカルリソースとすることも可能です。テーマ SWC ファイルもスタイル SWF ファイルにコンパイルし、実行時にロードすることができます。詳細については、695 ページの「実行時スタイルシートとしてのテーマ SWC ファイルの使用」を参照してください。

Flex アプリケーションにこの SWC ファイルの内容を適用するには、698 ページの「テーマの使用」の手順に従ってください。独自にテーマを作成するには、701 ページの「テーマ SWC ファイルの作成」の手順に従ってください。テーマを新規に作成しなくても、Flex アプリケーションのテーマを編集できます。このような編集には、`themeColor` プロパティを使用します。詳細については、778 ページの「テーマの作成」を参照してください。

デフォルトのテーマである Halo は、SWC ファイルにあるグラフィカルスキンとプログラムスキンの組み合わせです。この組み合わせは、"framework.swc" ファイルの中の "defaults.css" ファイルによって定義されます。"framework.swc" ファイルでは、Flex コンポーネントのプロパティが数多く設定されています。場合によっては、mx.skins.halo パッケージのクラスが使用されることもあります。Flex には、実際のアプリケーションに適用できる事前定義のテーマがいくつか用意されています。詳細については、700 ページの「付属のテーマファイルについて」を参照してください。

テーマの使用

テーマは通常テーマ SWC ファイルの形をとります。これらの SWC ファイルには、スタイルシートやスキニングアセットが含まれています。プログラムスキンや、SWF ファイル、GIF ファイル、JPEG ファイルのようなグラフィカルアセットは、テーマの SWC ファイルの中でアセットとして使用します。単独の CSS ファイルのみでテーマを構成することもできます。

テーマを、緩い結合で結び付けた各種ファイルの集合とせず、SWC ファイルとしてパッケージ化すると、次の利点が得られます。

- SWC ファイルなので、配布が容易です。
- SWC ファイルは、再コンパイルしない限り、改変してアプリケーションに適用し直すことができません。
- SWC ファイルはプリコンパイルされています。そのため、SWC ファイルに含まれていないスキンクラスのコンパイル時間と比べて、アプリケーションのコンパイル時間が短くなります。

テーマを Flex アプリケーションに適用するには、theme コンパイルオプションを使用して SWC ファイルまたは CSS ファイルを指定します。次の例では、BullyBuster テーマの SWC ファイルを使用するアプリケーションをコンパイルするために、mxmlc コマンドラインコンパイラを使用しています。

```
mxmlc -theme c:¥theme¥BullyBuster.swc c:¥myfiles¥flex2¥misc¥MainApp.mxml
```

"flex-config.xml" ファイルのオプションを使用してアプリケーションをコンパイルする場合は、次のようにテーマを指定します。

```
<compiler>
  <theme>
    <filename>c:¥theme¥BullyBuster.swc</filename>
  </theme>
</compiler>
```

テーマのリストに SWC ファイルを追加すると、コンパイラによって、その SWC ファイルに収められているクラスがアプリケーションの library-path に追加されます。さらに、その SWC ファイルにある CSS ファイルがアプリケーションに適用されます。しかし、この逆の操作は機能しません。library-path に SWC ファイルを追加しても、theme オプションを使用してその SWC ファイルを指定していなければ、コンパイラでは、その SWC ファイルにある CSS ファイルがアプリケーションに適用されません。

Flex コンパイラの使用方法の詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 9 章の「Flex コンパイラの使用」を参照してください。

アプリケーションに適用するテーマとして、複数のテーマファイルを指定できます。両方のファイルに重複するスタイルがなければ、それぞれのファイルにあるテーマがすべて適用されます。ただし、テーマファイルの適用順序は重要です。複数のテーマファイルにある同じスタイルプロパティを指定すると、リストの最後にあるテーマが使用されます。次の例では "Wooden.css" ファイルにあるスタイルプロパティが優先されますが、先頭の 2 つのファイルにのみ存在するプロパティはそのまま適用されます。

```
<compiler>
  <theme>
    <filename>../themes/Institutional.css</filename>
    <filename>../themes/Ice.css</filename>
    <filename>../themes/Wooden.css</filename>
  </theme>
</compiler>
```

付属のテーマファイルについて

Flex には、変更せずにそのまま使用できるテーマがいくつか付属しています。デフォルトの Halo テーマを除き、これらのテーマはすべて SDK の "flex_install_dir/frameworks/themes" ディレクトリにあります。Flex Data Services の場合、テーマは "Flex インストールディレクトリ /flex_sdk_2/frameworks/themes" ディレクトリにあります。Flex Builder の場合、テーマは "flex_install_dir/Flex SDK 2/frameworks/themes" ディレクトリにあります。

これらのテーマについて次の表で説明します。

テーマ	説明
Aeon Graphical	"AeonGraphical.css" ファイルおよび "AeonGraphical.swf" ファイルが収められています。これは、デフォルトの Halo テーマをグラフィカルにしたもので、"defaults.css" ファイル、および "framework.swc" ファイル内部のグラフィカルアセットによって定義されています。 グラフィカルアセットのソースである FLA ファイルは、"AeonGraphical Source" ディレクトリにあります。このファイルにあるグラフィカルアセットを変更するには、Flash IDE を使用します。Flex のテーマで使用する SWF ファイルは必ずエクスポートしておく必要があります。 アプリケーションをグラフィカルに再スキンするときは、このテーマから着手します。このテーマをそのまま使用すると作業が効率的になることもありますが、実行時スタイルを適用する機能のうち、利用できないものがあります。
Halo	Flex のデフォルトの外観と操作性を提供します。Halo テーマでは、"framework.swc" ファイルにある "defaults.css" ファイル、および mx.skins.halo パッケージにある各種クラスを使用します。 これはデフォルトのテーマです。 デフォルトのテーマアセットの詳細については、673 ページの「デフォルトのスタイルシートについて」を参照してください。
HaloClassic	Flex の旧バージョンの外観と操作性を提供します。このテーマは "haloclassic.swc" ファイルで構成されます。このファイルには、グラフィカルアセットを提供する CSS ファイルと SWF ファイルが収められています。
Ice	"Ice.css" ファイルが収められています。
Institutional	"Institutional.css" ファイルが収められています。
Smoke	"Smoke.css" ファイルおよび背景の JPEG ファイルが収められています。
Wooden	"Wooden.css" ファイルおよび背景の JPEG ファイルが収められています。

テーマ SWC ファイルの作成

テーマ SWC ファイルを構築するには、CSS ファイルを作成し、そのファイル、グラフィカルアセット、およびプログラムアセットを SWC ファイルに追加します。そのためには、compc ユーティリティの `include-file` オプションを使用します。

"framework.swc" ファイルにある "defaults.css" ファイルをカスタマイズして新しいテーマを作成することをお勧めします。以降のセクションでは、これらの手順を詳しく説明します。

テーマのスタイルシートの作成

テーマは、スタイルシートと、そのスタイルシートで使用するアセットで構成されることが普通です。これらのアセットは、グラフィックファイルまたはプログラムスキンクラスです。

テーマの CSS ファイルにグラフィックファイルを追加するには、`Embed` ステートメントを使用します。次の例では、Button クラスに新しいグラフィカルスキンを定義します。

```
Button {
  upSkin: Embed("upIcon.jpg");
  downSkin: Embed("downIcon.jpg");
  overSkin: Embed("overIcon.jpg");
}
```

`Embed` キーワードに指定する名前は、SWC ファイルをコンパイルするときにスキンアセットに使用する名前と同じにします。

テーマの CSS ファイルにプログラムスキンクラスを追加するには、次の例に示すように、`ClassReference` ステートメントを使用して目的のクラス名 (ファイル名ではありません) を指定します。

```
Button {
  upSkin:ClassReference('myskins.ButtonSkin');
  downSkin:ClassReference('myskins.ButtonSkin');
  overSkin:ClassReference('myskins.ButtonSkin');
}
```

前の例では、すべてのスキンが単一のプログラムスキンクラスファイルで定義されていました。プログラムスキンの作成の詳細については、751 ページの「プログラムスキン」を参照してください。

CSS ファイルには、任意の数のクラスセレクタおよびタイプセレクタを指定できます。テーマのスタイル定義でスキンを使用する必要はありません。次の例に示すように、スタイルシートの中でスタイルプロパティを設定するだけですみます。

```
ControlBar {
  borderStyle: "controlBar";
  paddingBottom: 10;
  paddingLeft: 10;
  paddingRight: 10;
  paddingTop: 10;
```

```
    verticalAlign: "middle";
}
```

ControlBar コンポーネントのこのタイプセレクタは、"framework.swc" ファイル内のデフォルトのスタイルシートである "defaults.css" ファイルから取得したものです。このファイルを使用してテーマの設計に着手することをお勧めします。"defaults.css" の詳細については、673 ページの「デフォルトのスタイルシートについて」を参照してください。

テーマ SWC ファイルのコンパイル

テーマ SWC ファイルをコンパイルするには、Flex コンポーネントコンパイラの include-file オプションおよび include-classes オプションを使用します。このコンパイラは、コンポーネントライブラリおよび RSL (Runtime Shared Library : ランタイム共有ライブラリ) の作成に使用するコンパイラと同じものです。

コンポーネントコンパイラは、compc コマンドラインユーティリティを使用して呼び出すか、Adobe Flex Builder でライブラリプロジェクトを作成する際に呼び出します。

include-file オプションを使用して、テーマ SWC ファイルに CSS ファイルとグラフィックファイルを追加します。この説明は、702 ページの「include-file オプションを使用したテーマ SWC ファイルのコンパイル」を参照してください。include-classes オプションを使用して、テーマ SWC ファイルにプログラムスキンクラスを追加します。この説明は、703 ページの「include-classes オプションを使用したテーマ SWC ファイルのコンパイル」を参照してください。

テーマ SWC ファイルのコンパイルに使用するコマンドを簡素化するには、設定ファイルを使用します。詳細については、703 ページの「設定ファイルを使用したテーマ SWC ファイルのコンパイル」を参照してください。

include-file オプションを使用したテーマ SWC ファイルのコンパイル

include-file オプションは、2 つのパラメータ、つまり名前とパスを受け取ります。name は、アセットを CSS ファイルで参照する場合と同じように参照するために使用する名前です。path は、アセットへのファイルシステムパスです。include-file を使用するときに、ソースパスにリソースを追加する必要はありません。次のコマンドラインの例では、テーマ SWC ファイルに "upIcon.jpg" アセットを追加します。

```
-include-file upIcon.jpg c:¥myfiles¥themes¥assets¥upIcon.jpg
```

テーマ SWC ファイルに追加するリソースとして、CSS ファイルも指定します。CSS ファイルの名前を指定するときは、.css 拡張子も含めて記述します。拡張子を記述しないと、目的のファイルがスタイルシートとして認識されないので、アプリケーションに適用されません。

コンポーネントコンパイラを使用して、スタイルシートやグラフィカルスキンなどのテーマのアセットを SWC ファイルにコンパイルします。次の例では、compc コマンドラインコンパイラを使用してテーマをコンパイルします。

```
compc -include-file mycss.css c:¥myfiles¥themes¥mycss.css
```

```
  -include-file upIcon.jpg c:¥myfiles¥themes¥assets¥upIcon.jpg
  -include-file downIcon.jpg c:¥myfiles¥themes¥assets¥downIcon.jpg
  -include-file overIcon.jpg c:¥myfiles¥themes¥assets¥overIcon.jpg
  -o c:¥myfiles¥themes¥MyTheme.swc
```

include-file オプションにリストの形式で複数のアセットを渡すことはできません。引数のペアごとに -include-file を記述し、その後に引数のペアを指定する必要があります。ほとんどのテーマでは、多くのスキンファイルやスタイルシートが使用されます。それが原因で、コンパイルするたびに負荷が大きくなることがあります。この処理を簡素化するには、load-config オプションを使用し、設定オプションを含むファイルを指定します。たとえば、複数の include-file オプションを指定します。詳細については、703 ページの「設定ファイルを使用したテーマ SWC ファイルのコンパイル」を参照してください。

include-classes オプションを使用したテーマ SWC ファイルのコンパイル

include-classes オプションは、単一のパラメータ、つまり SWC ファイルに含めるクラス名を受け取ります。クラスファイル名ではなく、クラス名を渡します。たとえば、"MyButtonSkin.as" ではなく、MyButtonSkin を渡します。SWC ファイルをコンパイルするときは、このクラスがソースパスに存在している必要があります。

次のコマンドラインの例では、CSS ファイルと単一のプログラムスキンクラスを持つテーマ SWC ファイルをコンパイルします。このスキンクラスは MyButtonSkin で、"themes" ディレクトリに置かれています。

```
compc -source-path c:¥myfiles¥flex2¥themes
  -include-file mycss.css c:¥myfiles¥flex2¥themes¥mycss.css
  -include-classes MyButtonSkin -o c:¥myfiles¥flex2¥themes¥MyTheme.swc
```

include-classes オプションには、リストの形式で複数のクラスを渡すことができます。次の例に示すように、各クラスを空白で区切って記述します。

```
-include-classes MyButtonSkin MyControlBarSkin MyAccordionHeaderSkin
```

スキンクラスの作成の詳細については、741 ページ、第 20 章の「スキンの使用」を参照してください。

設定ファイルを使用したテーマ SWC ファイルのコンパイル

設定ファイルを使用すると、一般的にコマンドライン上でオプションを渡すより冗長になりますが、その一方で、コンポーネントコンパイラオプションの読み取りと保守は簡単になります。たとえば、設定ファイル内でコマンドラインを次のエントリに置き換えることができます。

```
<?xml version="1.0"?>
<flex-config xmlns="http://www.adobe.com/2006/flex-config">
  <output>MyTheme.swc</output>
  <include-file>
    <name>mycss.css</name>
    <path>c:¥myfiles¥themes¥mycss.css</path>
  </include-file>
  <include-file>
```

```
    <name>upIcon.jpg</name>
    <path>c:¥myfiles¥themes¥assets¥upIcon.jpg</path>
  </include-file>
  <include-file>
    <name>downIcon.jpg</name>
    <path>c:¥myfiles¥hemes¥assets¥downIcon.jpg</path>
  </include-file>
  <include-file>
    <name>overIcon.jpg</name>
    <path>c:¥myfiles¥themes¥assets¥overIcon.jpg</path>
  </include-file>
  <include-classes>
    <class>MyButtonSkin</class>
    <class>MyAccordionHeaderSkin</class>
    <class>MyControlBarSkin</class>
  </include-classes>
</flex-config>
```

次の例に示すように、load-config オプションを指定すると、compc で設定ファイルを使用できます。

```
compc -load-config myconfig.xml
```

Flex Builder のコンポーネントコンパイラに設定ファイルを渡すこともできます。コンポーネントコンパイラの使用の詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 9 章の「Flex コンパイラの使用」を参照してください。

第 19 章

19 フォントの使用

Adobe Flex アプリケーションにはフォントを含めることができます。デフォルトのデバイスフォントを使用するほうがフォントの処理は容易ですが、別のフォントを埋め込むと、テキストベースのコントロールに回転やフェーディングなどの特殊効果を適用できます。このトピックでは、デバイスフォントと埋め込みフォントの違いについて説明します。また、2 バイトフォントの効率的な埋め込み方法について説明します。さらに、Adobe Flex アプリケーションにフォントを埋め込む方法をいくつか紹介します。この方法には、CSS (Cascading Style Sheet : カスケーディングスタイルシート) のシンタックスを使用する方法から、外部 SWF ファイルに埋め込まれたフォントを使用する方法まであります。

目次

フォントについて

Flex アプリケーションをコンパイルすると、テキストの作成に使用したフォントの名前がアプリケーションに保存されます。Adobe Flash Player 9 で Flex アプリケーションを実行する際には、そのフォント名を基に、ユーザーのシステムで使用できる同一または類似のフォントが検索されます。また、Flex アプリケーションにフォントを埋め込んでおけば、クライアントのシステムにそのフォントがなくても、そのアプリケーションでは正しいフォントが使用されます。

各コンポーネントに表示されるフォントは、fontFamily スタイルプロパティを使用して定義します。このプロパティは、外部スタイルシート、<mx:Style> ブロック、またはインラインで設定できます。次の例に示すように、このプロパティにはフォントをリストで指定することができます。

```
.myClass {
  fontFamily: Arial, Helvetica;
  color: Red;
  fontSize: 22;
  fontWeight: bold;
}
```

クライアントのシステムにリストの最初のフォントがない場合、Flash Player は 2 番目のフォントを検索します。以降同様に、一致するフォントが見つかるまで検索します。一致するフォントが見つからない場合、クライアントで使用されているフォントの中で最も近いフォントを推測して使用します。

フォントは、継承可能なスタイルプロパティです。したがって、次の例に示すように、コンテナにフォントスタイルを設定しておけば、そのコンテナに属するすべてのコントロールでそのスタイルが継承されます。

```
<?xml version="1.0"?>
<!-- fonts/InheritableExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     VBox {
        fontFamily: Helvetica;
        fontSize: 18pt;
     }
     HBox {
        fontFamily: Times;
        fontSize: 18pt;
     }
  </mx:Style>

  <mx:Panel title="Styles Inherited from VBox Type Selector">
     <mx:VBox>
        <mx:Button label="This Button uses Helvetica"/>
        <mx:Label text="This label is in Helvetica"/>
        <mx:TextArea width="400" height="75" text="The text in this
            control uses the Helvetica font because it is inherited from
            the VBox style."
        />
```

```
    </mx:VBox>
  </mx:Panel>
  <mx:Panel title="Styles Inherited from HBox Type Selector">
    <mx:HBox>
      <mx:Button label="This Button uses Helvetica"/>
      <mx:Label text="This label is in Helvetica"/>
      <mx:TextArea width="400" height="75"
          text="The text in this control uses the Times font
          because it is inherited from the HBox style."
      />
    </mx:HBox>
  </mx:Panel>
</mx:Application>
```

この例では、HBox および VBox タイプセレクタの fontSize および fontFamily プロパティを定義します。これらのプロパティをサポートするコンテナ内のすべてのコンポーネント、今回の場合は Button、Label、および TextField コントロールにこれらのスタイルが適用されます。

デバイスフォントの使用

fontFamily プロパティには任意のフォントを指定できます。ただし、すべてのシステムにすべてのフォントが備わっているわけではないので、コントロールが予期しない外観になることもあります。最も安全にフォントを指定するには、フォントリストの末尾にデバイスフォントをデフォルトとして含めます。デバイスフォントでは、フォントのアウトライン情報が書き出されず、SWF ファイルにも埋め込まれません。代わりに、Flash Player では、ローカルコンピュータにインストールされているフォントの中で、デバイスフォントに最も近いフォントを使用します。

次の例では、クライアントマシンでデバイスフォント _sans 以外のフォントがどれも見つからない場合に、デバイスフォント _sans を使用するように指定します。

```
<?xml version="1.0"?>
<!-- fonts/DeviceFont.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
    .myClass {
      fontFamily: Arial, Helvetica, "_sans";
      color: Red;
      fontSize: 22;
      fontWeight: bold;
    }
  </mx:Style>

  <mx:Panel title="myClass Class Selector with Device Font">
    <mx:VBox styleName="myClass">
      <mx:Button label="Click Me"/>
      <mx:Label text="Label"/>
      <mx:TextArea width="400" height="75" text="The text
          uses the myClass class selector."
```

```
        />
    </mx:VBox>
  </mx:Panel>
</mx:Application>
```

メモ スタイル宣言でデバイスフォントを定義する場合、デバイスフォント名を引用符で囲む必要があります。

Flash Player では、3 種類のデバイスフォントがサポートされています。これらのフォントについて次の表で説明します。

フォント名	説明
`_sans`	`_sans` デバイスフォントは、Helvetica や Arial などの sans-serif 書体です。
`_serif`	`_serif` デバイスフォントは、Times Roman などの serif 書体です。
`_typewriter`	`_typewriter` デバイスフォントは、Courier などの等幅フォントです。

デバイスフォントはクライアント上に常駐しているため、デバイスフォントを使用しても SWF ファイルのサイズには影響ありません。ただし、デバイスフォントを使用すると、Flash Player がローカルのオペレーティングシステムとデータをやり取りする必要があるため、アプリケーションのパフォーマンスが低下することがあります。また、デバイスフォントのみを指定すると、使用できるフォントは 3 種類に限定されることになります。

埋め込みフォントの使用

クライアントマシンに依存せずに、指定したフォントを利用できるよう、Flex アプリケーション内に TrueType フォントファミリを埋め込むことができます。これにより、このアプリケーションを実行するときは Flash Player で必ずこのフォントを使用できるため、フォントがない場合の影響を考慮せずにすみます。

埋め込みフォントには次の利点があります。

- クライアントの環境に目的のフォントをインストールしておく必要がありません。
- 埋め込みフォントはアンチエイリアス処理されます。つまりフォントの縁が滑らかになり、読みやすくなります。特にテキストのサイズが大きな場合に効果的です。
- 埋め込みフォントの一部またはすべてを、ユーザーから見えなくすることができます。
- 埋め込みフォントを回転することができます。
- 埋め込みフォントは、拡大したときに滑らかに表示されます。
- 埋め込みフォントを使用した場合、テキストは意図したとおりに正確に表示されます。
- フォントを埋め込むときは、クリアで品質の高いテキストレンダリングを SWF ファイルで実現する FlashType ヒント情報を使用できます。FlashType ヒントを使用すると、特に小さいフォントサイズでレンダリングするときに、テキストの読みやすさが大きく向上します。FlashType ヒントの詳細については、716 ページの「FlashType ヒントの使用」を参照してください。

しかし、埋め込みフォントは必ずしも最良のソリューションであるとは限りません。埋め込みフォントには、次のような制限と欠点があります。

- TrueType フォントだけが埋め込まれます。Type 1 PostScript フォントなどの他のフォントタイプを埋め込むには、Flash 8 で作成する SWF ファイルにそのフォントを埋め込み、その SWF ファイルを Flex アプリケーションに埋め込みます。詳細については、729 ページの「SWF ファイルからのフォントの埋め込み」を参照してください。
- ドキュメントにテキストのフォントアウトラインを含める必要があるため、埋め込みフォントを使用するとアプリケーションのファイルサイズが大きくなります。その結果、ユーザーがファイルをダウンロードするのにかかる時間が長くなります。
- 通常、埋め込みフォントを使用すると、テキストサイズが 10 ポイント未満のテキストの読みやすさが低下します。すべての埋め込みフォントは、アンチエイリアス処理を使用してフォント情報をクライアントの画面にレンダリングします。そのため、サイズが小さいとぼやけて表示されたり、読みにくくなったりします。これを避けるには、Flex アプリケーションで FlashType ヒントを使用してフォントをレンダリングします。詳細については、716 ページの「FlashType ヒントの使用」を参照してください。

Flex アプリケーションでは、埋め込みフォントに CSS シンタックスを使用することが普通です。`@font-face` “at-rule” 宣言を使用して埋め込みフォントのソースを指定し、`fontFamily` プロパティを使用してフォントの名前を定義します。使用する同じフォントファミリの書体ごとに `@font-face` 宣言を指定する必要があります。ソースには、ローカル JRE (Java Runtime Environment) フォントまたはファイルパスを使用してアクセスできるフォントを指定できます。このフォントファミリ名を MXML コード内で使用して、埋め込みフォントを参照します。

注
Flex アプリケーションにフォントファイルを埋め込む前に、フォントのライセンスを確認してください。フォントには、ベクター情報として格納することを禁じるライセンス制限がある場合があります。

Flex コンパイラが探し出せないフォントを埋め込もうとすると、Flex ではエラーがスローされます。アプリケーションはコンパイルされません。

埋め込みフォントのシンタックス

TrueType フォントを埋め込むには、スタイルシートまたは <mx:Style> タグ内で次のシンタックスを使用します。

```
@font-face {
  src:url("location") | local("name");
  fontFamily: alias;
  [fontStyle: italic | oblique | normal;]
  [fontWeight: bold | heavy | normal;]
  [flashType: true | false;]
}
```

src プロパティで、コンパイラがフォントを見つける方法、および見つけるフォントの名前を指定します。src:url を使用してファイル名でフォントをロードするか、src:local を使用してシステムフォント名でロードすることができます。このプロパティは必須です。詳細については、712 ページの「埋め込みフォントの検索」を参照してください。

fontFamily プロパティで、スタイルシートでフォントを適用するために使用するフォントのエイリアスを設定します。このプロパティは必須です。

fontStyle および fontWeight プロパティで、フォントの書体の値を設定します。これらのプロパティは省略可能です。デフォルト値は normal です。

flashType プロパティで、フォントの埋め込み時に FlashType ヒント情報を含めるかどうかを指定します。このプロパティはオプションです。デフォルト値は true です。SWF ファイルからフォントを埋め込むときは、このオプションは使用できません。729 ページの「SWF ファイルからのフォントの埋め込み」を参照してください。FlashType ヒントの使用の詳細については、716 ページの「FlashType ヒントの使用」を参照してください。

次の例では、フォントファイル MyriadWebPro.ttf を埋め込みます。

```
@font-face {
  src:url("../assets/MyriadWebPro.ttf");
  fontFamily: myFontFamily;
  flashType: true;
}
```

次の例でも同じフォントを埋め込みますが、システムフォント名を使用します。

```
@font-face {
  src:local("Myriad Web Pro");
  fontFamily: myFontFamily;
  flashType: true;
}
```

@font-face 宣言でフォントを埋め込んだ後は、タイプセレクタやクラスセレクタ内で新規の fontFamily 名やエイリアスを使用できます。次の例では myFontFamily 埋め込み fontFamily を VBox コントロールのタイプセレクタとして使用します。

```
<?xml version="1.0"?>
```

```
<!-- fonts/EmbeddedFontFace.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFontFamily;
        flashType: true;
     }

     VBox {
        fontFamily: myFontFamily;
     }
  </mx:Style>

  <mx:Panel title="Embedded Font Applied With Type Selector">
     <mx:VBox>
        <mx:Button label="Click Me"/>
        <mx:Label text="Label"/>
        <mx:TextArea width="400" height="75" text="The text
            uses the myClass class selector."
        />
     </mx:VBox>
  </mx:Panel>
</mx:Application>
```

また、次の例で示すように、埋め込みフォントをインラインで使用することもできます。

```
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontFaceInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFontFamily;
        flashType: true;
     }
  </mx:Style>

  <mx:Panel title="Embedded Font Applied Inline">
     <mx:VBox fontFamily="myFontFamily">
        <mx:Button label="Click Me"/>
        <mx:Label text="Label"/>
        <mx:TextArea width="400" height="75" text="This text
            is in Myriad Web Pro."
        />
     </mx:VBox>
  </mx:Panel>
</mx:Application>
```

この例を実行すると、Button コントロールのラベルが表示されません。これは、Button コントロールのラベルにボールド書体を使用しているからです。埋め込みフォントの書体 (MyriadWebPro) には、ボールド書体の定義が含まれていません。Button コントロールのラベルが正しい書体で表示されるようにするには、フォントのボールド書体も埋め込む必要があります。ボールド書体の埋め込みについては、721 ページの「複数の書体の使用」を参照してください。

埋め込みフォントの検索

@font-face 宣言の src 属性では、fontFamily の場所を指定します。url 関数または local 関数を指定できます。これらの関数について次の表で説明します。

属性	内容
url	フォントの有効な URI を指定することで、TrueType フォントを場所によって埋め込みます。相対 URI ("../fontfolder/akbar.ttf" など) または絶対 URI ("c:￥myfonts￥akbar.ttf" など) で指定できます。
local	ローカルでアクセスできる TrueType フォントを場所ではなく名前で埋め込みます。フォントは、ファイル名ではなく、フォント名で指定します。たとえば、"AkbarBI.ttf" ではなく、Akbar Bold Italic を指定します。 アプリケーションサーバーの JRE からローカルでアクセスできるフォントを埋め込むことができます。このようなフォントとして、"jre/lib/fonts" フォルダ内の *.ttf ファイル、"jre/lib/font.properties" ファイルで割り当てられたフォント、オペレーティングシステム (OS) によって JRE で使用可能となっているフォントなどがあります。Windows の場合、"/windows/fonts" (または "/winnt/fonts") ディレクトリ内の TTF ファイルを local 関数で使用できます。Solaris または Linux の場合、xfs などのフォントサーバーに登録されているフォントを利用できます。 指定するフォント名は、オペレーティングシステムによって決まります。通常、フォントファイルの拡張子は含めませんが、これはオペレーティングシステムによって異なります。詳細については、オペレーティングシステムのマニュアルを参照してください。

@font-face 宣言では、src 記述子の url 関数または local 関数を指定する必要があります。他の記述子はすべて省略可能です。一般的に local ではなく、url を使用します。ファイルを指定した方が、オペレーティングシステムで制御される参照を使用するよりも確実だからです。

同じ fontFamily 記述子の埋め込みフォントと非埋め込みフォントが、混在しないようにしてください。

ActionScript へのフォントの埋め込み

ActionScript で [Embed] メタデータタグを使用することで、位置または名前を指定して、TTF フォントファイルまたはシステムフォントを埋め込むことができます。位置を指定してフォントを埋め込むには、[Embed] メタデータタグで source プロパティを使用します。名前を指定してフォントを埋め込むには、[Embed] メタデータタグで systemFont プロパティを使用します。

次の例では、[Embed] タグのシンタックスを使用し、位置を指定してフォントを埋め込みます。

```
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontFaceActionScriptByLocation.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .mystyle1 {
        fontFamily:myMyriadFont;
        fontSize: 32pt;
```

```
    }
    .mystyle2 {
       fontFamily:myBoldMyriadFont;
       fontSize: 32pt;
       fontWeight: bold;
    }
  </mx:Style>

  <mx:Script>
    /*
     * Embed a font by location.
     */
    [Embed(source='../assets/MyriadWebPro.ttf',
       fontName='myMyriadFont',
       mimeType='application/x-font'
    )]
    // You do not use this variable directly. It exists so that
    // the compiler will link in the font.
    private var font1:Class;

    /*
     * Embed a font with bold typeface by location.
     */
    [Embed(source='../assets/MyriadWebPro-Bold.ttf',
       fontWeight='bold',
       fontName='myBoldMyriadFont',
       mimeType='application/x-font',
       flashType='true'
    )]
    private var font2:Class;

  </mx:Script>

  <mx:Panel title="Embedded Fonts Using ActionScript">
    <mx:VBox>
       <mx:Label
           width="100%"
           height="75"
           styleName="mystyle1"
           text="The text uses the MyriadWebPro font."
           rotation="15"
       />
       <mx:Label
           width="100%"
           height="75"
           styleName="mystyle2"
           text="The text uses the MyriadWebPro-Bold font."
           rotation="15"
       />
    </mx:VBox>
  </mx:Panel>
</mx:Application>
```

[Embed] タグで設定した fontName プロパティの値を、スタイル定義でエイリアス (fontFamily) として使用します。

ボールドやイタリックなど、異なる書体を持つフォントを埋め込むには、[Embed] ステートメントとスタイル定義で、fontWeight 属性または fontStyle 属性を指定します。異なる書体の埋め込みの詳細については、721 ページの「複数の書体の使用」を参照してください。

この例が正しく機能することを確認するために、Label コントロールを回転させます。フォントが正しく埋め込まれていない場合は、Label コントロールに表示されているテキストを回転させると、そのテキストが消失します。

ローカルフォント、すなわちシステムフォントを埋め込むには、フォントのファイル名ではなくシステムフォント名を使用します。また、[Embed] タグで source 属性ではなく、systemFont 属性を使用してその名前を指定します。それ以外の場合は、次の例と同じシンタックスを使用します。

```
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontFaceActionScriptByName.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .mystyle1 {
        fontFamily:myPlainFont;
        fontSize: 32pt;
     }
     .mystyle2 {
        fontFamily:myItalicFont;
        fontSize: 32pt;
        fontStyle: italic;
     }
  </mx:Style>

  <mx:Script>
     /*
      * Embed a font by name.
      */
     [Embed(systemFont='Myriad Web Pro',
        fontName='myPlainFont',
        mimeType='application/x-font'
     )]
     // You do not use this variable directly. It exists so that
     // the compiler will link in the font.
     private var font1:Class;

     /*
      * Embed a font with italic typeface by name.
      */
     [Embed(systemFont='Myriad Web Pro',
        fontStyle='italic',
        fontName='myItalicFont',
        mimeType='application/x-font',
        flashType='true'
     )]
```

```
    private var font2:Class;

  </mx:Script>

  <mx:Panel title="Embedded Fonts Using ActionScript">
     <mx:VBox>
        <mx:Label
            width="100%"
            height="75"
            styleName="mystyle1"
            text="The text uses the plain typeface."
            rotation="15"
        />
        <mx:Label
            width="100%"
            height="75"
            styleName="mystyle2"
            text="The text uses the italic typeface."
            rotation="15"
        />
     </mx:VBox>
  </mx:Panel>
</mx:Application>
```

ヒント

Windows でシステムフォント名を取得するには、フォントプロパティの拡張機能をインストールします。そして、Windows エクスプローラでフォントのファイルを右クリックし、[Names] タブを選択します。[Font Family Name] の下の値を `systemFont` の値として使用します。

`[Embed]` メタデータタグまたは `@font-face` 宣言で `unicodeRange` パラメータを指定することで、フォントの文字範囲のサブセットを指定できます。デフォルトのすべての文字ではなく、ある範囲の文字だけを埋め込むことで、埋め込みフォントのサイズが小さくなり、SWF ファイルの最終的な出力サイズが小さくなります。詳細については、725 ページの「文字範囲の設定」を参照してください。

FlashType ヒントの使用

フォントを埋め込むときに FlashType ヒントを使用して、埋め込むフォントにフォントに関する追加情報を含めることができます。通常、FlashType ヒント情報を使用する埋め込みフォントは、フォントサイズが小さいときによりクリアでシャープに表示されます。

デフォルトでは、Flex アプリケーションで埋め込むフォントには FlashType ヒント情報が使用されます。このデフォルトは、flex-config.xml ファイルのコンパイラオプション `fonts.flash-type` で設定されます。デフォルト値は `true` です。スタイルシートで値を設定するか、設定ファイルで値を変更すると、このデフォルト値をオーバーライドできます。スタイルシートで FlashType ヒントを無効にするには、次の例に示すように、`@font-face` 規則で `flashType` スタイルプロパティを `false` に設定します。

```
@font-face {
   src:url("../assets/MyriadWebPro.ttf");
   fontFamily: myFontFamily;
   flashType: false;
}
```

FlashType ヒントを使用すると、コンパイラのパフォーマンスが落ちることがあります。これは実行時には問題ありませんが、アプリケーションを頻繁にコンパイルする場合や、Flex Data Services でサポートされる要求時コンパイラを使用する場合には顕著になることがあります。FlashType ヒントを使用すると、SWF ファイルのロードが若干遅くなることもあります。特に、使用する文字セットの数が多くなると、このロード速度の低下が目立つようになります。したがって、使用するフォントの数に注意する必要があります。FlashType ヒント情報があることで、Flash Player でのメモリ使用量が増えることもあります。たとえば、4 ～ 5 種類のフォントを使用している場合、メモリ使用量がおよそ 4 MB 増加します。

Flex アプリケーションで FlashType ヒントを使用するフォントを埋め込むと、フォントは他の埋め込みフォントとまったく同じように機能します。これらのフォントは、アンチエイリアス処理され、回転が可能で、部分または全体を透明にすることができます。

FlashType ヒントを使用するフォント定義では、追加のスタイルプロパティの `fontAntiAliasType`、`fontGridFitType`、`fontSharpness`、および `fontThickness` がサポートされます。これらのプロパティはすべて継承スタイルです。

FlashType 関連のスタイルプロパティは CSS スタイルなので、`fontFamily` や `fontSize` などの標準のスタイルプロパティと同様に使用できます。たとえば、テキストベースのコンポーネントでは New Century 14 のサブピクセル FlashType ヒントをシャープネス 50、太さ -35 で使用し、すべての Button コントロールでは Tahoma 10 のピクセル FlashType ヒントをシャープネス 0、太さ 0 で使用するというようなことが可能です。これらのスタイルは、TextFiled のすべてのテキストに適用されます。一部のテキストにのみ適用することはできません。

FlashType のスタイルプロパティのデフォルト値は、"defaults.css" ファイルで定義されています。このファイルを別のファイルに置き換えた場合や、そのプロパティをオーバーライドする別のスタイルシートを使用した場合、Flash Player での FlashType ヒントを使用するフォントのレンダリングには標準のフォントレンダラーが使用されます。FlashType ヒントを使用するフォントを埋め込んでも、fontAntiAliasType プロパティを advanced に設定していないと、FlashType ヒント情報の利点を生かすことはできません。

これらのプロパティについて次の表で説明します。

スタイルプロパティ	内容
fontAntiAliasType	内部 TextField の antiAliasType プロパティを設定します。有効な値は、normal および advanced です。デフォルト値は advanced で、フォントの FlashType ヒントが有効になります。 コンパイラが FlashType ヒントを使用しないようにするには、このプロパティを normal に設定します。 このスタイルは、システムフォントや FlashType ヒント情報なしで埋め込まれるフォントには影響を与えません。
fontGridFitType	内部 TextField の gridFitType プロパティを設定します。有効な値は none、pixel、および subpixel です。デフォルト値は pixel です。詳細については、『Adobe Flex 2 リファレンスガイド』で、TextField クラスおよび GridFitType クラスを参照してください。 このプロパティは、システムフォントに対しては TextField コントロールの gridFitType スタイルプロパティと同じ効果を持ち、FlashType ヒントとともにフォントを埋め込んでいる場合にのみ適用されます。 fontAntiAliasType プロパティを advanced に設定していない限り、このプロパティの値を変更しても何の効果もありません。
fontSharpness	内部 TextField の sharpness プロパティを設定します。有効な値は -400 ～ 400 の範囲の数値で、デフォルト値は 0 です。 このプロパティは、システムフォントに対しては TextField コントロールの fontSharpness スタイルプロパティと同じ効果を持ち、FlashType ヒントとともにフォントを埋め込んでいる場合にのみ適用されます。 fontAntiAliasType プロパティを advanced に設定していない限り、このプロパティの値を変更しても何の効果もありません。
fontThickness	内部 TextField の thickness プロパティを設定します。有効な値は -200 ～ 200 の範囲の数値で、デフォルト値は 0 です。 このプロパティは、システムフォントに対しては TextField コントロールの fontThickness スタイルプロパティと同じ効果を持ち、FlashType ヒントとともにフォントを埋め込んでいる場合にのみ適用されます。 fontAntiAliasType プロパティを advanced に設定していない限り、このプロパティの値を変更しても何の効果もありません。

埋め込みフォントの検出

SystemManager の isFontFaceEmbedded() メソッドを使用すると、フォントが埋め込まれているかどうか、さらに埋め込まれているフォントがあれば、それが Font クラスの register() メソッドを使用してグローバルに登録されているかどうかを判断できます。isFontFaceEmbedded() メソッドは1つの引数を受け取ります。この引数は、フォントの TextFormat を記述するオブジェクトです。このメソッドは、次の例に示すように指定したフォントファミリが埋め込まれているかどうかを示すブール値を返します。

```
<?xml version="1.0"?>
<!-- fonts/DetectingEmbeddedFonts.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
     creationComplete="determineIfFontFaceIsEmbedded()">
  <mx:Style>
     @font-face {
        src: url(../assets/MyriadWebPro.ttf);
        fontFamily: myPlainFont;
        flashType: true;
     }

     .myStyle1 {
        fontFamily: myPlainFont;
        fontSize:12pt
     }
  </mx:Style>
  <mx:Script><![CDATA[
     import mx.managers.SystemManager;
     import flash.text.TextFormat;

     public function determineIfFontFaceIsEmbedded():void {
        var tf1:TextFormat = new TextFormat();
        tf1.font = "myPlainFont";

        var tf2:TextFormat = new TextFormat();
        tf2.font = "Arial";

        var b1:Boolean = Application.application.systemManager.
           isFontFaceEmbedded(tf1);
        var b2:Boolean = Application.application.systemManager.
           isFontFaceEmbedded(tf2);
        l1.text = tf1.font + " (" + b1 + ")";
        l2.text = tf2.font + " (" + b2 + ")";
     }
  ]]></mx:Script>
  <mx:Text id="text1" styleName="myStyle1" text="Rotate Me"/>

  <mx:Button label="Rotate +1" click="++text1.rotation;"/>
  <mx:Button label="Rotate -1" click="--text1.rotation;"/>
```

```
  <mx:Form>
     <mx:FormItem label="isFontFaceEmbedded:">
        <mx:Label id="l1"/>
     </mx:FormItem>
     <mx:FormItem label="isFontFaceEmbedded:">
        <mx:Label id="l2"/>
     </mx:FormItem>
  </mx:Form>
</mx:Application>
```

Font クラスの enumerateFonts() メソッドを使用して、デバイスや埋め込みフォントに関する情報を出力できます。次の例では埋め込みフォントの一覧を表示します。

```
<?xml version="1.0"?>
<!-- fonts/EnumerateFonts.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="listFonts()">
  <mx:Style>
     @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFont;
        flashType: true;
     }

     @font-face {
        src:url("../assets/MyriadWebPro-Bold.ttf");
        fontFamily: myFont;
        fontWeight: bold;
        flashType: true;
     }

     @font-face {
        src:url("../assets/MyriadWebPro-Italic.ttf");
        fontFamily: myFont;
        fontStyle: italic;
        flashType: true;
     }

     .myPlainStyle {
        fontSize: 32;
        fontFamily: myFont;
     }

     .myBoldStyle {
        fontSize: 32;
        fontFamily: myFont;
        fontWeight: bold;
     }

     .myItalicStyle {
        fontSize: 32;
```

```
        fontFamily: myFont;
        fontStyle: italic;
    }
  </mx:Style>

  <mx:Script><![CDATA[
     private function listFonts():void {
        var fontArray:Array = Font.enumerateFonts(false);
        for(var i:int = 0; i < fontArray.length; i++) {
           var thisFont:Font = fontArray[i];
           if (thisFont.fontType == "embedded") {
            trace("name: " + thisFont.fontName +
            "; typeface: " + thisFont.fontStyle +
            "; type: " + thisFont.fontType);
           }
        }
     }
  ]]></mx:Script>

  <mx:VBox>
     <mx:Label text="Plain Label" styleName="myPlainStyle"/>
     <mx:Label text="Italic Label" styleName="myBoldStyle"/>
     <mx:Label text="Bold Label" styleName="myItalicStyle"/>
  </mx:VBox>

</mx:Application>
```

この例による出力を次に示します。

```
name: myFont; typeface: regular; type: embedded
name: myFont; typeface: bold; type: embedded
name: myFont; typeface: italic; type: embedded
```

enumerateFonts() メソッドは単一のブール値の引数、enumerateDeviceFonts 受け取ります。enumerateDeviceFonts プロパティのデフォルト値は false で、これはデフォルトでは埋め込みフォントの配列が返されることを示します。

enumerateDeviceFonts 引数を true に設定すると、enumerateFonts() メソッドはクライアントシステムで使用できるデバイスフォントの配列を返しますが、それはクライアントの mms.cfg ファイルで、DisableDeviceFontEnumeration プロパティをデフォルト値の 0 に設定している場合だけです。DisableDeviceFontEnumeration プロパティを 1 に設定すると、クライアントで明示的に設定しない限り、Flash Player ではクライアントコンピュータ上にあるデバイスフォントのリストは作成されません。"mms.cfg" ファイルによるクライアントの設定の詳細については、Flash Player のマニュアルを参照してください。

複数の書体の使用

ほとんどのフォントには、4 つの書体スタイルがあります。それは、標準、ボールド、イタリック、およびボールドイタリックです。Flex アプリケーションには、任意の数の書体スタイルを埋め込むことができます。アプリケーションにボールド書体のみを埋め込むと、通常の書体 (標準書体) は使用できません。使用するには、標準書体も埋め込む必要があります。使用する書体ごとに、新規の @font-face 宣言をスタイルシートに追加する必要があります。

Button コントロールのように、デフォルトで標準書体ではなく、ボールド書体を使用する Flex コントロールもあります。Button ラベルにフォントを埋め込む場合は、そのフォントのボールド書体を埋め込む必要があります。

次の例では、Myriad Web Pro フォントのボールド、イタリック、標準の各書体を埋め込みます。書体を定義した後、fontFamily と同じエイリアスを使用してフォントのセレクタを定義します。ボールド、イタリック、標準の書体ごとに 1 つのセレクタを定義します。この例では、フォントスタイルを適用するために、クラスセレクタを Label コントロールにインラインで適用します。

```
<?xml version="1.0"?>
<!-- fonts/MultipleFaces.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFont;
        flashType: true;
     }

     @font-face {
        /* Note the different filename for boldface. */
        src:url("../assets/MyriadWebPro-Bold.ttf");
        fontFamily: myFont; /* Notice that this is the same alias. */
        fontWeight: bold;
        flashType: true;
     }

     @font-face {
        /* Note the different filename for italic face. */
        src:url("../assets/MyriadWebPro-Italic.ttf");
        fontFamily: myFont; /* Notice that this is the same alias. */
        fontStyle: italic;
        flashType: true;
     }

     .myPlainStyle {
        fontSize: 32;
        fontFamily: myFont;
     }
```

```
    .myBoldStyle {
       fontSize: 32;
       fontFamily: myFont;
       fontWeight: bold;
    }

    .myItalicStyle {
       fontSize: 32;
       fontFamily: myFont;
       fontStyle: italic;
    }
  </mx:Style>

  <mx:VBox>
     <mx:Label text="Plain Label" styleName="myPlainStyle"/>
     <mx:Label text="Italic Label" styleName="myItalicStyle"/>
     <mx:Label text="Bold Label" styleName="myBoldStyle"/>
  </mx:VBox>

</mx:Application>
```

オプションとして、ボールド書体やイタリック書体をインラインでコントロールに適用することもできます。次に例を示します。

```
<?xml version="1.0"?>
<!-- fonts/MultipleFacesAppliedInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFont;
        flashType: true;
     }

     @font-face {
        src:url("../assets/MyriadWebPro-Bold.ttf");
        fontFamily: myFont;
        fontWeight: bold;
        flashType: true;
     }

     @font-face {
        src:url("../assets/MyriadWebPro-Italic.ttf");
        fontFamily: myFont;
        fontStyle: italic;
        flashType: true;
     }

     .myStyle1 {
        fontSize: 32;
        fontFamily: myFont;
     }
```

```
  </mx:Style>

  <mx:VBox styleName="myStyle1">
     <mx:Label text="Plain Label"/>
     <mx:Label text="Italic Label" fontStyle="italic"/>
     <mx:Label text="Bold Label" fontWeight="bold"/>
  </mx:VBox>

</mx:Application>
```

場所を指定したフォントファイルではなく、名前を指定してシステムフォントを埋め込む場合は、@font-face の規則で、書体名 (MyriadWebPro-Bold など) ではなくフォントのファミリ名 (Myriad Web Pro など) を使用します。次の例では、システムフォント名を使用して標準書体とイタリック書体を埋め込みます。

```
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontFaceCSSByName.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     @font-face {
        src:local("Myriad Web Pro");
        fontFamily: myPlainFont;
        flashType: true;
     }

     @font-face {
        src:local("Myriad Web Pro");
        fontFamily: myItalicFont;
        fontStyle: italic;
        flashType: true;
     }

     .mystyle1 {
        fontFamily:myPlainFont;
        fontSize: 32pt;
     }
     .mystyle2 {
        fontFamily:myItalicFont;
        fontSize: 32pt;
        fontStyle: italic;
     }
  </mx:Style>

  <mx:Panel title="Embedded Fonts Using CSS">
     <mx:VBox>
        <mx:Label
            width="100%"
            height="75"
            styleName="mystyle1"
            text="The text uses the plain typeface."
            rotation="15"
```

```
        />
        <mx:Label
            width="100%"
            height="75"
            styleName="mystyle2"
            text="The text uses the italic typeface."
            rotation="15"
        />
    </mx:VBox>
  </mx:Panel>
</mx:Application>
```

Flex では、<b> HTML タグおよび <i> HTML タグを使用して、Label などのテキストベースコントロールのテキストにボールドやイタリックを適用することもできます。

ボールドイタリックのフォントを使用する場合は、そのフォントにボールドイタリック用の別の書体がある必要があります。次の例のように、@font-face とセレクタブロックに両方の記述子 (fontWeight および fontStyle) を指定します。

```
@font-face {
   src:url("../assets/KNIZIA-BI.TTF");
   fontStyle: italic;
   fontWeight: bold;
   fontFamily: myFont;
   flashType: true;
}
.myBoldItalicStyle {
   fontFamily:myFont;
   fontWeight:bold;
   fontStyle:italic;
   fontSize: 32;
}
```

In the @font-face 定義では、fontWeight 記述子と fontStyle 記述子を使用して、フォントがボールドであるかイタリックであるかを指定できます。ボールドフォントにするには、fontWeight を bold に設定するか、700 以上の整数に設定します。ボールドではないフォントにするには、fontWeight に plain または normal を指定します。イタリックフォントにするには、fontStyle を italic または oblique に設定します。イタリックではないフォントにするには、fontStyle に plain または normal を指定します。fontWeight または fontStyle を指定しないと、plain または normal を指定したと見なされます。

他のクラスセレクタやタイプセレクタと同様に、埋め込みフォント用の fontSize などの任意の記述子をセレクタに追加することもできます。

デフォルトでは、Flex にはアプリケーション内の各埋め込みフォントについてフォント定義全体が含まれるため、アプリケーションのサイズを小さくするには、使用するフォント数を制限する必要があります。フォントの文字範囲を定義することで、フォント定義のサイズを限定できます。詳細については、725 ページの「文字範囲の設定」を参照してください。

フォントマネージャについて

Flex には、デフォルトでフォントマネージャが 2 つ含まれています。フォントマネージャは、埋め込み TrueType フォント定義を使用し、Flash Player に各文字を描画します。2 つのフォントマネージャは Batik と JRE です。それぞれ BatikFontManager と JREFontManager クラスで表されます。Batik フォントマネージャではローカルフォントを使用できないので、その機能には制限があります。しかし、一般的には Batik フォントマネージャのほうが、より円滑なレンダリングと正確な線マトリックス (複数行のテキストや行の長さの計算に影響します) を提供します。ローカルフォントとは、オペレーティングシステムからアクセスできるフォントのことです。一方、JRE フォントマネージャは、オペレーティングシステムで使用されるローカルフォントにアクセスできますが、出力の品質は概して高くありません。

"flex-config.xml" ファイルで、どのフォントマネージャを使用するかを決定します。デフォルト設定では、次の例に示す優先順位で両方を使用します。

```
<fonts>
  <managers>
    <manager-class>flash.fonts.JREFontManager</manager-class>
    <manager-class>flash.fonts.BatikFontManager</manager-class>
  </managers>
</fonts>
```

`<manager>` 要素の優先順位は、これとは逆の順になります。つまり、デフォルトでは Batik フォントマネージャが、優先されるフォントマネージャです。ただし、Batik フォントマネージャはシステムフォントをサポートしていないので、それらのフォントをサポートするために JRE フォントマネージャも使用できるようになっています。

文字範囲の設定

埋め込みフォントのフェイスを構成するシンボルの範囲を指定することにより、埋め込みフォントのサイズを小さくできます。フォント内の各文字を記述する必要があります。これらの文字の一部を削除すると、各埋め込みフォントに対して必要な記述情報全体のサイズが小さくなります。

文字の範囲は、"flex-config.xml" ファイル、または各 MXML ファイルの `font-face` 宣言で設定できます。文字の Unicode 値を使用して、個別の文字または文字範囲を指定します。1 つのフォント宣言に対して複数の範囲を設定できます。

宣言された範囲外の文字を使用した場合、その文字の部分には何も表示されません。

このセクションでは、Flex アプリケーションに文字範囲を設定する方法を中心に説明します。なお、SWF ファイルからフォントを埋め込む場合は、Flash で文字範囲を制限できます。詳細については、729 ページの「SWF ファイルからのフォントの埋め込み」を参照してください。

文字範囲の詳細については、CSS-2 フォントの仕様 (www.w3.org/TR/1998/REC-CSS2-19980512/fonts.html#descdef-unicode-range) を参照してください。

font-face 宣言での範囲の設定

font-face 宣言の unicodeRange 属性を使用して、使用可能な文字範囲を MXML ファイル内で設定できます。次の例では、<mx:Style> タグ内で Myriad Web Pro フォントを埋め込み、フォントの文字範囲を定義します。

```
<?xml version="1.0"?>
<!-- fonts/EmbeddedFontCharacterRange.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     @font-face {
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily: myFontFamily;
        flashType: true;
        unicodeRange:
           U+0041-U+005A, /* Upper-Case [A..Z] */
           U+0061-U+007A, /* Lower-Case a-z */
           U+0030-U+0039, /* Numbers [0..9] */
           U+002E-U+002E; /* Period [.] */
     }

     TextArea {
        fontFamily: myFontFamily;
        fontSize: 32;
     }
  </mx:Style>

  <mx:Panel title="Embedded Font Character Range">
        <mx:TextArea
            width="400"
            height="150"
            text="The Text Uses Only Some of Available
            Characters 0 1 2 3 4 5 6 7 8 9."
        />
  </mx:Panel>
</mx:Application>
```

"flex-config.xml" での範囲の設定

<language-range> 子タグを使用して、"flex-config.xml" ファイルで埋め込みフォントの言語と文字範囲を指定できます。これにより、1回定義した範囲を複数の @font-face ブロックで使用できます。

次の例では、englishRange および otherRange という名前が付いた範囲を "flex-config.xml" ファイルで作成します。

```
<fonts>
  <languages>
     <language-range>
       <lang>englishRange</lang>
```

```
        <range>U+0020-U+007E</range>
      </language-range>
      <language-range>
        <lang>otherRange</lang>
        <range>U+00??</range>
      </language-range>
    <languages>
</fonts>
```

MXML ファイルでは、次の例に示すように、@font-face 宣言の unicodeRange 属性を使用して定義済みの範囲を指定します。

```
@font-face {
  fontFamily: myPlainFont;
  src: url("../assets/MyriadWebPro.ttf");
  flashType: true;
  unicodeRange: "englishRange";
}
```

Flex には、Flash の "UnicodeTable.xml" の文字範囲の割り当てを収めたファイルが用意されており、これを Flex の設定ファイルで利用できます。このファイルは、Adobe Flex Data Services では "flex_app_root/WEB-INF/flex/flash-unicode-table.xml" にあり、Adobe Flex SDK では "flex_install_dir/frameworks/flash-unicode-table.xml" にあります。

次の例は、Latin 1 にあらかじめ定義されている範囲です。

```
<language-range>
  <lang>Latin I</lang>
  <range>U+0020,U+00A1-U+00FF,U+2000-U+206F,U+20A0-U+20CF,U+2100-U+2183
  </range>
</language-range>
```

"flash-unicode-table.xml" に記載されている範囲を Flex アプリケーションで使用できるようにするには、このファイルの範囲をコピーして "flex-config.xml" ファイルに追加します。

使用可能な範囲の検出

Font クラスを使用すると、フォントで使用できる文字を検出できます。これには、hasGlyphs() メソッドを使用します。

次の例では、同じフォントを 2 回埋め込み、それぞれのフォントを異なる文字範囲に制限します。最初のフォントでは文字 A と文字 B のみを使用でき、2 番目のフォントファミリでは基本ラテンブロックにある 128 文字すべてを使用できます。

```
<?xml version="1.0"?>
<!-- charts/CharacterRangeDetection.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="checkCharacterSupport();">
  <mx:Style>
    @font-face{
```

```
        font-family: myABFont;
        flashType: true;
        src:url("../assets/MyriadWebPro.ttf");
        /* Limit range to the letters A and B. */
        unicodeRange: U+0041-U+0042;
     }
     @font-face{
        font-family: myWideRangeFont;
        flashType: true;
        src:url("../assets/MyriadWebPro.ttf");
        /* Set range to the 128 characters in
        the Basic Latin block. */
        unicodeRange: U+0041-U+007F;
     }
  </mx:Style>
  <mx:Script><![CDATA[
     public function checkCharacterSupport():void {
        var fontArray:Array = Font.enumerateFonts(false);
        for(var i:int = 0; i < fontArray.length; i++) {
           var thisFont:Font = fontArray[i];
           if (thisFont.hasGlyphs("DHARMA")) {
            trace(thisFont.fontName +
                 " supports these glyphs");
           } else {
            trace(thisFont.fontName +
              " does not support these glyphs");
           }
        }
     }
  ]]></mx:Script>
</mx:Application>
```

2 バイトフォントの埋め込み

Flex で 2 バイトフォントを使用する場合は、可能な限り少ない文字セットを埋め込む必要があります。フォントの文字セット全体を埋め込むと、アプリケーションの SWF ファイルが巨大なサイズになることがあります。Unicode 文字範囲のセットを "flex-config.xml" ファイルに定義し、その範囲の名前をスタイルの `@font-face` 宣言で参照できます。

Flex では、"flash-unicode-table.xml" ファイルに、タイ語、漢字、ハングル語、ヘブライ語などの一般的な 2 バイト文字の文字範囲があらかじめ定義されています。このファイルは Flex では処理されませんが、さまざまな文字範囲に使用できるようあらかじめ定義して用意されています。たとえば、次に示すタイ語の文字範囲は "flash-unicode-table.xml" ファイルに記載されています。

```
<language-range>
  <lang>Thai</lang>
  <range>U+0E01-U+0E5B</range>
</language-range>
```

Flex アプリケーションでこの言語を使用するには、この文字範囲を "flex-config.xml" ファイルにコピーするか、fonts.languages.language-range オプションを使用してコマンドライン上で渡します。次の例に示すように、定義全体を子タグとして <languages> タグに追加します。

```
<flex-config>
  <fonts>
    <languages>
      <language-range>
        <lang>thai</lang>
        <range>U+0E01-U+0E5B</range>
      </language-range>
    </languages>
  </fonts>
  ...
</flex-config>
```

<lang> エレメントの値は、任意の値に変更できます。CSS を使用してフォントを埋め込む場合、次の例に示すように、@font-face 宣言の unicodeRange プロパティでこの値を使用して言語を参照します。

```
@font-face {
  fontFamily:"Thai_font";
  src: url("../assets/THAN.TTF"); /* ファイルから埋め込みます。*/
  flashType: true;
  unicodeRange:"thai"
}
```

SWF ファイルからのフォントの埋め込み

SWF ファイルに埋め込まれたフォントを、Flex アプリケーションに埋め込むことができます。

SWF ファイルからのフォントの埋め込みについて

Flex アプリケーションで使用する SWF ファイルにフォントを埋め込むときは、Flash 8 でサポートされる任意のフォントを埋め込むことができます。これには、Type 1 PostScript およびビットマップ (Macintosh のみ) フォントや、FlashType ヒント付きのフォントも含まれます。そうするには、目的のフォントを収めた SWF ファイルを Flash 8 で作成します。次に、その SWF ファイルに収めたフォントの内容を参照し、Flex アプリケーションにフォントが埋め込まれるようにします。

一般的に、埋め込むフォントごとに、基本的な 4 書体である標準、ボールド、イタリック、およびボールドイタリックを埋め込む必要があります。書体ごとに独立した TTF ファイルを持たないフォントを埋め込む場合でも、この書体の処理は必要になります。これは、標準書体以外の書体を使用する Flex コントロールが存在するからです。

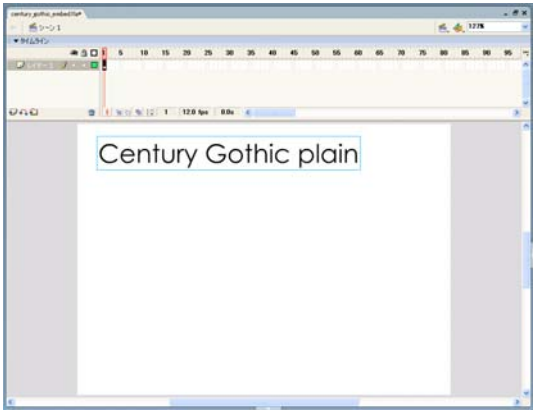
たとえば、Button コントロールのラベルには、そのコントロールで使用するフォントのボールド書体が使用されます。書体を 1 つだけ埋め込み、それを Button コントロールで使おうとしても、Button コントロールではボールド書体が必要となるので、この埋め込んだ書体は適用されません。

埋め込みフォントを使用した Flash 8 ファイルの作成

Flex アプリケーションに SWF ファイルからフォントを埋め込むには、まず目的のフォントを収めた SWF ファイルを Flash で作成します。このセクションの手順を終了した後、732 ページの「SWF ファイルから Flex アプリケーションへのフォントの埋め込み」の手順に従って処理を完了します。

フォントを埋め込んだ Flash 8 SWF ファイルを作成する手順は次のとおりです。

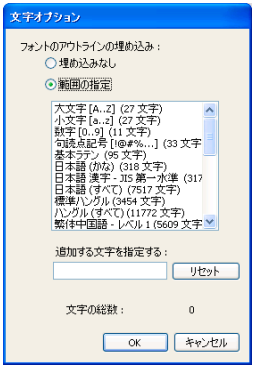
1. Flash 8 でドキュメントを作成します。
2. テキストツールを選択し、マウスをドラッグしてステージ上にテキスト領域を作成します。
3. [プロパティ] パネルで、フォントドロップダウンリストから埋め込むフォントを選択します。
4. このテキスト領域に 1 文字以上入力します。複数のフォントを埋め込む場合は、次の例のように、フォント名とその書体名を入力しておくと便利です。



5. [プロパティ] パネルで [ダイナミックテキスト] ドロップダウンリストを選択し、このテキストをダイナミックテキストにします。
6. フォントで FlashType ヒントを使用する場合は、[Anti-Alias for Readability] または [Custom Anti-Alias] オプションが選択されたアンチエイリアスモードであることを確認してください。他のオプションを選択すると、フォントの SWF ファイルに FlashType ヒント情報が含められません。FlashType ヒントの使用の詳細については、716 ページの「FlashType ヒントの使用」を参照してください。

7. [プロパティ] パネルで [埋め込み] ボタンをクリックします。

   [文字の埋め込み] ダイアログボックスが表示されます。



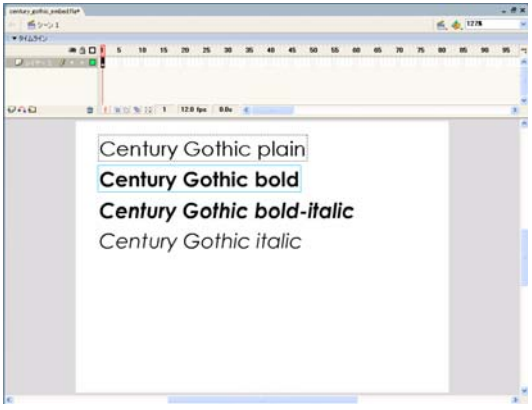
8. 使用する文字範囲を 1 つ以上選択します。必要な範囲のみを選択するようにします。

   All を選択するのは、すべての範囲がどうしても必要な場合のみとします。SWF ファイルに追加する文字が増加するほど、そのファイルサイズも大きくなります。たとえば、Century Gothic について All を選択すると、得られる SWF ファイルのサイズは約 270 KB になります。Uppercase、Lowercase、および Punctuation のみを選択すれば、完成した SWF ファイルのサイズは 14 KB 程度に収まります。

   どの文字を Flex アプリケーションで使用するかが正確にわかっていない限り、[リセット] を選択しないようにします。

9. ボールドやイタリックなど、必要な書体ごとに追加のテキストベースコントロールを作成します。ボールドコントロールにボールド書体を適用します。イタリックコントロールにイタリック書体を適用します。必要な場合は、ボールドイタリックコントロールにボールドイタリック書体を適用します。これらの書体ごとに、SWF ファイルに追加する文字範囲を選択します。

   ステージは最終的に次の例のようになります。



10. [ファイル]-[書き出し]-[ムービーの書き出し] を選択します。

Flex アプリケーションに読み込む SWF ファイルが生成されます。

SWF ファイルから Flex アプリケーションへのフォントの埋め込み

SWF ファイルから Flex アプリケーションに FlashType フォントを埋め込む処理は 2 つの手順で構成されます。このセクションでは、その 2 番目の手順について説明します。このセクションの手順を実行する前に、フォントを埋め込む SWF ファイルを Flash で作成しておく必要があります。詳細については、730 ページの「埋め込みフォントを使用した Flash 8 ファイルの作成」を参照してください。

Flash 8 の SWF ファイルから Flex アプリケーションにフォントを埋め込む手順は次のとおりです。

1. 次の例に示すように、`@font-face` 宣言を使用して各書体を埋め込みます。

```
/* assets/FlashTypeStyles.css */
@font-face {
    src:url("../assets/MyriadWebProEmbed.swf");
    fontFamily: "Myriad Web Pro";
}
@font-face {
    src:url("../assets/MyriadWebProEmbed.swf");
    fontFamily: "Myriad Web Pro";
    fontWeight: bold;
}
@font-face {
    src:url("../assets/MyriadWebProEmbed.swf");
    fontFamily: "Myriad Web Pro";
    fontStyle: italic;
}
```

`src` 属性を使用して、SWF ファイルの場所を指定します。`fontFamily` 属性の値を、使用できるフォントのリストとして Flash に表示されるフォント名に設定します。書体が標準でない場合は、書体のプロパティごとに新しい `@font-face` エントリを指定する必要があります。`@font-face` 宣言の使用の詳細については、710 ページの「埋め込みフォントのシンタックス」を参照してください。

`@font-face` 宣言で `flashType` プロパティの値を指定しないでください。Flash のアンチエイリアス設定によって、FlashType ヒント情報を含めるかどうかが決まるからです。フォントが含まれる SWF ファイルを作成した後は、Flex コンパイラや Flex アプリケーションでヒント情報を SWF ファイルに追加することはできません。FlashType ヒントの使用の詳細については、716 ページの「FlashType ヒントの使用」を参照してください。

2. 埋め込みフォントの書体ごとにスタイルを定義します。このスタイルは外部スタイルシートとして定義できるほか、次の例に示すように、<mx:Style> ブロックで定義することもできます。

```
/* assets/FlashTypeClassSelectors.css */
.myPlainStyle {
    fontFamily: "Myriad Web Pro";
    fontSize: 24;
}
.myItalicStyle {
    fontFamily: "Myriad Web Pro";
    fontSize: 24;
    fontStyle: italic;
}
.myBoldStyle {
    fontFamily: "Myriad Web Pro";
    fontSize: 24;
    fontWeight: bold;
}
```

スタイル定義で fontFamily プロパティを指定する必要があります。このプロパティの値は、@font-face 宣言で設定されている fontFamily プロパティの値と一致していることが必要です。定義する書体に関係なく、fontFamily プロパティの値は同一にします。たとえば、書体をボールドにする場合、fontFamily プロパティは “Myriad Web Pro Bold” ではなく、“Myriad Web Pro” に設定します。

また、@font-face 宣言と同様に、スタイル定義ですべての書体プロパティを指定することも必要です。

3. 次の例に示すように、Flex コントロールに新しいスタイルを適用します。

```
<?xml version="1.0"?>
<!-- fonts/EmbedFlashTypeFonts.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style source="../assets/FlashTypeStyles.css"/>
  <mx:Style source="../assets/FlashTypeClassSelectors.css"/>

  <mx:Panel title="Embedded Font">
     <mx:VBox>
        <mx:Label text="Plain Label" styleName="myPlainStyle"/>
        <mx:Label text="Italic Label" styleName="myBoldStyle"/>
        <mx:Label text="Bold Label" styleName="myItalicStyle"/>
     </mx:VBox>
  </mx:Panel>
</mx:Application>
```

スタイルをスタイルシートで定義し、styleName プロパティを使用して適用する代わりに、スタイルをインラインで定義して適用することもできます。次の例では、fontFamily プロパティと fontStyle プロパティの値をインラインで設定し、Myriad Web Pro フォントのイタリックボールド書体を Label コントロールに適用します。

```
<?xml version="1.0"?>
<!-- fonts/EmbedFlashTypeFontsInline.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style source="../assets/FlashTypeStyles.css"/>

  <mx:Panel title="Embedded FlashType Font">
     <mx:VBox>
        <mx:Label text="Plain Label" fontFamily="Myriad Web Pro"
fontSize="24"/>
        <mx:Label text="Italic Label" fontFamily="Myriad Web Pro"
fontStyle="italic" fontSize="24"/>
        <mx:Label text="Bold Label" fontFamily="Myriad Web Pro"
fontWeight="bold" fontSize="24"/>
     </mx:VBox>
  </mx:Panel>
</mx:Application>
```

次の例では、Myriad Web Pro フォントを Flex アプリケーションに埋め込みます。各書体を埋め込み、それら書体のスタイルを定義した上で、Text コントロールにこれらのスタイルを適用します。

```
<?xml version="1.0"?>
<!-- fonts/EmbedFlashTypeFontsFull.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     @font-face {
        src:url("../assets/MyriadWebProEmbed.swf");
        fontFamily: "Myriad Web Pro";
     }
     @font-face {
        src:url("../assets/MyriadWebProEmbed.swf");
        fontFamily: "Myriad Web Pro";
        fontWeight: bold;
     }
     @font-face {
        src:url("../assets/MyriadWebProEmbed.swf");
        fontFamily: "Myriad Web Pro";
        fontStyle: italic;
     }
     .myPlainStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
     }
     .myItalicStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
        fontStyle: italic;
     }
     .myBoldStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
        fontWeight: bold;
     }
```

```
  </mx:Style>

  <mx:Panel title="Embedded SWF-based Font">
     <mx:VBox>
        <!-- Apply each custom style to Text controls. -->
        <mx:Text id="text1" styleName="myPlainStyle" text="Plain Text"/>
        <mx:Text id="text2" styleName="myBoldStyle" text="Bold Text"/>
        <mx:Text id="text3" styleName="myItalicStyle" text="Italic Text"/>
     </mx:VBox>
  </mx:Panel>

  <!-- Rotate the Text controls. If the text disappears when the control is
     rotated, then the font is not properly embedded. -->
  <mx:Button label="Rotate +1" click="++text1.rotation;++text2.rotation;
     ++text3.rotation;"/>
  <mx:Button label="Rotate -1" click="--text1.rotation;--text2.rotation;
     --text3.rotation;"/>
</mx:Application>
```

この例を実行するときは、Button コントロールをクリックしてテキストを回転させます。これによって、フォントが正しく埋め込まれていることを確認できます。フォントの埋め込みが正しくない場合は、回転させたテキストが消失します。

@font-face CSS シンタックスを使用する代わりに、[Embed] メタデータキーワードを使用して Flex アプリケーションに SWF ファイルからフォントを埋め込むこともできます。これを利用すると、ActionScript でのフォントの制御幅が広がります。そうするには、前の例で作成したものと同じ SWF ファイルを使用します。Flex アプリケーションで、次の例に示すように各書体をそれぞれの変数に関連付けます。

```
<?xml version="1.0"?>
<!-- fonts/EmbedFlashTypeFontsActionScriptSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myPlainStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
     }
     .myItalicStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
        fontStyle: italic;
     }
     .myBoldStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
        fontWeight: bold;
     }
  </mx:Style>

  <mx:Script><![CDATA[
```

```
        [Embed(source='../assets/MyriadWebProEmbed.swf',
            fontName='Myriad Web Pro'
        )]
        private static var plainFont:Class;

        [Embed(source='../assets/MyriadWebProEmbedWithFontSymbols.swf',
            fontName='Myriad Web Pro',
            fontStyle='italic'
        )]
        private static var italicFont:Class;

        [Embed(source='../assets/MyriadWebProEmbedWithFontSymbols.swf',
            fontName='Myriad Web Pro',
            fontWeight='bold'
        )]
        private static var boldFont:Class;

  ]]></mx:Script>

  <mx:Panel title="Embedded SWF-Based Font">
     <mx:VBox>
        <!-- Apply each custom style to Text controls. -->
        <mx:Text id="text1" styleName="myPlainStyle" text="Plain Text"/>
        <mx:Text id="text2" styleName="myBoldStyle" text="Bold Text"/>
        <mx:Text id="text3" styleName="myItalicStyle" text="Italic Text"/>
     </mx:VBox>
  </mx:Panel>

  <!-- Rotate the Text controls. If the text disappears when the control is
     rotated, then the font is not properly embedded. -->
  <mx:Button
    label="Rotate +1"
    click="++text1.rotation;++text2.rotation;++text3.rotation;"
  />
  <mx:Button
    label="Rotate -1"
    click="--text1.rotation;--text2.rotation;--text3.rotation;"
  />
</mx:Application>
```

Class 型の変数を定義し、Flex アプリケーションの SWF ファイルにフォントがコンパイラでリンクされるようにする必要があります。この例では、静的変数 fontPlain、fontBold、および fontItalic の値を設定しますが、これらの値はアプリケーションのこれ以降の部分では使用されません。

[Embed] ステートメントを使用して Flex アプリケーションにフォントを埋め込む場合でも、それらのフォントのスタイルを定義する必要があります。これは、@font-face 宣言を使用してフォントを埋め込む場合と同様です。この処理によって、Flex アプリケーションにこれらのスタイルが適用されるようになります。

フォントシンボルとして SWF ファイルからのフォントにアクセスすることもできます。

フォントシンボルである SWF ファイルからのフォントを使用するには :

1. Flash 8 で、[ウィンドウ]-[ライブラリ] を選択してライブラリの内容を表示します。内容は空です。
2. マウスをライブラリの上に置いて右クリックし、[新しいフォント] を選択して新しいフォントシンボルを作成します。
3. [Font Symbol Properties] ダイアログボックスで、新しいフォントシンボルに名前を付け、適用可能なフォントシンボルプロパティ (ボールドやイタリックなど) を選択します。ここで指定するシンボル名は、[Embed] ステートメントで使用します。フォントの書体 (標準、ボールド、イタリック、ボールドイタリックなど) ごとにこの操作を繰り返します。
4. ライブラリ内のフォントシンボルを作成した後、ライブラリ内のシンボルを右クリックし、[リンケージ] を選択します。
5. [リンケージプロパティ] ダイアログボックスで、[ActionScript に書き出し] を選択し、[OK] をクリックします。このダイアログボックスの [Identifier] は、前のステップで指定したシンボル名と同じになっているはずです。ライブラリ内のフォントシンボルごとにこの操作を繰り返します。
6. Flex アプリケーションの [Embed] ステートメントでは、シンボルの属性を使用してフォントシンボルを指します。[Embed] ステートメントでは、MIME タイプ、fontStyle、fontWeight、fontName などの他の特性を指定しないでください。

次の例では、Flash 8 からライブラリ内のフォントシンボルとともに書き出された Myriad Web Pro フォントを埋め込みます。

```
<?xml version="1.0"?>
<!-- fonts/EmbedFlashTypeFontsActionScript.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     .myPlainStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
     }
     .myItalicStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
        fontStyle: italic;
     }
     .myBoldStyle {
        fontFamily: "Myriad Web Pro";
        fontSize: 24;
        fontWeight: bold;
     }
  </mx:Style>

  <mx:Script><![CDATA[

        [Embed(source='../assets/MyriadWebProEmbedWithFontSymbols.swf',
```

```
            symbol='MyriadPlain'
        )]
        private var font1:Class;

        [Embed(source='../assets/MyriadWebProEmbedWithFontSymbols.swf',
            symbol='MyriadBold'
        )]
        private var font2:Class;

        [Embed(source='../assets/MyriadWebProEmbedWithFontSymbols.swf',
            symbol='MyriadItalic'
        )]
        private var font3:Class;

  ]]></mx:Script>

  <mx:Panel title="Embedded SWF-based Font">
     <mx:VBox>
        <!-- Apply each custom style to Text controls. -->
        <mx:Text id="text1"
            styleName="myPlainStyle"
            text="Plain Text"
        />
        <mx:Text id="text2"
            styleName="myBoldStyle"
            text="Bold Text"
        />
        <mx:Text id="text3"
            styleName="myItalicStyle"
            text="Italic Text"
        />
     </mx:VBox>
  </mx:Panel>

  <!-- Rotate the Text controls. If the text disappears when the control is
     rotated, then the font is not properly embedded. -->
  <mx:Button
    label="Rotate +1"
    click="++text1.rotation;++text2.rotation;++text3.rotation;"
  />
  <mx:Button
    label="Rotate -1"
    click="--text1.rotation;--text2.rotation;--text3.rotation;"
  />
</mx:Application>
```

トラブルシューティング

ここでは、Flex アプリケーションにフォントを正しく埋め込むために利用できる方法について説明します。

コンパイラエラーの解決

次の表は、よく見られるコンパイラエラーとその解決方法を示しています。

エラー	解決方法
トランスコーディングで 'swf_file_name' を解決できない。	これは、目的のフォントがコンパイラから見つからないことを示しています。`@font-face` 宣言または `[Embed]` タグに記述したフォントへのパスが正しいことを確認します。また、そのパスがコンパイラからアクセス可能であることを確認します。`local` プロパティを使用してフォントの位置を `@font-face` 宣言で指定している場合は、目的のフォントがローカルでアクセス可能であることを確認します。その位置が "fonts.properties" ファイルのエントリであること、またはフォントがシステムフォントの検索パスに存在することを確認します。詳細については、710 ページの「埋め込みフォントのシンタックス」を参照してください。
style_description を持つフォント 'font_name' が見つからない。	これは、`[Embed]` ステートメントで使用している `fontName` パラメータが、実際のフォント名と一致していない可能性があることを示しています。 SWF ファイル内のフォントを使用している場合は、Flash で使用できるフォントのリストにあるフォント名が、`[Embed]` ステートメントの `fontName` 属性、およびスタイル定義で使用している `fontFamily` プロパティと、綴りや空白の使用などの面で一致していることを確認します。 フォントのスタイルが Flash で正しく埋め込まれていないために、このエラーが発生することもあります。FLA ファイルを開き、フォントとスタイルを記述したテキスト領域が存在すること、そのテキストがダイナミックテキストであること、およびそのテキストについて文字範囲を選択していることを確認します。

ランタイムエラーの解決

フォントが正しく埋め込まれていることを確認するには、そのフォントを使用しているコントロールを回転させてみます。フォントが埋め込まれていれば、フォントも回転して表示されます。コントロールを回転させたときにテキストが消失する場合は、フォントが正しく埋め込まれていません。

フォントを正しく埋め込むには、次の方法を試します。

- 特定の 1 種類のコントロールでテキストが正しく表示されない場合は、適切な書体を埋め込んでいるかどうかを確認します。たとえば、Button コントロールのテキストラベルではボールド書体が必要です。ボールド書体を埋め込んでいないと、Button コントロールには埋め込みフォントが表示されません。
- Flex アプリケーションで、各フォント書体のすべてのプロパティを @font-face 宣言または [Embed] ステートメントで設定していることを確認します。ボールド書体を埋め込むには、次の例で示すように、fontWeight プロパティを bold に設定します。

```
@font-face {
  src: url(../assets/MyriadWebProEmbed.ttf);
  fontFamily: "Myriad Web Pro";
  fontWeight: bold;
}
```

スタイル定義で fontWeight スタイルプロパティを設定する必要もあります。

```
.myStyle2 {
  fontFamily:"Myriad Web Pro";
  fontWeight:bold;
  fontSize:12pt;
}
```

[Embed] ステートメントを使用している場合は、次の例に示すように、fontWeight プロパティを bold に設定します。

```
[Embed(source="MyriadWebProEmbed.ttf", fontName="Myriad Web Pro",
  fontWeight="bold")]
```

- SWF ファイル内のフォントを使用している場合は、Flash で FLA ファイルを開き、すべての書体を正しく追加していることを確認します。各テキスト領域を選択して、次の方法を試します。
  - フォント名が正しいことを確認します。Flash で使用できるフォントのリストにあるフォント名が、@font-face 宣言の fontFamily プロパティ、または [Embed] ステートメントの fontName 属性と、綴りや空白の使用などの面で一致していることを確認します。この値が、スタイル定義で使用している fontFamily プロパティと一致していることも必要です。
  - スタイルを正しく適用していることを確認します。たとえば、ボールド書体のテキスト領域を選択して、書体が実際にボールドになっていることを確認します。
  - [埋め込み] ボタンをクリックし、Flex アプリケーションで使用する文字が、埋め込み文字範囲に含まれていることを確認します。
  - 各テキスト領域の設定が、静的テキストや入力テキストではなく、ダイナミックテキストになっていることを確認します。テキストの型は、テキストの [プロパティ] タブにある最初のドロップダウンボックスに表示されています。
- SWF ファイル内のフォントの場合は、使用している SWF ファイルが最新であること、目的のフォントがその SWF ファイルに収められていること、その SWF ファイルが Flash で生成されたものであることを確認します。必要に応じ、Flash で SWF ファイルを生成し直します。

第 20 章

スキンの使用

20

このトピックでは、ActionScript クラスファイルまたはイメージファイルを使用して Adobe Flex コンポーネントにスキンを追加する方法について説明します。

目次

スキニングについて

" スキニング " とは、コンポーネントのビジュアルエレメントを修正または置換することで外観を変更する処理です。これらのエレメントを構成するのは、イメージ、SWF ファイル、または描画 API メソッドが含まれるクラスファイルなどです。

スキンでは、さまざまな状態のコンポーネント全体またはその一部が定義されます。たとえば、Button コントロールには多くのスキンがあり、各スキンはボタンの状態を表しています。スキンには downSkin、upSkin、および overSkin があり、それぞれボタンを押したとき、離したとき、およびマウスをボタン上に置いたときにボタンがどのように表示されるかを示します。これらの 3 つのデフォルトスキンは、次のように表示されます。

upSkin　overSkin　downSkin

他のコントロールにも同様の状態があります。たとえば、Button を継承した RadioButton コントロールおよび RadioButton コントロールにも、upSkin、downSkin、および overSkin プロパティがあります。ComboBox などのコントロールにも、コントロールが特定の状態になったときのコントロールの外観を定義するスキンがあります。これらのスキンには、disbaledSkin、downSkin および overSkin があります。Accordion のような Button とは直接の階層関係がないコントロールには、borderSkin や focusSkin などのスキンがあります。

スキンはスタイルプロパティとして適用されます。スキンは、MXML タグプロパティ、StyleManager クラス、<mx:Style> ブロックを使用して適用するか、スタイルシートで適用します。複雑なスキニングでは、通常、スキンを整理して Flex アプリケーションに適用するためスタイルシートを使用します。スタイルシートは、コンパイル時にロードするか、実行時にロードします。実行時におけるスタイルシートのロードの詳細については、688 ページの「実行時のスタイルシートのロード」を参照してください。

スキンには、グラフィカルスキンとプログラムスキンの 2 種類があります。" グラフィカルスキン " はスキンの外観を定義するイメージです。これらのイメージとして JPEG ファイル、GIF ファイル、PNG ファイルを使用できるほか、SWF ファイルに埋め込んだシンボルを使用することもできます。

プログラムスキンは、ActionScript ステートメントを使用して描画するスキンであり、クラスファイルで定義します。プログラムスキンを使用するコントロールの外観を変更するには、ActionScript クラスのファイルを編集し、Flex アプリケーションを使用してコンパイルします。

プログラムスキンの利点の 1 つは、スタイルをより柔軟に制御できることです。たとえば、スタイルシートやグラフィカルスキンでは、Button コントロールの角の丸み半径を制御することはできません。Adobe Flash などのグラフィックツールを使用しなくても、Flex オーサリング環境や任意のテキストエディタで、プログラムスキンを直接開発することもできます。プログラムスキンには外部のイメージファイルが組み込まれていないので、メモリの消費量が少ないという特性があります。

アイコンについて

アイコンは、スキンと同様に扱われる特殊なスタイルプロパティです。アイコンスタイルプロパティ (icon、upIcon、および downIcon など) は、コントロールに追加されるビジュアルエレメントを定義します。このビジュアルエレメントはコンポーネントスキンの外観を置換するのではなく、そのスキンに追加されます。

Button コントロールの場合、アイコンによりボタンの外観は強調されます。アイコンはボタンに追加され、ボタンラベルの横に表示されます。CheckBox と RadioButton コントロールの場合、コントロール上のチェックボックスやボタンの外観がアイコンに置き換えられます。ただし、コントロールラベルの外観には影響しません。

リストベースのコントロールの場合、アイコンは、リスト内におけるアイテムのグラフィカル表現の外観を定義します。コンテナの場合、アイコンは、一部のナビゲータコンテナ内 (TabNavigator のタブ、Accordion のヘッダ内など) のコンテナシンボルを定義します。

ボタンの場合、アイコンプロパティを設定すると、upIcon や downIcon などの他のすべてのアイコン設定はオーバーライドされます。

スキン用のリソース

Flex 2 には、次に示すスキン用のグラフィックソースファイルおよびプログラムソースファイルがあります。

mx.skins パッケージの基本スキンクラス　これらの抽象スキンクラスでは、Flex におけるスキンクラスの基本的な機能を定義します。詳細については、751 ページの「プログラムスキン」を参照してください。

mx.skins.halo パッケージのプログラム Halo スキン　これらの具象スキンクラスは、mx.skins パッケージの基本スキンクラスを拡張したものです。これらのスキンを拡張または編集し、Flex のデフォルトの外観と操作性に基づいて、新規のプログラムスキンを作成できます。詳細については、751 ページの「プログラムスキン」を参照してください。

HaloClassic スキン　これらのスキンは Flex 1.x で使用されていたものです。これらのスキンを使用すると、Flex 1.x アプリケーションの外観を維持できます。"HaloClassic.swc" ファイルは、framework/themes ディレクトリ内にあります。詳細については、698 ページの「テーマについて」を参照してください。

グラフィカル Aeon テーマ　Aeon テーマは、"AeonGraphical.css" ファイル、およびスキンシンボルを定義する "AeonGraphical.swf" ファイルで構成されます。これらのファイルは、"framework/themes" ディレクトリにあります。さらに、Flex は、"AeonGraphical.swf" ファイルの FLA ソースファイルも備えています。詳細については、698 ページの「テーマについて」を参照してください。

Flex の外観と操作性を基にこれらのファイルを使用してスキンを作成するか、このトピックの指示に従って独自のスキンを作成できます。

スキニングの詳細については、www.adobe.com/go/flex2_skinning_jp を参照してください。

グラフィカルスキニング

グラフィカルスキンを使用するには、Flex アプリケーションにイメージファイルを埋め込みます。次に、これらのファイルをスタイルプロパティとして定義し、これを使用してコンポーネントの既存のスキンを置き換えます。

たとえば、Button コントロールの外観を変更するには、"orb_up_skin.gif"、"orb_down_skin.gif"、"orb_over_skin.gif" という 3 つのイメージファイルを作成します。


orb_up_skin.gif


orb_over_skin.gif


orb_down_skin.gif

スタイルシートでは、Button コントロールの `upSkin`、`downSkin`、および `overSkin` の各スタイルプロパティの値を定義し、これらのイメージファイルを指定します。Button コントロールには、これらの新スキンが反映されます。

```
<?xml version="1.0"?>
<!-- skins/EmbedImagesTypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        overSkin: Embed("../assets/orb_over_skin.gif");
        upSkin: Embed("../assets/orb_up_skin.gif");
        downSkin: Embed("../assets/orb_down_skin.gif");
     }
  </mx:Style>
  <mx:Button id="b1" label="Click Me"/>
</mx:Application>
```

スキンはスタイルプロパティとして適用するものですが、それぞれのスキンはクラスとして考える必要があります。Button コントロールの upSkin プロパティの値を、たとえば “myimage.jpg” に設定すると、実際には、参照先を現在のスキンクラスからこの新しいクラスに変更したことになります。Flex では、埋め込まれた各アセットが、現在のドキュメントのクラスに内部クラスとしてラップされます。

スキンは、Flex アプリケーションの最終的な SWF ファイルに埋め込まれます。スキンを構成するグラフィックファイルを変更した場合は、変更を有効にするため、Flex アプリケーションを再コンパイルする必要があります。

スキンプロパティは、次の方法で定義できます。

- カスケードスタイルシート (CSS ファイル)。詳細については、745 ページの「スタイルシートを使用したスキンの適用」を参照してください。
- インライン。詳細については、747 ページの「インラインでのスキンの適用」を参照してください。
- setStyle() メソッド。詳細については、747 ページの「setStyle() メソッドの使用」を参照してください。

イメージをスキンとして埋め込むことの欠点は、スキンを持つコンポーネントのサイズを変更するときに、イメージが変形する可能性があるという点です。scale-9 という手法を使用すると、コンポーネントのサイズを変更しても変形しないスキンを作成できます。scale-9 手法の詳細については、1025 ページの「埋め込みイメージでの Scale-9 フォーマットシステムの使用」を参照してください。

グラフィカルスキンのアセットは、目的のアプリケーションの SWF ファイルに " 埋め込んでおく " 必要があります。その理由は、コンポーネントの最小サイズと優先サイズを判別するには、コンポーネントが作成された直後にスキンアセットが存在している必要があるためです。実行時に外部アセットを参照する場合、Flex にはサイズを決定するための情報がないので、スキンを正しくレンタリングできません。

次の例は、@Embed キーワードをインラインで使用して外部アセットを埋め込む方法を示しています。

```
<?xml version="1.0"?>
<!-- skins/EmbedImagesInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Button id="b1"
     label="Click Me"
```

```
    overSkin="@Embed(source='../assets/orb_over_skin.gif')"
    upSkin="@Embed(source='../assets/orb_up_skin.gif')"
    downSkin="@Embed(source='../assets/orb_down_skin.gif')"
  />
</mx:Application>
```

スタイルシートを使用したスキンの適用

color や fontSize などの通常のスタイルプロパティを適用する場合と同じように、<mx:Style> タグまたは外部スタイルシート内で、CSS ファイルのプロパティとしてスキンを埋め込むことができます。

CSS を使用してスキンを適用するときは、タイプセレクタまたはクラスセレクタを使用できます。これにより、1 つのコンポーネントのみにスキンを適用することもできれば、同じタイプの全コンポーネントにスキンを適用することもできます。

次の例では、Button タイプセレクタでスキンを定義します。この例では、すべての Buttons コントロールが新しいスキン定義を取得します。

```
<?xml version="1.0"?>
<!-- skins/EmbedImagesTypeSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
    Button {
      overSkin: Embed("../assets/orb_over_skin.gif");
      upSkin: Embed("../assets/orb_up_skin.gif");
      downSkin: Embed("../assets/orb_down_skin.gif");
    }
  </mx:Style>
  <mx:Button id="b1" label="Click Me"/>
</mx:Application>
```

クラスセレクタを定義することにより、コンポーネントの単一のインスタンスにスキンを適用することもできます。次の例では、カスタムスタイルを 2 番目のボタンにのみ適用します。

```
<?xml version="1.0"?>
<!-- skins/EmbedImagesClassSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
    .myButtonStyle {
      overSkin: Embed("../assets/orb_over_skin.gif");
      upSkin: Embed("../assets/orb_up_skin.gif");
      downSkin: Embed("../assets/orb_down_skin.gif");
    }
  </mx:Style>
  <mx:Button id="b1" label="Click Me"/>
  <mx:Button id="b2" label="Click Me" styleName="myButtonStyle"/>
</mx:Application>
```

場合によっては、サブコンポーネントのスキンを交換することもできます。次の例では、List コントロール内に表示される垂直 ScrollBar コントロールのスキンを交換します。

```
<?xml version="1.0"?>
<!-- skins/SubComponentSkins.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     [Bindable]
     private var theText:String = "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. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
id est laborum.";
  ]]></mx:Script>

  <mx:Style>
     .myScrollStyle {
        upArrowUpSkin: Embed("../assets/uparrow_up_skin.gif");
       downArrowUpSkin: Embed("../assets/downarrow_up_skin.gif");
     }
  </mx:Style>

  <mx:TextArea id="ta1"
     width="400"
     height="50"
     verticalScrollPolicy="on"
     verticalScrollBarStyleName="myScrollStyle"
     text="{theText}"
  />

</mx:Application>
```

List タイプセレクタで verticalScrollBarStyleName プロパティの値を設定すると、List コンポーネントのすべての垂直 ScrollBar コントロールに 1 つのカスタムスキンが適用されます。アプリケーションの他の部分の ScrollBar コントロールには、カスタムスキンは追加されません。

すべてのスタイルシートの場合と同様、独立した CSS ファイルでスキンを定義して、<mx:Style> タグの source 属性で目的のファイルを指定できます。次に例を示します。

```
<mx:Style source="../stylesheets/MySkins.css"/>
```

これらを SWF ファイルにコンパイルし、実行時にスタイルシートをロードすることもできます。それから、StyleManager クラスの loadStyleDeclarations() メソッドを使用して、実行時に CSS ベースの SWF ファイルをロードします。次に例を示します。

```
<?xml version="1.0"?>
<!-- styles/runtime/BasicApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
```

```
        import mx.styles.StyleManager;

        public function applyRuntimeStyleSheet():void {
            StyleManager.loadStyleDeclarations("assets/BasicStyles.swf")
        }
        ]]>
    </mx:Script>

    <mx:Label text="Click the button to load a new CSS-based SWF file"/>
    <mx:Button id="b1" label="Click Me" click="applyRuntimeStyleSheet()"/>

</mx:Application>
```

詳細については、688 ページの「実行時のスタイルシートのロード」を参照してください。

インラインでのスキンの適用

グラフィカルスキンをインラインで適用するには、次の例に示すとおり、コントロールのスキンプロパティを使用してスキンを埋め込みます。

```
<?xml version="1.0"?>
<!-- skins/EmbedImagesInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Button id="b1"
     label="Click Me"
     overSkin="@Embed(source='../assets/orb_over_skin.gif')"
     upSkin="@Embed(source='../assets/orb_up_skin.gif')"
     downSkin="@Embed(source='../assets/orb_down_skin.gif')"
  />
</mx:Application>
```

スキンアセットの場所は、それを埋め込む MXML ファイルの場所を基準として相対的に指定します。

インラインで埋め込むときは、CSS ファイルで使用する Embed ではなく、接頭辞として @ 記号を付けた @Embed を使用します。

setStyle() メソッドの使用

スキンはスタイルプロパティとして定義されるので、スキンには setStyle() メソッドおよび getStyle() メソッドでアクセスできます。これにより、コンパイル時にグラフィカルアセットを埋め込んでおけば、実行時にスキンを変更することも、動的にスキンを定義することもできます。

setStyle() メソッドで使用できるようにイメージを埋め込むには、[Embed] メタデータタグを使用して、変数に参照を割り当てます。それから、setStyle() メソッドを使用して、このイメージをスキンとしてコンポーネントに適用します。

次の例では、3 つのイメージを埋め込み、これらのイメージをスキンとして Button コントロールのインスタンスに適用します。

```
<?xml version="1.0"?>
```

```
<!-- skins/EmbedWithSetStyle.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="init()">
  <mx:Script><![CDATA[
     [Embed("../assets/orb_over_skin.gif")]
     public var os:Class;

     [Embed("../assets/orb_down_skin.gif")]
     public var ds:Class;

     [Embed("../assets/orb_up_skin.gif")]
     public var us:Class;

     private function init():void {
        b1.setStyle("upSkin",us);
        b1.setStyle("overSkin",os);
        b1.setStyle("downSkin",ds);
     }
  ]]></mx:Script>

  <mx:Button label="Click Me" id="b1"/>

</mx:Application>
```

スキンをコントロールの全インスタンスに適用するには、次の例のように StyleManager を使用します。

```
<?xml version="1.0"?>
<!-- skins/EmbedWithStyleManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="init()">
  <mx:Script><![CDATA[
     import mx.styles.StyleManager;

     [Embed("../assets/orb_over_skin.gif")]
     public var os:Class;

     [Embed("../assets/orb_down_skin.gif")]
     public var ds:Class;

     [Embed("../assets/orb_up_skin.gif")]
     public var us:Class;

     private function init():void {
        StyleManager.getStyleDeclaration("Button").setStyle("upSkin", us);
        StyleManager.getStyleDeclaration("Button").setStyle("overSkin", os);
        StyleManager.getStyleDeclaration("Button").setStyle("downSkin", ds);
     }
  ]]></mx:Script>

  <mx:Button label="Click Me" id="b1"/>

</mx:Application>
```

setStyle() メソッドの使用の詳細については、682 ページの「setStyle() メソッドと getStyle() メソッドの使用」を参照してください。StyleManager の使用方法の詳細については、677 ページの「StyleManager クラスの使用」を参照してください。

スキンとしての SWF ファイルの使用

SWF ファイルをスキンとして使用することができます。SWF ファイル全体を単一のスキンとして使用できるほか、SWF ファイル内の 1 つまたは複数のシンボルを 1 つのスキンとして使用することもできます。SWF ファイル全体を埋め込むには、次のように Embed ステートメントの source プロパティで SWF ファイルの場所を指定します。

```
<?xml version="1.0"?>
<!-- skins/EmbedSWFSource.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        upSkin: Embed(source="../assets/SubmitButtonUpSkin.swf");
     }
  </mx:Style>
  <mx:Button id="b1"/>
</mx:Application>
```

シンボルを SWF ファイルから読み込むには、source プロパティで SWF ファイルの場所を指定した上で、使用するシンボル名を symbol プロパティで指定します。Embed ステートメントの各属性は、カンマで区切る必要があります。

次の例では、Button コントロールの upSkin プロパティを SWF ファイル全体に置き換えます。ただし、overSkin と downSkin プロパティは、2 番目の SWF ファイルの個々のシンボルに置き換わります。

```
<?xml version="1.0"?>
<!-- skins/EmbedSymbolsCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        upSkin: Embed(source='../assets/SubmitButtonSkins.swf',
symbol='MyUpSkin');
        overSkin: Embed(source='../assets/SubmitButtonSkins.swf',
symbol='MyOverSkin');
        downSkin: Embed(source='../assets/SubmitButtonSkins.swf',
symbol='MyDownSkin');
     }
  </mx:Style>
  <mx:Button id="b1"/>
</mx:Application>
```

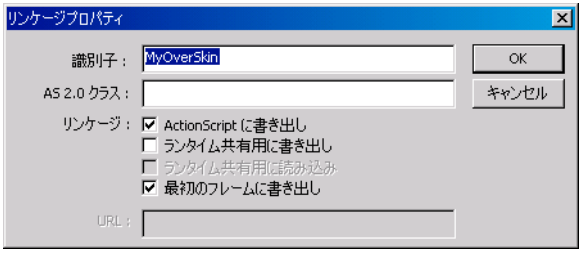
インラインでスキンシンボルを埋め込むときは、次に示すように同じシンタックスを使用します。

```
<?xml version="1.0"?>
<!-- skins/EmbedSymbolsInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Button id="b1"
     overSkin="@Embed(source='../assets/SubmitButtonSkins.swf',
symbol='MyOverSkin')"
     upSkin="@Embed(source='../assets/SubmitButtonSkins.swf',
symbol='MyUpSkin')"
```

```
    downSkin="@Embed(source='../assets/SubmitButtonSkins.swf',
symbol='MyDownSkin')"
  />
</mx:Application>
```

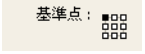
使用するすべてのシンボルは、ソース FLA ファイルの中で次の条件を満たしている必要があります。

- シンボルはステージ上に置かれている必要があります。イメージファイルを作成して、それをシンボルに変換した後、そのシンボルをライブラリからステージにドラッグする必要があります。Flash は、ステージ上にないシンボルは書き出しません。別の方法として、[リンケージプロパティ] ダイアログボックスの [最初のフレームに書き出し] オプションを選択することもできます。
- シンボルは、リンケージ名を指定して ActionScript 用に書き出されている必要があります。Flash で、シンボルの [リンケージプロパティ] ダイアログボックスの [ActionScript に書き出し] オプションを選択します。次に例を示します。



  リンケージ名は、Flex で使用される名前です。シンボル名は無視されます。

- FLA ファイルには、いずれの ActionScript も含めることはできません。
- シンボルの左上には、[シンボルに変換] ダイアログボックスで基準点を設定しておく必要があります。次の例は、左上の基準点を示しています。



- 9 つの領域を持つグリッドである scale-9 イメージファイルを使用すると、コンポーネントのサイズを変更しても、それに応じてスキンが正常に拡大縮小されます。scale-9 イメージの属性は、次の例で示すように、CSS でスキンを適用するときに定義します。

```
Button {
  overSkin: Embed(
    "../assets/orb_over_skin.gif",
    scaleGridTop=6,
    scaleGridLeft=12,
    scaleGridBottom=44,
    scaleGridRight=49
  );
}
```

  scale-9 の手法を使用するアセットの埋め込みの詳細については、1025 ページの「埋め込みイメージでの Scale-9 フォーマットシステムの使用」を参照してください。

プログラムスキン

このセクションでは、プログラムスキンを ActionScript クラスとして記述する方法、Flash Graphics (flash.display.Graphics) パッケージの基本描画メソッドを使用する方法、およびこれらのスキンを Flex コントロールに適用する方法について説明します。

Flex に付属のプログラムスキンを修正して使用できるほか、独自のスキンを作成することもできます。既存スキンの修正の詳細については、743 ページの「スキン用のリソース」を参照してください。プログラムスキンのレシピの使用方法の詳細については、752 ページの「プログラムスキンのレシピ」を参照してください。

Flex コンポーネントのスキンを交換する最も簡単な方法は、既存スキンを使用して、CSS を使用してそのスキンを適用することです。mx.skins.halo パッケージに含まれるすべてのスキンが使用可能です。これらのクラスを他のコントロールに適用することもできます。たとえば、CheckBox を RadioButton のように見せるため、次の例に示すように、mx.skins.halo.RadioButton スキンを CheckBox コントロールのさまざまな状態に適用することができます。

```
<?xml version="1.0"?>
<!-- skins/UseHaloSkins.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     RadioButton {
        /*
           Defaults. Shown here for illustrative purposes.
        */
        disabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        downIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        overIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedDisabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedDownIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedOverIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedUpIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        upIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
     }

     CheckBox {
        /*
           Applies the RadioButtonIcon skin to the CheckBox component's states.
        */
        disabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        downIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        overIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedDownIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedDisabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedOverIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        selectedUpIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
        upIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
     }
  </mx:Style>
```

```
  <mx:CheckBox id="cb1" label="Click Me"/>
  <mx:RadioButton id="rb1" label="Click Me"/>

</mx:Application>
```

コンポーネントがプログラムスキンを使用するすべての状態を確認するには、"defaults.css" ファイルのコンポーネントの項目を参照してください。

既存のプログラムスキンを、デフォルトでそのスキンクラスを使用しないコンポーネントに適用する場合、そのコンポーネントが、意図しているスキンオーナーと同じ状態 (downIcon、upIcon など) をサポートしている必要があります。それ以外の場合は、カスタムプログラムスキンクラスを作成する必要があります。カスタムプログラムスキンの作成の詳細については、752 ページの「プログラムスキンのレシピ」を参照してください。

カスタムプログラムスキンを作成するときは、コンパイルする必要があります。768 ページの「プログラムスキンのコンパイル」を参照してください。

コンパイルしたプログラムスキンは、Flex アプリケーションのコンポーネントに適用できます。CSS を使用してスキンを適用するか、インラインシンタックスを使用するか、`setStyle()` メソッドを使用することができます。詳細については、772 ページの「プログラムスキンの適用」を参照してください。

Flex コンポーネントで使用するプログラムスキンは、mx.skins.halo パッケージに収められています。これらのスキンはすべて、抽象基本クラスのいずれかを拡張したものです。抽象基本クラスには、ProgrammaticSkin、Border、および RectangularBorder があります。たとえば、Button コントロールの外観と操作性を変更するには、抽象クラス ProgrammaticSkin を拡張して、Button コントロールとしてのロジックを追加します。

このセクションで説明した手法を使用すると、基本スキンクラスのほか、デフォルトの Halo テーマにあるあらゆるスキンを拡張できます。

プログラムスキンのレシピ

プログラムスキンのクラスは、少なくとも、コンストラクタ、`updateDisplayList()` メソッド、およびスキンプロパティ用の getter と setter で構成されています。一般的に、プログラムスキンは mx.skins パッケージの抽象クラスのいずれかを拡張したものです。

プログラムスキンのレシピに従ったスキンの例を見るには、mx.skins.halo パッケージの具象クラスを参照してください。このクラスには、Flex コンポーネントで使用されるスキンが収められています。これらのスキンは、ここで紹介するものと同じレシピに従っています。

次の例は、プログラムスキンの代表的なアウトラインを示しています。

```
// skins/MySkinOutline.as

package { // Use unnamed package if this skin is not in its own package.

  // Import necessary classes here.
```

```
  import flash.display.Graphics;
  import mx.skins.Border;
  import mx.skins.ProgrammaticSkin;
  import mx.styles.StyleManager;

  public class MySkinOutline extends ProgrammaticSkin {

     // Constructor.
     public function MySkinOutline():void {
        // Set default values here.
     }

     override protected function updateDisplayList(w:Number,
        h:Number):void {
        // Add styleable properties here.
        // Add logic to detect components state and set properties here.
        // Add drawing methods here.
     }
  }
} // Close unnamed package.
```

Flex アプリケーションでは、CSS で ClassReference ステートメントを使用して、プログラムスキンを適用できます。

```
<?xml version="1.0"?>
<!-- skins/ApplyMySkinOutline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        overSkin: ClassReference("MySkinOutline");
        upSkin: ClassReference("MySkinOutline");
        downSkin: ClassReference("MySkinOutline");
     }
  </mx:Style>
  <mx:Button id="b1" label="Click Me"/>
</mx:Application>
```

他のメソッドを使用して、Flex アプリケーションにプログラムスキンを適用することもできます。詳細については、772 ページの「プログラムスキンの適用」を参照してください。

多くの場合、プログラムスキンでは、複数の状態のコントロールのためにスキンの外観を定義します。たとえば、1 つのボタンには upSkin、downSkin、および overSkin の各状態用に別個のスキンを定義できます。これらはそれぞれ、デフォルトの状態、ボタンを押した状態、およびマウスをボタン上に置いた状態を表しています。

次の手順のリストに従って、実際の Flex コントロール用のプログラムスキンを作成してください。各手順については、続く項で詳しく説明します。

Flex コントロールのプログラムスキンを作成するには :

1. 次の抽象クラスのいずれかを、目的のプログラムスキンのスーパークラスとして選択します。
   - ProgrammaticSkin
   - Border
   - RectangularBorder

   mx.skins.Halo パッケージにある具象クラスのいずれかを拡張することもできます。詳細については、754 ページの「スーパークラスの選択」を参照してください。
2. `updateDisplayList()` メソッドを実装します。このメソッドに、すべての描画やスタイル変更の呼び出しを入れます。

   詳細については、755 ページの「updateDisplayList() メソッドの実装」を参照してください。
3. (オプション) `measuredWidth` プロパティおよび `measuredHeight` プロパティの getter を実装します。

   詳細については、759 ページの「measuredWidth および measuredHeight の getter の実装」を参照してください。
4. スキンが Border または RectangularBorder のサブクラスである場合は、`borderMetrics` プロパティの getter を実装します。

   詳細については、760 ページの「borderMetrics プロパティの getter の実装」を参照してください。
5. プロパティをスタイル対応にします。

   作成したスキンのプロパティを、ユーザーが CSS または `setStyle()` メソッドへの呼び出しを使用して設定できるようにするには、スキンクラスにコードを追加する必要があります。詳細については、762 ページの「スタイル対応プロパティの作成」を参照してください。

スキンクラスを操作している場合は、プログラムスキンの親への参照を取得できます。この参照を使用すると、親コンポーネントのプロパティへのアクセスや、親コンポーネントに対するメソッドの呼び出しが可能になります。詳細については、765 ページの「親コンポーネントへのアクセス」を参照してください。

スーパークラスの選択

プログラムスキン作成の最初の手順は、スキン用のスーパークラスを選択することです。

mx.skins パッケージの抽象基本クラスまたは mx.skins.halo パッケージの具象クラスを拡張できます。mx.skins パッケージの抽象基本クラスを拡張すると、スキンの外観と操作性を広範囲に制御できます。Flex コンポーネントのデフォルト動作を使用した上で、スタイルをいくつか追加するような場合は、mx.skins.halo パッケージの具象クラスを拡張する方法が適しています。

このセクションでは、mx.skins パッケージにある抽象基本スキンクラスのいずれかを拡張する方法について説明します。ここで説明する手法は、mx.skins.halo パッケージの具象クラスにも適用できます。

ほとんどのスキンは mx.skins.ProgrammaticSkin クラスを拡張したものですが、次に示す任意のクラスと、スキン用のスーパークラスとして選択することができます。

- ProgrammaticSkin クラスは、IFlexDisplayObject、ILayoutManagerClient、IInvalidating、および ISimpleStyleClient の各インターフェイスを実装しているので、最も扱いやすく、広く使用されるスーパークラスです。
- Border クラスは、ProgrammaticSkin クラスを拡張して borderMetrics プロパティのサポートを追加したものです。スキンでコンポーネントの境界線を定義する場合は、このクラスまたは RectangularBorder クラスを使用します。
- RectangularBorder クラスは、Border クラスを拡張して backgroundImage スタイルに対するサポートを追加したものです。

プログラムスキンクラスは、IFlexDisplayObject、IInvalidating、および ILayoutManagerClient の各インターフェイスを実装する必要があります。スキンの外観が CSS スタイルの値で決まる場合、このスキンには ISimpleStyleClient インターフェイスも実装する必要があります。これらのインターフェイスが実装されているかぎり、スキンのスーパークラスには、DisplayObject クラスの任意のサブクラスを指定できます。BitmapAsset クラスもこのサブクラスに該当します。

次の例では、ProgrammaticSkin がスーパークラスとして指定されている MySkin クラスを定義します。

```
package {
  import mx.skins.ProgrammaticSkin;
  public class MySkin extends ProgrammaticSkin {
    ...
  }
}
```

updateDisplayList() メソッドの実装

updateDisplayList() メソッドは、スキンの外観を定義します。このメソッドは、スキンの構築の後に呼び出され、最初にスキンを描画します。それ以降は、コンポーネントのサイズ変更、スタイル変更、移動、または何らかの方法でやり取りが発生するたびに呼び出されます。

描画メソッドを使用して、updateDisplayList() メソッドでプログラムスキンを描画します。

updateDisplayList() メソッドを実装する場合は、次の操作を行います。

- override キーワードを使用して、スーパークラスの実装をオーバーライドします。
- 戻り値の型を void に設定します。
- このメソッドを protected として宣言します。

updateDisplayList() メソッドは、コンポーネントの高さと幅をパラメータとして受け取ります。これらの引数の値を、描画可能な領域の境界線として使用します。このメソッドは void を返します。

スキンをレンダリングするには、Graphics クラスのメソッドを使用します。このメソッドには、lineTo() や drawRect() があります。コンポーネントのシェイプを追加する前に領域がクリアされていることを確認するには、描画を実行する前に clear() メソッドを呼び出します。このメソッドを実行すると、それまでの updateDisplayList() メソッドの呼び出しで得られた結果が消去され、その時点までに描画メソッドを使用して作成したイメージがすべて削除されます。lineStyle() メソッドで指定された線スタイルもすべてリセットされます。

Graphics パッケージのメソッドを使用するには、flash.display.Graphics クラスのほか、GradientType や Font など、flash.display パッケージ内の使用するクラスをすべて読み込む必要があります。次の例では、flash.display パッケージのすべてのクラスを読み込みます。

```
import flash.display.*;
```

次の例では、drawRect() メソッドを使用し、コンポーネントの周囲に境界として長方形を描画します。

```
g.drawRect(0, 0, width, height);
```

次の例では、X とその周囲に境界線を描きます。

```
// skins/CheckboxSkin.as

package { // Use unnamed package if this skin is not in its own package.

  // Import necessary classes here.
  import flash.display.Graphics;
  import mx.skins.Border;
  import mx.skins.ProgrammaticSkin;
  import mx.styles.StyleManager;

  public class CheckboxSkin extends ProgrammaticSkin {

     // Constructor.
     public function CheckboxSkin():void {
        // Set default values here.
     }

  override protected function updateDisplayList(w:Number, h:Number):void {
     var g:Graphics = graphics;
     g.clear();
     g.beginFill(0xFFFFFF,1.0);
     g.lineStyle(2, 0xFF0000);
     g.drawRect(0, 0, w, h);
     g.endFill();
     g.moveTo(0, 0);
     g.lineTo(w, h);
     g.moveTo(0, h);
     g.lineTo(w, 0);
  }
  }
} // Close unnamed package.
```

Graphics パッケージの一般的なメソッドの説明については、768 ページの「プログラムによる描画」を参照してください。これらのメソッドの詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

updateDisplayList() メソッドで実行する一般的なタスクとして、コントロールの現在の状態に応じてスキンのプロパティを変更する処理があります。たとえば、Button コントロール用のプログラムスキンを定義した場合、ユーザーが Button コントロールにマウスを移動するかクリックしたときに、その境界線の太さや背景色を変更することができます。

この状態を確認するには、スキンの name プロパティを使用します。name は、スキンの現在の名前を示しています。たとえば、Button コントロール用のプログラムスキンを定義した場合、name プロパティはスキン状態を示す任意の値、つまり downSkin、upSkin、overSkin、disabledSkin、selectedDisabledSkin、selectedDownSkin、selectedOverSkin、および selectedUpSkin に設定できます。

次の例では、Button コントロールの現在の状態をチェックし、さらに線の太さと背景の塗りの色を適切に調整します。具体的には、この Button コントロールをクリックすると、線の太さが 2 ポイントになるようにスキンが再描画されます。クリックしたマウスボタンを離すと、スキンがもう一度再描画され、線の太さはそのデフォルト値である 4 ポイントに戻ります。また、背景の塗りの色も、Button コントロールの状態に応じて変化します。

```
// skins/ButtonStatesSkin.as

package { // Use unnamed package if this skin is not in its own package.

  // Import necessary classes here.
  import flash.display.Graphics;
  import mx.skins.Border;
  import mx.skins.ProgrammaticSkin;
  import mx.styles.StyleManager;

  public class ButtonStatesSkin extends ProgrammaticSkin {

     public var backgroundFillColor:Number;
     public var lineThickness:Number;

     // Constructor.
     public function ButtonStatesSkin():void {
        // Set default values.
        backgroundFillColor = 0xFFFFFF;
        lineThickness = 4;
     }

     override protected function updateDisplayList(w:Number, h:Number):void {
        // Depending on the skin's current name, set values for this skin.
        switch (name) {
           case "upSkin":
            lineThickness = 4;
            backgroundFillColor = 0xFFFFFF;
```

```
            break;
           case "overSkin":
            lineThickness = 4;
            backgroundFillColor = 0xCCCCCC;
            break;
           case "downSkin":
            lineThickness = 2;
            backgroundFillColor = 0xFFFFFF;
            break;
           case "disabledSkin":
            lineThickness = 2;
            backgroundFillColor = 0xCCCCCC;
            break;
        }

        // Draw the box using the new values.
        var g:Graphics = graphics;
        g.clear();
        g.beginFill(backgroundFillColor,1.0);
        g.lineStyle(lineThickness, 0xFF0000);
        g.drawRect(0, 0, w, h);
        g.endFill();
        g.moveTo(0, 0);
        g.lineTo(w, h);
        g.moveTo(0, h);
        g.lineTo(w, 0);
    }
  }
} // Close unnamed package.
```

単一のプログラムスキンクラスを使用してコントロールの複数の状態を定義する場合、次の例に示すように、Flex アプリケーションでコントロールの適切な状態すべてにその単一のスキンを適用する必要があります。

```
<?xml version="1.0"?>
<!-- skins/ApplyButtonStatesSkin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     Button {
        overSkin: ClassReference("ButtonStatesSkin");
        upSkin: ClassReference("ButtonStatesSkin");
        downSkin: ClassReference("ButtonStatesSkin");
     }
  </mx:Style>
  <mx:Button id="b1" label="Click Me"/>
</mx:Application>
```

スキンが RectangularBorder のサブクラスの場合は、updateDisplayList() メソッドの本体から super.updateDisplayList() も呼び出す必要があります。

measuredWidth および measuredHeight の getter の実装

measuredWidth プロパティおよび measuredHeight プロパティは、コンポーネントのデフォルトの幅と高さを定義します。measuredWidth プロパティおよび measuredHeight プロパティの getter メソッドを実装することは可能ですが、ほとんどのスキンでは不要です。ScrollBar 矢印を定義しているスキンのように、getter の実装が必要なスキンも一部にはあります。これらの getter を実装する場合は、スーパークラスの getter メソッドを実装するときに override キーワードを指定します。また、実装する getter はパブリックとする必要があります。

measuredWidth および measuredHeight の getter は、定数を返すことが普通です。通常、Flex アプリケーションは計算サイズに従いますが、必ず従うというわけではありません。これらの getter を省略すると、measuredWidth および measuredHeight の値はデフォルト値の 0 に設定されます。

次の例では、measuredWidth プロパティおよび measuredHeight プロパティを 10 に設定した後、getter をオーバーライドします。

```
// skins/ButtonStatesWithMeasuredSizesSkin.as

package { // Use unnamed package if this skin is not in its own package.

  // Import necessary classes here.
  import flash.display.Graphics;
  import mx.skins.Border;
  import mx.skins.ProgrammaticSkin;
  import mx.styles.StyleManager;

  public class ButtonStatesWithMeasuredSizesSkin extends ProgrammaticSkin {

     public var backgroundFillColor:Number;
     public var lineThickness:Number;

     private var _measuredWidth:Number;
     private var _measuredHeight:Number;

     // Constructor.
     public function ButtonStatesWithMeasuredSizesSkin():void {
        // Set default values.
        backgroundFillColor = 0xFFFFFF;
        lineThickness = 4;

        _measuredHeight = 100;
        _measuredWidth = 150;
     }

     override public function get measuredWidth():Number {
        return _measuredWidth;
     }

     override public function get measuredHeight():Number {
        return _measuredHeight;
     }
```

```
    override protected function updateDisplayList(w:Number, h:Number):void {
       // Depending on the skin's current name, set values for this skin.
       switch (name) {
          case "upSkin":
           lineThickness = 4;
           backgroundFillColor = 0xFFFFFF;
           break;
          case "overSkin":
           lineThickness = 4;
           backgroundFillColor = 0xCCCCCC;
           break;
          case "downSkin":
           lineThickness = 2;
           backgroundFillColor = 0xFFFFFF;
           break;
          case "disabledSkin":
           lineThickness = 2;
           backgroundFillColor = 0xCCCCCC;
           break;
       }

       // Draw the box using the new values.
       var g:Graphics = graphics;
       g.clear();
       g.beginFill(backgroundFillColor,1.0);
       g.lineStyle(lineThickness, 0xFF0000);
       g.drawRect(0, 0, w, h);
       g.endFill();
       g.moveTo(0, 0);
       g.lineTo(w, h);
       g.moveTo(0, h);
       g.lineTo(w, 0);
    }
  }
} // Close unnamed package.
```

borderMetrics プロパティの getter の実装

borderMetrics プロパティは、プログラムスキンの 4 辺すべての境界線の太さを定義します。Border または RectangularBorder のサブクラスとなっているプログラムスキンでは、borderMetrics プロパティの getter を実装する必要があります。それ以外の場合、この手順はオプションになります。このプロパティは EdgeMetrics タイプなので、戻り値の型として EdgeMetrics を設定する必要があります。

次の例では、borderThickness スタイルを取得し、その値を使用して、EdgeMetrics のコンストラクタで定義されているとおりに境界線の 4 辺の幅を定義します。

```
// skins/ButtonStatesWithBorderMetricsSkin.as

package { // Use unnamed package if this skin is not in its own package.
```

```
  // Import necessary classes here.
  import flash.display.Graphics;
  import mx.skins.Border;
  import mx.skins.ProgrammaticSkin;
  import mx.styles.StyleManager;
  import mx.core.EdgeMetrics;

  public class ButtonStatesWithBorderMetricsSkin extends ProgrammaticSkin {

     public var backgroundFillColor:Number;
     public var lineThickness:Number;

     private var _borderMetrics:EdgeMetrics;

     // Constructor.
     public function ButtonStatesWithBorderMetricsSkin():void {
        // Set default values.
        backgroundFillColor = 0xFFFFFF;
        lineThickness = 4;
     }

     public function get borderMetrics():EdgeMetrics {
        if (_borderMetrics) {
           return _borderMetrics;
        }
        var borderThickness:Number = getStyle("borderThickness");
        _borderMetrics = new EdgeMetrics(borderThickness, borderThickness,
borderThickness, borderThickness);
        return _borderMetrics;
     }

     override protected function updateDisplayList(w:Number, h:Number):void {
        // Depending on the skin's current name, set values for this skin.
        switch (name) {
           case "upSkin":
            lineThickness = 4;
            backgroundFillColor = 0xFFFFFF;
            break;
           case "overSkin":
            lineThickness = 4;
            backgroundFillColor = 0xCCCCCC;
            break;
           case "downSkin":
            lineThickness = 2;
            backgroundFillColor = 0xFFFFFF;
            break;
           case "disabledSkin":
            lineThickness = 2;
            backgroundFillColor = 0xCCCCCC;
            break;
        }
```

```
        // Draw the box using the new values.
        var g:Graphics = graphics;
        g.clear();
        g.beginFill(backgroundFillColor,1.0);
        g.lineStyle(lineThickness, 0xFF0000);
        g.drawRect(0, 0, w, h);
        g.endFill();
        g.moveTo(0, 0);
        g.lineTo(w, h);
        g.moveTo(0, h);
        g.lineTo(w, 0);
    }
  }
} // Close unnamed package.
```

スタイル対応プロパティの作成

多くの場合、スキンの背景色、境界線の太さ、角の丸みなどのスタイルプロパティを定義するプログラムスキンを定義します。これらのプロパティをスタイル対応にすることで、ユーザーは CSS ファイルで、または Flex アプリケーション内の `setStyle()` メソッドを使用して、この値を変更できます。インラインシンタックスを使用してプラグラムスキンに定義されているスタイルの設定は変更できません。

カスタムプロパティをスタイル対応にするには、`updateDisplayList()` メソッドに `getStyle()` メソッドの呼び出しを追加し、そのメソッドのパラメータとして目的のカスタムプロパティを指定します。Flex がスキンをレンダリングするときは、このプロパティに対して `getStyle()` が呼び出され、CSS または表示リスト内で設定が検出されます。それにより、スキンを描画するときにスタイルプロパティの値を使用できます。

`getStyle()` メソッドへのこの呼び出しをチェックにラップし、スタイルが存在するかどうかを確認します。プロパティが設定されていない場合、`getStyle()` の結果は予期できない結果になることがあります。

次の例では、プロパティに値を割り当てる前に、そのプロパティが定義されているかどうかを確認します。

```
if (getStyle("lineThickness")) {
  _lineThickness = getStyle("lineThickness");
}
```

スキンのスタイル対応プロパティには、デフォルト値を定義する必要があります。通常、この操作は、スキンのコンストラクタ関数内で行います。デフォルト値を設定しないと、Flex アプリケーションでこのスタイルが定義されない場合のスタイルプロパティは、`NaN` または `undefined` に設定されます。これが原因で、ランタイムエラーが発生することがあります。

次の MyButtonSkin プログラムスキンクラスの例では、スタイル対応の _lineThickness プロパティおよび _backgroundFillColor プロパティのデフォルト値を、スキンのコンストラクタで定義します。次に、getStyle() メソッドの呼び出しを updateDisplayList() メソッドに追加し、これらのプロパティをスタイル対応にします。

```
// skins/ButtonStylesSkin.as

package { // Use unnamed package if this skin is not in its own package.

  // Import necessary classes here.
  import flash.display.Graphics;
  import mx.skins.Border;
  import mx.skins.ProgrammaticSkin;
  import mx.styles.StyleManager;

  public class ButtonStylesSkin extends ProgrammaticSkin {

     public var _backgroundFillColor:Number;
     public var _lineThickness:Number;

     // Constructor.
     public function ButtonStylesSkin():void {
        // Set default values.
        _backgroundFillColor = 0xFFFFFF;
        _lineThickness=2;
     }

     override protected function updateDisplayList(w:Number, h:Number):void {
        if (getStyle("lineThickness")) {
           // Get value of lineThickness style property.
           _lineThickness = getStyle("lineThickness");
        }
        if (getStyle("backgroundFillColor")) {
           // Get value of backgroundFillColor style property.
           _backgroundFillColor = getStyle("backgroundFillColor");
        }

        // Draw the box using the new values.
        var g:Graphics = graphics;
        g.clear();
        g.beginFill(_backgroundFillColor,1.0);
        g.lineStyle(_lineThickness, 0xFF0000);
        g.drawRect(0, 0, w, h);
        g.endFill();
        g.moveTo(0, 0);
        g.lineTo(w, h);
        g.moveTo(0, h);
        g.lineTo(w, 0);
     }
  }
} // Close unnamed package.
```

Flex アプリケーションでは、CSS または setStyle() メソッドを使用して、スタイル対応プロパティの値を設定できます。

次の例では、すべての Button コントロールのスタイル対応プロパティの値を、CSS で設定します。

```
<?xml version="1.0"?>
<!-- skins/ApplyButtonStylesSkin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="600">
  <mx:Style>
     Button {
        upSkin:ClassReference('ButtonStylesSkin');
        downSkin:ClassReference('ButtonStylesSkin');
        overSkin:ClassReference('ButtonStylesSkin');
        disabledSkin:ClassReference('ButtonStylesSkin');
        lineThickness:4;
        backgroundFillColor:#CCCCCC;
     }
  </mx:Style>

  <mx:Script><![CDATA[
     public function changeLineThickness(e:Event):void {
        var t:int = Number(b1.getStyle("lineThickness"));
        if (t == 4) {
           b1.setStyle("lineThickness",1);
        } else {
           b1.setStyle("lineThickness",4);
        }
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Change Line Thickness"
click="changeLineThickness(event)"/>

  <mx:Button id="b2"/>
</mx:Application>
```

Flex アプリケーションで、setStyle() メソッドを使用してスタイルプロパティの値を設定する場合、前の例のようにスタイル対応プロパティの値を、コンポーネントの単一のインスタンスにのみ設定できるほか、そのすべてのインスタンスに対して設定することもできます。次の例では、setStyle() メソッドを使用し、コントロールのすべてのインスタンスに対してスタイル対応プロパティの値を設定します。この場合のインスタンスは、すべての Button コントロールです。

```
<?xml version="1.0"?>
<!-- skins/ApplyGlobalButtonStylesSkin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="600">
  <mx:Style>
     Button {
        upSkin:ClassReference('ButtonStylesSkin');
        downSkin:ClassReference('ButtonStylesSkin');
        overSkin:ClassReference('ButtonStylesSkin');
        disabledSkin:ClassReference('ButtonStylesSkin');
```

```
        lineThickness:4;
        backgroundFillColor:#CCCCCC;
    }
  </mx:Style>

  <mx:Script><![CDATA[
     public function changeLineThickness(e:Event):void {
        var t:int = Number(b1.getStyle("lineThickness"));
        if (t == 4) {
           StyleManager.getStyleDeclaration("Button").setStyle("lineThickness",
1);
        } else {
           StyleManager.getStyleDeclaration("Button").setStyle("lineThickness",
4);
        }
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Change Line Thickness"
click="changeLineThickness(event)"/>

</mx:Application>
```

CSS または setStyle() メソッドのいずれかを使用してこれらのプロパティの値を設定しない場合、Flex では、スキンのコンストラクタで設定されたプロパティのデフォルト値が使用されます。

color や fontSize などの既存のスタイルプロパティの値を取得するには、プロパティの存在のチェックに setStyle() メソッドへの呼び出しをラップする必要はありません。これは、Flex により、すべてのコンポーネントスタイルのデフォルト値を定義する CSSStyleDeclaration が作成されるからです。既存のスタイルプロパティが、未定義ということはありえません。カスタムスキンに追加したスタイルプロパティは、この CSSStyleDeclaration には追加されません。コンポーネントは、そのプロパティがスタイルプロパティであることを認識していないからです。

スタイル対応として定義したカスタムスキンプロパティは継承されません。そのため、このコンポーネントのサブクラスや子は、このプロパティの値を継承しません。

親コンポーネントへのアクセス

プログラムスキンクラスからプログラムスキンの親への参照を取得できます。この参照を使用すると、親コンポーネントのプロパティへのアクセスや、親コンポーネントに対するメソッドの呼び出しが可能になります。

スキンの parent プロパティを使用することにより、updateDisplayList() メソッドから親にアクセスできます。このスキンはまだ親コントロールに追加されていないため、スキンのコンストラクタ内で親コンポーネントにアクセスすることはできません。親コンポーネントが addChild() メソッドを呼び出して、スキンを子として追加すると、そのスキンの parent プロパティの値が設定されます。

プログラムスキンを使用してコンポーネントのインスタンスを作成するときは、次の順序で各種イベントが発生します。

1. 親コンポーネントのインスタンスを作成します。
2. スキンクラスのインスタンスを作成します。
3. 親コンポーネントで addChild() メソッドを呼び出して、スキンクラスを追加します。

スキンの親への参照を取得するには、スキンの parent プロパティを UIComponent 型にキャストする必要があります。このスキンは、IFlexDisplayObject インターフェイスからこの読み取り専用のプロパティを継承します。また、is 演算子を使用して、親が UIComponent 型であることも確認する必要があります。これは、キャストを完了できないと、Flex によってランタイムエラーがスローされるからです。

次の例では、親コントロールのクラス名を取得し、親コンポーネントのタイプに応じて境界線を描画し塗ります。

```
package {

import flash.display.GradientType;
import flash.display.Graphics;
import mx.skins.Border;
import mx.styles.StyleManager;
import mx.utils.ColorUtil;
import mx.skins.halo.HaloColors;
import mx.core.UIComponent;

public class IconSkin extends Border {

  public function IconSkin() {
     //super();
  }

  override public function get measuredWidth():Number {
     return 14;
  }

  override public function get measuredHeight():Number {
     return 14;
  }

  override protected function updateDisplayList(w:Number, h:Number):void {
     super.updateDisplayList(w, h);

     // User-defined styles
     var borderColor:uint = getStyle("borderColor");
     var fillAlphas:Array = getStyle("fillAlphas");
     var fillColors:Array = getStyle("fillColors");
     StyleManager.getColorNames(fillColors);
     var highlightAlphas:Array = getStyle("highlightAlphas");
     var themeColor:uint = getStyle("themeColor");

     var r:Number = width / 2;
```

```
     var upFillColors:Array;
     var upFillAlphas:Array;

     var disFillColors:Array;
     var disFillAlphas:Array;

     var g:Graphics = graphics;
     g.clear();

     var myParent:String;

     switch (name) {
        case "upIcon": {
           upFillColors = [ fillColors[0], fillColors[1] ];
           upFillAlphas = [ fillAlphas[0], fillAlphas[1] ];

           if (parent is UIComponent) {
            myParent = String(UIComponent(parent).className);
           }
           if (myParent=="RadioButton") {
            // RadioButton border
            g.beginGradientFill(GradientType.LINEAR,[ borderColor, 0x000000 ],
[100,100], [0,0xFF], verticalGradientMatrix(0,0,w,h));
            g.drawCircle(r,r,r);
            g.drawCircle(r,r,(r-1));
            g.endFill();

            // RadioButton fill
           g.beginGradientFill(GradientType.LINEAR, upFillColors, upFillAlphas,
[0,0xFF], verticalGradientMatrix(1,1,w-2,h-2));
            g.drawCircle(r,r,(r-1));
            g.endFill();
           } else if (myParent=="CheckBox") {
            // CheckBox border
            drawRoundRect(0,0,w,h,0,[borderColor, 0x000000],1,
verticalGradientMatrix(0,0,w,h), GradientType.LINEAR, null, {x: 1,y:1,w:w-2,h:h-
2,r:0});

            // CheckBox fill
            drawRoundRect(1,1,w-2,h-2,0,upFillColors, upFillAlphas,
verticalGradientMatrix(1,1,w-2,h-2));
           }

           // top highlight
           drawRoundRect(1,1,w-2,(h-2)/2,{tl:r,tr:r,bl:0,br:0},[0xFFFFFF,
0xFFFFFF], highlightAlphas, verticalGradientMatrix(0,0,w-2,(h-2)/2));
        }

        // Insert other cases such as downIcon and overIcon here.

     }
  }
}

}
```

プログラムスキンのコンパイル

プログラムスキンを使用するアプリケーションをコンパイルするときは、任意の ActionScript クラスを扱う場合と同じように、プログラムスキンを扱います。スキンをコンパイラの source-path パラメータに追加する必要があります。コンパイルする MXML ファイルと同じディレクトリにスキンが存在する場合は、source-path をピリオドに設定します。次の例では、そのことを mxmlc コマンドラインコンパイラで示しています。

```
$ ./mxmlc -source-path=. c:¥flex¥MyApp.mxml
```

プログラムスキンがパッケージに含まれていない場合は、それらのスキンを外部から認識できるようにするため、名前のないパッケージに追加する必要があります。そのようにしないと、mxmlc はコンパイラエラーをスローします。スキンを追加するには、次の例のようにクラスを package ステートメントで囲みます。

```
package { // 名前のないパッケージを開きます。
   import flash.display.*;
   import mx.skins.ProgrammaticSkin;

   public class MySkin extends ProgrammaticSkin {
      ...
   }
} // 名前のないパッケージを閉じます。
```

Flex アプリケーションでは、目的のプログラムスキンをスクリプトブロックまたはヘルパークラスで定義している適切なクラスまたはパッケージを読み込む必要があります。

プログラムによる描画

プログラムスキンの構成部品を描画するには、Graphics クラスの描画メソッドを使用します。これらの描画メソッドでは、塗りやグラデーションの塗りの記述、線のサイズやシェイプの定義、および線の描画ができます。このような単純な描画メソッドを組み合わせることにより、コンポーネントのスキンを構成する複雑なシェイプを作成できます。

次の表は、Graphics パッケージの中で最も頻繁に使用される描画メソッドを簡単に示しています。

メソッド	概要
beginFill()	塗りの描画を開始します。例: beginFill(0xCCCCFF,1); 開いたパス、つまり現在の描画位置が、moveTo() メソッドで指定した前の座標と等しくないパスがあり、そのパスに関連付けられた塗りが指定されている場合、この開いたパスは線で閉じられ、関連付けられている塗りが適用されます。
beginGradientFill()	グラデーションの塗りの描画を開始します。開いたパス、つまり現在の描画位置が、moveTo() メソッドで指定した前の座標と等しくないパスがあり、そのパスに関連付けられた塗りが指定されている場合、この開いたパスは線で閉じられ、関連付けられている塗りが適用されます。

メソッド	概要
clear()	現在のオブジェクトに関する描画出力の内容をすべて削除します。clear() メソッドには引数がありません。

curveTo()	現在の線のスタイルを使用して曲線を描画します。例:
	moveTo(500, 500);
	curveTo(600, 500, 600, 400);
	curveTo(600, 300, 500, 300);
	curveTo(400, 300, 400, 400);
	curveTo(400, 500, 500, 500);

drawCircle()	事前に設定された線のスタイルと塗りを使用して、円を描画します。次の例に示すように、描画する円の x 位置、y 位置、および半径をこのメソッドに渡します。 drawCircle(10,10,50);
drawRect()	事前に設定された線のスタイルと塗りを使用して、矩形を描画します。次の例に示すように、描画する矩形の x 位置、y 位置、長さ、および幅をこのメソッドに渡します。 drawRect(10,10,100,20);
drawRoundRect()	事前に設定された線のスタイルと塗りを使用して、角の丸い矩形を描画します。次の例に示すように、描画する矩形の x 位置、y 位置、長さ、高さ、および丸い角の描画に使用する楕円の幅と高さを、このメソッドに渡します。 drawRoundRect(10,10,100,20,9,5)
endFill()	beginFill() メソッドまたは beginGradientFill() メソッドで指定された塗りを終了します。endFill() メソッドには引数がありません。moveTo() メソッドで指定された前の座標と現在の描画位置が等しくなく、塗りが定義されている場合は、パスが線で閉じられ、定義されている塗りが適用されます。
lineStyle()	この呼び出し以降に呼び出される lineTo() メソッドおよび curveTo() メソッドで作成する線のストロークを定義します。次の例では、線のスタイルを、太さは 2 ポイント、色はグレー、不透明度は 100% に設定します。 lineStyle(2,0xCCCCCC,1) パスを描画している途中で lineStyle() メソッドを呼び出すと、そのパスに属する線のセグメントごとに異なるスタイルを指定できます。clear() メソッドを呼び出すと、線のスタイルが undefined に戻ります。
lineTo()	現在の線のスタイルを使用して線を描画します。次の例では、三角形を描画します。 moveTo (200, 200); lineTo (300, 300); lineTo (100, 300); lineTo (200, 200); moveTo() メソッドを呼び出す前に lineTo() を呼び出すと、現在の描画位置はデフォルト値の (0, 0) に戻ります。
moveTo()	現在の描画位置を指定された座標に移動します。例: moveTo(100,10);

次の例では、三角形を描画します。

```
// skins/CheckBoxAsArrowSkin.as

package { // Use unnamed package if this skin is not in its own package.

  // Import necessary classes here.
  import flash.display.Graphics;
  import mx.skins.Border;
  import mx.skins.ProgrammaticSkin;
  import mx.styles.StyleManager;

  public class CheckBoxAsArrowSkin extends ProgrammaticSkin {

     // Constructor.
     public function CheckBoxAsArrowSkin():void {
        // Set default values.
     }

     override protected function updateDisplayList(w:Number, h:Number):void {
        var unscaledHeight:Number = 2;
        var unscaledWidth:Number = 2;

        var arrowColor:Number;

        var g:Graphics = graphics;
        g.clear();

        switch (name) {
           case "upIcon":
           case "selectedUpIcon": {
            arrowColor = 0x666666;
            break;
           }
           case "overIcon":
           case "downIcon":
           case "selectedOverIcon":
           case "selectedDownIcon": {
            arrowColor = 0xCCCCCC;
            break;
           }
        }

        // Draw an arrow.
        graphics.lineStyle(1, 1, 1);
        graphics.beginFill(arrowColor);
        graphics.moveTo(unscaledWidth, unscaledHeight-20);
        graphics.lineTo(unscaledWidth-30, unscaledHeight+20);
        graphics.lineTo(unscaledWidth+30, unscaledHeight+20);
        graphics.lineTo(unscaledWidth, unscaledHeight-20);
        graphics.endFill();
     }
  }
} // Close unnamed package.
```

ProgrammaticSkin クラスでも描画メソッドが定義されていますが、その中で最も多く使用するものは drawRoundRect() メソッドです。このメソッドでは、プログラムで矩形を描画します。そのプログラムで、角の丸み半径やグラデーションなどのプロパティを設定できます。このメソッドを使用してコンテナの境界をカスタマイズすると、たとえば、次のような描画が可能です。

次のコードでは、drawRoundRect() メソッドを使用して、このカスタマイズした VBox の境界を描画します。

```
// skins/CustomContainerBorderSkin.as

package { // Use unnamed package if this skin is not in its own package.

  // Import necessary classes here.
  import flash.display.Graphics;
  import mx.graphics.RectangularDropShadow;
  import mx.skins.RectangularBorder;

  public class CustomContainerBorderSkin extends RectangularBorder {

     private var dropShadow:RectangularDropShadow;

     // Constructor.
     public function CustomContainerBorderSkin() {
     }

     override protected function updateDisplayList(unscaledWidth:Number,
unscaledHeight:Number):void {

        super.updateDisplayList(unscaledWidth, unscaledHeight);

        var cornerRadius:Number = getStyle("cornerRadius");
        var backgroundColor:int = getStyle("backgroundColor");
        var backgroundAlpha:Number = getStyle("backgroundAlpha");
        graphics.clear();

        // Background
        drawRoundRect(0, 0, unscaledWidth, unscaledHeight, {tl: 0,
tr:cornerRadius, bl: cornerRadius, br: 0}, backgroundColor, backgroundAlpha);

        // Shadow
        if (!dropShadow)
           dropShadow = new RectangularDropShadow();

        dropShadow.distance = 8;
        dropShadow.angle = 45;
        dropShadow.color = 0;
        dropShadow.alpha = 0.4;
```

```
            dropShadow.tlRadius = 0;
            dropShadow.trRadius = cornerRadius;
            dropShadow.blRadius = cornerRadius;
            dropShadow.brRadius = 0;
            dropShadow.drawShadow(graphics, 0, 0, unscaledWidth, unscaledHeight);
        }
    }
}
```

Flex アプリケーションでは、次の例に示すようにこのスキンを適用します。

```
<?xml version="1.0"?>
<!-- skins/ApplyContainerBorderSkin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:VBox id="vb1"
     borderSkin="CustomContainerBorderSkin"
     backgroundColor="0xCCCC99"
     backgroundAlpha="0.8"
     cornerRadius="14"
     paddingLeft="20"
     paddingTop="20"
     paddingRight="20"
     paddingBottom="20"
  >
     <mx:Label text="Text Goes Here"/>
  </mx:VBox>

</mx:Application>
```

前の例にある“拡大 / 縮小されない”幅と高さのプロパティは、そのスキン自身がこれらのプロパティを認識したときのスキンのサイズを指しています。これらのサイズでは、スキンのサイズが外部のコンポーネントにより変更されている可能性がある点は無視されています。コンポーネント内部で作業する際は、拡大 / 縮小されないサイズの使用が最適です。

任意のメソッドを使用して、スキンを描画できます。詳細については、Adobe Flex 2 リファレンスガイドを参照してください。

プログラムスキンの適用

グラフィカルスキンと同様、次に示す方法により、アプリケーションの中でプログラムスキンをコントロールに適用できます。

- CSS ファイル。773 ページの「CSS ファイルの使用」を参照してください。
- インライン。774 ページの「インラインシンタックスの使用」を参照してください。
- `setStyle()` メソッドおよび StyleManager クラスの使用。775 ページの「スキンでの setStyle() メソッドおよび StyleManager の使用」を参照してください。

これらのどの場合であっても、コンポーネントにプログラムスキンを適用するときは、完全なファイル名ではなく、プログラムスキンのクラス名で指定します。たとえば、完全なファイル名である“SampleSkin.as”ではなく、“SampleSkin”を使用します。

アプリケーションのコンパイル時には、このプログラムスキンのクラスがソースパスに置かれている必要があります。さらに、このスキンのクラスを、Flex アプリケーションの ActionScript ブロックまたはヘルパークラスに読み込むことも必要です。

プログラムスキンが名前付きのパッケージに含まれている場合、パッケージのレベルをピリオドで区切ってください。たとえば、SampleButtonSkin クラスが my.skins パッケージに含まれている場合、これを適用するときは "my.skins.SampleButtonSkin" を使用します。これはパッケージ内の任意のクラスを参照するときと同じです。プログラムスキンが名前付きのパッケージに含まれていない場合、それらのスキンは名前のないパッケージに含まれている必要があります。それらのスキンを名前のないパッケージに追加するには、クラスを package ステートメントで囲みます。

詳細については、768 ページの「プログラムスキンのコンパイル」を参照してください。

CSS ファイルの使用

CSS ファイル内のプログラムスキンを、ClassReference ディレクティブを使用して適用します。このディレクティブは、引数としてクラス名を取ります。アプリケーションのコンパイル時には、このプログラムスキンのクラスがソースパスに置かれている必要があります。

次の CSS ファイルの例では、SampleAccordionHeaderSkin および SampleButtonSkin プログラムスキンを、AccordionHeader および Button コントロールの選択された状態に適用します。

```
/* assets/UseHaloSkins.css */
RadioButton {
    /*
        Defaults. Shown here for illustrative purposes.
    */
    disabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    downIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    overIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedDisabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedDownIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedOverIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedUpIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    upIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
}

CheckBox {
    /*
        Applies the RadioButtonIcon skin to the CheckBox component's states.
    */
    disabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    downIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    overIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedDownIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedDisabledIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedOverIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    selectedUpIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
    upIcon: ClassReference("mx.skins.halo.RadioButtonIcon");
}
```

CSS ファイルと同様、次の例に示すように、<mx:Style> タグを使用してスタイルシートを定義するか、外部スタイルシートを参照できます。

```
<?xml version="1.0"?>
<!-- skins/UseHaloSkinsStyleSheet.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style source="../assets/UseHaloSkins.css"/>

  <mx:CheckBox id="cb1" label="Click Me"/>
  <mx:RadioButton id="rb1" label="Click Me"/>

</mx:Application>
```

インラインシンタックスの使用

スキンを交換する状態ごとにプログラムスキンクラスを指定することにより、そのクラスをインラインで埋め込むことができます。クラス名は、中カッコ { } で囲みます。アプリケーションのコンパイル時には、このプログラムスキンのクラスがソースパスに置かれている必要があります。クラスファイルが Flex アプリケーションと同じディレクトリに置かれている場合は、その場所をソースパスに追加する必要はありません。

次の例では、SampleButtonSkin プログラムスキンを Button コントロールの upSkin 状態、overSkin 状態、downSkin 状態、disabledSkin 状態に適用します。

```
<?xml version="1.0"?>
<!-- skins/ApplyProgrammaticSkinsInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Button id="b1"
     label="Click Me"
     overSkin="{ButtonStatesSkin}"
     upSkin="{ButtonStatesSkin}"
     downSkin="{ButtonStatesSkin}"
     disabledSkin="{ButtonStatesSkin}"
  />
</mx:Application>
```

スキンクラスがパッケージに含まれている場合、コンパイラで該当のクラス名を解決できるように、クラスをアプリケーションに読み込む必要があります。次に例を示します。

```
<mx:Script>
   import myPackage.*;
</mx:Script>
```

スキンでの setStyle() メソッドおよび StyleManager の使用

setStyle() メソッドを使用すると、プログラムスキンをコントロールの単一のインスタンスに適用できます。このメソッド内で、置換するスキンのプロパティおよびプログラムスキンのクラス名を指定します。アプリケーションのコンパイル時には、このプログラムスキンのクラスがソースパスに置かれている必要があります。

次の例では、setStyle() メソッドを使用して、button1 コントロールの upSkin 状態に ButtonStatesSkin スキンを適用します。

```
<?xml version="1.0"?>
<!-- skins/ApplyProgrammaticSkinsSetStyle.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     public function changeSkins(e:Event):void {
        b1.setStyle("upSkin",ButtonStatesSkin);
        b1.setStyle("downSkin",ButtonStatesSkin);
        b1.setStyle("overSkin",ButtonStatesSkin);
        b1.setStyle("disabledSkin",ButtonStatesSkin);
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Change Skins" click="changeSkins(event)"/>

</mx:Application>
```

setStyle() メソッドで ButtonStatesSkin クラスを参照すると、コンパイラは、コンパイル時に ButtonStatesSkin クラス全体にリンクを設定します。生成される SWF ファイルは、changeSkins() メソッドが一度も呼び出されない場合でも、このクラスへの参照がない場合より大きくなります。

StyleManager クラスを使用すると、1つのプログラムスキンをすべての Button コントロールの状態に適用できます。次の例では、すべての Button コントロールの upSkin 状態に SampleButtonSkin スキンを適用します。

```
<?xml version="1.0"?>
<!-- skins/ApplyProgrammaticSkinsStyleManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     public function changeSkins(e:Event):void {
        StyleManager.getStyleDeclaration("Button").setStyle("upSkin",
ButtonStatesSkin);
        StyleManager.getStyleDeclaration("Button").setStyle("downSkin",
ButtonStatesSkin);
        StyleManager.getStyleDeclaration("Button").setStyle("overSkin",
ButtonStatesSkin);
        StyleManager.getStyleDeclaration("Button").setStyle("disabledSkin",
ButtonStatesSkin);
     }
  ]]></mx:Script>

  <mx:Button id="b1" label="Change Skins" click="changeSkins(event)"/>

</mx:Application>
```

スキンがパッケージに含まれている場合、コンパイラで該当のクラス名を解決できるように、クラスをアプリケーションに読み込む必要があります。次に例を示します。

```
<mx:Script>
  import myPackage.ButtonStatesSkin;
</mx:Script>
```

setStyle() メソッドの使用方法の詳細については、682 ページの「setStyle() メソッドと getStyle() メソッドの使用」を参照してください。StyleManager の使用方法の詳細については、677 ページの「StyleManager クラスの使用」を参照してください。

ツールヒントのスキン交換

このセクションでは、ToolTip コントロールにプログラムスキンを適用する方法について説明します。ここでは、このトピックで既に紹介した概念のいくつかについて説明します。

ツールヒントのスキンは、ToolTipBorder プログラムスキンで定義します。このファイルは、mx.skins.halo パッケージに収められています。

ツールヒントのスキンを交換するには、ToolTipBorder クラスファイルを編集し、CSS を使用してその新しいスキンをツールヒントに適用します。その基本的な手順を次で説明します。

CSS を使用してツールヒントのスキンを交換するには :

1. "mx.skins.halo.ToolTipBorder.as" ファイルを開きます。このファイルは、743 ページの「スキン用のリソース」の説明にあるように、ソースファイルに付属しています。
2. "ToolTipBorder.as" ファイルを別名で、元の場所とは異なる場所に保存します。このファイル名は、新しいクラスの名前と同じものにする必要があります。
3. 次の例に示すように、パッケージを mx.skins.halo から、使用するパッケージまたは空のパッケージに変更します。

   ```
   //package mx.skins.halo { // 古いパッケージ名
   package { // 新しい、空のパッケージ
   ```

4. 次の例に示すように、ファイル内のクラス名を変更します。

   ```
   //public class ToolTipBorder extends RectangularBorder // 古い名前
   public class MyToolTipBorder extends RectangularBorder // 新しい名前
   ```

5. 次の例に示すように、新しいクラス名が反映されるようにコンストラクタを変更します。

   ```
   //public function ToolTipBorder() // 古いコンストラクタ
   public function MyToolTipBorder() // 新しいコンストラクタ
   ```

6. 次の例に示すように、バージョン管理の行をコメントアウトします。

   ```
   //include "../../core/Version.as";
   ```

7. クラスファイルを編集して、ツールヒントの外観を変更します。この編集操作では、updateDisplayList() メソッドの中で "toolTip" を処理する部分を編集します。このメソッドは、ツールヒントの境界を描画し、デフォルトのスタイルを設定するためのものです。一般的には、drawRoundRect() メソッドのパラメータを変更してツールヒントの外観を変更します。

   次の例では、backgroundColor プロパティのデフォルト値をカラーの配列に置き換えることで、ツールヒントのボックスの背景に淡い赤色を追加します。

```
var highlightAlphas:Array = [0.3,0.0];
drawRoundRect(3, 1, w-6, h-4, cornerRadius, [0xFF0000, 0xFFFFBB],
   backgroundAlpha);
```

8. ファイルを保存します。

9. Flex アプリケーションでは、新しいスキンを指定するように、ツールヒントの borderSkin スタイルプロパティを編集します。この編集を実行するには、アプリケーション内の <mx:Style> タグ、外部 CSS ファイル、またはカスタムのテーマファイルを使用します。次の例では、borderSkin プロパティを新しいスキンクラスに設定します。

```
<?xml version="1.0"?>
<!-- skins/ApplyCustomToolTipSkin.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     ToolTip {
        borderSkin: ClassReference("MyToolTipBorder");
     }
  </mx:Style>

  <mx:Button id="b1" label="Click Me" toolTip="Click this button"/>
</mx:Application>
```

   CSS の参照では完全なパッケージ名を指定する必要があります。この例では、"MyToolTipBorder.as" ファイルが空のパッケージに属しているので、mx.skins.halo のようなパッケージの指定が存在しません。

10. 新しいスキンファイルをソースパスに置いて、アプリケーションをコンパイルし、実行します。このためには、コンパイラの source-path オプションの値を設定します。

    アプリケーションと同じディレクトリに目的のスキンクラスを置いている場合は、その場所をソースパスに追加する必要はありません。この場合は、現在のディレクトリで実行すると見なされます。詳細については、『Flex 2 アプリケーションの構築とデプロイ』の第 9 章の「Flex コンパイラの使用」を参照してください。

テーマの作成

テーマとは、Flex アプリケーションの外観と操作性を定義するスタイル定義とスキンのコレクションです。テーマファイルでは、スタイルシートと同様に、グラフィカルスキンとプログラムスキンの両方を扱うことができます。

テーマは、Flex アプリケーションに適用できる SWC フィイルの形式をとります。SWC ファイルをコンパイルするには、compc コマンド行コンパイラユーティリティを使用します。

SWC ファイルをコンパイルしておき、その SWC ファイルをアプリケーションでテーマとして使用すれば、アプリケーション本体をコンパイルするときにすべてのスキンファイルをコンパイルする手間を省くことができます。その結果、コンパイル時間を短縮でき、アプリケーションのコードに存在する問題も容易にデバッグできます。さらに、クラスとイメージファイルの組み合わせによる運用に比べ、SWC ファイルは他のアプリケーションへの移植が容易です。

テーマの作成および適用の詳細については、698 ページの「テーマについて」を参照してください。

第 21 章

アイテムレンダラーとアイテムエディタの使用

21

セルの外観や動作は、Adobe Flex アプリケーションで使用できるリストベースのデータプロバイダコントロールでカスタマイズできます。使用できるコントロールには、DataGrid、HorizontalList、List、Menu、MenuBar、TitleList、Tree などがあります。リストコントロールでセルの外観を制御するには、カスタムアイテムレンダラーとカスタムアイテムエディタを作成します。

このトピックではアイテムレンダラーとアイテムエディタの詳細と、Flex アプリケーションにこれらを作成するさまざまな方法について説明します。次のトピック 823 ページ、第 22 章の「アイテムエディタの操作」では、さらに高度なアイテムエディタの作成に必要な詳細情報について説明します。

目次

アイテムレンダラーについて

Flex では、アイテムのリストの表示に使用できるいくつかのコントロールをサポートしています。これらのコントロールを使用すると、アイテムリストをスクロールし、リストからアイテムを選択できるようになります。Flex リストコンポーネントはすべて ListBase クラスから派生したもので、次のようなコントロールがあります。

- DataGrid
- HorizontalList
- List
- Menu
- MenuBar
- TileList
- Tree

リストコントロールは、オブジェクトのコレクションである " データプロバイダ " からデータを取得します。たとえば、Tree コントロールはデータプロバイダからデータを読み取ることによって、ツリー構造と、各ツリーノードに割り当てられた関連データを定義します。

Flex コンポーネントと、そのコンポーネントの生成に使用するデータとの間には、データプロバイダによって 1 つの抽象化層が形成されます。同一のデータプロバイダから複数のコンポーネントを作成できます。また、実行時にコンポーネントに使用するデータプロバイダを切り替えることもできます。さらに、データプロバイダを変更した場合に、そのデータプロバイダを使用するすべてのコンポーネントに変更が反映されるようにすることもできます。

データプロバイダをモデルとすれば、Flex コンポーネントはそのモデルのビューと考えることができます。モデルとビューを分離することにより、変更が生じたときに両方を修正するのではなく、どちらか一方だけを修正するだけで済むというメリットが生まれます。

それぞれのリストコントロールにはデータの表示を制御するデフォルトのメカニズム ("ビュー") があり、そのデフォルトはオーバーライドできます。デフォルトビューをオーバーライドするには、カスタム " アイテムレンダラー " を作成します。

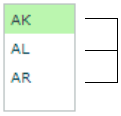
アイテムレンダラーを使用してデータ表示を制御する以外に、DataGrid、List、Tree の各コントロールでも、ユーザーにデータを編集させることができます。たとえば、DataGrid コントロールの Quantity 列をショッピングカートに使用している場合、ユーザーにこの列を更新させることができます。データ表示と同様に、リストコントロールには、ユーザーのデータ編集を可能にするデフォルトメカニズムがあり、このメカニズムはカスタム " アイテムエディタ " を作成するとオーバーライドできます。

デフォルトのアイテムレンダリングとセル編集

それぞれのリストコントロールには、そのコントロールのために定義されたデフォルトのアイテムレンダラーがあります。最も単純なアイテムレンダラーは DataGridItemRenderer クラスで、DataGrid コントロールのアイテムレンダラーを定義します。このアイテムレンダラーでは、データプロバイダの各エレメントはテキストストリングであると見なされます。

アイテムレンダラーには他に、ListItemRenderer、MenuItemRenderer、MenuBarItem、TileListItemRenderer、TreeItemRenderer などがあります。デフォルトでは、これらのアイテムレンダラーはイメージをテキストに組み合わせます。

たとえば次の図は、デフォルトのアイテムレンダラーを使用して 3 つのアイテムを表示する List コントロールを示しています。



デフォルトのアイテムレンダラー

この例では、List コントロールはデフォルトのアイテムレンダラーを使用して、アラスカ、アラバマ、およびアーカンソーの郵便番号を示す 3 つのストリングを表示しています。この List コントロールを作成するには、次のコードを使用します。

```
<?xml version="1.0"?>
<!-- itemRenderers\list\ListDefaultRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:List width="50" height ="75">
        <mx:dataProvider>
            <mx:String>AK</mx:String>
            <mx:String>AL</mx:String>
            <mx:String>AR</mx:String>
        </mx:dataProvider>
    </mx:List>
</mx:Application>
```

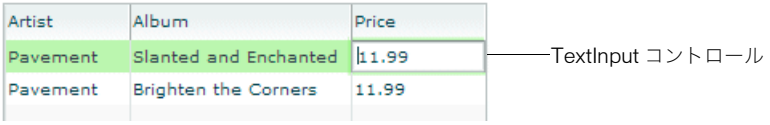
次の例では、DataGrid コントロールのデータプロバイダに、アーチスト、アルバム名、および価格を表示するフィールドが含まれています。これらのフィールドはそれぞれ、データをテキストとして表示するデフォルトのアイテムレンダラーを使用して、DataGrid コントロールに表示されています。

次のコードを使用すると、前の図に示した例が作成されます。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\DGDefaultRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[
          import mx.collections.ArrayCollection;

          [Bindable]
          private var initDG:ArrayCollection = new ArrayCollection([
            {Artist:'Pavement', Album:'Slanted and Enchanted', Price:11.99},
            {Artist:'Pavement', Album:'Brighten the Corners', Price:11.99}
          ]);
        ]]>
    </mx:Script>

    <mx:Panel paddingTop="10" paddingBottom="10"
        paddingLeft="10" paddingRight="10">
```

```
        <mx:DataGrid id="myGrid" dataProvider="{initDG}"
            width="100%" editable="true">
            <mx:columns>
                <mx:DataGridColumn dataField="Artist" resizable="true"/>
                <mx:DataGridColumn dataField="Album" resizable="true"/>
                <mx:DataGridColumn dataField="Price" resizable="true"/>
            </mx:columns>
        </mx:DataGrid>
    </mx:Panel>
</mx:Application>
```

編集可能なコンポーネントを使用すると、リストコントロールのデータを変更して、その変更内容を、コントロールの基になるデータプロバイダに戻せます。DataGrid、List、Tree の各コントロールには、editable という名前のプロパティがあり、このプロパティが true に設定されている場合、セルの内容を編集できます。デフォルトでは、リストコントロールは TextInput コントロールをアイテムエディタとして使用します。つまりリストコントロールでセルを選択すると、次の図のように、セルの現在の内容を含む TextInput コントロールが自動的に開き、内容を編集できるようになります。



この図では、デフォルトのアイテムエディタを使用して、DataGrid コントロールの最初のアルバムの価格を編集しています。

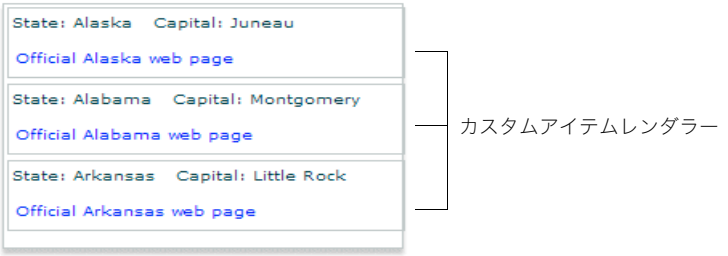
アイテムエディタの詳細については、823 ページ、第 22 章の「アイテムエディタの操作」を参照してください。

カスタムアイテムレンダラーとカスタムアイテムエディタの使用

リストコンポーネントの表示を制御するには、カスタムアイテムレンダラーまたはカスタムアイテムエディタを作成します。カスタムアイテムレンダラーとカスタムアイテムエディタでもリストコントロールの基になる機能を使用しますが、この 2 つを使用すると、データの表示と編集を制御できるようになります。カスタムアイテムレンダーとカスタムアイテムエディタには、次のようないくつかの利点があります。

- テキストの表示をユーザーによりわかりやすい外観に変えることで、さらに効果的なユーザーインターフェイスを作成できます。
- 複数のエレメントを、ラベルやイメージなどの 1 つのリストアイテムに組み込めます。
- データの表示をプログラムで制御できます。

次の List コントロールは、780 ページの「デフォルトのアイテムレンダリングとセル編集」で説明している List コントロールを変更したものです。この例では、カスタムアイテムレンダラーを使用して、州名、州都、および各州の公式ウェブサイトの URL をそれぞれのリストアイテムに表示しています。

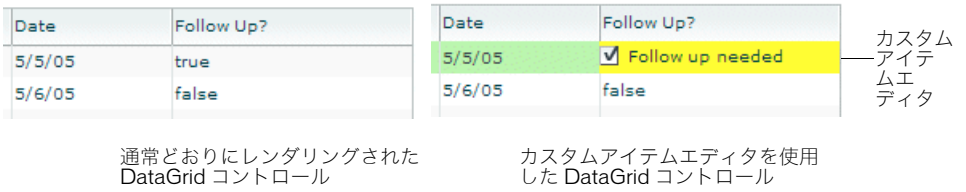


この例のコードについては、818 ページの「例 : アイテムレンダラーを List コントロールと共に使用する」を参照してください。

次の図では、DataGrid コントロールの最初と 3 番目の列にデフォルトのアイテムレンダラーを使用しています。また 2 番目の列にはカスタムアイテムレンダラーを使用して、DataGrid コントロールにアルバムカバーとアルバムタイトルを表示しています。

この例のコードについては、802 ページの「複雑なインラインアイテムレンダラーまたはインラインアイテムエディタの作成」を参照してください。

カスタムアイテムレンダラーを定義してセルの表示を制御できるのと同じように、カスタムアイテムエディタを使用してセルの内容を編集できます。たとえば、カスタムアイテムレンダラーでイメージを表示する場合、ComboBox コントロールを使用するカスタムアイテムエディタを定義して、使用可能なイメージのリストからユーザーにイメージを選択させることができます。または次の例のように、CheckBox コントロールを使用して、セルの `true` または `false` の値をユーザーに設定させることもできます。



この例のコードについては、810 ページの「単純なアイテムエディタコンポーネントの作成」を参照してください。

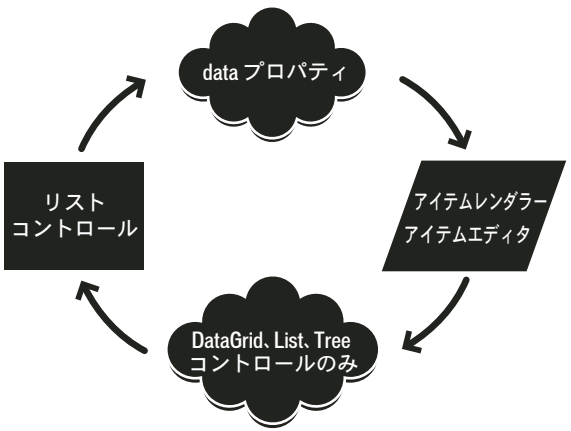
アイテムレンダラーを使用していても、そのコントロールにアイテムエディタが存在しないこともあります。多くの場合、コントロールは表示専用で、編集できません。

また、対応するアイテムレンダラーを使用せずに、アイテムエディタを使用することもできます。たとえばデフォルトのアイテムレンダラーを使用して、男性や女性、または true や false などの情報をテキストとして表示できます。ただしその後でカスタムアイテムエディタを ComboBox コントロールと共に使用し、値の変更時にセルに入力できる値の選択肢を表示することができます。

アイテムレンダラーとアイテムエディタのアーキテクチャ

アイテムレンダラーまたはアイテムエディタでリストコントロールの内容を正しく処理するため、リストコントロールからアイテムレンダラーまたはアイテムエディタに情報を渡せるようにする必要があります。また、アイテムエディタからリストコントロールに更新された情報を戻せるようにします。

次の図は、リストコントロールとアイテムレンダラーまたはアイテムエディタとの関係を示しています。



アイテムレンダラーまたはアイテムエディタに情報を渡すために、Flex では、セルデータをアイテムレンダラーまたはアイテムエディタの `data` プロパティに設定します。前の図に、アイテムレンダラーまたはアイテムエディタにデータを渡すために使用されている `data` プロパティが示されています。

デフォルトでは、アイテムエディタからリストコントロールに戻せる値は、編集されたセルの新しい値になる1つだけです。複数の値を戻すには、List コントロールにデータを戻す追加のコードを作成する必要があります。詳細については、843 ページの「例 : アイテムエディタから複数の値を受け取る」を参照してください。

アイテムレンダラーまたはアイテムエディタで使用して、リストコントロールのセルデータにアクセスする必要がある Flex コンポーネントは、IDataRenderer インターフェイスを実装して `data` プロパティをサポートする必要があります。リストコントロールから渡されたデータにアクセスする必要がなければ、`data` プロパティをサポートしないコンポーネントでも、アイテムレンダラーで使用できます。

`data` プロパティの内容は、次の表のようにコントロールのタイプによって異なります。

コントロール	data プロパティの内容
DataGrid	DataGrid コントロールの行全体で使用するデータプロバイダエレメントを含みます。つまり、DataGrid コントロールのそれぞれのセルに異なるアイテムレンダラーを使用する場合でも、行の各セルは `data` プロパティの同じ情報を受け取ります。
Tree List	List コントロールまたは Tree コントロールのノードで使用するデータプロバイダエレメントを含みます。
HorizontalList TileList	セルで使用するデータプロバイダエレメントを含みます。
Menu	メニューアイテムで使用するデータプロバイダエレメントを含みます。
MenuBar	メニューバーのアイテムで使用するデータプロバイダエレメントを含みます。

アイテムエディタの詳細については、823 ページ、第 22 章の「アイテムエディタの操作」を参照してください。

アイテムレンダラーとアイテムエディタのインターフェイスについて

Flex のアイテムレンダラーとアイテムエディタのアーキテクチャは、複数のインターフェイスによって定義されます。アイテムレンダラーやアイテムエディタとして使用できる Flex コンポーネントは、これらのインターフェイスを実装します。独自のカスタムコンポーネントを作成して、アイテムレンダラーまたはアイテムエディタとして使用する場合は、使用目的に応じて、これらのインターフェイスをコンポーネントに実装する必要があります。

次の表で、アイテムレンダラーとアイテムエディタで使用するインターフェイスについて説明します。

インターフェイス	用途
IDataRenderer	アイテムレンダラーまたはアイテムエディタに情報を渡すために使用する `data` プロパティを定義します。すべてのアイテムレンダラーとアイテムエディタに、このインターフェイスを実装する必要があります。チャートレンダラークラスなどの多くの Flex コンポーネントや、多くの Flex コントロールがこのインターフェイスを実装しています。
IDropInListItemRenderer	ドロップインのアイテムレンダラーやアイテムエディタに必要な `listData` プロパティを定義します。ドロップインのアイテムレンダラーとアイテムエディタすべてに、このインターフェイスを実装する必要があります。詳細については、796 ページの「ドロップインアイテムレンダラーとドロップインアイテムエディタの作成」を参照してください。 `listData` プロパティのタイプは BaseListData で、BaseListData クラスには、DataGridListData、ListData、TreeListData の 3 つのサブクラスがあります。`listData` プロパティの値の実際のデータタイプは、ドロップインのアイテムレンダラーまたはアイテムエディタを使用するコントロールによって異なります。DataGrid コントロールの場合、値のタイプは DataGridListData で、List コントロールの場合は ListData です。また Tree コントロールの場合は TreeListData です。
IListItemRenderer	Chart コントロール以外のすべての Flex コントロールで使用するために、アイテムレンダラーまたはアイテムエディタに実装する必要があるインターフェイスの完全セットを定義します。Chart コントロールのレンダラーに実装する必要があるのは、IDataRenderer インターフェイスのみです。 インターフェイスの完全セットに含まれるのは、IDataRenderer、IFlexDisplayObject、ILayoutClient、IStyleable、IUIComponent です。 UIComponent クラスは、IDataRenderer インターフェイスを除くこれらのインターフェイスをすべて実装しています。そのため、カスタムアイテムレンダラーまたはカスタムアイテムエディタを UIComponent クラスのサブクラスとして作成する場合、実装する必要があるのは IDataRenderer インターフェイスのみです。ドロップインのアイテムレンダラーまたはアイテムエディタを UIComponent クラスのサブクラスとして実装する場合、実装する必要があるのは IDataRenderer インターフェイスと IDropInListItemRenderer インターフェイスのみです。

アイテムレンダラーとアイテムエディタを使用したアプリケーションレイアウト

TileList コントロールと HorizontalList コントロールを除くすべてのリストコントロールは、そのコンテンツのデフォルトの高さと幅を計算する際に、アイテムレンダラーを無視します。そのため、デフォルト以外のサイズを必要とするカスタムアイテムレンダラーを定義する場合は、コントロールのサイズを明示的に指定する必要があります。

List コントロールや Tree コントロールなどの垂直方向のコントロールは、`rowHeight` プロパティを使用してデフォルトのセルの高さを決定します。行ごとに高さが異なる場合は、`variableRowHeight` プロパティを `true` に設定することもできます。この設定を行わないと、Flex によって各セルの高さが 20 ピクセルに設定されます。

HorizontalList コントロールの場合、`columnWidth` プロパティの値が指定されていれば、Flex ではこの値を使用します。TileList コントロールの場合、Flex では `columnWidth` プロパティと `rowHeight` プロパティの値を使用します。これらのプロパティを省略すると、Flex はコントロールのデータプロバイダを調べ、アイテムレンダラーが指定されている場合はそのサイズも含めて、各セルのデフォルトのサイズを決定します。これ以外の場合、Flex は各セルの高さと幅を 50 ピクセルに設定します。

アイテムレンダラーとアイテムエディタの作成

カスタムアイテムレンダラーやカスタムアイテムエディタを使用する場合、最初に決定しなければならないことはこれらの実装方法です。アイテムレンダラーとアイテムエディタの定義方法には、次の 3 とおりあります。

ドロップインアイテムレンダラーとドロップインアイテムエディタ　コンポーネントを 1 つ挿入して、アイテムレンダラーまたはアイテムエディタを定義します。詳細については、788 ページの「ドロップインアイテムレンダラーまたはドロップインアイテムエディタの使用」を参照してください。

インラインアイテムレンダラーとインラインアイテムエディタ　リストコントロールの子タグを使用してコンポーネントを定義し、アイテムレンダラーまたはアイテムエディタを定義します。詳細については、789 ページの「インラインアイテムレンダラーまたはインラインアイテムエディタの使用」を参照してください。

アイテムレンダラーとアイテムエディタのコンポーネント　アイテムレンダラーまたはアイテムエディタを再利用可能なコンポーネントとして定義します。詳細については、790 ページの「アイテムレンダラーまたはアイテムエディタとしてのコンポーネントの使用」を参照してください。

次のセクションで、それぞれの方法について説明します。

ドロップインアイテムレンダラーまたはドロップインアイテムエディタの使用

NumericStepper コントロールを使用して DataGrid コントロールのフィールドを編集するなどの単純なアイテムレンダラーやアイテムエディタでは、ドロップインアイテムエディタを使用できます。ドロップインアイテムレンダラーまたはドロップインアイテムエディタは、リストコントロールの itemRenderer プロパティまたは itemEditor プロパティの値として指定する Flex コンポーネントです。

次の例では、NumericStepper コントロールを DataGrid コントロールの列のアイテムエディタとして指定します。

```
<?xml version="1.0"?>
<!-- itemRenderers\dropin\DropInNumStepper.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection = new ArrayCollection([
                {label1:"Order #2314", quant:3, Sent:true},
                {label1:"Order #2315", quant:3, Sent:false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myDG" dataProvider="{myDP}"
        variableRowHeight="true"
        editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="label1"
                headerText="Order #" />
            <mx:DataGridColumn dataField="quant"
                itemEditor="mx.controls.NumericStepper"
                editorDataField="value" />
        </mx:columns >
    </mx:DataGrid>
</mx:Application>
```

この例では、編集のために列のセルを選択すると、NumericStepper コントロールがそのセルに表示されます。さらに data プロパティが自動的に使用され、その列の現在の値で NumericStepper コントロールが設定されます。

editorDataField プロパティを使用して、セルに新しいデータを返すアイテムエディタのプロパティを指定します。デフォルトでは、リストコントロールは TextInput コントロールの text フィールドを使用して新しい値を提供します。そのため、editorDataField プロパティのデフォルトの値は "text" です。この例では、NumericStepper の value フィールドからセルに新しい値が渡されるように指定しています。

ドロップインアイテムレンダラーやドロップインアイテムエディタとして使用できるのは、IDropInListItemRenderer インターフェイスを実装している、Flex コンポーネントのサブセットのみです。ドロップインアイテムレンダラーとドロップインアイテムエディタの詳細、およびこの 2 つをサポートするコントロールについては、796 ページの「ドロップインアイテムレンダラーとドロップインアイテムエディタの作成」を参照してください。

インラインアイテムレンダラーまたはインラインアイテムエディタの使用

788 ページの「ドロップインアイテムレンダラーまたはドロップインアイテムエディタの使用」のセクションの例は、ドロップインアイテムレンダラーとドロップインアイテムエディタの使いやすさを示しています。これらを使用する上での唯一の欠点は設定できないことです。ドロップインアイテムレンダラーとドロップインアイテムエディタを指定できるのは、リストコントロールのプロパティの値としてのみです。

柔軟性の高いアイテムレンダラーやアイテムエディタを作成するには、これらをインラインコンポーネントとして開発します。次の例では、前の例を変更して、NumericStepper コントロールをインラインアイテムエディタとして使用しています。インラインアイテムエディタでは、スタンドアローンのコントロールとして使用している場合と同じように、NumericStepper コントロールを設定できます。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineNumStepper.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection = new ArrayCollection([
                {label1:"Order #2314", quant:3, Sent:true},
                {label1:"Order #2315", quant:3, Sent:false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myDG" dataProvider="{myDP}"
        variableRowHeight="true"
        editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="label1" headerText="Order #"/>
            <mx:DataGridColumn dataField="quant" editorDataField="value">
                <mx:itemEditor>
                    <mx:Component>
                        <mx:NumericStepper stepSize="1" maximum="50"/>
                    </mx:Component>
                </mx:itemEditor>
```

```
            </mx:DataGridColumn>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、NumericStepper コントロールを列のアイテムエディタとして定義し、NumericStepper コントロールの最大値を 50、増分単位を 1 に設定しています。

インラインアイテムレンダラーとインラインアイテムエディタの作成の詳細については、800 ページの「インラインアイテムレンダラーとインラインアイテムエディタの作成」を参照してください。

アイテムレンダラーまたはアイテムエディタとしてのコンポーネントの使用

ドロップインアイテムレンダラー、ドロップインアイテムエディタ、インラインアイテムレンダラー、およびインラインアイテムエディタを使用することの短所は、アイテムレンダラーやアイテムエディタをアプリケーションファイルの中で定義するため、そのアプリケーションの他の場所や、他のアプリケーションで再利用できないことです。Flex コンポーネントとして再利用可能なアイテムレンダラーやアイテムエディタを作成すると、そのアイテムレンダラーを必要とするアプリケーションのどの場所でも使用できます。

たとえば、次のコードは NumericStepper コントロールを使用して、カスタムアイテムエディタを MXML コンポーネントとして定義しています。

```
<?xml version="1.0"?>
<!-- itemRenderers\component\myComponents\NSEditor.mxml -->
<mx:NumericStepper xmlns:mx="http://www.adobe.com/2006/mxml"
    stepSize="1"
    maximum="50"/>
```

ここでは、カスタム MXML コンポーネントで、アイテムエディタを NumericStepper コントロールとして定義しています。この例では、カスタムアイテムエディタを NSEditor と命名し、MXML コンポーネントとして "NSEditor.mxml" ファイルに実装しています。この "NSEditor.mxml" ファイルを、アプリケーションのメインディレクトリの下の myComponents ディレクトリに置きます。

その後は、アプリケーションのどの場所でもこのコンポーネントを使用できます。次に例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\component\MainNSEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection = new ArrayCollection([
                {label1:"Order #2314", quant:3, Sent:true},
                {label1:"Order #2315", quant:3, Sent:false}
            ]);
```

```
        ]]>
    </mx:Script>

    <mx:DataGrid id="myDG" dataProvider="{myDP}"
            variableRowHeight="true"
            editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="label1"
                headerText="Order #"/>
            <mx:DataGridColumn dataField="quant"
                itemEditor="myComponents.NSEditor"
                editorDataField="value"/>
        </mx:columns >
    </mx:DataGrid>
</mx:Application>
```

DataGrid コントロールの quant 列でセルを選択すると、NumericStepper コントロールが現在のセルの値で表示されます。

アプリケーションの中に、ユーザーが数値を変更できる場所がいくつかある場合があります。このアイテムエディタはカスタムコンポーネントとして定義されているので、アプリケーションのどの場所でも再利用できます。

コンポーネントとしてのアイテムレンダラーとアイテムエディタの作成の詳細については、808 ページの「アイテムレンダラーとアイテムエディタコンポーネントの作成」を参照してください。

アイテムレンダラーでの編集可能なコントロールの使用

アイテムレンダラーは、その中で使用できる Flex コンポーネントのタイプを制限しません。たとえば、Label、LinkButton、Button、Text などのコントロールを使用してデータを表示することはできますが、これらのコントロールで、コントロールの内容を変更することはできません。

または CheckBox、ComboBox、TextInput などのコントロールを使用すると、データを表示できるだけでなく、コントロールを操作してコントロール自体を変更できます。たとえば CheckBox コントロールを使用して、選択された (`true` 値) または選択されていない (`false` 値) セルを DataGrid コントロールに表示できます。

CheckBox コントロールを含む DataGrid コントロールのセルを選択すると、コントロールを操作してその状態を変更できます。ユーザーには、DataGrid コントロールが編集可能であるように表示されます。

ただしデフォルトではアイテムレンダラーはリストコントロールの編集機能と連携していないため、ユーザーがセルを変更しても、変更内容はリストコントロールのデータプロバイダに反映されず、またイベントも送出されません。ユーザーにはリストコントロールが編集可能に見えていても、実際には編集できません。

他の例で、ユーザーが CheckBox コントロールの値を変更し、その後で DataGrid 列をソートしたとします。しかし、DataGrid は CheckBox コントロールの値ではなく、データプロバイダの値でセルをソートするため、ユーザーはソートが正しく実行されていないことに気付きます。

この状況は、次のようないくつかの方法で対処できます。

- ユーザーが変更できるコントロール (CheckBox や ComboBox など) をアイテムレンダラーで使用しないようにします。
- これらのコントロールのカスタムバージョンを作成して、ユーザーがこれらを操作できないようにします。
- リストコントロールの `rendererIsEditor` プロパティを使用して、そのアイテムレンダラーがアイテムエディタにもなるように指定します。詳細については、847 ページの「例 : アイテムレンダラーをアイテムエディタとして使用する」を参照してください。
- アイテムレンダラーとそれを受け入れるコントロールのコードを独自に作成し、ユーザーにアイテムレンダラーを操作させたときはデータを戻すようにします。詳細および使用例については、843 ページの「例 : アイテムエディタから複数の値を受け取る」を参照してください。

ActionScript での itemRenderer プロパティまたは itemEditor プロパティの設定

`itemRenderer` プロパティと `itemEditor` プロパティのタイプは IFactory です。この 2 つのプロパティを MXML で設定すると、プロパティの値は、MXML コンパイラーによって自動的に ClassFactory タイプにキャストされます。ClassFactory は、IFactory インターフェイスを実装するクラスです。

これらのプロパティを ActionScript で設定する場合は、次の例のように、プロパティの値を ClassFactory に明示的にキャストする必要があります。

```
<?xml version="1.0"?>
<!-- itemRenderers\list\AppListStateRendererEditorAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

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

            // Cast the value of the itemRenderer property
            // to ClassFactory.
            public function initCellEditor():void {
                myList.itemRenderer=new ClassFactory(RendererState);
            }
        ]]>
    </mx:Script>

    <mx:List id="myList" variableRowHeight="true"
            height="180" width="250"
```

```
            backgroundColor="white"
            initialize="initCellEditor();">
        <mx:dataProvider>
            <mx:Object label="Alaska"
                data="Juneau"
                webPage="http://www.state.ak.us/"/>
            <mx:Object label="Alabama"
                data="Montgomery"
                webPage="http://www.alabama.gov/" />
            <mx:Object label="Arkansas"
                data="Little Rock"
                webPage="http://www.state.ar.us/"/>
        </mx:dataProvider>
    </mx:List>
</mx:Application>
```

アイテムレンダラーとアイテムエディタのライフサイクルについて

アイテムレンダラーとアイテムエディタのインスタンスは、表示や測定のためにアプリケーションで必要になったときに Flex によって作成されます。そのため、アプリケーション内のアイテムレンダラーまたはアイテムエディタのインスタンスの数は、表示されているアイテムレンダラーの数とは必ずしも直接関連しません。また、リストコントロールをスクロールまたはサイズ変更すると、アイテムレンダラーの 1 つのインスタンスが再利用されて複数のデータアイテムを示す場合があります。そのため、同時にアクティブになっているアイテムレンダラーやアイテムエディタの数を推測で判断することは避けてください。

Flex ではアイテムレンダラーを再利用できるため、その状態を完全に定義する必要があります。たとえば、アイテムレンダラーの CheckBox コントロールを使用して、data プロパティの現在の値を基に true (チェックマークあり) または false (チェックマークなし) の値を表示するとします。ここでよくある間違いは、アイテムレンダラーの CheckBox コントロールは常にチェックなしのデフォルト状態だと思い込むことです。そこでデベロッパーは、data プロパティの true の値を調べて、見つかったら CheckBox コントロールにチェックを付けるアイテムレンダラーを作成します。

しかし、CheckBox が既にチェックされている可能性があることも考慮する必要があります。そのため、data プロパティの false の値を調べ、コントロールがチェックされている場合はそのチェックを明示的にオフにすることが必要です。

listData プロパティへのアクセス

コンポーネントが IDropInListItemRenderer インターフェイスを実装している場合、アイテムレンダラーまたはアイテムエディタでそのコンポーネントを使用すると、コンポーネントに渡されたデータについての情報を listData プロパティで取得できます。listData プロパティのタイプは BaseListData で、BaseListData クラスには次のプロパティが定義されています。

プロパティ	説明
owner	アイテムレンダラーまたはアイテムエディタを使用するリストコントロールへの参照。
rowIndex	DataGrid、List、または Tree コントロールの行の、現在表示されているコントロールの行に対するインデックス。最初の行はインデックス 1 になります。
text	List クラスの itemToLabel() メソッドに基づくアイテムデータのテキスト表示。

BaseListData クラスには、追加のプロパティを定義する DataGridListData、ListData、TreeListData という 3 つのサブクラスがあります。たとえば DataGridListData クラスでは、アイテムレンダラーまたはアイテムエディタからアクセスできる columnIndex プロパティと dataField プロパティを追加します。

listData プロパティの値のデータタイプは、アイテムレンダラーまたはアイテムエディタを使用するコントロールによって異なります。DataGrid コントロールの場合、値のタイプは DataGridListData で、List コントロールの場合は ListData です。また Tree コントロールの場合は TreeListData です。

TextArea コントロールは、IDropInListItemRenderer インターフェイスを実装している Flex コントロールの 1 つです。次の例のアイテムレンダラーは、TextArea コントロールの listData プロパティを使用して、各アイテムレンダラーの行と列を DataGrid コントロールに表示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\myComponents\RendererDGListData.mxml -->
<mx:TextArea xmlns:mx="http://www.adobe.com/2006/mxml"
    preinitialize="initTA();">

    <mx:Script>
        <![CDATA[

            import mx.controls.dataGridClasses.DataGridListData;
            import flash.events.Event;

            public function initTA():void {
                addEventListener("dataChange", handleDataChanged);
            }

            public function handleDataChanged(event:Event):void {
                // Cast listData to DataGridListData.
                var myListData:DataGridListData =
                    DataGridListData(listData);
```

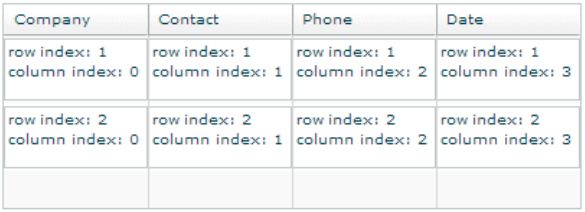
```
                // Access information about the data passed
                // to the cell renderer.
                text="row index: " + String(myListData.rowIndex) +
                    " column index: " + String(myListData.columnIndex);
            }
        ]]>
    </mx:Script>
</mx:TextArea>
```

このアイテムレンダラーは DataGrid コントロールで使用されているため、listData プロパティの値のデータタイプは DataGridListData です。この例では、data プロパティが変更されるたびに、dataChange イベントを使用して TextArea コントロールの内容を設定します。

次の例の DataGrid コントロールは、このアイテムレンダラーを使用します。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\MainDGListDataRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            public var initDG:ArrayCollection = new ArrayCollection([
                { Company: 'Acme', Contact: 'Bob Jones',
                    Phone: '413-555-1212', Date: '5/5/05'},
                { Company: 'Allied', Contact: 'Jane Smith',
                    Phone: '617-555-3434', Date: '5/6/05'}
            ]);
        ]]>
    </mx:Script>

    <mx:Panel paddingTop="10" paddingBottom="10"
        paddingLeft="10" paddingRight="10" >

        <mx:DataGrid id="myGrid" dataProvider="{initDG}"
                variableRowHeight="true">
            <mx:columns>
                <mx:DataGridColumn dataField="Company"
                    itemRenderer="myComponents.RendererDGListData"/>
                <mx:DataGridColumn dataField="Contact"
                    itemRenderer="myComponents.RendererDGListData"/>
                <mx:DataGridColumn dataField="Phone"
                    itemRenderer="myComponents.RendererDGListData"/>
                <mx:DataGridColumn dataField="Date"
                    itemRenderer="myComponents.RendererDGListData"/>
            </mx:columns>
        </mx:DataGrid>
    </mx:Panel>
</mx:Application>
```

このコードで、次の図の DataGrid コントロールが生成されます。



この図からわかるように、DataGrid コントロールの各セルは該当する列と行のインデックスを表示しています。

ドロップインアイテムレンダラーとドロップインアイテムエディタの作成

いくつかの Flex コントロールは、アイテムレンダラーやアイテムエディタとして機能するように設計されています。そのためこれらのコントロールを、`itemRenderer` プロパティまたは `itemEditor` プロパティの値として指定できます。プロパティの値として指定されたコントロールを、ドロップインアイテムレンダラーまたはドロップインアイテムエディタと呼びます。

コンポーネントをドロップインアイテムレンダラーまたはドロップインアイテムエディタとして使用するには、コンポーネントに IDropInListItemRenderer インターフェイスを実装する必要があります。次のコントロールには IDropInListItemRenderer インターフェイスが実装されているため、ドロップインアイテムレンダラーまたはドロップインアイテムエディタとしてそのまま再利用できます。

- Button
- CheckBox
- DateField
- Image
- Label
- NumericStepper
- Text
- TextArea
- TextInput

ドロップインアイテムレンダラーまたはドロップインアイテムエディタとして使用するために、独自のコンポーネントを定義することができます。その際に必要なことは、定義したコンポーネントにも IDropInListItemRenderer インターフェイスを実装することだけです。

ドロップインアイテムレンダラーとドロップインアイテムエディタの使用

コントロールをドロップインアイテムレンダラーとして使用すると、そのコントロールのデフォルトプロパティが Flex によってセルの現在の値に設定されます。コントロールをアイテムエディタとして使用すると、セルの初期値がコントロールの現在の値になります。セルに対して行った編集はすべて、リストコントロールのデータプロバイダにコピーされます。

次の表は、ドロップインアイテムレンダラーとドロップインアイテムエディタとして使用できるコンポーネントと、そのコンポーネントのデフォルトプロパティの一覧です。

コントロール	デフォルトプロパティ	注意事項
Button	selected	Button コントロールにはラベルを指定できません。
CheckBox	selected	Tree コントロールでドロップインアイテムレンダラーとして使用する場合、Tree ではテキストのないチェックボックスのみを表示します。
DateField	selectedDate	リストコントロールのデータプロバイダには Date タイプのフィールドが必要です。
Image	source	rowHeight プロパティを使用して明示的に行の高さを指定するか、variableRowHeight プロパティを true に設定して行を適切な大きさにします。
Label	text	
NumericStepper	value	
Text	text	
TextArea	text	
TextInput	text	

次の例では、NumericStepper、DateField、CheckBox の各コントロールを、DataGrid コントロールのドロップインアイテムレンダラーとドロップインアイテムエディタとして使用しています。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\CBInlineCellEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection = new ArrayCollection([
                {label1:"Order #2314", contact:"John Doe",
                    quant:3, solddate:new Date(2005, 0, 1), Sent:true},
                {label1:"Order #2315", contact:"Jane Doe",
```

```
                    quant:3, solddate:new Date(2005, 0, 5), Sent:false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myDG"
            dataProvider="{myDP}"
            variableRowHeight="true"
            width="500" height="250"
            editable="true">
        <mx:columns>
            <mx:DataGridColumn dataField="label1"
                headerText="Order #"
                editable="false"/>
            <mx:DataGridColumn dataField="quant"
                headerText="Quantity"
                itemEditor="mx.controls.NumericStepper"
                editorDataField="value"/>
            <mx:DataGridColumn dataField="solddate"
                headerText="Date"
                itemRenderer="mx.controls.DateField"
                rendererIsEditor="true"
                editorDataField="selectedDate"/>
            <mx:DataGridColumn dataField="Sent"
                itemRenderer="mx.controls.CheckBox"
                rendererIsEditor="true"
                editorDataField="selected"/>
        </mx:columns >
    </mx:DataGrid>
</mx:Application>
```

ドロップインアイテムレンダラーの設定に使用するデータプロバイダのフィールドを決定するために、Flex では <mx:DataGridColumn> タグの dataField プロパティの値を使用します。この例では、次のことを実行します。

- 2 番目の列の dataField プロパティを quant に設定し、NumericStepper コントロールをアイテムエディタとして定義します。そのためこの列ではセルの値がテキストとして表示され、編集のためにセルを選択すると NumericStepper コントロールが開きます。NumericStepper コントロールによってセルの現在の値が表示され、セルに対して行った変更はすべて DataGrid コントロールのデータプロバイダにコピーされます。
- 3 番目の列の dataField プロパティを solddate に設定し、rendererIsEditor プロパティを true に設定して、DateField コントロールをアイテムレンダラーとアイテムエディタとして定義します。データプロバイダではこのデータは Date タイプのオブジェクトとして定義されています。DateField コントロールをアイテムレンダラーまたはアイテムエディタとして使用するためには、データは Date タイプに定義されている必要があります。そのため、列では DateField コントロールを使用してデータの値を表示し、またセルの値の編集にも DateField コントロールを使用します。

■ 4 番目の列の dataField プロパティを Sent に設定し、CheckBox コントロールをアイテムレンダラーとアイテムエディタとして定義します。通常、アイテムエディタとして異なるクラスを指定するには itemEditor プロパティを使用し、同一のコントロールをアイテムレンダラーとアイテムエディタの両方に使用するように指定するには、rendererIsEditor プロパティを true に設定します。行の Sent フィールドを true に設定している場合、CheckBox コントロールによってセルにチェックマークが表示されます。Sent フィールドを false に設定している場合、空のチェックボックスが表示されます。

アイテムレンダラーをアイテムエディタとして使用する際の詳細については、824 ページの「編集可能なセルの作成」を参照してください。

Image コントロールをドロップインアイテムレンダラーとして使用する場合、通常はイメージが収まるように行の高さを設定する必要があります。次に例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\dropin\DGDropInImageRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                { Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99, Cover: 'slanted.jpg' },
                { Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99, Cover: 'brighten.jpg'}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid" dataProvider="{initDG}" rowHeight="50">
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
            <mx:DataGridColumn dataField="Cover"
                itemRenderer="mx.controls.Image"/>
            <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、Image コントロールを使用して、DataGrid コントロールのセルにアルバムカバーを表示しています。

List コントロールでのドロップインアイテムレンダラーの要件

List コントロールの itemRenderer プロパティを使用してドロップインアイテムレンダラーを指定する場合、データプロバイダのデータは、アイテムレンダラーのコントロールが使用するタイプでなければなりません。たとえば、Image コントロールをドロップインレンダラーとして使用する場合、List コントロールのデータプロバイダのそのフィールドは String データである必要があります。

インラインアイテムレンダラーとインラインアイテムエディタの作成

インラインアイテムレンダラーとインラインアイテムエディタの定義は、コンポーネントの MXML 定義の中で行います。インラインアイテムレンダラーとインラインアイテムエディタを使用すると、アイテムレンダリングとセルの編集を完全に制御できます。

単純なインラインアイテムレンダラーまたはインラインアイテムエディタの作成

単純なインラインアイテムレンダラーには、data プロパティをサポートするコントロールが 1 つ入っています。Flex によって現在のセルのデータが自動的にアイテムレンダラーにコピーされます。次に例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineImageRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99, Cover: 'slanted.jpg'},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99, Cover: 'brighten.jpg'}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid" dataProvider="{initDG}" rowHeight="50">
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
```

```
            <mx:DataGridColumn dataField="Cover">
                <mx:itemRenderer>
                    <mx:Component>
                        <mx:Image height="45"/>
                    </mx:Component>
                </mx:itemRenderer>
            </mx:DataGridColumn>
            <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、Image コントロールを使用してアルバムカバーを表示しています。

単純なインラインアイテムエディタには、data プロパティをサポートするコントロールが1つ入っています。Flex によって現在のセルのデータがアイテムレンダラーまたはアイテムエディタにコピーされ、新しいセルのデータが editorDataField プロパティの値を基にリストコントロールにコピーされます。次にアイテムエディタの例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineNumStepper.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection = new ArrayCollection([
                {label1:"Order #2314", quant:3, Sent:true},
                {label1:"Order #2315", quant:3, Sent:false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myDG" dataProvider="{myDP}"
        variableRowHeight="true"
        editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="label1" headerText="Order #"/>
            <mx:DataGridColumn dataField="quant" editorDataField="value">
                <mx:itemEditor>
                    <mx:Component>
                        <mx:NumericStepper stepSize="1" maximum="50"/>
                    </mx:Component>
                </mx:itemEditor>
            </mx:DataGridColumn>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、NumericStepper コントロールの value プロパティを使用して、新しいセルの値を戻しています。

複雑なインラインアイテムレンダラーまたはインラインアイテムエディタの作成

複雑なアイテムレンダラーまたはアイテムエディタでは、複数のコントロールを定義します。たとえば、780 ページの「デフォルトのアイテムレンダリングとセル編集」のセクションでは、3 つのテキストフィールドを使用してアルバムの情報を表示する DataGrid コントロールを示しました。DataGrid コントロールにビジュアルエレメントを追加すると、視覚効果を高めることができます。そのために、データプロバイダを変更して、アルバムカバーの JPEG イメージの URL が含まれるようにします。

インラインアイテムレンダラーを使用すると、次の例のように、アルバム名とアルバムイメージを DataGrid コントロールの別個のセルではなく、1 つのセルに表示できます。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineDGRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.collections.ArrayCollection;

            // Variable in the Application scope.
            public var localVar:String="Application localVar";

            // Data includes URL to album cover.
            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99, Cover:'slanted.jpg'},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99, Cover:'brighten.jpg'}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid" dataProvider="{initDG}"
            variableRowHeight="true">
        <mx:columns>
                <mx:DataGridColumn dataField="Artist"/>
                <mx:DataGridColumn dataField="Album">
                    <mx:itemRenderer>
                        <mx:Component>
                            <mx:VBox>
                                <mx:Text id="albumName"
                                    width="100%" text="{data.Album}"/>
                                <mx:Image id="albumImage"
                                    height="45" source="{data.Cover}"/>
                            </mx:VBox>
                        </mx:Component>
```

```
                    </mx:itemRenderer>
                </mx:DataGridColumn>
                <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

前の例では、DataGrid コントロールに 3 つの列を定義し、アイテムレンダラーを 2 番目の列に割り当てています。このアプリケーションによる結果の図については、782 ページの「カスタムアイテムレンダラーとカスタムアイテムエディタの使用」を参照してください。

アイテムレンダラーの Text コントロールと Image コントロールは両方とも、data プロパティを使用して値を初期化してください。初期化が必要なのは、アイテムレンダラーに複数のコントロールが定義されていて、データプロバイダのどのデータエレメントがアイテムレンダラーのそれぞれのコントロールに関連付けられているかを Flex では自動的に判断できないためです。

インラインアイテムレンダラーのトップレベルコンテナに子コントロールが 1 つしかない場合でも、次の例のように、data プロパティで子コントロールを初期化する必要があります。

```
<mx:DataGridColumn dataField="Cover">
  <mx:itemRenderer>
    <mx:Component>
      <mx:VBox horizontalAlign="center">
        <mx:Image height="45" source="{data.Cover}"/>
      </mx:VBox>
    </mx:Component>
  </mx:itemRenderer>
</mx:DataGridColumn>
```

前の例では、セルの中でイメージを整列できるように、Image コントロールを VBox コンテナの子にしています。これで Image コントロールは VBox コンテナの子になったため、data プロパティを使用して初期化する必要があります。

複雑なインラインアイテムエディタでは、複数のコントロールを定義できます。これによって、リストコントロールで使用するデータプロバイダの複数の値を編集できるようになります。また、複雑なインラインアイテムエディタを定義して、String 以外の値を返すこともできます。詳細については、823 ページ、第 22 章の「アイテムエディタの操作」を参照してください。

インラインコンポーネントで許可されるアイテム

インラインアイテムレンダラーまたはインラインアイテムエディタで行えないことは、空の <mx:Component></mx:Component> タグを作成することだけです。たとえば、独自のレンダリングロジックや編集ロジックと共に、エフェクトとスタイルの定義をインラインアイテムレンダラーまたはインラインアイテムエディタで組み合わせることができます。

次のアイテムをインラインアイテムレンダラーまたはインラインアイテムエディタに組み込めます。

- バインディングタグ
- エフェクトタグ
- メタデータタグ
- モデルタグ
- スクリプトタグ
- サービスタグ
- ステートタグ
- スタイルタグ
- XML タグ
- id 属性 (最上位コンポーネントを除く)

<mx:Component> タグの使用

MXML ファイルにインラインアイテムレンダラーまたはインラインアイテムエディタを定義するには、<mx:Component> タグを使用します。このセクションでは、このタグの使用方法について説明します。

<mx:Component> タグでのスコープの定義

新しいスコープは、<mx:Component> タグで MXML ファイルに定義します。このファイルの中で、アイテムレンダラーまたはアイテムエディタのローカルスコープは、<mx:Component> タグと </mx:Component> タグで区切られた MXML コードブロックによって定義されます。アイテムレンダラーまたはアイテムエディタのローカルスコープ外のエレメントにアクセスするには、エレメント名の先頭に outerDocument キーワードを付けます。

たとえば、メインアプリケーションのスコープに localVar という名前の変数を定義し、アイテムレンダラーのスコープに同名の別の変数を定義したとします。アイテムレンダラーの中からアプリケーションの localVar にアクセスするには、この変数の先頭に outerDocument キーワードを付けます。次に例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineDGImageScope.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
<mx:Script>
    <![CDATA[
        import mx.collections.ArrayCollection;

        // Variable in the Application scope.
        [Bindable]
        public var localVar:String="Application localVar";

        // Data includes URL to album cover.
        [Bindable]
        private var initDG:ArrayCollection = new ArrayCollection([
            { Artist:'Pavement', Album:'Slanted and Enchanted',
               Price:11.99, Cover:'slanted.jpg'},
            { Artist:'Pavement', Album:'Brighten the Corners',
               Price:11.99, Cover:'brighten.jpg'}
        ]);
    ]]>
</mx:Script>

<mx:DataGrid id="myGrid" dataProvider="{initDG}" width="100%"
        variableRowHeight="true">
    <mx:columns>
        <mx:DataGridColumn dataField="Artist"/>
        <mx:DataGridColumn dataField="Album"/>
        <mx:DataGridColumn dataField="Cover">
            <mx:itemRenderer>
              <mx:Component>
                <mx:VBox>
                    <mx:Script>
                      <![CDATA[
                        // Variable in the renderer scope.
                        [Bindable]
                        public var localVar:String="Renderer localVar";
                      ]]>
                     </mx:Script>

                     <mx:Text id="albumName"
                        width="100%"
                        selectable="false"
                        text="{data.Album}"/>
                     <mx:Image id="albumImage"
                        height="45"
                        source="{data.Cover}"/>
                     <mx:TextArea
                        text="{'Renderer localVar= ' + localVar}"/>
                     <mx:TextArea
        text="{'Application localVar= ' + outerDocument.localVar}"/>
                </mx:VBox>
              </mx:Component>
            </mx:itemRenderer>
        </mx:DataGridColumn>
        <mx:DataGridColumn dataField="Price"/>
```

```
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

outerDocument キーワードの用途の 1 つは、インラインアイテムエディタでコントロールのデータプロバイダを初期化することです。たとえば、Web サービスなどのメカニズムを使用して米国の州の一覧などのデータをアプリケーションに渡したとします。このキーワードを使用すれば、アイテムエディタとして使用されているすべての ComboBox コントロールを、米国の州の一覧を渡されたアプリケーションの 1 つのプロパティから初期化することができます。

インラインコンポーネントへのクラス名の指定

<mx:Component> タグの className プロパティをオプションで指定することで、Flex によって自動生成されるインラインコンポーネントのクラスに明示的に名前を指定できます。クラス名を指定することによって、インラインコンポーネント内のエレメントを参照する方法が定義されます。

className プロパティの使用例については、843 ページの「例 : アイテムエディタから複数の値を受け取る」を参照してください。

再利用可能なインラインアイテムレンダラーまたはインラインアイテムエディタの作成

インラインアイテムレンダラーまたはインラインアイテムエディタは、コンポーネントの定義の中ではなく、再利用可能なものとして定義すると、アプリケーションの複数の場所で使用することができます。

たとえば、<mx:Component> タグを使用して、住所の州の部分を選択する ComboBox コントロールで構成されたインラインアイテムエディタを定義します。その後で、定義したインラインアイテムエディタを 2 つの異なる DataGrid コントロールで使用します。次に例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineDGEditorCBReUse.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700" >

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            public var initDG:ArrayCollection = new ArrayCollection([
                {Company: 'Acme', Contact: 'Bob Jones',
                    Phone: '413-555-1212', City: 'Boston', State: 'MA'},
                {Company: 'Allied', Contact: 'Jane Smith',
                    Phone: '617-555-3434', City: 'SanFrancisco', State: 'CA'}
            ]);
        ]]>
    </mx:Script>
```

```
    <mx:Component id="inlineEditor">
        <mx:ComboBox>
            <mx:dataProvider>
                <mx:String>AL</mx:String>
                <mx:String>AK</mx:String>
                <mx:String>AR</mx:String>
            </mx:dataProvider>
        </mx:ComboBox>
    </mx:Component>

    <mx:DataGrid id="myGrid"
        variableRowHeight="true"
        dataProvider="{initDG}"
        editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="Company" editable="false"/>
            <mx:DataGridColumn dataField="Contact"/>
            <mx:DataGridColumn dataField="Phone"/>
            <mx:DataGridColumn dataField="City"/>
            <mx:DataGridColumn dataField="State"
                width="150"
                editorDataField="selectedItem"
                itemEditor="{inlineEditor}"/>
        </mx:columns>
    </mx:DataGrid>

    <mx:DataGrid id="myGrid2"
        variableRowHeight="true"
        dataProvider="{initDG}"
        editable="true">
        <mx:columns>
            <mx:DataGridColumn dataField="Company" editable="false"/>
            <mx:DataGridColumn dataField="Contact"/>
            <mx:DataGridColumn dataField="Phone"/>
            <mx:DataGridColumn dataField="City"/>
            <mx:DataGridColumn dataField="State"
                width="150"
                editorDataField="selectedItem"
                itemEditor="{inlineEditor}"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、<mx:Component> タグで定義したインラインアイテムエディタの id プロパティを指定しています。次にデータバインディングを使用して、2 つの DataGrid コントロールの itemEditor プロパティの値としてアイテムエディタを指定しています。

<mx:Component> タグで定義したインラインアイテムエディタまたはインラインアイテムレンダラーは、DataGrid コントロールで使用されている場所にのみ表示されます。それ以外は、アプリケーションをレイアウトするときに Flex によって無視されます。

アイテムレンダラーとアイテムエディタコンポーネントの作成

MXML コンポーネントを使用してカスタムアイテムレンダラーまたはカスタムアイテムエディタを定義すると、ドロップインアイテムレンダラーやドロップインアイテムエディタを使用する場合に比べて、柔軟性と機能性が高まります。アイテムレンダラーとアイテムエディタをカスタムコンポーネントとして定義する際の規則の多くは、インラインアイテムレンダラーやインラインアイテムエディタを使用する場合と同じです。詳細については、800 ページの「インラインアイテムレンダラーとインラインアイテムエディタの作成」を参照してください。

このセクションでは、MXML コンポーネントを使用したカスタムアイテムレンダラーとカスタムアイテムエディタの定義について説明します。カスタムコンポーネントの操作方法の詳細については、『Flex 2 コンポーネントの作成と拡張』を参照してください。

アイテムレンダラーコンポーネントの作成

780 ページの「デフォルトのアイテムレンダリングとセル編集」のセクションでは、3 つのテキストフィールドを使用してアルバムの情報を表示する DataGrid コントロールを示しました。DataGrid コントロールにビジュアルエレメントを追加すると、視覚効果を高めることができます。そのために、データプロバイダを変更して、アルバムカバーの JPEG イメージの URL が含まれるようにします。

DataGrid コントロールのデフォルトアイテムレンダラーでは、データをテキストとして表示します。DataGrid コントロールでアルバムカバーのイメージを表示するには、次の例のように、"RendererDGImage.mxml" ファイルに定義されたカスタムアイテムレンダラーを使用します。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\myComponents\RendererDGCheckBox.mxml -->
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="center" >

    <mx:Image id="albumImage" height="175" source="{data.Cover}"/>
</mx:HBox>
```

このアイテムレンダラーでは、Image コントロールは HBox コンテナに入っています。イメージをコンテナの中央に配置する場合は、HBox コンテナで指定します。それ以外の場合は、イメージは左寄せでセルに表示されます。Image コントロールでイメージの高さを 75 ピクセルに指定します。デフォルトではイメージの高さは 0 ピクセルです。そのため高さを省略すると、イメージは表示されません。

data プロパティのフィールドをアイテムレンダラーまたはアイテムエディタのコントロールと関連付けるには、データバインディングを使用します。この例では、アイテムレンダラーに渡される data プロパティに、DataGrid コントロールの行全体で使用されるデータプロバイダのエレメントが含まれています。次に data プロパティの Cover フィールドを Image コントロールにバインドします。

カスタムアイテムレンダラーを DataGrid コントロールと共に使用します。次に例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\component\MainDGImageRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99, Cover: 'slanted.jpg'},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99, Cover: 'brighten.jpg'}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        variableRowHeight="true">
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
            <mx:DataGridColumn dataField="Cover"
                itemRenderer="myComponents.RendererDGImage"/>
            <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

DataGrid コントロールにはアルバムカバーの列が入っています。この列では itemRenderer プロパティを使用して、その列のアイテムレンダラーを含む MXML ファイルの名前を指定しています。ここでこの例を実行すると、DataGrid コントロールは Cover 列用に作成されたカスタムアイテムレンダラーを使用して、アルバムカバーのイメージを表示します。

アイテムレンダラーを使用すると、次の例のように、アルバム名とアルバムイメージを DataGrid コントロールの別個のセルではなく、1 つのセルに表示させることができます。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\myComponents\RendererDGTitleImage.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="center" height="75">

    <mx:Text id="albumName"
        width="100%"
        selectable="false"
        text="{data.Album}"/>
    <mx:Image id="albumImage"
        source="{data.Cover}"/>
</mx:VBox>
```

このアイテムレンダラーを "RendererDGTitleImage.mxml" ファイルに保存します。メインアプリケーションの DataGrid コントロールは、次の例のように、このアイテムレンダラーを参照します。

```
<?xml version="1.0"?>
<!-- itemRenderers\component\MainDGTitleRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99, Cover: 'slanted.jpg'},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99, Cover: 'brighten.jpg'}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        variableRowHeight="true">
        <mx:columns>
            <mx:DataGridColumn dataField="Artist" />
            <mx:DataGridColumn dataField="Album"
                itemRenderer="myComponents.RendererDGTitleImage" />
            <mx:DataGridColumn dataField="Price"  />
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

前の例では、DataGrid コントロールに 3 つの列のみを定義し、アイテムレンダラーを 2 番目の列に割り当てています。このアプリケーションによる結果の図については、782 ページの「カスタムアイテムレンダラーとカスタムアイテムエディタの使用」を参照してください。

単純なアイテムエディタコンポーネントの作成

単純なアイテムエディタコンポーネントでは、セルの編集に使用するコントロールを 1 つ定義し、リストコントロールに値を 1 つだけ返します。単純なアイテムエディタコンポーネントの例については、790 ページの「アイテムレンダラーまたはアイテムエディタとしてのコンポーネントの使用」を参照してください。

複雑なアイテムエディタには、複数のコンポーネントを含めることができます。または、リストコントロールに複数の値を戻せます。詳細については、823 ページ、第 22 章の「アイテムエディタの操作」を参照してください。

データプロパティのオーバーライド

カスタムアイテムレンダラーまたはカスタムアイテムエディタで使用し、レンダラーに渡されたデータへのアクセスを必要とするすべてのコンポーネントには、mx.core.IDataRenderer インターフェイスを実装して、data プロパティを定義する必要があります。このプロパティは、すべての Flex コンテナと多数の Flex コンポーネントでサポートされています。

クラスでは、次のシグネチャで data プロパティを setter および getter メソッドとして定義することで、mx.core.IDataRenderer インターフェイスを実装します。

```
override public function set data(value:Object):void
public function get data():Object
```

setter メソッドで、*value* はアイテムレンダラーに渡された data プロパティです。

カスタムコンポーネントを定義して data プロパティをサポートするには、コンポーネントに mx.core.IDataRenderer インターフェイスを実装する必要があります。定義したコンポーネントが、mx.core.IDataRenderer インターフェイスを実装しているクラスのサブクラスである場合は、このインターフェイスを再度実装する必要はありません。

既に data プロパティを実装しているアイテムレンダラーまたはアイテムエディタのコントロールにプログラムロジックを追加するには、data プロパティで setter または getter メソッドをオーバーライドします。通常は、渡された値を基に一部の処理を実行できるように、setter メソッドをオーバーライドします。

たとえば次の DataGrid コントロールは、データプロバイダの SalePrice フィールドの true 値と false 値を使用しています。これらの値を DataGrid コントロールに表示することもできますが、イメージを表示することによって、次のアイテムレンダラーのようにより効果的な DataGrid コントロールを作成できます。

```
<?xml version="1.0"?>
<!-- itemRenderers\component\myComponents\RendererDGImageSelect.mxml -->
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="center">

    <mx:Script>
        <![CDATA[

            import mx.events.FlexEvent;

            [Embed(source="saleIcon.jpg")]
            [Bindable]
            public var sale:Class;

            [Embed(source="noSaleIcon.jpg")]
            [Bindable]
            public var noSale:Class;

            override public function set data(value:Object):void {
                if(value != null)  {
```

```
                    super.data = value;
                    if (value.SalePrice == true) onSale.source=new sale();
                    else onSale.source=new noSale();
                }
                // Dispatch the dataChange event.
                dispatchEvent(new FlexEvent(FlexEvent.DATA_CHANGE));
            }
        ]]>
    </mx:Script>

    <mx:Image id="onSale" height="20"/>
</mx:HBox>
```

この例では、HBox コンテナの setter メソッドをオーバーライドしています。オーバーライドでは、super.data を使用して基本クラスの data プロパティを設定し、次に SalePrice フィールドの値を基に Image コントロールの source プロパティを設定しています。このフィールドが true の場合、商品は特売中なので、それを示すアイコンを表示します。この商品が特売中でない場合、別のアイコンを表示します。

またオーバーライドでは、dataChange イベントを送出して data プロパティが変更されたことを通知します。通常このイベントは setter メソッドから送出されます。

creationComplete イベントと dataChange イベントの使用について

コンポーネントが作成および初期化されると、Flex ではそのコンポーネントに対して 1 回だけ creationComplete イベントを送出します。多くのカスタムコンポーネントでは、creationComplete イベントのイベントリスナーを定義し、コンポーネントが完全に作成および初期化された後に実行する必要があるすべての後処理タスクを処理します。

ただし、アイテムレンダラーやアイテムエディタの場合、それらのインスタンスが Flex によって再利用されることがあります。しかし再利用されたインスタンスは、creationComplete イベントを再送出しません。そのためこのイベントの代わりに、dataChange イベントをアイテムレンダラーやアイテムエディタと共に使用することができます。これによって、data プロパティが変更されるたびに dataChange イベントが送出されます。794 ページの「listData プロパティへのアクセス」のセクションの例では、dataChange イベントを使用して DataGrid コントロールのアイテムレンダラーの TextArea を更新しています。

ActionScript でのアイテムレンダラーの作成

一般的に、アイテムレンダラーやアイテムエディタは MXML で作成しますが、ActionScript でも作成することができます。次にアイテムレンダラーの例を示します。

```
package myComponents {

    // myComponents/CellField.as
    import mx.controls.*;
    import mx.core.*;
    import mx.controls.dataGridClasses.DataGridListData;

    public class CellField extends TextInput
    {
        // Define the constructor and set properties.
        public function CellField() {
            super();
            height=60;
            width=80;
            setStyle("borderStyle", "none");
            editable=false;
        }

        // Override the set method for the data property.
        override public function set data(value:Object):void {
            super.data = value;

            if (value != null)
            {
                text = value[DataGridListData(listData).dataField];
                if(Number(text) > 100)
                {
                    setStyle("backgroundColor", 0xFF0000);
                }
            }

            super.invalidateDisplayList();
        }
    }
}
```

前の例では、アイテムレンダラーとして TextInput コントロールのサブクラスを作成しています。アイテムレンダラーまたはアイテムエディタとして使用するには、そのクラスはパブリックである必要があります。DataGrid セルの値が 100 より大きい場合、このアイテムレンダラーは背景を赤で表示します。

このアイテムレンダラーは DataGrid コントロールで使用できます。次に例を示します。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- itemRenderers\asRenderer\MainASItemRenderer.mxml -->

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="600" height="600">
```

```
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Monday: 12, Tuesday: 22, Wednesday: 452, Thursday: 90},
                {Monday: 258, Tuesday: 22, Wednesday: 45, Thursday: 46},
                {Monday: 4, Tuesday: 123, Wednesday: 50, Thursday: 95},
                {Monday: 12, Tuesday: 52, Wednesday: 111, Thursday: 20},
                {Monday: 22, Tuesday: 78, Wednesday: 4, Thursday: 51}
            ]);
        ]]>
    </mx:Script>

    <mx:Text text="All cells over 100 are red" />

    <mx:DataGrid id="myDataGrid"
        dataProvider="{initDG}"
        variableRowHeight="true">
        <mx:columns>
            <mx:DataGridColumn dataField="Monday"
                itemRenderer="myComponents.CellField" />
            <mx:DataGridColumn dataField="Tuesday"
                itemRenderer="myComponents.CellField" />
            <mx:DataGridColumn dataField="Wednesday"
                itemRenderer="myComponents.CellField" />
            <mx:DataGridColumn dataField="Thursday"
                itemRenderer="myComponents.CellField" />
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

アイテムレンダラーの操作

このセクションでは、Flex コントロールと共にアイテムレンダラーを使用する例を説明します。アイテムエディタの使用例については、823 ページ、第 22 章の「アイテムエディタの操作」を参照してください。

例 : アイテムレンダラーを TileList コントロールと HorizontalList コントロールと共に使用する

TileList コントロールと HorizontalList コントロールは、アイテムをタイル状に並べて表示します。TileList コントロールは、アイテムを垂直列に並べます。HorizontalList コントロールは、アイテムを水平行に並べます。TileList コントロールと HorizontalList コントロールは、カスタムアイテムレンダラーと組み合わせて、イメージなどのデータの一覧を表示する場合に特に便利です。

次の図は、製品カタログの表示に使用される HorizontalList コントロールを示しています。

HorizontalList コントロールのアイテムにはそれぞれ、イメージ、説明のテキストストリング、および価格が含まれています。次のコードは、カタログを表示するアプリケーションを示しています。HorizontalList コントロールのアイテムレンダラーは、Thumbnail という名前の MXML コンポーネントです。

```
<?xml version="1.0"?>
<!-- itemRenderers\htlistMainlistThumbnailRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Model id="catalog" source="catalog.xml"/>

    <mx:HorizontalList id="myList"
        columnWidth="125"
        rowHeight="125"
        columnCount="4"
        dataProvider="{catalog.product}"
        itemRenderer="myComponents.Thumbnail"/>

    <mx:LinkButton label="Product images courtesy of Lavish"
        click="navigateToURL(new URLRequest('http://www.shoplavish.com'),
'_blank')"/>
</mx:Application>
```

"catalog.xml" ファイルには HorizontalList コントロールのデータプロバイダが定義されています。

```
<?xml version="1.0"?>
<catalog>
    <product id="1">
        <name>USB Watch</name>
        <price>129.99</price>
        <image>assets/products/usbwatch.jpg</image>
        <thumbnail>assets/products/usbwatch_sm.jpg</thumbnail>
```

```
    </product>
    <product id="2">
        <name>007 Digital Camera</name>
        <price>99.99</price>
        <image>assets/products/007camera.jpg</image>
        <thumbnail>assets/products/007camera_sm.jpg</thumbnail>
    </product>
    <product id="3">
        <name>2-Way Radio Watch</name>
        <price>49.99</price>
        <image>assets/products/radiowatch.jpg</image>
        <thumbnail>assets/products/radiowatch_sm.jpg</thumbnail>
    </product>
    <product id="4">
        <name>USB Desk Fan</name>
        <price>19.99</price>
        <image>assets/products/usbfan.jpg</image>
        <thumbnail>assets/products/usbfan_sm.jpg</thumbnail>
    </product>
    <product id="5">
        <name>Caffeinated Soap</name>
        <price>19.99</price>
        <image>assets/products/soap.jpg</image>
        <thumbnail>assets/products/soap_sm.jpg</thumbnail>
    </product>
    <product id="6">
        <name>Desktop Rovers</name>
        <price>49.99</price>
        <image>assets/products/rover.jpg</image>
        <thumbnail>assets/products/rover_sm.jpg</thumbnail>
    </product>
</catalog>
```

次の例は Thumbnail.mxml という MXML コンポーネントを示しています。この例では、アイテムレンダラーを定義して、1 つの Image コントロールと 2 つの Label コントロールという計 3 つのコントロールを設定します。これらのコントロールで、アイテムレンダラーに渡された data プロパティを調べ、表示する内容を判断します。

```
<?xml version="1.0" ?>
<!-- itemRenderers\htlist\myComponents\Thumbnail.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="center"
    verticalGap="0" borderStyle="none" backgroundColor="white" >

    <mx:Image id="image" width="60" height="60" source="{data.image}"/>
    <mx:Label text="{data.name}" width="120" textAlign="center"/>
    <mx:Label text="${data.price}" fontWeight="bold"/>
</mx:VBox>
```

TileList コントロールと HorizontalList コントロールの詳細については、411 ページ、第 12 章の「データ駆動型コントロールの使用」を参照してください。

例 : アイテムレンダラーを DataGrid コントロールと共に使用する

DataGrid コントロールは DataGridColumn クラスを操作してグリッドを構成します。DataGrid コントロールでは、2 つのタイプのレンダラーを指定できます。1 つは各列のセル用で、もう 1 つは各列の最上部にあるヘッダセル用です。DataGrid コントロールの列にアイテムレンダラーを指定するには、DataGridColumn.itemRenderer プロパティを使用します。列のヘッダセルにアイテムレンダラーを指定するには、DataGridColumn.headerRenderer プロパティを使用します。

たとえば DataGrid コントロールの列を強調するために、列のヘッダセルに特価を示すアイコンを使用することができます。次にアイテムレンダラーの例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\myComponents\RendererDGHeader.mxml -->
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[

            [Embed(source="saleIcon.jpg")]
            [Bindable]
            public var sale:Class;
        ]]>
    </mx:Script>

    <mx:Label text="Sale Price!"/>
    <mx:Image height="20" source="{sale}"/>

</mx:HBox>
```

このアイテムレンダラーでは、Label コントロールを使用して “Sale Price!” というテキストを挿入し、Image コントロールを使用して列のヘッダにアイコンを挿入しています。

次の例はメインアプリケーションです。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\MainDGHeaderRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
          import mx.collections.ArrayCollection;

          [Bindable]
          private var initDG:ArrayCollection = new ArrayCollection([
            {Artist:'Pavement', Album:'Slanted and Enchanted',
                Price:11.99, SalePrice: true },
            {Artist:'Pavement', Album:'Brighten the Corners',
                Price:11.99, SalePrice: false }
          ]);
        ]]>
    </mx:Script>
```

```
    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        variableRowHeight="true">
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
            <mx:DataGridColumn dataField="Price"/>
            <mx:DataGridColumn width="150" dataField="SalePrice"
                headerRenderer="myComponents.RendererDGHeader"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、DataGrid コントロールで String の `true` または `false` を列に表示し、その価格が特価かどうかを示しています。またその列のアイテムレンダラーを定義して、テキストよりも視覚効果の高いアイコンを表示することもできます。DataGrid コントロールと共にアイテムレンダラーを使用する例については、782 ページの「カスタムアイテムレンダラーとカスタムアイテムエディタの使用」を参照してください。

例 : アイテムレンダラーを List コントロールと共に使用する

カスタムアイテムレンダラーを List コントロールと共に使用する場合は、次の例のように、`List.itemRenderer` プロパティを使用して指定します。

```
<?xml version="1.0"?>
<!-- itemRenderers\list\MainListStateRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="700" width="700">

    <mx:List id="myList"
        height="180" width="250"
        variableRowHeight="true"
        itemRenderer="myComponents.RendererState"
        backgroundColor="white" >
        <mx:dataProvider>
            <mx:Object label="Alaska"
                data="Juneau"
                webPage="http://www.state.ak.us/"/>
            <mx:Object label="Alabama"
                data="Montgomery"
                webPage="http://www.alabama.gov/" />
            <mx:Object label="Arkansas"
                data="Little Rock"
                webPage="http://www.state.ar.us/"/>
        </mx:dataProvider>
    </mx:List>
</mx:Application>
```

前の例では、アイテムレンダラーが List コントロールのデフォルトの行の高さを超える内容を表示するため、rowHeight プロパティを 75 ピクセルに設定しています。このアプリケーションのイメージについては、782 ページの「カスタムアイテムレンダラーとカスタムアイテムエディタの使用」を参照してください。

次の例に示す RendererState.mxml アイテムレンダラーは、各 List アイテムの部分を表示し、州の Web サイトを開くことができる LinkButton コントロールを作成します。

```
<?xml version="1.0"?>
<!-- itemRenderers\list\myComponents\RendererState.mxml -->
    <mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" >
        <mx:Script>
            <![CDATA[
                // Import Event and URLRequest classes.
                import flash.events.Event;
                import flash.net.URLRequest;

                private var u:URLRequest;

                // Event handler to open URL using
                // the navigateToURL() method.
                private function handleClick(eventObj:Event):void {
                    u = new URLRequest(data.webPage);
                    navigateToURL(u);
                }
            ]]>
        </mx:Script>

        <mx:HBox >
            <!-- Use Label controls to display state and capital names. -->
            <mx:Label id="State" text="State: {data.label}"/>
            <mx:Label id="Statecapital" text="Capital: {data.data}" />
        </mx:HBox>

        <!-- Define the Link control to open a URL. -->
        <mx:LinkButton id="webPage" label="Official {data.label} web page"
            click="handleClick(event);" color="blue"  />
    </mx:VBox>
```

例 : アイテムレンダラーを Tree コントロールと共に使用する

Tree コントロールでは、itemRenderer プロパティを使用して、ツリーのすべてのノードに対して 1 つのレンダラーを指定します。カスタムアイテムレンダラーを定義する場合は、テキストやアイコンなど、ノード全体の表示を処理する必要があります。

アイテムレンダラーを、デフォルトのアイテムレンダラークラスである TreeItemRenderer クラスのサブクラスとして定義する方法もあります。その場合は、アイテムレンダラー全体を実装せずに、アプリケーションの必要に合わせてアイテムレンダラーを修正できます。次に例を示します。

```
package myComponents
{
    // itemRenderers/tree/myComponents/MyTreeItemRenderer.as
    import mx.controls.treeClasses.*;
    import mx.collections.*;

    public class MyTreeItemRenderer extends TreeItemRenderer
    {

        // Define the constructor.
        public function MyTreeItemRenderer() {
            super();
        }

        // Override the set method for the data property
        // to set the font color and style of each node.
        override public function set data(value:Object):void {
            super.data = value;
            if(TreeListData(super.listData).hasChildren)
            {
                setStyle("color", 0xff0000);
                setStyle("fontWeight", 'bold');
            }
            else
            {
                setStyle("color", 0x000000);
                setStyle("fontWeight", 'normal');
            }
        }

        // Override the updateDisplayList() method
        // to set the text for each tree node.
        override protected function updateDisplayList(unscaledWidth:Number,
            unscaledHeight:Number):void {

            super.updateDisplayList(unscaledWidth, unscaledHeight);
            if(super.data)
            {
                if(TreeListData(super.listData).hasChildren)
                {
                    var tmp:XMLList =
                        new XMLList(TreeListData(super.listData).item);
                    var myStr:int = tmp[0].children().length();
                    super.label.text =  TreeListData(super.listData).label +
                        "(" + myStr + ")";
                }
            }
        }
    }
}
```

このアイテムレンダラーは、子ノードを持つノードのテキストを赤で表示し、子ノードの数を括弧で囲んで表示します。次の例では、このアイテムレンダラーをアプリケーションで使用しています。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- itemRenderers\tree\MainTreeItemRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="initCollections();">

    <mx:Script>
        <![CDATA[

            import mx.collections.*;

            public var xmlBalanced:XMLList =
                <>
                    <node label="Containers">
                        <node label="DividedBoxClasses">
                            <node label="BoxDivider" data="BoxDivider.as"/>
                        </node>
                        <node label="GridClasses">
                            <node label="GridRow" data="GridRow.as"/>
                            <node label="GridItem" data="GridItem.as"/>
                            <node label="Other File" data="Other.as"/>
                        </node>
                    </node>
                    <node label="Data">
                        <node label="Messages">
                            <node label="DataMessage"
                                data="DataMessage.as"/>
                            <node label="SequenceMessage"
                                data="SequenceMessage.as"/>
                        </node>
                        <node label="Events">
                            <node label="ConflictEvents"
                                data="ConflictEvent.as"/>
                            <node label="CommitFaultEvent"
                                data="CommitFaultEvent.as"/>
                        </node>
                    </node>
                </>;

            [Bindable]
            public var xlcBalanced:XMLListCollection;

            private function initCollections():void {
                xlcBalanced = new XMLListCollection(xmlBalanced);
            }
        ]]>
    </mx:Script>

    <mx:Text width="400"
        text="The nodes with children are in bold red text, with the number of
children in parenthesis.)"/>
```

```
    <mx:Tree id="compBalanced"
        width="400" height="500"
        dataProvider="{xlcBalanced}"
        labelField="@label"
        itemRenderer="myComponents.MyTreeItemRenderer"/>
</mx:Application>
```

第 22 章

アイテムエディタの操作

22

アイテムエディタを使用すると、List コントロールのセルの値を変更できます。DataGrid、List、および Tree の各コントロールでアイテムエディタを使用できます。このトピックでは、セルの編集プロセス、セルの編集に伴うイベント、およびアイテムエディタの作成方法と使用方法の例について説明します。

アイテムレンダラーとアイテムエディタの概要については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

目次

セルの編集プロセス

List コントロールのセルを編集するときは、次の一連の手順が発生します。

1. セルを編集しようとするユーザーが、目的のセルをクリックしてマウスボタンを離す操作や、Tab キーで目的のセルに移動する操作を実行します。
2. Adobe Flex から `itemEditBeginning` イベントが送出されます。このイベントを使用すると、特定のセル (複数指定可) の編集を無効にすることができます。詳細については、840 ページの「例 : セルを編集できないようにする」を参照してください。
3. Flex から `itemEditBegin` イベントが送出され、アイテムエディタが開きます。このイベントを使用すると、アイテムエディタに渡すデータを変更できます。詳細については、841 ページの「例 : アイテムエディタとの間で受け渡しするデータを変更する」を参照してください。
4. ユーザーがセルを編集します。
5. ユーザーが編集セッションを終了します。普通は、ユーザーがセルからフォーカスを移動したときに、そのセルの編集セッションが終了します。
6. Flex から `itemEditEnd` イベントが送出され、アイテムエディタが閉じて、List コントロールが新しいセルデータで更新されます。このイベントを使用すると、セルに返されるデータの変更、新しいデータの検証、アイテムエディタから返された形式とは異なる形式でデータを返す処理などが可能になります。詳細については、841 ページの「例 : アイテムエディタとの間で受け渡しするデータを変更する」を参照してください。
7. 新しいデータの値がセルに表示されます。

編集可能なセルの作成

DataGrid、List、および Tree の各コントロールには `editable` プロパティがあり、これを `true` に設定しておくとユーザーはそのコントロールの内容を編集できます。デフォルトでは `editable` プロパティの値は `false` で、そのセルは編集できないようになっています。DataGrid コントロールの場合は、`editable` プロパティを `true` に設定すると、そのグリッドのすべての列を編集できるようになります。`DatagridColumn.editable` プロパティを `false` に設定することで、任意の列を編集不可にできます。

List コントロールでは、デフォルトのアイテムエディタ (TextInput コントロール)、またはカスタムのアイテムエディタを使用できるほか、カスタムのアイテムレンダラーをエディタとして使用することもできます。アイテムレンダラーをアイテムエディタとして使用できるようにするかどうかは、以下の表に示すように、List コントロールの rendererIsEditor プロパティで指定します。

rendererIsEditor プロパティ	itemRenderer プロパティ	itemEditor プロパティ
false (デフォルト)	アイテムレンダラーを指定します。	アイテムエディタを指定します。セルを選択すると、itemEditor プロパティで定義されているアイテムエディタが開きます。itemEditor プロパティが定義されていない場合は、デフォルトのアイテムエディタ (TextInput コントロール) を使用します。
true	セルの内容を表示するアイテムレンダラーを指定します。このアイテムレンダラーをアイテムエディタとして使用できます。	無視されます。

この表が示すように、カスタムのアイテムレンダラーをアイテムエディタとして使用するか、またはカスタムのアイテムエディタを使用するかは、rendererIsEditor プロパティの状態で定義します。rendererIsEditor プロパティを true に設定すると、アイテムレンダラーがアイテムエディタとして使用され、itemEditor プロパティは無視されます。

例については、847 ページの「例 : アイテムレンダラーをアイテムエディタとして使用する」を参照してください。

アイテムエディタからデータを返す

デフォルトでは、アイテムエディタから List コントロールには単一の値が返されます。List コントロールの editorDataField プロパティを使用して、アイテムエディタの、新しいデータを持つプロパティを指定します。Flex により、この値が、目的のセルに適したデータ型に変換されます。

デフォルトのアイテムエディタは TextInput コントロールです。したがって、editorDataField プロパティのデフォルト値は "text" で、これは TextInput コントロールの text プロパティを指しています。カスタムのアイテムエディタを使用する場合は、次の例に示すように、editorDataField プロパティを、そのアイテムエディタの該当するプロパティに設定することも必要です。

```
<?xml version="1.0"?>
<!-- itemRenderers\dropin\DropInNumStepper.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
```

```
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection = new ArrayCollection([
                {label1:"Order #2314", quant:3, Sent:true},
                {label1:"Order #2315", quant:3, Sent:false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myDG" dataProvider="{myDP}"
        variableRowHeight="true"
        editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="label1"
                headerText="Order #" />
            <mx:DataGridColumn dataField="quant"
                itemEditor="mx.controls.NumericStepper"
                editorDataField="value" />
        </mx:columns >
    </mx:DataGrid>
</mx:Application>
```

前述の例では、アイテムエディタとして NumericStepper コントロールを使用するので、editorDataField プロパティを、新しいセルデータを収めた NumericStepper のプロパティ名である "value" に設定します。

アイテムエディタから複数の値を返す方法の詳細については、843 ページの「例 : アイテムエディタから複数の値を受け取る」を参照してください。

データを返すプロパティの定義

アイテムエディタには、複数のコンポーネントを指定できます。たとえば、次の例に示すアイテムエディタは、セルの編集に使用する CheckBox コントロールを子として持つ親コンテナ VBox を含んでいます。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\myComponents\EditorDGCheckBox.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
     backgroundColor="yellow">

    <mx:Script>
        <![CDATA[
            // Define a property for returning the new value to the cell.
            public var cbSelected:Boolean;
        ]]>
    </mx:Script>

    <mx:CheckBox id="followUpCB" label="Follow up needed"
        height="100%" width="100%"
```

```
        selected="{data.FollowUp}"
        click="cbSelected=followUpCB.selected"
        updateComplete="cbSelected=followUpCB.selected;"/>
</mx:VBox>
```

前述の例では、セルを選択すると、親の VBox コンテナで定義されているように、黄色の背景に CheckBox コントロールが表示されます。ここで、この CheckBox コントロールを選択または選択解除することで、目的のセルに新しい値が設定されます。

List コントロールに値を返すために、VBox コンテナでは cbSelected という新しいプロパティを定義します。このような処理が必要な理由は、ユーザー側で editorDataField プロパティに設定できる値が、このアイテムエディタが持つ最上位コンポーネントのプロパティ名に限られているからです。つまり、CheckBox コントロールが VBox コンテナの子である場合、editorDataField プロパティに、その CheckBox コントロールのプロパティ名 "selected" を設定することはできません。

ユーザーがセルを選択して CheckBox コントロールを開いても、その CheckBox コントロールの状態を変更していない場合は、updateComplete イベントを使用して cbSelected プロパティをの値を設定します。

次の例では、顧客連絡先担当者のリストを表示する DataGrid コントロールで、このアイテムエディタを使用します。このリストを構成するデータは、会社名、連絡先担当者名、電話番号、およびこの連絡先担当者にフォローアップが必要かどうかを示す列です。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\MainDGCheckBoxEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Company: 'Acme', Contact: 'Bob Jones',
                    Phone: '413-555-1212', Date: '5/5/05' , FollowUp: true },
                {Company: 'Allied', Contact: 'Jane Smith',
                    Phone: '617-555-3434', Date: '5/6/05' , FollowUp: false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="Company" editable="false"/>
            <mx:DataGridColumn dataField="Contact"/>
            <mx:DataGridColumn dataField="Phone"/>
            <mx:DataGridColumn dataField="Date"/>
            <mx:DataGridColumn dataField="FollowUp"
                width="150"
```

```
                headerText="Follow Up?"
                itemEditor="myComponents.EditorDGCheckBox"
                editorDataField="cbSelected"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

editorDataField プロパティは、VBox コンテナの新しいプロパティである cbSelected に設定されています。

このメカニズムは、複数のコントロールを持つインラインアイテムレンダラーでも同様に使用できます。たとえば、以下のコードに示すように、前述の DataGrid の例を、アイテムエディタコンポーネントではなく、インラインアイテムエディタを使用するように変更できます。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineDGCheckBoxEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Company: 'Acme', Contact: 'Bob Jones',
                    Phone: '413-555-1212', Date: '5/5/05' , FollowUp: true },
                {Company: 'Allied', Contact: 'Jane Smith',
                    Phone: '617-555-3434', Date: '5/6/05' , FollowUp: false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        editable="true">
        <mx:columns>
            <mx:DataGridColumn dataField="Company" editable="false"/>
            <mx:DataGridColumn dataField="Contact"/>
            <mx:DataGridColumn dataField="Phone"/>
            <mx:DataGridColumn dataField="Date"/>
            <mx:DataGridColumn dataField="FollowUp"
                width="150"
                headerText="Follow Up?"
                editorDataField="cbSelected">

                <mx:itemEditor>
                    <mx:Component>
                        <mx:VBox backgroundColor="yellow">
                            <mx:Script>
                                <![CDATA[
                                    // Define a property for returning
                                    // the new value to the cell.
                                    [Bindable]
```

```
                                public var cbSelected:Boolean;
                            ]]>
                        </mx:Script>

                        <mx:CheckBox id="followUpCB"
                            label="Follow up needed"
                            height="100%" width="100%"
                            selected="{data.FollowUp}"
                            click="cbSelected=followUpCB.selected"/>
                    </mx:VBox>
                </mx:Component>
            </mx:itemEditor>
        </mx:DataGridColumn>
    </mx:columns>
</mx:DataGrid>
</mx:Application>
```

アイテムエディタのサイズと位置の決定

アイテムエディタが表示されると、それは List コントロールの選択したセルの上にポップアップコントロールとして表示されます。List コントロールによって、セルの現在の値も隠されます。

826 ページの「データを返すプロパティの定義」にある前述の例の CheckBox コントロールでは、`width` および `height` の両方を 100% としてサイズが決定されています。また、VBox コンテナでは、その `backgroundColor` スタイルプロパティが黄色に設定されています。アイテムエディタをセルと同じサイズに設定し、コンテナに背景色を設定することで、下に表示されているセルを完全に覆い隠すことができます。

次の表は、アイテムエディタのサイズを設定するために使用できる List コントロールのプロパティを示しています。

プロパティ	説明
`editorHeightOffset`	DataGridColumn コントロールのセルまたは Tree コントロールのテキストフィールドのサイズを基準にして、アイテムエディタの高さをピクセル数で指定します。
`editorWidthOffset`	DataGridColumn コントロールのセルまたは Tree コントロールのテキストフィールドのサイズを基準にして、アイテムエディタの幅をピクセル数で指定します。
`editorXOffset`	DataGridColumn コントロールのセルの左上隅を基準として、または Tree コントロールのテキストフィールドの左上隅からの距離として、アイテムエディタの左上隅の x 位置をピクセル数で指定します。
`editorYOffset`	DataGridColumn コントロールのセルの左上隅を基準として、または Tree コントロールのテキストフィールドの左上隅からの距離として、アイテムエディタの左上隅の y 位置をピクセル数で指定します。

次に示すコードでは、前のセクションにある DataGridColumn コントロールの定義を変更し、editorXOffset プロパティと editorYOffset プロパティを使用して、アイテムエディタを下方および右方に 15 ピクセルずつ移動し、DataGrid コントロールの中でより目立つ外観となるようにします。

```
<?xml version="1.0"?>
<!-- itemRenderers\inline\InlineDGCheckBoxEditorWithOffsets.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Company: 'Acme', Contact: 'Bob Jones',
                    Phone: '413-555-1212', Date: '5/5/05' , FollowUp: true },
                {Company: 'Allied', Contact: 'Jane Smith',
                    Phone: '617-555-3434', Date: '5/6/05' , FollowUp: false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        editable="true" >
        <mx:columns>
            <mx:DataGridColumn dataField="Company" editable="false"/>
            <mx:DataGridColumn dataField="Contact"/>
            <mx:DataGridColumn dataField="Phone"/>
            <mx:DataGridColumn dataField="Date"/>
            <mx:DataGridColumn dataField="FollowUp"
                width="150"
                headerText="Follow Up?"
                editorDataField="cbSelected"
                editorXOffset="15"
                editorYOffset="15">

                <mx:itemEditor>
                    <mx:Component>
                        <mx:VBox backgroundColor="yellow">
                            <mx:Script>
                                <![CDATA[
                                    // Define a property for returning
                                    // the new value to the cell.
                                    [Bindable]
                                    public var cbSelected:Boolean;
                                ]]>
                            </mx:Script>

                            <mx:CheckBox id="followUpCB"
                                label="Follow up needed"
                                height="100%" width="100%"
```

```
                            selected="{data.FollowUp}"
                            click="cbSelected=followUpCB.selected"/>
                    </mx:VBox>
                </mx:Component>
            </mx:itemEditor>
        </mx:DataGridColumn>
    </mx:columns>
</mx:DataGrid>
</mx:Application>
```

Enter キーに応答するアイテムエディタの作成

List コントロールでデフォルトのアイテムエディタを使用している場合は、セルの値を編集した後で Enter キーを押すと、そのコントロールの中でフォーカスを次のセルに移動できます。コンポーネント 1 つのみで構成したシンプルなアイテムエディタを作成し、そのコンポーネントに IFocusable インターフェイスを実装すると、そのアイテムエディタも Enter キーによる操作に応答します。Accordion、Button、ButtonBar、ComboBase、DateChooser、DateField、ListBase、MenuBar、NumericStepper、TabNavigator、TextArea、および TextInput の各コンポーネントに IFocusable インターフェイスが実装されています。

次の例では、VBox コンテナを最上位のコンポーネントとして、その子コンポーネントに CheckBox コントロールを持つ、複雑なアイテムレンダラーを定義します。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\myComponents\EditorDGCheckBox.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
     backgroundColor="yellow">

    <mx:Script>
        <![CDATA[
            // Define a property for returning the new value to the cell.
            public var cbSelected:Boolean;
        ]]>
    </mx:Script>

    <mx:CheckBox id="followUpCB" label="Follow up needed"
        height="100%" width="100%"
        selected="{data.FollowUp}"
        click="cbSelected=followUpCB.selected"
        updateComplete="cbSelected=followUpCB.selected;"/>
</mx:VBox>
```

このアイテムエディタを開くと、CheckBox コントロールでフォーカスを取得してその値を編集できますが、親の VBox コンテナは IFocusManagerComponent インターフェイスを実装していないので、Enter キーを押してもフォーカスは次のセルに移動しません。次の例に示すように、IFocusManagerComponent インターフェイスを実装するようにこの例を変更できます。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\myComponents\EditorDGCheckBoxFocusable.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
     backgroundColor="yellow"
     implements="mx.managers.IFocusManagerComponent" >

    <mx:Script>
        <![CDATA[
            // Define a property for returning the new value to the cell.
            public var cbSelected:Boolean;

            // Implement the drawFocus() method for the VBox.
            override public function drawFocus(isFocused:Boolean):void {
                // This method can be empty, or you can use it
                // to make a visual change to the component.
            }
        ]]>
    </mx:Script>

    <mx:CheckBox id="followUpCB"
        label="Follow up needed"
        height="100%" width="100%"
        selected="{data.FollowUp}"
        click="cbSelected=followUpCB.selected"
        updateComplete="cbSelected=followUpCB.selected;"/>
</mx:VBox>
```

IFocusManagerComponent インターフェイスにはプロパティとメソッドがいくつか定義されていますが、UIComponent クラスは、これらすべての実装を定義または継承します。ただし、drawFocus() メソッドだけは例外です。したがって、Enter キーの操作に応答するアイテムエディタとするには、例外であるこのメソッドを実装するだけですみます。

セルの編集イベントの使用

リストコンポーネントでは、セルの編集プロセスの過程で、itemEditBeginning、itemEditBegin、および itemEditEnd の 3 つのイベントが送出されます。List コントロールでは、これら 3 つのイベントすべてについてデフォルトのイベントリスナーが定義されます。

これらのイベントの任意のものについて独自のイベントリスナーを記述し、編集プロセスをカスタマイズできます。独自のイベントリスナーを記述する場合は、目的のコンポーネントで定義しているデフォルトのイベントリスナーの前にその独自のイベントリスナーが実行され、その後でデフォルトのイベントリスナーが実行されるようにします。たとえば、アイテムエディタに渡すデータや、アイテムエディタから返されるデータを変更できます。このデータは、itemEditBegin イベントのイベントリスナーで変更できます。このイベントリスナーの実行が終了すると、デフォルトのイベントリスナーが実行されて、編集プロセスが継続します。

なお、コンポーネントのデフォルトのイベントリスナーを独自のイベントリスナーに置き換えることができます。デフォルトのイベントリスナーが実行されないようにするには、独自のイベントリスナーの任意の場所から preventDefault() メソッドを呼び出します。

アイテムエディタを作成するときは、次の各種イベントを使用します。

- itemEditBeginning

  セルをクリックしてマウスボタンを離す操作や、Tab キーでセルに移動する操作など、セルを編集しようとする何らかの操作が発生したときに送出されます。

  List コントロールには、itemEditBeginning イベントに対するデフォルトのイベントリスナーがあります。このイベントリスナーは、List コントロールの editedItemPosition プロパティを、フォーカスのあるセルに設定します。

  特定のセル (複数指定可) を編集できないようにするには、このイベントについて独自のイベントリスナーを記述することが普通です。セルを編集できないようにするには、独自のイベントリスナーから preventDefault() メソッドを呼び出します。これにより、デフォルトのイベントリスナーが実行されなくなり、セルは編集不可になります。詳細については、840 ページの「例 : セルを編集できないようにする」を参照してください。

- itemEditBegin

  アイテムエディタが開く前に送出されます。

  リストコンポーネントには、itemEditBegin イベントに対するデフォルトのイベントリスナーがあります。このイベントリスナーは、createItemEditor() メソッドを呼び出して次のアクションを実行します。

  - アイテムエディタオブジェクトを作成し、data プロパティをセルからエディタにコピーします。デフォルトでは、アイテムエディタオブジェクトは、TextInput コントロールのインスタンスです。List コントロールの itemEditor プロパティを使用して、独自のアイテムエディタクラスを指定します。
  - アイテムエディタオブジェクトを参照するように、List コントロールの itemEditorInstance プロパティを設定します。

このイベントのイベントリスナーを記述して、アイテムエディタに渡されるデータを変更できます。たとえば、データ、データのフォーマットなど、アイテムエディタで使用する情報を変更できます。詳細については、841 ページの「例 : アイテムエディタとの間で受け渡しするデータを変更する」を参照してください。

また、セルの編集に使用するアイテムエディタを指定するためのイベントリスナーを作成することもできます。たとえば、2 つの異なるアイテムエディタがあるとします。イベントリスナーの中で、編集対象のデータを調べ、itemEditor プロパティを適切なアイテムエディタに設定することでそのアイテムエディタを開いた後、createItemEditor() メソッドを呼び出すことができます。この場合は、まず preventDefault() を呼び出し、デフォルトのイベントリスナーが実行される過程で createItemEditor() が呼び出されないようにします。

あとは、itemEditBegin イベントのイベントリスナーから createItemEditor() メソッドを呼び出すだけです。これ以外の時点でエディタを作成するには、itemEditBegin イベントを生成するように editedItemPosition プロパティを設定します。

- itemEditEnd

  セルの編集セッションが終了すると送出されます。これは、目的のセルからフォーカスが他に移動した時点であることが普通です。

  リストコンポーネントには、このイベントに対するデフォルトのイベントリスナーがあります。このイベントリスナーは、アイテムエディタから List コントロールのデータプロバイダにデータをコピーします。デフォルトのイベントリスナーは、次のアクションを実行します。

  - List コントロールの editorDataField プロパティを使用して、新しいデータを含むアイテムエディタのプロパティを指定します。デフォルトのアイテムエディタは TextInput コントロールなので、editorDataField プロパティのデフォルト値は "text" です。これは、TextInput コントロールの text プロパティにセルの新しいデータが収められていることを示します。
  - 編集セッションを終了する理由によっては、デフォルトのイベントリスナーが destroyItemEditor() メソッドを呼び出し、アイテムエディタを閉じます。詳細については、837 ページの「itemEditEnd イベントの発生理由の判断」を参照してください。

  通常では、このイベントのイベントリスナーを記述して、次のアクションを実行します。

  - アイテムエディタから返されたデータを変更します。

    独自のイベントリスナーでは、エディタから List コントロールに返されたデータを変更できます。たとえば、List コントロールにデータを返す前に、データを再フォーマットできます。デフォルトでは、アイテムエディタは単一の値のみ返すことができます。複数の値を返す場合は、itemEditEnd イベントのイベントリスナーを記述する必要があります。
  - アイテムエディタに入力された値を確認します。

    イベントリスナーでは、アイテムエディタに入力されたデータを確認できます。データが正しくない場合は、preventDefault() メソッドを呼び出して、新しいデータが List コントロールに渡されないようにした上で、エディタを開いたままにしておくことができます。

セルの編集イベントクラス

次の表に示すように、編集可能な List コントロールにはそれぞれ、セルの編集イベントのイベントオブジェクトを定義するクラスが対応付けられています。

List クラス	Event クラス
DataGrid	DataGridEvent
List	ListEvent
Tree	ListEvent

List コントロールと Tree コントロールのイベントクラスは ListEvent です。

List コントロールのイベントリスナーを定義する際は、そのイベントリスナーに渡すイベントオブジェクトのタイプを必ず正しく指定するようにします。DataGrid コントロールについての例を次に示します。

```
public function myCellEndEvent(event:DataGridEvent):void {
   // イベントリスナーを定義する。
}
```

イベントリスナーでのセルデータとアイテムエディタへのアクセス

イベントリスナーから、編集中のセルに現在設定されている値、ユーザーが入力した新しい値、またはそのセルの編集に使用されているアイテムエディタにアクセスできます。

セルに現在設定されている値にアクセスするには、List コントロールの editedItemRenderer プロパティを使用します。editedItemRenderer プロパティには、編集中のセルに対応したデータが収められています。List コントロールおよび Tree コントロールの場合、このプロパティに収められているのは、セルのデータプロバイダエレメントです。また、DataGrid コントロールの場合は、DataGrid の行全体のデータプロバイダエレメントです。

セルの新しい値またはアイテムエディタにアクセスするには、List コントロールの itemEditorInstance プロパティを使用します。itemEditorInstance プロパティは、cellBeginEvent リスナーのイベントリスナーを実行した後でない限り、初期化されません。したがって、itemEditorInstance プロパティには、itemEditEnd イベントのイベントリスナーからのみアクセスすることが普通です。

次の例は、これらのプロパティを使用する itemEditEnd イベントのイベントリスナーを示しています。

```
<?xml version="1.0"?>
<!-- itemRenderers\events\EndEditEventAccessEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
```

```
            import mx.controls.TextInput;
            import mx.events.DataGridEvent;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99 }
            ]);

            // Define event listener for the itemEditEnd event.
            private function getCellInfo(event:DataGridEvent):void {

                // Get the cell editor and cast it to TextInput.
                var myEditor:TextInput =
                    TextInput(event.currentTarget.itemEditorInstance);

                // Get the new value from the editor.
                var newVal:String = myEditor.text;

                // Get the old value.
                var oldVal:String =
            event.currentTarget.editedItemRenderer.data[event.dataField];

                // Write out the cell coordinates, new value,
                // and old value to the TextArea control.
                cellInfo.text = "cell edited.\n";
                cellInfo.text += "Row, column: " + event.rowIndex + ", " +
                    event.columnIndex + "\n";
                cellInfo.text += "New value: " + newVal + "\n";
                cellInfo.text += "Old value: " + oldVal;
            }
        ]]>
    </mx:Script>

    <mx:TextArea id="cellInfo" width="300" height="150" />

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        editable="true"
        itemEditEnd="getCellInfo(event);" >
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
            <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、アイテムエディタを取得して、それを適切なエディタクラスにキャストします。デフォルトのアイテムエディタは TextInput コントロールなので、この場合のキャスト先は TextInput です。カスタムのアイテムエディタを定義済みであれば、その該当のクラスにキャストします。アイテムエディタへの参照を設定すれば、そのプロパティにアクセスしてセルの新しい値を入手できます。

セルの古い値にアクセスするには、DataGrid コントロールの editedItemRenderer プロパティを使用します。さらに、イベントオブジェクトの dataField プロパティを使用して、編集された列の data プロパティにアクセスします。

itemEditEnd イベントの発生理由の判断

ユーザーがセルの編集セッションを終了するにはいくつかの方法があります。itemEditEnd イベントのイベントリスナーの本体で、このイベントの発生理由を判断し、それに従ってイベントを処理できます。

List コントロールの各イベントクラスでは、イベントの発生理由を示す値を収めた reason プロパティが定義されています。reason プロパティに設定される値は次のとおりです。

値	内容
CANCELLED	ユーザーが編集作業をキャンセルし、編集したデータの保存も必要としていないことを示します。itemEditEnd イベントに設定した独自のイベントリスナーから preventDefault() メソッドを呼び出しても、エディタを閉じる処理では destroyItemEditor() メソッドが呼び出されます。
NEW_COLUMN	(DataGrid のみ) 今までの列と同じ行にある別の列に、ユーザーがフォーカスを移動したことを示します。イベントリスナーでは、フォーカスの移動を許可することもできれば、許可しないこともできます。 たとえば、現在編集中のセルにユーザーが入力した値が有効なものかどうかをイベントリスナーで確認します。無効な値が入力されている場合は、preventDefault() メソッドを呼び出すことにより、新しいセルへのフォーカス移動を禁止できます。この場合、アイテムエディタは開いたままなので、ユーザーは引き続き現在のセルを編集できます。preventDefault() メソッドに加え、destroyItemEditor() メソッドも呼び出すと、新しいセルに移動できないようにした上で、アイテムエディタを閉じることができます。
NEW_ROW	ユーザーがフォーカスを新しい行に移動したことを示します。reason プロパティでは、NEW_COLUMN の場合と同じようにこの値を扱うことができます。
OTHER	List コントロールがフォーカスを失った場合やスクロールされた場合など、何らかの理由で編集できない状態になったことを示します。itemEditEnd イベントに設定した独自のイベントリスナーから preventDefault() メソッドを呼び出しても、エディタを閉じる処理では destroyItemEditor() メソッドが呼び出されます。

次の例では、itemEditEnd イベントを使用して、ユーザーがセルに空のストリングを入力していないことを確認します。空のストリングが入力されている場合は、itemEditEnd イベントで preventDefault() メソッドを呼び出し、有効な値が入力されない限り、現在のセルからフォーカスを移動できないようにします。ただし、itemEditEnd イベントの reason プロパティに設定されている値が CANCELLED である場合、イベントリスナーは何も実行しません。

```
<?xml version="1.0"?>
<!-- itemRenderers\events\EndEditEventFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.controls.TextInput;
            import mx.events.DataGridEvent;
            import mx.events.DataGridEventReason;
            import mx.formatters.NumberFormatter;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99 }
            ]);

            private var myFormatter:NumberFormatter=new NumberFormatter();

           public function formatData(event:DataGridEvent):void {
                // Check the reason for the event.
                if (event.reason == DataGridEventReason.CANCELLED)
                {
                    // Do not update cell.
                    return;
                }

                // Get the new data value from the editor.
                var newData:String=
                    TextInput(event.currentTarget.itemEditorInstance).text;

                if(newData == "")
                {
                    // Prevent the user from removing focus,
                    // and leave the cell editor open.
                    event.preventDefault();
                    // Write a message to the errorString property.
                    // This message appears when the user
                    // mouses over the editor.
                    TextInput(myGrid.itemEditorInstance).errorString=
                        "Enter a valid string.";
                }
```

```
            }

        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        editable="true"
        itemEditEnd="formatData(event);">
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
            <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

この例では、ユーザーがイベントを発生させた理由が CANCELLED であると、イベントリスナーは何も実行しません。

この理由が NEW_COLUMN、NEW_ROW、または OTHER であると、イベントリスナーは次のアクションを実行します。

1. セルの新しい値が空のストリングではないことを確認します。
2. 空のストリングが入力されている場合、イベントリスナーは preventDefault() メソッドを呼び出し、アイテムエディタが閉じないようにした上で、セルが空のストリングで更新されないようにします。
3. TextInput コントロールの errorString プロパティにメッセージを書き込みます。これにより、TextInput コントロールの周囲に赤いボックスが表示され、ユーザーがセルの上にマウスを置くと、このメッセージがツールヒントとして表示されます。

アイテムエディタの例

このセクションでは、次の機能を持つアイテムエディタの例を紹介します。

- 840 ページの「例 : セルを編集できないようにする」
- 841 ページの「例 : アイテムエディタとの間で受け渡しするデータを変更する」
- 843 ページの「例 : アイテムエディタから複数の値を受け取る」
- 847 ページの「例 : アイテムレンダラーをアイテムエディタとして使用する」
- 849 ページの「例 : カスタムアイテムエディタでデータバリデータを使用する」

例 : セルを編集できないようにする

itemEditBeginning イベントのイベントリスナーから、編集中のセルを調べ、編集できないようにすることができます。特定のセル (複数可) のみを編集禁止とし、それ以外のセルは編集できるようにする場合に、この方法が便利です。

たとえば、DataGrid コントロールで editable プロパティを使用して、DataGrid コントロールにあるセルをすべて編集可能にします。DataGridColumn の editable プロパティを使用することで、特定の列についてこの設定をオーバーライドすることはできますが、特定のセルについて編集を有効または無効にすることはできません。

セルを編集できないようにするには、次の例に示すように、itemEditBeginning イベントに設定した独自のイベントリスナーから preventDefault() メソッドを呼び出します。

```
<?xml version="1.0"?>
<!-- itemRenderers\events\EndEditEventPreventEdit.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.events.DataGridEvent;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99}
            ]);

            // Define event listener for the cellEdit event
            // to prohibit editing of the Album column.
            private function disableEditing(event:DataGridEvent):void {
                if(event.columnIndex==1)
                {
                    event.preventDefault();
                }
            }

        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        editable="true"
        itemEditBeginning="disableEditing(event);" >
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
```

```
            <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

前の例では列のインデックスを使用していましたが、DataGrid または編集しているセルの任意のプロパティを調べて、そのセルを編集可能にするかどうかを決めることができます。

例 : アイテムエディタとの間で受け渡しするデータを変更する

itemEditBegin イベントおよび itemEditEnd イベントを使用して、アイテムエディタとの間で受け渡しするデータを調べ、必要に応じてそのデータを変更できます。たとえば、データの再フォーマット、編集対象となるデータの部分的抽出、検証のためのデータの調査などが可能です。

Flex のいずれかのフォーマッタクラスを使用して、アイテムエディタから返されるデータをフォーマットできます。次の例では、DataGrid コントロールの Price 列をユーザーが編集できるようにします。次に、itemEditEnd イベントのイベントリスナーを宣言します。このイベントリスナーでは、次のコードに示すように、NumberFormatter クラスを使用して、セルの新しい値の小数点以下桁数を 2 桁に制限します。

```
<?xml version="1.0"?>
<!-- itemRenderers\events\EndEditEventFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[

            import mx.controls.TextInput;
            import mx.events.DataGridEvent;
            import mx.events.DataGridEventReason;
            import mx.formatters.NumberFormatter;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {Artist:'Pavement', Album:'Slanted and Enchanted',
                    Price:11.99},
                {Artist:'Pavement', Album:'Brighten the Corners',
                    Price:11.99 }
            ]);

            // Define the number formatter.
            private var myFormatter:NumberFormatter=new NumberFormatter();

            // Define the eventlistner for the itemEditEnd event.
            public function formatData(event:DataGridEvent):void {
                // Check the reason for the event.
                if (event.reason == DataGridEventReason.CANCELLED)
```

```
                {
                    // Do not update cell.
                    return;
                }

                // Get the new data value from the editor.
                var newData:String=
                    TextInput(event.currentTarget.itemEditorInstance).text;

                // Determine if the new value is an empty String.
                if(newData == "") {
                    // Prevent the user from removing focus,
                    // and leave the cell editor open.
                    event.preventDefault();
                    // Write a message to the errorString property.
                    // This message appears when the user
                    // mouses over the editor.
                    TextInput(myGrid.itemEditorInstance).errorString=
                        "Enter a valid string.";
                    return;
                }

                // For the Price column, return a value
                // with a precision of 2.
                if(event.dataField == "Price") {
                    myFormatter.precision=2;
                    TextInput(myGrid.itemEditorInstance).text=
                        myFormatter.format(newData);
                }
            }
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        dataProvider="{initDG}"
        editable="true"
        itemEditEnd="formatData(event);" >
        <mx:columns>
            <mx:DataGridColumn dataField="Artist"/>
            <mx:DataGridColumn dataField="Album"/>
            <mx:DataGridColumn dataField="Price"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

例 : アイテムエディタから複数の値を受け取る

セル編集のメカニズムは、単一の値を List コントロールに返すアイテムエディタでの使用に最適化されています。この場合は、List コントロールの定義で editorDataField プロパティを使用して、セルの新しい値を保持するアイテムエディタのプロパティを指定します。

一方、単一の値ではないフォーマットでデータを返すアイテムエディタを作成することもあります。複数の値を返す方法として、複数のスカラー値を返す方法と、複数値を収めたオブジェクトを返す方法があります。

単一ではない値を返すには、cellEndEvent について、データをアイテムエディタから取得して List コントロールの editedItemRenderer プロパティに直接書き込むイベントリスナーを記述します。editedItemRenderer プロパティにデータを書き込むことによって、List コントロールの対応するデータプロバイダも新しい値で更新されます。

次の例では、コンポーネントとして実装するアイテムエディタを示しています。このコンポーネントでは、TextInput コントロールおよび ComboBox コントロールを使用して、住所の都市名と州名の部分をユーザーが設定できるようにします。

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="yellow">
<!-- itemRenderers\events\myComponents\CityStateEditor.mxml -->

    <mx:TextInput id="setCity"  width="130" text="{data.City}"/>

    <mx:ComboBox id="pickState" selectedItem="{data.State}">
        <mx:dataProvider>
            <mx:String>AL</mx:String>
            <mx:String>AK</mx:String>
            <mx:String>AR</mx:String>
            <mx:String>CA</mx:String>
            <mx:String>MA</mx:String>
        </mx:dataProvider>
    </mx:ComboBox>
</mx:VBox>
```

cellEndEvent イベントのイベントリスナーで、List コントロールの itemEditorInstance プロパティを使用して、TextInput コントロールおよび ComboBox コントロールにある新しい値にアクセスします。次に、editedItemRenderer プロパティにこの新しい値を直接書き込んで List コントロールを更新できます。次にこの例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\events\ComplexDGEditorReturnObject.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700">

    <mx:Script>
        <![CDATA[

            import mx.events.DataGridEvent;
```

```
            import mx.events.DataGridEventReason;
            import mx.collections.ArrayCollection;
            import myComponents.CityStateEditor;

            [Bindable]
            public var initDG:ArrayCollection = new ArrayCollection([
                {Company: 'Acme', Contact: 'Bob Jones',
                    Phone: '413-555-1212', City: 'Boston', State: 'MA'},
                {Company: 'Allied', Contact: 'Jane Smith',
                    Phone: '617-555-3434', City: 'SanFrancisco', State: 'CA'}
             ]);

            // Define the event listener.
            public function processData(event:DataGridEvent):void {
                // Check the reason for the event.
                if (event.reason == DataGridEventReason.CANCELLED){
                    // Do not update cell.
                    return;
                }

                if(event.dataField == "City and State")
                {
                    // Disable copying data back to the control.
                    event.preventDefault();

                    // Get new city from editor.
                    myGrid.editedItemRenderer.data.City =
CityStateEditor(DataGrid(event.target).itemEditorInstance).setCity.text;

                    // Get new state from editor.
                    myGrid.editedItemRenderer.data.State =
CityStateEditor(DataGrid(event.target).itemEditorInstance).pickState.selectedIte
m;

                    // Close the cell editor.
                    myGrid.destroyItemEditor();

                    // Notify the list control to update its display.
                    myGrid.dataProvider.itemUpdated(event.itemRenderer.data);
                }
            }
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        rowHeight="75"
        dataProvider="{initDG}"
        editable="true"
        itemEditEnd="processData(event);">
        <mx:columns>
            <mx:DataGridColumn dataField="Company" editable="false"/>
            <mx:DataGridColumn dataField="Contact"/>
            <mx:DataGridColumn dataField="Phone"/>
```

```
            <mx:DataGridColumn dataField="City and State" width="150"
                    itemEditor="myComponents.CityStateEditor">
                <mx:itemRenderer>
                    <mx:Component>
                        <mx:Text selectable="false" width="100%"
                            text="{data.City}, {data.State}"/>
                    </mx:Component>
                </mx:itemRenderer>
            </mx:DataGridColumn>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

前の例では、インラインアイテムレンダラーを使用して、DataGrid コントロールの単一の列に都市名と州名を表示していました。

itemEditEnd イベントのイベントリスナーでは、編集中の列が都市名と州名の列かどうかを判断します。都市名と州名の列であれば、このイベントリスナーは次のアクションを実行します。

1. preventDefault() メソッドを呼び出して、ストリング値が返されないようにします。
2. List コントロールの itemEditorInstance プロパティを使用して、アイテムエディタから都市と州の新しい値を入手します。
3. destroyItemEditor() メソッドを呼び出して、アイテムエディタを閉じます。
4. itemUpdated() メソッドを呼び出して、List コントロールに渡した新しいデータでその表示が更新されるようにします。このメソッドを呼び出さないと、List コントロールの表示が別の機会に更新されるまで、新しいデータはコントロールに表示されません。

List コントロールおよび Tree コントロールの使用例については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

次の例では上記と同じアクションを実行しますが、コンポーネントアイテムエディタではなく、インラインアイテムエディタを使用しています。

```
<?xml version="1.0"?>
<!-- itemRenderers\events\InlineDGEditorReturnObject.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700">

    <mx:Script>
        <![CDATA[

            import mx.events.DataGridEvent;
            import mx.collections.ArrayCollection;
            import mx.controls.TextInput;

            public var newCity:String;
            public var newState:String;

            [Bindable]
            public var initDG:ArrayCollection = new ArrayCollection([
```

```
                {Company: 'Acme', Contact: 'Bob Jones',
                    Phone: '413-555-1212', City: 'Boston', State: 'MA'},
                {Company: 'Allied', Contact: 'Jane Smith',
                    Phone: '617-555-3434', City: 'SanFrancisco', State: 'CA'}
            ]);

            public function processData(event:DataGridEvent):void {
                if(event.dataField=='City/State')
                {
                    // Disable copying data back to the control.
                    event.preventDefault();

                    myGrid.editedItemRenderer.data.City=
                        myEditor(myGrid.itemEditorInstance).setCity.text;
                    myGrid.editedItemRenderer.data.State=

myEditor(myGrid.itemEditorInstance).pickState.selectedItem;

                    myGrid.destroyItemEditor();

                    // Notify the list control to update its display.
                    myGrid.dataProvider.itemUpdated(myGrid.editedItemRenderer);
                }
            }
        ]]>
    </mx:Script>

    <mx:DataGrid id="myGrid"
        rowHeight="75"
        dataProvider="{initDG}"
        editable="true"
        itemEditEnd="processData(event);">
        <mx:columns>
            <mx:DataGridColumn dataField="Company" editable="false"/>
            <mx:DataGridColumn dataField="Contact"/>
            <mx:DataGridColumn dataField="Phone"/>
            <mx:DataGridColumn dataField="City/State" width="150">
                <mx:itemRenderer>
                    <mx:Component>
                        <mx:Text selectable="false" width="100%"
                            text="{data.City}, {data.State}"/>
                    </mx:Component>
                </mx:itemRenderer>

                <mx:itemEditor>
                    <mx:Component className="myEditor">
                        <mx:VBox backgroundColor="yellow">

                            <mx:TextInput id="setCity" width="130"
                                text="{data.City}"/>

                            <mx:ComboBox id="pickState"
                                selectedItem="{data.State}">
```

```
                                    <mx:dataProvider>
                                        <mx:String>AL</mx:String>
                                        <mx:String>AK</mx:String>
                                        <mx:String>AR</mx:String>
                                        <mx:String>CA</mx:String>
                                        <mx:String>MA</mx:String>
                                    </mx:dataProvider>
                                </mx:ComboBox>
                            </mx:VBox>
                        </mx:Component>
                    </mx:itemEditor>
                </mx:DataGridColumn>
            </mx:columns>
        </mx:DataGrid>
</mx:Application>
```

<mx:Component> タグには、値 "myEditor" が設定された className プロパティがあります。この className プロパティは、インラインアイテムエディタを表すために Flex で作成されたクラスの名前を定義しています。これは前の例で、コンポーネントアイテムエディタのクラス名を、MXML ファイルの名前で定義していたことと同じです。itemEditEnd イベントのイベントリスナーで、myEditor クラスのプロパティとしての TextInput コントロールおよび ComboBox コントロールにアクセスできます。

例 : アイテムレンダラーをアイテムエディタとして使用する

DataGrid、List、または Tree のいずれかのコントロールの rendererIsEditor プロパティを true に設定すると、このコントロールではデフォルトの TextInput コントロールがアイテムエディタとして使用されます。この設定がない場合は、itemRenderer プロパティで指定されているアイテムレンダラーが使用されます。アイテムレンダラーを指定している場合は、編集可能なコントロールをそのレンダラーで指定し、コントロールの値をユーザーが編集できるようにする必要があります。

たとえば、次の例に示すアイテムレンダラーでは、TextInput コントロールを使用してセルの情報を表示し、その内容を編集できるようにしています。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\myComponents\MyContactEditable.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" >

   <mx:Script>
        <![CDATA[
            // Define a property for returning the new value to the cell.
            [Bindable]
            public var newContact:String;
        ]]>
    </mx:Script>

    <mx:Label id="title" text="{data.label1}"/>
    <mx:Label id="contactLabel" text="Last Contacted By:"/>
```

```
    <mx:TextInput id="contactTI"
        editable="true"
        text="{data.Contact}"
        change="newContact=contactTI.text;"/>
</mx:VBox>
```

次の例に示すように、このアイテムレンダラーを DataGrid コントロールで使用できます。

```
<?xml version="1.0"?>
<!-- itemRenderers\dataGrid\MainAppEditable.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="700" width="700">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
                {label1: "Order #2314", Contact: "John Doe",
                    Confirmed: false, Photo: "john_doe.jpg", Sent: false},
                {label1: "Order #2315", Contact: "Jane Doe",
                    Confirmed: true, Photo: "jane_doe.jpg", Sent: false}
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="myDG"
        width="500" height="250"
        dataProvider="{initDG}"
        variableRowHeight="true"
        editable="true">
        <mx:columns>
            <mx:DataGridColumn dataField="Photo"
                editable="false"/>
            <mx:DataGridColumn dataField="Contact"
                width="200"
                editable="true"
                rendererIsEditor="true"
                itemRenderer="myComponents.MyContactEditable"
                editorDataField="newContact"/>
            <mx:DataGridColumn dataField="Confirmed"
                editable="true"
                rendererIsEditor="true"
                itemRenderer="mx.controls.CheckBox"
                editorDataField="selected"/>
            <mx:DataGridColumn dataField="Sent"
                editable="true"
                rendererIsEditor="false"
                itemEditor="mx.controls.CheckBox"
                editorDataField="selected"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

前の例では、DataGrid コントロールの 2 番目の列と 3 番目の列で rendererIsEditor プロパティを true に設定することにより、アイテムレンダラーをアイテムエディタとして使用していました。

例 : カスタムアイテムエディタでデータバリデータを使用する

リストベースのコントロールにも、他のコントロールと同様、セルに格納されたデータを検証する機能を設定できます。この機能を使用するには、データバリデータを実装するアイテムレンダラーまたはアイテムエディタを作成します。データバリデータの詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

次の例は、アイテムエディタコンポーネントを検証するコードを示しています。このコードでは、TextInput コントロールを使用してフィールドを編集します。この例では、ユーザーの入力を検証するために、TextInput コントロールの text プロパティに PhoneNumberValidator バリデータを割り当てます。

```
<?xml version="1.0"?>
<!-- itemRenderers\validator\myComponents\EditorPhoneValidator.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:Script>
        <![CDATA[
            // Define a property for returning the new value to the cell.
            [Bindable]
            public var returnPN:String;
        ]]>
    </mx:Script>

    <mx:PhoneNumberValidator id="pnV"
        source="{newPN}"
        property="text"
        trigger="{newPN}"
        triggerEvent="change"
        required="true"/>
    <mx:TextInput id="newPN"
        text="{data.phone}"
        updateComplete="returnPN=newPN.text;"
        change="returnPN=newPN.text;"/>
</mx:VBox>
```

ユーザーが入力した電話番号が不適切な場合、PhoneNumberValidator によってエディタの周囲に赤いボックスが描画されます。そこにマウスを置くと、検証エラーメッセージが表示されます。

次の例は、このアイテムエディタを使用したアプリケーションのコードを示しています。

```
<?xml version="1.0" ?>
<!-- itemRenderers\validator\MainDGValidatorEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var initDG:ArrayCollection = new ArrayCollection([
```

```
                {name: 'Bob Jones', phone: '413-555-1212',
                    email: 'bjones@acme.com'},
                {name: 'Sally Smith', phone: '617-555-5833',
                    email: 'ssmith@acme.com'},
            ]);
        ]]>
    </mx:Script>

    <mx:DataGrid id="dg"
        width="500" height="200"
        editable="true"
        dataProvider="{initDG}">
        <mx:columns>
            <mx:DataGridColumn dataField="name"
                headerText="Name" />
            <mx:DataGridColumn dataField="phone"
                headerText="Phone"
                itemEditor="myComponents.EditorPhoneValidator"
                editorDataField="returnPN"/>
            <mx:DataGridColumn dataField="email" headerText="Email" />
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

List コントロールでアイテムエディタを使用した例

このセクションでは、DataGrid、List、および Tree の各コントロールのアイテムエディタを作成する方法について説明します。アイテムエディタの詳細については、823 ページ、第 22 章の「アイテムエディタの操作」を参照してください。

例 : DataGrid コントロールでアイテムエディタを使用する

DataGrid コントロールでアイテムエディタを使用する例については、823 ページ、第 22 章の「アイテムエディタの操作」を参照してください。

例 : List コントロールでカスタムアイテムエディタを使用する

次のアイテムエディタに示すように List コントロールにアイテムエディタを追加すると、ユーザーが各州の情報を編集できるようになります。

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
<!-- itemRenderers\list\myComponents\EditorStateInfo.mxml -->

    <mx:TextInput id="newLabel" text="{data.label}" />
    <mx:TextInput id="newData" text="{data.data}" />
    <mx:TextInput id="newWebPage" text="{data.webPage}" />
</mx:VBox>
```

州名、州都名、および Web アドレスをユーザーが編集できるように、3 種類の TextInput コントロールを持つアイテムエディタを定義します。このアイテムエディタは 3 つの値を返します。したがって、List コントロールのデータプロバイダにそれらの値を書き込むイベントリスナーを itemEditEnd イベントについて記述します。次にこの例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\list\MainListStateRendererEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="700" width="700">

    <mx:Script>
        <![CDATA[

            import mx.events.ListEvent;
            import myComponents.EditorStateInfo;

            // Define the event listener.
            public function processData(event:ListEvent):void {

                // Disable copying data back to the control.
                event.preventDefault();

                // Get new label from editor.
                myList.editedItemRenderer.data.label =
EditorStateInfo(event.currentTarget.itemEditorInstance).newLabel.text;

                // Get new data from editor.
                myList.editedItemRenderer.data.data =
EditorStateInfo(event.currentTarget.itemEditorInstance).newData.text;

                // Get new webPage from editor.
                myList.editedItemRenderer.data.webPage =
EditorStateInfo(event.currentTarget.itemEditorInstance).newWebPage.text;

                // Close the cell editor.
                myList.destroyItemEditor();

                // Notify the list control to update its display.
        myList.dataProvider.notifyItemUpdate(myList.editedItemRenderer);
            }
        ]]>
    </mx:Script>

    <mx:List id="myList"
        height="180" width="250"
        editable="true"
        itemRenderer="myComponents.RendererState"
        itemEditor="myComponents.EditorStateInfo"
        variableRowHeight="true"
        backgroundColor="white"
        itemEditEnd="processData(event);">
        <mx:dataProvider>
```

```
            <mx:Object label="Alaska"
                data="Juneau"
                webPage="http://www.state.ak.us/"/>
            <mx:Object label="Alabama"
                data="Montgomery"
                webPage="http://www.alabama.gov/" />
            <mx:Object label="Arkansas"
                data="Little Rock"
                webPage="http://www.state.ar.us/"/>
        </mx:dataProvider>
    </mx:List>
</mx:Application>
```

この例は、RendererState.mxml レンダラーを使用しています。このレンダラーについては、818 ページの「例 : アイテムレンダラーを List コントロールと共に使用する」を参照してください。

ドロップインアイテムエディタとしての DateField コントロールまたは ComboBox コントロールの使用

List コントロールで、DateField コントロールまたは ComboBox コントロールをドロップインアイテムエディタとして使用できます。ただし、目的のデータプロバイダがオブジェクトのコレクションである場合は、List コントロールの `labelField` プロパティを、DateField コントロールまたは ComboBox コントロールで変更する、そのデータプロバイダのフィールド名に設定する必要があります。次にその例を示します。

```
<?xml version="1.0"?>
<!-- itemRenderers\list\ListEditorDateField.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.collections.*;
            import mx.controls.DateField;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var catalog:ArrayCollection = new ArrayCollection([
                {confirmed: new Date(), Location: "Spain"},
                {confirmed: new Date(2006,0,15), Location: "Italy"},
                {confirmed: new Date(2004,9,24), Location: "Bora Bora"},
                {confirmed: new Date(), Location: "Vietnam"}
            ]);
        ]]>
    </mx:Script>

    <mx:List id="myList"
        width="300" height="300"
        rowHeight="50"
        dataProvider="{catalog}"
        editable="true"
```

```
        labelField="confirmed"
        itemEditor="mx.controls.DateField"
        editorDataField="selectedDate"/>
</mx:Application>
```

この例では、labelField プロパティの値として "confirmed" を指定し、データプロバイダの confirmed フィールドを DateField コントロールで変更することを指定します。

例 : Tree コントロールでカスタムアイテムエディタを使用する

Tree コントロールでは、通常、ツリーのノードごとに単一のラベルを表示します。ただし、各ノードのデータプロバイダには、通常は表示されない追加のデータが収められていることがあります。

この例では、Tree コントロールを使用してさまざまな会社の連絡先情報を表示します。この Tree コントロールでは、会社名をブランチノードとして、その会社内の部署名をリーフノードとして表示します。これらのどのノードを選択してもカスタムアイテムエディタが開き、該当の会社またはその会社にある任意の部署の電話番号や連絡先ステータスを変更できます。

Tree コントロールの最上位ノードは編集できません。したがって、この例では itemEditBeginning イベントを使用して、ユーザーが最上位ノードを選択していないか判断します。最上位ノードが選択されている場合は、次の例に示すように、itemEditBeginning イベントのイベントリスナーで編集を禁止します。

```
<?xml version="1.0"?>
<!-- itemRenderers\tree\MainTreeEditor.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="600" height="600">

    <mx:Script>
        <![CDATA[

        import mx.events.ListEvent;
        import myComponents.TreeEditor;

        private var contacts1:Object =
          {label: "top", children: [
            {label: "Acme", status: true, phone: "243-333-5555", children: [
              {label: "Sales", status: true, phone: "561-256-5555"},
              {label: "Support", status: false, phone: "871-256-5555"}
            ]},
            {label: "Ace", status: true, phone: "444-333-5555", children: [
              {label: "Sales", status: true, phone: "232-898-5555"},
              {label: "Support", status: false, phone: "977-296-5555"},
            ]},
            {label: "Platinum", status: false, phone: "521-256-5555"}
        ]};

        private function initCatalog(cat:Object):void {
```

```
            myTree.dataProvider = cat;
        }

        // Define the event listener for the itemEditBeginning event
        // to disable editing when the user selects
        // the top node in the tree.
        private function disableEditing(event:ListEvent):void {
            if(event.rowIndex==0) {
                event.preventDefault();
            }
        }

        // Define the event listener for the itemEditEnd event
        // to copy the updated data back to the data provider
        // of the Tree control.
        public function processData(event:ListEvent):void {

            // Disable copying data back to the control.
            event.preventDefault();

            // Get new phone number from editor.
            myTree.editedItemRenderer.data.phone =
    TreeEditor(event.currentTarget.itemEditorInstance).contactPhone.text;

            // Get new status from editor.
            myTree.editedItemRenderer.data.status =
    TreeEditor(event.currentTarget.itemEditorInstance).confirmed.selected;

            // Close the cell editor.
            myTree.destroyItemEditor();

            // Notify the list control to update its display.
    myTree.dataProvider.notifyItemUpdate(myTree.editedItemRenderer);
        }
        ]]>
    </mx:Script>

    <mx:Tree id="myTree"
        width="400" height="400"
        editable="true"
        itemEditor="myComponents.TreeEditor"
        editorHeightOffset="75" editorWidthOffset="-100"
        editorXOffset="40" editorYOffset="30"
        creationComplete="initCatalog(contacts1);"
        itemEditBeginning="disableEditing(event);"
        itemEditEnd="processData(event);"/>
</mx:Application>
```

Tree コントロールの itemEditor プロパティを使用して、カスタムアイテムエディタを指定します。また、アイテムエディタの位置を指定するには、editorHeightOffset、editorWidthOffset、editorXOffset、および editorYOffset の各プロパティを使用します。

"TreeEditor.mxml" ファイルで定義された次のアイテムエディタでは、Tree コントロールの各アイテムに関連付けられたデータを編集できます。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- itemRenderers/tree/myComponents/TreeEditor.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Define variables for the new data.
            public var newPhone:String;
            public var newConfirmed:Boolean;

        ]]>
    </mx:Script>

    <!-- Display item label.-->
    <mx:Label text="{data.label}"/>

    <!-- Display the text 'Phone:' and let the user edit it.-->
    <mx:HBox>
        <mx:Text text="Phone:"/>
        <mx:TextInput id="contactPhone"
            width="150"
            text="{data.phone}"
            change="newPhone=contactPhone.text;"/>
    </mx:HBox>

    <!-- Display the status using a CheckBox control
        and let the user edit it.-->
    <mx:CheckBox id="confirmed"
        label="Confirmed"
        selected="{data.status}"
        click="newConfirmed=confirmed.selected;"/>
</mx:VBox>
```

第 23 章

ツールヒントの使用

23

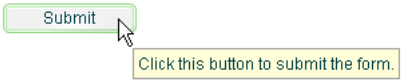
Adobe Flex ツールヒントを使用すると、ユーザーに役立つ情報を提供できます。ユーザーがグラフィカルコンポーネントにマウスポインタを重ねると、ツールヒントがポップアップし、テキストが表示されます。ツールヒントでアプリケーションの操作手順を示したり、ツールヒントをカスタマイズして別の機能を実装することもできます。このトピックでは、Flex アプリケーションでのツールヒントの使用方法について説明します。

目次

ツールヒントについて

ツールヒントは、多くのデスクトップアプリケーションが標準で備えている機能です。ユーザーが画面上のエレメント (Button コントロールなど) にマウスポインタを重ねたときにメッセージが表示され、アプリケーションが使いやすくなります。

ユーザーがボタンの上にマウスポインタを重ねると、次の図のようなツールヒントテキストが表示されます。



マウスポインタをコンポーネントから離すか、マウスボタンをクリックするとツールヒントが消えます。マウスポインタをコンポーネント上に置いたままにした場合も、しばらくするとツールヒントが消えます。デフォルトの動作では、1 度に 1 つのツールヒントだけを表示します。

マウスポインタをコンポーネントに重ねたときにツールヒントが表示されるまでの時間を設定できます。同様に、ツールヒントが消えるまでの時間を設定することもできます。

コンテナにツールヒントを定義すると、コンテナの子にツールヒントが設定されていない場合、子に対しても親のツールヒントが表示されます。

Flex ツールヒントは、スタイルシートやツールヒントテキストのダイナミックロードをサポートしています。ツールヒントのテキストに、埋め込み HTML を使用することはできません。スタイルシートおよびテキストのダイナミックロードの詳細については、860 ページの「ツールヒントのスタイルの設定」および 868 ページの「ダイナミックツールヒントテキストの使用」を参照してください。

Flex コンポーネントの中には、ツールヒントによく似た独自の " データヒント " を持つものがあります。そのようなコンポーネントとしては、List コントロール、チャートコントロールの大部分、DataGrid コントロールなどが挙げられます。詳細については、コンポーネントのマニュアルを参照してください。

ツールヒントは、Flex 内でアクセシビリティを実装するための基盤となります。アクセシビリティをサポートするすべてのコントロールは、ツールヒントテキストを JAWS などのスクリーンリーダーで読み上げ可能にすることで、アクセシビリティを実現します。アクセシビリティの詳細については、1109 ページ、第 36 章の「アクセシビリティアプリケーションの作成」を参照してください。

ツールヒントの作成

UIComponent クラスを拡張したビジュアル Flex コンポーネントはすべて、toolTip プロパティをサポートします (UIComponent クラスは IToolTipManagerClient インターフェイスを実装しています)。このプロパティは、UIComponent クラスから継承されます。toolTip プロパティにはテキストストリングを設定します。コンポーネントの上にマウスポインタを重ねると、設定しておいたテキストストリングが表示されます。次の例では、Button コントロールの toolTip プロパティにテキストを設定します。

```
<?xml version="1.0"?>
<!-- tooltips/BasicToolTip.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Button id="b1" label="Click Me" toolTip="This Button does nothing."/>

</mx:Application>
```

ツールヒントの値を ActionScript で設定するには、コンポーネントの toolTip プロパティを使用します。次の例では、新しい Button を作成し、Button コントロールの toolTip プロパティを設定します。

```
<?xml version="1.0"?>
<!-- tooltips/BasicToolTipActionScript.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     public function createNewButton(event:MouseEvent):void {
        var myButton:Button = new Button();
        myButton.label = "Create Another Button";
        myButton.toolTip = "Click this new button to create another button.";
```

```
        myButton.addEventListener(MouseEvent.CLICK, createNewButton);
        addChild(myButton);
    }
  ]]></mx:Script>
  <mx:Button id="b1" label="Create Another Button"
click="createNewButton(event);"/>
</mx:Application>
```

コンテナの子にツールヒントを定義していない場合は、親のツールヒントが表示されます。たとえば、ツールヒントを持つ Panel コンテナに Button コントロールを追加した場合、ユーザーが Panel コンテナの上にマウスポインタを移動すると、Panel のツールヒントテキストが表示されます。マウスポインタを Button コントロールの上に移動すると、Panel のツールヒントが引き続き表示されます。コンテナのツールヒントテキストをオーバーライドするには、子の `toolTip` プロパティの値を設定します。

次の例では、ツールヒントテキストの継承を示します。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipInheritance.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:VBox toolTip="VBOX">
     <mx:Button id="b1" label="Button 1" toolTip="BUTTON"/>
     <mx:Button id="b2" label="Button 2"/>
  </mx:VBox>

</mx:Application>
```

マウスポインタを b1 ボタンの上に重ねると、ツールヒントに "BUTTON" と表示されます。マウスポインタを b2 ボタンの上か、VBox コンテナ上の b1 ボタン以外の場所に重ねた場合は、ツールヒントに "VBOX" と表示されます。

TabNavigator コンテナは、その子コンテナのツールヒントを使用します。TabNavigator コンテナの子ビューにツールヒントを追加した場合、そのビューのタブにマウスを重ねたときにツールヒントが表示されます。ビュー自体にマウスを重ねてもツールヒントは表示されません。TabNavigator コンテナにツールヒントを追加した場合、そのツールヒントがタブまたはビューにオーバーライドされない限り、タブまたはビューのいずれかにマウスを重ねるとツールヒントが表示されます。次のコントロールでも、これと同じようにツールヒントが動作します。

- Accordion
- ButtonBar
- LinkBar
- TabBar
- ToggleButtonBar

ツールヒントテキストのサイズに制限はありませんが、メッセージが長いと読みづらくなることがあります。ツールヒントテキストがツールヒントボックスの幅に到達すると、テキストが次の行に折り返されます。ツールヒントテキストに改行を追加することもできます。ActionScript では、改行のエスケープ文字 (\n) を使用します。MXML タグの場合は、XML エンティティの
 を使用します。

次に、改行のエスケープ文字 (\n) とエンティティ参照 (
) の使用例を示します。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipNewlines.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="doSomething(event)">
  <mx:Script><![CDATA[
     public function doSomething(event:Event):void {
        // Use the \n to force a line break in ActionScript.
        b1.toolTip = "Click this button \n to clear the form.";
     }
  ]]></mx:Script>
  <mx:Button id="b1" label="Clear" width="100"/>
  <!-- Use &#13; to force a line break in MXML tags. -->
  <mx:Button id="b2" label="Submit" width="100" toolTip="Click this button &#13;
to submit the form."/>
</mx:Application>
```

ツールヒントのテキストにスタイルを適用することもできます。ツールヒントの CSS タイプセレクタを使用して、アプリケーション内のすべてのツールヒントに対してスタイルを適用し、その他の設定値を変更できます。以降のセクションでは、ツールヒントのテキストおよびボックスにスタイルを適用する方法について説明します。

ツールヒントのスタイルの設定

CSS (Cascading Style Sheet : カスケーディングスタイルシート) のシンタックスまたは mx.styles.StyleManager クラスを使用して、ツールヒントテキストやツールヒントボックスの外観を変更できます。ツールヒントのスタイルに対して行った変更は、現在のアプリケーション内のすべてのツールヒントに適用されます。

ツールヒントのデフォルトスタイルは、"framework.swc" ファイル内の "defaults.css" ファイルの中のツールヒントタイプセレクタによって定義されています。<mx:Style> タグ内でタイプセレクタを使用して、ツールヒントのデフォルトスタイルをオーバーライドできます。次の例では、CSS シンタックスを使用して、ツールヒントタイプセレクタのスタイルを設定しています。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
     ToolTip {
        fontFamily: "Arial";
        fontSize: 19;
        fontStyle: "italic";
```

```
        color: #FF6699;
        backgroundColor: #33CC99;
    }
  </mx:Style>

  <mx:Button id="b1" label="Click Me" toolTip="This Button does nothing."/>

</mx:Application>
```

StyleManager クラスを使用してツールヒントのスタイルを設定するには、次の例に示すように、setStyle() メソッドを使用してツールヒントタイプセレクタにスタイルを適用します。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipStyleManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="setToolTipStyle()">

  <mx:Script><![CDATA[
     import mx.styles.StyleManager;
     private function setToolTipStyle():void {

StyleManager.getStyleDeclaration("ToolTip").setStyle("fontStyle","italic");
        StyleManager.getStyleDeclaration("ToolTip").setStyle("fontSize","19");

StyleManager.getStyleDeclaration("ToolTip").setStyle("fontFamily","Arial");
        StyleManager.getStyleDeclaration("ToolTip").setStyle("color","#FF6699");

StyleManager.getStyleDeclaration("ToolTip").setStyle("backgroundColor","#33CC99"
);
     }
]]></mx:Script>

  <mx:Button id="b1" label="Click Me" toolTip="This Button does nothing."/>

</mx:Application>
```

ツールヒントでは、開発者がグローバルに設定した継承可能なスタイルが使用されます。たとえば、StyleManager を使用して、ツールヒントの fontWeight をグローバルセレクタに設定できます。次に例を示します。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipGlobalStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="setToolTipStyle()">

  <mx:Script><![CDATA[
     import mx.styles.StyleManager;
     private function setToolTipStyle():void {

StyleManager.getStyleDeclaration("global").setStyle("fontWeight","bold");
     }
]]></mx:Script>

  <mx:Button id="b1" label="Click Me" toolTip="This Button does nothing."/>

</mx:Application>
```

グローバルセレクタに fontWeight プロパティを設定すると、ツールヒント以外の多くのコントロールのテキストにもその変更が反映されるため、グローバルセレクタを使用する際は注意が必要です。

スタイルプロパティとプログラムスキンを使用してツールヒントのスキンを交換できます。詳細については、776 ページの「ツールヒントのスキン交換」を参照してください。

ツールヒントでサポートされるスタイルの一覧については、『Adobe Flex 2 リファレンスガイド』を参照してください。スタイルの使用の詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

ToolTip コントロールのスキンを交換して、ツールヒントにまったく新しい外観を与えることができます。そのためには、ToolTipBorder プログラムスキンを使用するか、グラフィカルスキンに交換します。ツールヒントのスキン設定の詳細については、741 ページ、第 20 章の「スキンの使用」を参照してください。ToolTip コントロールをプログラムスキンに交換する例については、776 ページの「ツールヒントのスキン交換」を参照してください。

ツールヒントの幅の設定

ツールヒントボックスの幅は、mx.controls.ToolTip クラスの maxWidth プロパティを変更することによって設定できます。このプロパティは静的であるため、このプロパティを設定すると、すべてのツールヒントボックスに対して設定され、ツールヒントのインスタンスには設定されません。

maxWidth プロパティには、新しいツールヒントボックスの最大幅をピクセル単位で指定します。たとえば、次の行によりツールヒントボックスの最大幅が 100 ピクセルに変更されます。

```
<?xml version="1.0"?>
<!-- tooltips/SetMaxWidth.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
     import mx.controls.ToolTip;

     public function init():void {
        ToolTip.maxWidth = 100;
     }

     public function createNewButton(event:MouseEvent):void {
        var myButton:Button = new Button();
        myButton.label = "Create Another Button";
        myButton.toolTip = "Click this new button to create another button.";
        myButton.addEventListener(MouseEvent.CLICK, createNewButton);
        addChild(myButton);

     }
  ]]></mx:Script>
  <mx:Button id="b1" label="Create Another Button"
click="createNewButton(event);" toolTip="Click this button to create a new
one."/>
</mx:Application>
```

Flex では、ツールヒントの幅がこの値を超えないように、ツールヒントのテキストを折り返して複数行で表示します。ツールヒントボックスのテキスト幅が maxWidth プロパティよりも狭い場合は、テキストがちょうど収まるように、ツールヒントボックスの幅が調整されます。

maxWidth のデフォルト値は 300 です。maxWidth の値がアプリケーションの幅よりも大きい場合は、ツールヒントボックス内のテキストがクリッピングされます。

ツールヒントイベントの使用

ツールヒントは、ライフサイクルの間に多数のイベントをトリガします。これらのイベントのタイプは ToolTipEvent です。

ToolTipEvent オブジェクトには、type プロパティや target プロパティに加えて、ツールヒントを参照する toolTip プロパティがあります。ツールヒントを参照することで、text プロパティを使用してツールヒントテキストにアクセスできます。

<mx:Script> ブロック内で ToolTipEvent タイプのイベントを使用するには、mx.events.ToolTipEvent クラスを読み込む必要があります。

次の例は、TOOL_TIP_SHOW イベントに応答してサウンドを再生します。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipsWithSoundEffects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="init()">
  <mx:Script><![CDATA[
     import mx.events.ToolTipEvent;
     import flash.media.Sound;

     [Embed(source="../assets/sound1.mp3")]
     private var beepSound:Class;
     private var myClip:Sound;

     public function playSound():void {
        myClip.play();
     }
     private function myListener(event:ToolTipEvent):void {
        playSound();
     }
     private function init():void {
        myLabel.addEventListener(ToolTipEvent.TOOL_TIP_SHOW, myListener);
        myClip = new beepSound();
     }
  ]]></mx:Script>
  <mx:Label id="myLabel" toolTip="ToolTip" text="Mouse Over Me"/>
</mx:Application>
```

NavBar コントロールでのツールヒントの使用

NavBar と TabBar のサブクラス (ButtonBar、LinkBar、ToggleButtonBar) は、データプロバイダでツールヒントをサポートします。データプロバイダ配列の中に、ナビゲーションアイテムの toolTip を指定する toolTip フィールドを含めることができます。

次の例では、ナビゲーションアイテムごとにツールヒントを作成しています。

```
<?xml version="1.0"?>
<!-- tooltips/NavItemToolTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ButtonBar>
     <mx:Object label="OK" toolTip="OK tip"/>
     <mx:Object label="Cancel" toolTip="Cancel tip"/>
  </mx:ButtonBar>
</mx:Application>
```

コンポーネントのデータプロバイダ内の子エレメントでツールヒントを使用できます。コンポーネントは、それらのツールヒントを認識し、表示します。次の例では、ツールヒントは LinkBar にまで反映されます。

```
<?xml version="1.0"?>
<!-- tooltips/DataProviderToolTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:VBox>
     <!-- Create a LinkBar control to navigate the ViewStack. -->
     <mx:LinkBar dataProvider="{vs1}" borderStyle="solid"/>

     <!-- Define the ViewStack and the three child containers. -->
     <mx:ViewStack id="vs1" borderStyle="solid" width="100%" height="150">
        <mx:Canvas id="search" label="Search" toolTip="Search Screen">
           <mx:Label text="Search Screen"/>
        </mx:Canvas>
        <mx:Canvas id="custInfo" label="Customer"
           toolTip="Customer Info Screen">
           <mx:Label text="Customer Info"/>
        </mx:Canvas>
        <mx:Canvas id="accountInfo" label="Account"
           toolTip="Account Info Screen">
           <mx:Label text="Account Info"/>
        </mx:Canvas>
     </mx:ViewStack>
  </mx:VBox>
</mx:Application>
```

また、NavBar の toolTipField プロパティの値を、ツールヒントを提供するデータプロバイダのフィールドを指すように設定することもできます。次の例のデータプロバイダは、myToolTip フィールドでツールヒントを定義しています。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipFieldExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ButtonBar toolTipField="myToolTip">
     <mx:Object label="OK" myToolTip="OK tip"/>
     <mx:Object label="Cancel" myToolTip="Cancel tip"/>
  </mx:ButtonBar>
</mx:Application>
```

ツールヒントマネージャの使用

ToolTipManager クラスを使用すると、表示の遅延時間やツールヒントの無効化など、ツールヒントの基本的な機能を設定できます。設定は、mx.managers package に格納されます。ToolTipManager を使用するときは、このクラスを読み込む必要があります。また、ToolTipManager クラスの currentToolTip プロパティには、現在のツールヒントへの参照が含まれます。

このセクションでは、ツールヒントマネージャの使用方法について説明します。

ツールヒントの有効化と無効化

Flex アプリケーションのツールヒントは有効または無効にすることができます。ツールヒントを無効にすると、コンポーネントの toolTip プロパティが設定されているかどうかにかかわらず、可視コンポーネントの上にマウスポインタを重ねてもツールヒントボックスが表示されなくなります。

ツールヒントを有効または無効にするには、ToolTipManager の enabled プロパティを使用します。このプロパティを true に設定するとツールヒントが有効になり、false に設定するとツールヒントが無効になります。デフォルト値は true です。

次の例では、ユーザーが [Toggle ToolTips] ボタンをクリックしたときにツールヒントのオンとオフを切り替えます。

```
<?xml version="1.0"?>
<!-- tooltips/ToggleToolTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script><![CDATA[
     import mx.managers.ToolTipManager;

     private function toggleToolTips():void {
        if (ToolTipManager.enabled) {
           ToolTipManager.enabled = false;
        } else {
           ToolTipManager.enabled = true;
        }
     }
  ]]></mx:Script>

  <mx:Button label="Toggle ToolTips" width="100" click="toggleToolTips()"
toolTip="Click me to enable/disable tooltips."/>

</mx:Application>
```

遅延時間の設定

" 遅延時間 " とは、何かが発生するまでの経過時間を表す指標です。たとえば、コンポーネント上にマウスポインタを移動すると、少し遅れてツールヒントが表示されます。ツールヒントを必要としないユーザーは、この間にマウスポインタを別の場所に移動して、テキストがポップアップ表示されるのを防ぐことができます。

ToolTipManager を使用すると、ツールヒントボックスが表示されるまでの時間や、マウスポインタをコンポーネント上に重ねたときにツールヒントを画面上に表示し続ける時間を設定できます。また、ツールヒント間の遅延を設定することもできます。

ToolTipManager の showDelay プロパティ、hideDelay プロパティ、および scrubDelay プロパティの値は、ActionScript のコードブロックで設定します。次の表は、遅延時間に関する ToolTipManager のプロパティとその説明です。

プロパティ	説明
showDelay	ツールヒントを持つコンポーネント上にマウスポインタを移動したとき、ツールヒントボックスを表示させるまでに Flex が待機する時間 (ミリ秒) です。 ツールヒントをすぐに表示させるには、showDelay プロパティを 0 に設定します。 デフォルト値は、500 ミリ秒 (0.5 秒) です。

hideDelay	表示されたツールヒントボックスを非表示にするまでに Flex が待機する時間 (ミリ秒) です。Flex がツールヒントボックスを非表示にした後、再びツールヒントボックスを表示するには、いったんコンポーネントの外側にマウスポインタを移動し、再度そのコンポーネントにマウスポインタを重ねる必要があります。
	hideDelay プロパティを 0 に設定した場合、ツールヒントは表示されません。デフォルト値の 10,000 ミリ秒 (10 秒) を使用することをお勧めします。
	hideDelay プロパティを Infinity に設定した場合は、ユーザーが何らかのイベント (マウスポインタをコンポーネントの外側に移動するなど) をトリガするまでツールヒントは表示されたままになります。次の例では、hideDelay プロパティを Infinity に設定します。
	`import mx.managers.ToolTipManager;`
	`ToolTipManager.hideDelay = Infinity;`

scrubDelay	ユーザーがコントロール間でマウスポインタを移動したときに、あるツールヒントボックスが表示されてから別のツールヒントボックスが表示されるまで (showDelay で指定された時間) の待機時間です (ミリ秒単位)。 この設定は、ユーザーがコントロール間をすばやく移動する場合に役立ちます。つまり、最初のツールヒントの表示後、showDelay の設定時間を待たなくても、すぐに別のツールヒントが表示されます。scrubDelay の設定が短いほど、次のツールヒントが表示されるまで showDelay プロパティに指定された時間待たなければならない可能性が高くなります。 このプロパティは、ツールバーに複数のボタンがあり、ユーザーがそれぞれの簡潔な説明を見てすばやくそれらの機能を把握する場合に有効です。 デフォルト値は 100 です。

次の例では、Application コントロールの initialize イベントを使用して、ToolTipManager の初期値を設定します。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipTiming.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initApp();">
  <mx:Script><![CDATA[
     import mx.managers.ToolTipManager;
     private function initApp():void {
        ToolTipManager.enabled = true;// Optional. Default value is true.
        ToolTipManager.showDelay = 0;// Display immediately.
        ToolTipManager.hideDelay = 3000; // Hide after 3 seconds.
     }
  ]]></mx:Script>

  <mx:Button label="Click Me" toolTip="Click this Button to do something"/>
  <mx:Button label="Click Me" toolTip="Click this Button to do something else"/>
  <mx:Button label="Click Me" toolTip="Click this Button to do another thing"/>
  <mx:Button label="Click Me" toolTip="Click this Button to do the same thing"/>

</mx:Application>
```

ツールヒントにおけるエフェクトの使用

ツールヒントには、Flex の標準エフェクトのほか、カスタムエフェクトを使用することもできます。ツールヒントが表示または非表示になったときにトリガされるエフェクトを定義するには、ToolTipManager の showEffect プロパティまたは hideEffect プロパティを設定します。

次の例では、フェードエフェクトを使用し、ユーザーがツールヒントのあるコンポーネントの上にマウスポインタを重ねると、ツールヒントがフェードインします。

```
<?xml version="1.0"?>
<!-- tooltips/FadeInToolTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="600" initialize="app_init();">

  <mx:Script><![CDATA[
     import mx.managers.ToolTipManager;
     public function app_init():void {
        ToolTipManager.showEffect = fadeIn;
     }
  ]]></mx:Script>

  <mx:Fade id="fadeIn" alphaFrom="0" alphaTo="1" duration="1000"/>

  <mx:Button id="b1" label="Click me" toolTip="This is a ToolTip."/>

</mx:Application>
```

デフォルトでは、この例で使用されているフォントはフェードしません。埋め込みエフェクトを実現するためには、埋め込みフォントを使用する必要があります。埋め込みフォントの使用の詳細については、708 ページの「埋め込みフォントの使用」を参照してください。

エフェクトの使用とカスタムエフェクトの定義の詳細については、603 ページ、第 17 章の「ビヘイビアの使用」を参照してください。

ダイナミックツールヒントテキストの使用

ツールヒントは、ユーザーに静的ヘルプテキストを表示するだけではありません。ツールヒントテキストをデータやコンポーネントのテキストにバインドすることもできます。これにより、ツールヒントをユーザーインターフェイスの一部として使用することが可能となり、ドリルダウン情報やクエリ結果など、ユーザーの操作に合わせてカスタマイズした、より有用なテキストを表示できます。

ツールヒントテキストの値を別のコントロールのテキストにバインドするには、中カッコ ({}) シンタックスを使用します。

次の例では、ユーザーがマウスポインタを Button コントロール上に重ねたときに、TextInput コントロールの値を Button コントロールのツールヒントテキストに挿入しています。

```
<?xml version="1.0"?>
<!-- tooltips/BoundToolTipText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:TextInput id="txtTo" width="300"/>
  <mx:Button label="Send" toolTip="Send e-mail to {txtTo.text}"/>

</mx:Application>
```

この例では、ユーザーが TextInput ボックスに「fred@fred.com」と入力し、その後マウスポインタをボタンの上に重ねると、ツールヒントボックスに “Send e-mail to fred@fred.com” というメッセージが表示されます。

ツールヒントのダイナミックテキストを作成する別の方法として、toolTipShow イベントハンドラでツールヒントを取得して、text プロパティの値を変更する方法があります。次の例では、Button コントロールの toolTipShow イベントのリスナーとして、myToolTipChanger() メソッドを登録します。そのメソッドの中で、ToolTipManager.currentToolTip.text プロパティの値を実行時までわからない値に設定しています。

```
<?xml version="1.0"?>
<!-- tooltips/DynamicToolTipText.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initApp()">
  <mx:Script><![CDATA[
     import mx.managers.ToolTipManager;
     import mx.controls.ToolTip;
     import mx.events.ToolTipEvent;

     public function initApp():void {
```

```
        b1.addEventListener(ToolTipEvent.TOOL_TIP_SHOW, myToolTipChanger)
    }
    public function myToolTipChanger(event:ToolTipEvent):void {
        // Append ?myName=fred to your request string or pass the value of
        // myName in to your application some other way.
        ToolTipManager.currentToolTip.text = "Click the button, " +
            Application.application.parameters.myName;
    }
  ]]>   </mx:Script>
  <mx:Button id="b1" label="Click Me" toolTip="Click the button"/>
</mx:Application>
```

オブジェクトに最初からツールヒントがある場合、toolTipShow イベントハンドラでは現在のツールヒントのテキストのみ作成できます。たとえば、上の例のボタンで toolTip プロパティの値が設定されていない場合、myToolTipChanger() メソッドが呼び出されてもツールヒントは表示されません。

Application.application.parameters オブジェクトの使用の詳細については、1080 ページの「Application.application.parameters オブジェクトの使用」を参照してください。

カスタムツールヒントの作成

ToolTipManager には、プログラムでツールヒントを使用できるメソッドが 2 つあります。createToolTip() メソッドは新しい ToolTip オブジェクトを作成し、destroyToolTip() メソッドは ToolTip オブジェクトを破棄します。ToolTip オブジェクトを作成する場合は、その他のオブジェクトと同様に、プロパティ、スタイル、イベント、およびエフェクトにアクセスしてカスタマイズできます。

createPopUp() メソッドのシグネチャを次に示します。

```
createToolTip(text:String, x:Number, y:Number, errorTipBorderStyle:String,
   context:IUIComponent):IToolTip
```

createToolTip() メソッドは、ToolTip クラスが実装する IToolTip インターフェイスを返します。このメソッドの戻り値をツールヒントにキャストしない方が効率的ですが、頻繁にキャストする必要があります。このキャストは次のいずれかの方法で行います。

- 次の例に示すように、as キーワードを使用します。

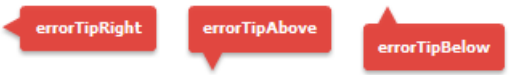
  ```
  myTip = ToolTipManager.createToolTip(s,10,10) as ToolTip;
  ```

- 次の例に示すように、*type*(*object*) のキャストシンタックスを使用します。

  ```
  myTip = ToolTip(ToolTipManager.createToolTip(s,10,10));
  ```

これらのキャスト方法で異なるのは、キャストに失敗したときの動作のみです。

text パラメータは、ツールヒントの内容を定義します。*x* パラメータおよび *y* パラメータは、ツールヒントの x 座標および y 座標をアプリケーションコンテナに対して相対的に定義します。*errorTipBorderStyle* プロパティは、エラーヒントのポインタの位置を設定します。このパラメータはオプションです。createToolTip() メソッドでこの値を指定した場合、そのツールヒントはエラーヒントのスタイルになります。次の例は、エラーヒントの有効な値とポインタの位置を示します。



context パラメータは現在は使用されていません。

エラーヒントの使用の詳細については、873 ページの「エラーヒントの使用」を参照してください。

ツールヒントは破棄されるまで表示されます。通常、一度に複数のツールヒントを表示するとユーザーが混乱するため、一度に表示するツールヒントは 1 つにしてください。

destroyToolTip() メソッドを使用すると、指定した ToolTip オブジェクトを破棄できます。destroyToolTip() メソッドのシグネチャは次のとおりです。

```
destroyToolTip(toolTip:IToolTip):void
```

toolTip パラメータは、破棄する ToolTip オブジェクトです。これは、createToolTip() メソッドによって返されたオブジェクトです。

次の例では、3 つの Button コントロールのある Panel コンテナにマウスポインタを重ねたときに表示されるカスタムツールヒントを作成します。各 Button コントロールには、マウスポインタを特定のコントロールに重ねたときに表示されるそれぞれのツールヒントがあります。マウスポインタを Panel コンテナから離したときには、大きなツールヒントだけが非表示になります。

```
<?xml version="1.0"?>
<!-- tooltips/CreatingToolTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.managers.ToolTipManager;
  import mx.controls.ToolTip;

  public var myTip:ToolTip;

  private function createBigTip():void {
     var s:String = "These buttons let you save, exit, or continue with the
current operation."
     myTip = ToolTipManager.createToolTip(s,10,10) as ToolTip;
     myTip.setStyle("backgroundColor",0xFFCC00);
     myTip.width = 150;
     myTip.height = 200;
  }

  private function destroyBigTip():void {
     ToolTipManager.destroyToolTip(myTip);
```

```
  }
  ]]></mx:Script>
  <mx:Panel rollOver="createBigTip()" rollOut="destroyBigTip()">
     <mx:Button label="OK" toolTip="Save your changes and exit."/>
     <mx:Button label="Apply" toolTip="Apply changes and continue."/>
     <mx:Button label="Cancel" toolTip="Cancel and exit."/>
  </mx:Panel>
</mx:Application>
```

Panel コンテナや VBox コンテナなどの既存のコントロールを拡張して IToolTip インターフェイスを実装する方法でも、カスタムツールヒントを作成することができます。次の例では、IToolTip インターフェイスの新たな実装の基礎に Panel コンテナを使用します。

```
<?xml version="1.0"?>
<!-- tooltips/ToolTipComponents/PanelToolTip.mxml -->
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="mx.core.IToolTip"
    width="200"
    alpha=".8"
    borderThickness="2"
    backgroundColor="0xCCCCCC"
    dropShadowEnabled="true"
    borderColor="black"
    borderStyle="solid"
>
    <mx:Script><![CDATA[
        [Bindable]
        public var bodyText:String = "heh";

        //  Implement required methods of the IToolTip interface; these
        //  methods are not used in this example, though.
        public var _text:String;

        public function get text():String {
            return _text;
        }
        public function set text(value:String):void {
        }
    ]]></mx:Script>

    <mx:Text text="{bodyText}" percentWidth="100"/>

</mx:Panel>
```

アプリケーション内でカスタムツールヒントを作成するには、ターゲットコンポーネントの toolTipCreate イベントハンドラを受け取ります。イベントハンドラで、新しいツールヒントをインスタンス化し、そのプロパティを設定します。続いて、ToolTipEvent オブジェクトの toolTip プロパティを新しいツールヒントに指定します。

次の例では、アプリケーションの最初の 2 つのボタンが、CustomToolTips パッケージのカスタム PanelToolTip に使用されます。3 番目のボタンは、デフォルトのツールヒントを使用して 2 つのボタンの違いを示します。この例を実行するには、"PanelToolTip.mxml" ファイルを CustomToolTips という名前のサブディレクトリに保存します。

```
<?xml version="1.0"?>
<!-- tooltips/MainCustomApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import ToolTipComponents.PanelToolTip;
        import mx.events.ToolTipEvent;

        private function createCustomTip(title:String, body:String,
event:ToolTipEvent):void {
           var ptt:PanelToolTip = new PanelToolTip();
           ptt.title = title;
           ptt.bodyText = body;
           event.toolTip = ptt;
        }
    ]]></mx:Script>

    <mx:Button id="b1"
        label="Delete"
        toolTip=" "
       toolTipCreate="createCustomTip('DELETE','Click this button to delete the
report.', event)"
    />

    <mx:Button id="b2"
        label="Generate"
        toolTip=" "
       toolTipCreate="createCustomTip('GENERATE','Click this button to generate
the report.', event)"
    />

    <mx:Button id="b3"
        label="Stop"
        toolTip="Click this button to stop the creation of the report. This
button uses a standard ToolTip style."
    />

</mx:Application>
```

エラーヒントの使用

エラーヒントは、errorTip クラスセレクタからそのスタイルを取得する ToolTip クラスのインスタンスです。通常は、データの無効により検証メカニズムが警告を表示するときにエラーヒントが表示されます。ただし、errorTip スタイルの定義を使用してそれをツールヒントに適用し、カスタム検証警告メカニズムを作成することもできます。

エラーヒントのスタイルは、"framework.swc" ファイル内の "defaults.css" ファイルに定義されています。errorTip には、次のデフォルト設定が指定されています。errorTip の前のピリオドは、それがクラスセレクタであることを示します。

```
.errorTip {
  color: #FFFFFF;
  fontSize: 9;
  fontWeight: "bold";
  shadowColor: #000000;
  borderColor: #CE2929;
  borderStyle: "errorTipRight";
  paddingBottom: 4;
  paddingLeft: 4;
  paddingRight: 4;
  paddingTop: 4;
}
```

エラーヒントの外観をカスタマイズするには、デフォルトのスタイルをオーバーライドする新しいテーマを作成するか、またはアプリケーション内でスタイルプロパティをオーバーライドします。テーマの作成の詳細については、698 ページの「テーマについて」を参照してください。

errorTip スタイルをツールヒントに適用して、検証エラーヒントによく似たツールヒントを作成できます。次の例は、検証ロジックは含まれていませんが、errorTip スタイルを使用して検証エラーヒントによく似たツールヒントを作成する方法を示しています。この例を実行するときは、TextInput コントロールにテキストを入力して Enter キーを押します。

```
<?xml version="1.0"?>
<!-- tooltips/ErrorTipStyle.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="20">
  <mx:Script><![CDATA[
     import mx.controls.ToolTip;
     import mx.managers.ToolTipManager;

     private var errorTip:ToolTip;
     private var myError:String;

     private function validateEntry(type:String, event:Object):void {
        // NOTE: Validation logic would go here.
        switch(type) {
           case "ssn":
            myError="Use SSN format (NNN-NN-NNNN)";
            break;
           case "phone":
```

```
            myError="Use phone format (NNN-NNN-NNNN)";
            break;
        }
        // Use the target's x and y positions to set position of error tip.
        errorTip = ToolTipManager.createToolTip(myError,event.currentTarget.x +
event.currentTarget.width,event.currentTarget.y) as ToolTip;

        // Apply the errorTip class selector.
        errorTip.setStyle("styleName", "errorTip");
    }
  ]]></mx:Script>

  <mx:TextInput id="ssn" enter="validateEntry('ssn',event)"/>
  <mx:TextInput id="phone" enter="validateEntry('phone',event)"/>
</mx:Application>
```

エラーヒントのもう1つの使用方法は、コンポーネントの errorString プロパティの値の設定です。この方法で設定すると、ツールヒントのインスタンスが ToolTipManager によって作成され、そのツールヒントに errorTip スタイルが適用されるので、ユーザー側でコーディングする必要がありません。

次の例は、errorString プロパティの値を設定してエラーヒントを作成する方法を示しています。

```
<?xml version="1.0"?>
<!-- tooltips/ErrorString.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="20">
  <mx:Script><![CDATA[
     import mx.controls.ToolTip;
     import mx.managers.ToolTipManager;

     private var errorTip:ToolTip;
     private var myError:String;

     private function validateEntry(type:String, event:Object):void {
        // NOTE: Validation logic would go here.
        switch(type) {
           case "ssn":
            myError="Use SSN format";
            break;
           case "phone":
            myError="Use phone format";
            break;
        }

        event.currentTarget.errorString = myError;

     }
  ]]></mx:Script>

  <mx:TextInput id="ssn" enter="validateEntry('ssn',event)"/>
  <mx:TextInput id="phone" enter="validateEntry('phone',event)"/>
</mx:Application>
```

また、次の例に示すように、errorTipBorderStyle プロパティの値を指定して、createToolTip() メソッドを呼び出す際にエラーヒントが作成されるようにすることもできます。

```
<?xml version="1.0"?>
<!-- tooltips/CreatingErrorTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.managers.ToolTipManager;
  import mx.core.IToolTip;

  public var myTip:IToolTip;

  private function createBigTip(event:Event):void {
     var myError:String = "These buttons let you save, exit, or continue with the
current operation."
     myTip = ToolTipManager.createToolTip(myError, event.currentTarget.x +
event.currentTarget.width, event.currentTarget.y, "errorTipRight");
  }

  private function destroyBigTip():void {
     ToolTipManager.destroyToolTip(myTip);
  }

  ]]></mx:Script>
  <mx:Panel rollOver="createBigTip(event)" rollOut="destroyBigTip()">
     <mx:Button label="OK" toolTip="Save your changes and exit."/>
     <mx:Button label="Apply" toolTip="Apply changes and continue."/>
     <mx:Button label="Cancel" toolTip="Cancel and exit."/>
  </mx:Panel>
</mx:Application>
```

バリデータの使用の詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

第 24 章

Cursor Manager の使用

24

Adobe Flex Cursor Manager を使用して、Flex アプリケーションのカーソルイメージを制御できます。Cursor Manager を利用することで、ユーザーに処理が完了するまで待つように知らせたり、許可されるアクションを知らせたりする視覚的なフィードバックや、あるいはその他のフィードバックを行うことができます。カーソルイメージには、JPEG、GIF、PNG、および SVG の各イメージファイルのほか、Sprite オブジェクトや SWF ファイルも使用できます。

このトピックでは、Cursor Manager を使用して Flex アプリケーションでカーソルイメージを制御する方法について説明します。

目次

Cursor Manager について

Flex では、デフォルトでアプリケーションカーソルとしてシステムカーソルが使用されます。システムカーソルは、オペレーティングシステムの設定で制御します。

Flex Cursor Manager を使用して、Flex アプリケーションのカーソルイメージを制御できます。たとえば、完了するまでユーザーが待つ必要があるような処理がアプリケーションで実行される場合、カーソルを変更することで、今は待ち状態であることを表すことができます。この場合は、カーソルを砂時計などのイメージに変更します。

また、ユーザーが実行できるアクションを示すために、カーソルを変更してユーザーにフィードバックすることもできます。たとえば、入力が有効な場合を示すカーソルイメージと、入力が無効な場合を示すカーソルイメージを変えることができます。

カーソルイメージには、JPEG、GIF、PNG、または SVG イメージや、Sprite オブジェクト、SWF ファイルを使用できます。

Cursor Manager の使用

Cursor Manager を使用するには、アプリケーションに mx.managers.CursorManager クラスを読み込み、このクラスのプロパティとメソッドを参照します。

Cursor Manager ではカーソルの優先順位リストが制御されており、リスト内で優先度が最も高いカーソルが表示されます。カーソルリストに、同じ優先度を持つカーソルが複数ある場合、最も新しく作成されたカーソルが表示されます。

新しいカーソルを作成し、必要に応じて CursorManager クラスの静的な `setCursor()` メソッドを使用して、カーソルの優先度を設定します。このメソッドにより、カーソルリストに新しいカーソルが追加されます。新しいカーソルが最も高い優先度を持つ場合、すぐに表示されます。優先度がリスト内の既存のカーソルよりも低い場合は、高い優先度を持つカーソルが削除されるまで、新しいカーソルは表示されません。

カーソルをリストから削除するには、静的な `removeCursor()` メソッドを使用します。削除されたカーソルが現在表示されているカーソルの場合、リスト内に次のカーソルがあれば、それが表示されます。リストが空になると、デフォルトのシステムカーソルが表示されます。

`setCursor()` メソッドのシグネチャを次に示します。

```
public static setCursor(cursorClass:Class, priority:int = 2,  xOffset:Number = 0,
   yOffset:Number = 0) : int
```

`setCursor()` メソッドのパラメータを次の表に示します。

パラメータ	説明	必須 / オプション
`cursorClass`	表示するカーソルのクラス名です。	必須
`priority`	カーソルの優先度レベルです。使用できる値は、`CursorManagerPriority.HIGH`、`CursorManagerPriority.MEDIUM`、および `CursorManagerPriority.LOW` です。デフォルト値は 2 です。これは `CursorManagerPriority.MEDIUM` に対応します。	オプション
`xOffset`	マウスポインタの位置を基準としたカーソルの x 方向オフセットです。デフォルト値は 0 です。	オプション
`yOffset`	マウスポインタの位置を基準としたカーソルの y 方向オフセットです。デフォルト値は 0 です。	オプション

このメソッドは、新しいカーソルの ID を返します。この ID を `removeCursor()` メソッドに渡すと、カーソルが削除されます。このメソッドのシグネチャは次のとおりです。

```
static removeCursor(cursorID:int):void
```

カーソルの作成と削除

次の例では、大きなイメージファイルのロード中に、カーソルをカスタムの待機カーソルまたはビジーカーソルに変更します。ロードが完了すると、ビジーカーソルは削除され、カーソルがシステムカーソルに戻ります。

メモ

Flex フレームワークには、デフォルトのビジーカーソルが用意されています。このカーソルの使用方法の詳細については、880 ページの「ビジーカーソルの使用」を参照してください。

```
<?xml version="1.0"?>
<!-- cursors\CursorManagerApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.managers.CursorManager;
            import flash.events.*;

            // Define a variable to hold the cursor ID.
            private var cursorID:Number = 0;

            // Embed the cursor symbol.
            [Embed(source="assets/wait.jpg")]
            private var waitCursorSymbol:Class;

            // Define event listener to display the wait cursor
            // and to load the image.
            private function initImage(event:MouseEvent):void {
                // Set busy cursor.
                cursorID = CursorManager.setCursor(waitCursorSymbol);
                // Load large image.
                image1.load("assets/DSC00034.JPG");
            }

            // Define an event listener to remove the wait cursor.
            private function loadComplete(event:Event):void {
                CursorManager.removeCursor(cursorID);
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <!-- Image control to load the image. -->
        <mx:Image id="image1" complete="loadComplete(event);"/>

        <!-- Button triggers the load. -->
        <mx:Button id="myButton" label="Show" click="initImage(event);"/>
    </mx:VBox>
</mx:Application>
```

この例では、カーソルイメージに JPEG イメージが使用されています。次の例に示すように、Sprite オブジェクトや、PNG、GIF、SVG、SWF などのファイルも使用できます。

```
[Embed(source="assets/wait.swf")]
var waitCursorSymbol:Class;
```

または、次の例に示すように SWF ファイルからシンボルを参照することもできます。

```
[Embed(source="assets/cursorList.swf", symbol="wait")]
var waitCursorSymbol:Class;
```

SWF ファイルを使用する利点は、アニメーションカーソルを作成できることです。

ビジーカーソルの使用

Flex では、デフォルトのビジーカーソルが定義されています。ビジーカーソルを使用して、アプリケーションが処理中であり、処理が完了してアプリケーションが入力に応答するようになるまで待つ必要があることをユーザーに知らせることができます。デフォルトのビジーカーソルは時計のアニメーションです。

ビジーカーソルは、次に挙げるいくつかの方法で制御できます。

- CursorManager のメソッドを使用して、ビジーカーソルの設定と削除が可能です。
- SWFLoader、WebService、HttpService、および RemoteObject の各クラスの `showBusyCursor` プロパティを使用して、ビジーカーソルを自動的に表示できます。

ビジーカーソルの設定

Cursor Manager に用意されている次の静的メソッドでビジーカーソルを制御します。

メソッド	説明
`setBusyCursor()`	ビジーカーソルを表示します。
`removeBusyCursor()`	カーソルリストからビジーカーソルを削除します。他のビジーカーソル要求がカーソルリスト内でまだアクティブな場合、つまり `setBusyCursor()` メソッドを 2 回以上呼び出していた場合には、ビジーカーソルがリストからすべて削除されるまで、ビジーカーソルは消えません。

879 ページの「カーソルの作成と削除」にある例を修正して、デフォルトのビジーカーソルを使用するようにしたものを、次の例に示します。

```
<?xml version="1.0"?>
<!-- cursors\DefBusyCursorApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.managers.CursorManager;
            import flash.events.*;

            private function initImage(event:MouseEvent):void {
                CursorManager.setBusyCursor();
                image1.load("assets/DSC00034.JPG");
            }

            private function loadComplete(event:Event):void {
                CursorManager.removeBusyCursor();
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <!-- Image control to load the image. -->
        <mx:Image id="image1" complete="loadComplete(event);"/>

        <!-- Button triggers the load. -->
        <mx:Button id="myButton" label="Show" click="initImage(event);"/>
    </mx:VBox>
</mx:Application>
```

ビジーカーソルを設定しても、ユーザーによるアプリケーションの操作に影響はなく、ユーザーはテキストの入力やボタンの選択を行うことができます。ただし、すべてのコンテナは、enabled プロパティをサポートします。デフォルトでは true に設定され、コンテナとその子に対するユーザー操作が可能となっています。あるコンテナの enabled プロパティを false に設定すると、ビジーカーソルが表示されているときは、そのコンテナとそのすべての子のカラーがグレー表示になり、そのコンテナおよび子には入力できなくなります。

また、アプリケーション全体についてユーザー操作を無効にするには、Application.application.enabled プロパティを false に設定します。

ビジーカーソルの優先度は CursorManagerPriority.LOW です。したがって、より高い優先度のカーソルがカーソルリストにある場合は、そのカーソルを削除しない限り、ビジーカーソルは表示されません。デフォルトで高い優先度を持つビジーカーソルを作成するには、次の例に示すように setCursor() メソッドを使用します。

```
<?xml version="1.0"?>
<!-- cursors\ShowBusyCursorAppHighP.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:Script>
        <![CDATA[
            import mx.managers.CursorManager;
            import mx.managers.CursorManagerPriority;
            import flash.events.*;

            // Define a variable to hold the cursor ID.
            private var cursorID:Number = 0;

            // Define event listener to display the busy cursor
            // and to load the image.
            private function initImage(event:MouseEvent):void {
                // Set busy cursor.
                cursorID=CursorManager.setCursor(
StyleManager.getStyleDeclaration("CursorManager").getStyle("busyCursor"),
CursorManagerPriority.HIGH);
                // Load large image.
                image1.load("assets/DSC00034.JPG");
            }

            // Define an event listener to remove the wait cursor.
            private function loadComplete(event:Event):void {
                CursorManager.removeCursor(cursorID);
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <!-- Image control to load the image. -->
        <mx:Image id="image1" complete="loadComplete(event);"/>

        <!-- Button triggers the load. -->
        <mx:Button id="myButton" label="Show" click="initImage(event);"/>
    </mx:VBox>
</mx:Application>
cursorID=CursorManager.setCursor(StyleManager.getStyleDeclaration
  ("CursorManager").getStyle("busyCursor"), CursorManagerPriority.HIGH);
```

このステートメントは、StyleManager クラスの静的な getStyleDeclaration() メソッドを使用して Cursor Manager の CSStyleDeclaration オブジェクトを取得し、このオブジェクトの getStyle() メソッドを使用してビジーカーソルを取得します。これにより、ビジーカーソルに高い優先度を設定します。

この方法を使用するときは、removeCursor() メソッドでカーソル ID を使用してビジーカーソルを削除することも必要です。

showBusyCursor プロパティの使用

SWFLoader コントロール、および <mx:WebService>、<mx:HttpService>、<mx:RemoteObject> の各タグには、showBusyCursor プロパティがあります。このプロパティを使用すると、該当のクラスでデータのロードが完了するまで自動的にデフォルトのビジーカーソルが表示されます。デフォルト値は、SWFLoader コントロールでも、<mx:WebService> タグ、<mx:HttpService> タグ、および <mx:RemoteObject> タグでも false です。

SWFLoader コントロールの場合、showBusyCursor プロパティを true に設定すると、コントロールの最初の progress イベントがトリガされたときにビジーカーソルが表示され、complete イベントがトリガされたときにビジーカーソルが非表示になります。次の例では、showBusyCursor プロパティを使用して、880 ページの「ビジーカーソルの設定」で挙げた例を簡素化する方法を示しています。

```
<?xml version="1.0"?>
<!-- cursors\ShowBusyCursorApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:VBox>
        <!-- SWFLoader control to load the image. -->
        <mx:SWFLoader id="image1" showBusyCursor="true"/>

        <!-- Button triggers the load. -->
        <mx:Button id="myButton" label="Show"
            click="image1.load('assets/DSC00034.jpg');"/>
    </mx:VBox>
</mx:Application>
```

第 25 章

Flex アプリケーションのローカライズ

25

Adobe Flex では、Flex フレームワークとカスタムコンポーネントおよびクラスのローカリゼーションがサポートされます。

内容

ローカライズした Flex アプリケーションについて

Flex を使用すると、ローカライズした Flex アプリケーションを作成し、ローカライズしたコンポーネントにアクセスすることができます。Flex は、ローカライズしたリソースの静的組み込みはサポートしていますが、実行時におけるリソースの動的取得はサポートしていません。

リソースバンドルとプロパティファイルについて

リソースバンドル、すなわちプロパティファイルでは、単純な "key=value" シンタックスを使用してキーと値をマッピングします。これは Java プロパティファイルと同様のものです。Java プロパティフォーマットとの主な違いは、Flex プロパティファイルは ASCII 以外の文字が含まれている場合に UTF-8 として保存される点です。

Flex コンパイラは、他のソースファイルを検索するのと同じ方法でプロパティファイルを検索します。プロパティファイルは、コンパイラが検索するディレクトリに置いてください。

リソースバンドル内の値には、次のようにアクセスできます。

MXML MXML タグ内で `@Resource` ディレクティブを使用します。

ActionScript `[ResourceBundle]` メタデータタグを使用して、ActionScript で ResourceBundle クラスのインスタンスを作成します。

場合によっては、内部のデータにアクセスするためにリソースバンドルの名前も指定する必要があります。コード内では、リソースバンドルにはプロパティファイルと同じ名前が使用されます。

ローカリゼーションのワークフロー

アプリケーションをローカライズするには、ResourceBundle API を使用します。それには、まず ActionScript に [ResourceBundle] メタデータタグを追加するか、MXML に @Resource 呼び出しを追加します。次にローカライズしたプロパティファイルとローカリゼーションクラスを作成して、ロケールディレクトリに入れます。最後に mxmlc コンパイラでメインアプリケーションをコンパイルします。

オプションで、ローカライズしたプロパティファイルと ActionScript クラスのライブラリを、compc コンパイラで作成した SWC ファイルにパッケージ化することができます。ただし、リソースバンドルの SWC ファイルが必要なのは、アプリケーションではなくライブラリを作成するときだけです。

mxmlc ユーティリティと compc ユーティリティの詳細については、『Flex 2 アプリケーションの構築とデプロイ』の第 9 章の「Flex コンパイラの使用」を参照してください。

ローカライズしたアプリケーションの作成

このセクションでは、ローカライズした簡単なサンプルアプリケーションの作成を始めます。ローカライズしたアプリケーションの作成に関する重要な要素について説明します。

ローカライズした簡単なアプリケーションを構築するには :

1. 次のテキストを含む "locale/en_US/RegistrationForm.properties" ファイルを作成します。

```
registration_title=Registration
submit=Submit Form
personname=Name
street_address=Street Address
city=City
state=State!
zip=ZIP Code
thanks=Thank you for registering!
subtext=[0] alphabetically comes before [1]
```

ロケールディレクトリの場所は重要ではありませんが、その場所をコンパイラのソースパスに追加できる必要があります。

2. 次のテキストを含む "locale/es_ES/RegistrationForm.properties" ファイルを作成します。このファイルは、必ず UTF-8 エンコーディングで保存してください。

```
registration_title=Registro
submit=Someta La Forma
personname=Nombre
street_address=Direcci 溶 De la Calle
city=Ciudad
state=Estado
zip=C 妖 igo postal
thanks=¡Gracias por colocarse!
subtext=[0] viene alfab 師 icamente antes [1]
```

3. 次のコードを含む LocalizedForm.mxml アプリケーションを作成します。

```
<?xml version="1.0"?>
<!-- l10n/LocalizedForm.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.resources.ResourceBundle;
     import mx.controls.Alert;

     [ResourceBundle("RegistrationForm")]
     private static var rb:ResourceBundle;

     private function registrationComplete():void {
        Alert.show(rb.getString('thanks'));
     }

  ]]></mx:Script>

  <mx:Form>
     <mx:FormItem label="@Resource(key='personname',
bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='street_address',
bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='city',
bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='state',
bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='zip',
bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
  </mx:Form>
  <mx:Button id="b1" label="@Resource(key='submit',
bundle='RegistrationForm')" click="registrationComplete()"/>

</mx:Application>
```

4. 次の mxmlc コマンドラインを使用して、en_US リソースバンドルとともにアプリケーションをコンパイルします。

```
mxmlc -locale en_US source-path+=path_to_locale_dir/{locale}
  -o Form-us.swf LocalizedForm.mxml
```

5. 次の mxmlc コマンドラインを使用して、en_ES リソースバンドルとともにアプリケーションをコンパイルします。

```
mxmlc -locale es_US source-path+=path_to_locale_dir/{locale}
   -o Form-es.swf LocalizedForm.mxml
```

6. 2 つの SWF ファイルを実行して、各アプリケーションで異なるフォームラベルと警告を確認します。

ResourceBundle API の使用

Flex のコードをローカライズする最初の手順は、mx.resources.ResourceBundle API を使用して、アプリケーション内でローカライズする必要のあるハードコードされた内容をすべて置き換えることです。ResourceBundle API は ActionScript コードまたは MXML コードで使用できます。

ローカライズする内容はすべて、メインアプリケーションから削除する必要があります。ActionScript の [ResourceBundle] メタデータタグか、MXML の @Resource ディレクティブでリソースバンドルにアクセスすることで、内容の呼び出しを実装します。

ResourceBundle メタデータタグの使用

[ResourceBundle] メタデータは、mx.resources.ResourceBundle タイプの変数の真上に配置する必要があります。次に ActionScript コードの例を示します。

```
...
[ResourceBundle("bundlename")]
private static var rb:ResourceBundle;
...
```

bundlename は、ローカライズしたデータが格納されたコンパイラのソースパス内のプロパティファイルを参照します。new キーワードでは、ActionScript に ResourceBundle インスタンスを作成できません。必ず [ResourceBundle] メタデータタグを使用してください。

コンパイラによって変数のインスタンスが作成されます。コードでは、ResourceBundle API を使用してリソースバンドルのキーにアクセスできます。この API には、ResourceBundle.getString() および ResourceBundle.getObject() メソッドが含まれます。次の例では、[ResourceBundle] メタデータタグを使用し、getString() メソッドを使用してリソースバンドルの内容にアクセスします。

```
<?xml version="1.0"?>
<!-- l10n/LocalizedFormGetString.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.resources.ResourceBundle;
     import mx.controls.Alert;

     [Bindable]
     [ResourceBundle("RegistrationForm")]
     private var rb:ResourceBundle;
```

```
    private function registrationComplete():void {
       Alert.show(rb.getString('thanks'));
    }

]]></mx:Script>

  <mx:Button id="b1" label="{rb.getString('submit')}"
click="registrationComplete()"/>

</mx:Application>
```

getObject() メソッドを使用してオブジェクトにアクセスする方法については、890ページの「ローカライズした ActionScript クラスの使用」を参照してください。

@Resource ディレクティブの使用

[ResourceBundle] メタデータタグの代わりに、MXML タグで @Resource ディレクティブを使用することができます。@Resource ディレクティブにリソースバンドル名とキーを指定して、リソースバンドルから値を取得します。キーのみを指定すると、現在のクラス名がバンドル名として使用されます。bundlename 名は、キーを含むプロパティファイルまたは ResourceBundle サブクラスの名前です。@Resource ディレクティブには次の 2 つの形式があります。

- @Resource("<キー>")
- @Resource(bundle="*bundlename*", key="key")

bundlename は、ローカライズしたデータが格納されたコンパイラのソースパス内のプロパティファイルを参照します。

次の例では、@Resource ディレクティブを使用してリソースバンドルのキーにアクセスしています。

```
<?xml version="1.0"?>
<!-- l10n/LocalizedFormResourceDirective.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Form>
     <mx:FormItem label="@Resource(key='personname',
bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='street_address',
bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='city', bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='state', bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
     <mx:FormItem label="@Resource(key='zip', bundle='RegistrationForm')">
        <mx:TextInput/>
     </mx:FormItem>
  </mx:Form>
```

```
  <mx:Button id="b1" label="@Resource(key='submit', bundle='RegistrationForm')"/
>

</mx:Application>
```

ローカライズしたストリング値の置換

他のストリング値を同様に、ローカライズしたストリング値でストリング置換を使用できます。たとえば、次のキーを含んでいるプロパティファイルもあります。

```
subtext=[0] alphabetically comes before [1]
```

次の例では、StringUtil.substitute() メソッドを使用して、ローカライズしたストリングに ActionScript で値を挿入します。

```
<?xml version="1.0"?>
<!-- l10n/StringSubstitution.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.resources.ResourceBundle;
     import mx.utils.StringUtil;

     [Bindable]
     [ResourceBundle("RegistrationForm")]
     private var rb:ResourceBundle;

     [Bindable]
     private var s:String;

     private function initApp():void {
        s = rb.getString('subtext');
        s = StringUtil.substitute(s, "Anant", "Nick");
     }
  ]]></mx:Script>

  <mx:Label id="l1" text="{s}"/>

</mx:Application>
```

ローカライズした ActionScript クラスの使用

任意の ActionScript クラスをローカライズしたリソースとして使用できます。たとえば、ロケールごとに 2 つのクラスを使用して、ローカライズしたタスクを Flex アプリケーションで実行することができます。リソースバンドルのストリングのリポジトリとして機能するプロパティファイルを作成するのではなく、リソースバンドル内のすべてのクラスを参照するクラスを作成します。プロパティファイルからストリングを抽出する場合と同様に、この新しいクラスからローカライズしたクラスへの参照を抽出します。

ローカライズしたクラスを使用するには、次の手順に従う必要があります。

1. [ResourceBundle] メタデータタグを使用して、Flex アプリケーションで ResourceBundle を埋め込みます。
2. Flex アプリケーションで、ResourceBundle.getObject() メソッドを使用します。
3. すべてのローカライズしたクラスリソースを参照する Object を返すバンドルクラスを作成します。
4. ローカライズした ActionScript クラスを locale/{locale} ディレクトリに格納します。

リソースバンドルに含めることで、イメージやサウンドファイルなどのローカライズした埋め込みアセットを使用することもできます。これを行うには、このセクションで説明する手順、および 893 ページの「埋め込みアセットの使用」で説明する手順をすべて実行します。

getObject() メソッドの使用

ローカライズしたオブジェクトや、イメージなどの埋め込まれたアセットをアプリケーションで使用するには、[ResourceBundle] メタデータタグを使用して、リソースバンドルへの参照を取得します。そして、次の例に示すように、ResourceBundle.getObject() メソッドを呼び出して、リソースバンドルから特定のクラスを取得します。

```
<?xml version="1.0"?>
<!-- l10n/SimpleClassApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">

  <mx:Script><![CDATA[
     import mx.resources.ResourceBundle;

     [Bindable]
     [ResourceBundle("MyObjectClassesBundle")]
     private var rb:ResourceBundle;

     private function initApp():void {
        var scClass:Class = Class(rb.getObject("SimpleClass"));
        var sc:SimpleClass = new scClass();

        b1.label = sc.returnString();
     }

  ]]></mx:Script>

  <mx:Button id="b1"/>

</mx:Application>
```

この例では、ローカライズした ActionScript クラスを埋め込むために作成する ResourceBundle サブクラスの名前は MyObjectClassesBundle です。

バンドルクラスの作成

埋め込まれたアセットを使用するには、バンドルクラスを使用する必要があります。このクラスは、ResourceBundle サブクラスを拡張する必要があります。通常は、`super()` メソッドを呼び出すだけのコンストラクタを含めます。このクラスは、`getContent()` メソッドでオブジェクトも返します。このオブジェクトは、ResourceBundle で必要なすべてのクラスを参照します。

サポートされるロケールごとにこのクラスの 1 つのバージョンを作成し、locale/{locale} ディレクトリに格納します。

次の例に、MyObjectClassesBundle クラスを示します。このクラスはローカライズした ActionScript クラスを読み込み、返されるコンテンツオブジェクトにそれらへの参照を追加します。

```
package {
import mx.resources.ResourceBundle;
import MyBird;
import SimpleClass;

public class MyObjectClassesBundle extends ResourceBundle {
  public function MyObjectClassesBundle() {
     super();
  }

  override protected function getContent():Object {
     var contentObj:Object = new Object();

     // Add a reference to the localized ActionScript class.
     contentObj["SimpleClass"] = SimpleClass;

     return contentObj;
  }
}
}
```

この例では、ローカライズした ActionScript クラスは、ストリングを返すメソッドのある簡単なクラスです。

```
package {

  public class SimpleClass {

     // Constructor
     public function SimpleClass() {
     }

     public function returnString():String {
        return "hello from SimpleClass";
     }
  }

}
```

ローカライズした ActionScript クラスファイルは、locale/{locale} ディレクトリ、またはバンドルクラスがアクセスできる任意のディレクトリに格納できます。

埋め込みアセットの使用

リソースバンドルで埋め込みクラスを使用できます。これらの埋め込みアセットには、イメージ、サウンドファイル、または Flex アプリケーションで埋め込み可能な任意の種類のファイルを含めることができます。

埋め込みアセットをリソースバンドルに含めるときは、埋め込むアセットまたはオブジェクトごとに ActionScript ラッパークラスを作成する必要があります。ローカライズした埋め込みアセットごとに、ラッパークラスを locale/{locale} ディレクトリに追加します。

ラッパークラスはアセットの種類に一致するクラスを拡張します。イメージの場合、ラッパークラスは通常は BitmapAsset を拡張します。リソースバンドルで使用できるその他のアセットタイプ、およびそれらの関連クラスの詳細については、1020 ページの「各種アセットの埋め込み」を参照してください。

ラッパークラスには、空のコンストラクタと [Embed] メタデータタグだけを含める必要があります。このクラスでは、@Embed ディレクティブではなく、Embed のクラスベースのバージョンを使用する必要があります。

たとえば、GIF イメージをローカライズするには、次の MyBird クラスを作成します。

```
package {
  import mx.core.BitmapAsset;
  [Embed(source='bird.gif')]
  public class MyBird extends BitmapAsset {
     public function MyBird() {
        // Empty constructor.
     }
  }
}
```

バンドルクラスは、ローカライズした ActionScript クラスを使用するために作成したクラスと同じです。892 ページの「バンドルクラスの作成」を参照してください。

```
package {
import mx.resources.ResourceBundle;
import MyBird;
import SimpleClass;

public class MyEmbeddedClassesBundle extends ResourceBundle {
  public function MyEmbeddedClassesBundle() {
     super();
  }

  override protected function getContent():Object {
     var contentObj:Object = new Object();

     // Add a reference to the embedded graphic asset.
     contentObj["MyBird"] = MyBird;

     // Add a reference to the localized ActionScript class.
     contentObj["SimpleClass"] = SimpleClass;
```

```
        return contentObj;
    }
}
}
```

Flex アプリケーションで、ローカライズした ActionScript クラスを使用したときと同じように、ResourceBundle のインスタンスを作成します。埋め込みアセットを取得するために、getObject() メソッドも使用します。次の例では、イメージクラス MyBird を取得します。

```
<?xml version="1.0"?>
<!-- l10n/EmbeddedAssetApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.resources.ResourceBundle;

     [Bindable]
     [ResourceBundle("MyEmbeddedClassesBundle")]
     private var rb2:ResourceBundle;

     [Bindable]
     [ResourceBundle("MyObjectClassesBundle")]
     private var rb:ResourceBundle;

     [Bindable]
     private var bird:Object = rb2.getObject("MyBird");

     private function initApp():void {
        var sc:Object = rb.getObject("SimpleClass");
        b1.label = sc.returnString();
     }

  ]]></mx:Script>

  <mx:Panel id="panel1" title="Image from Resource Bundle">
     <mx:Image id="image1" source="{bird}"/>
  </mx:Panel>

  <mx:Button id="b1"/>

</mx:Application>
```

実行時ロードへの変換を容易にするために、ローカライズしたクラスへの直接参照は作成しないでください。次の例のように、直接参照の代わりに、ActionScript で可視の子を追加する必要があります。

```
<?xml version="1.0"?>
<!-- l10n/EmbeddedAssetAppAddChild.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.resources.ResourceBundle;
     import mx.controls.Image;

     [Bindable]
     [ResourceBundle("MyEmbeddedClassesBundle")]
```

```
        private var rb2:ResourceBundle;

        [Bindable]
        [ResourceBundle("MyObjectClassesBundle")]
        private var rb:ResourceBundle;

        [Bindable]
        private var bird:Object = rb2.getObject("MyBird");

        private function initApp():void {
            var sc:Object = rb.getObject("SimpleClass");
            b1.label = sc.returnString();

            var i:Image = new Image();
            i.source = bird;
            panel1.addChild(i);
        }

    ]]></mx:Script>

    <mx:Panel id="panel1" title="Image from Resource Bundle">
    </mx:Panel>

    <mx:Button id="b1"/>

</mx:Application>
```

ローカライズしたプロパティファイルと ResourceBundle サブクラスの使用

ローカライズしたプロパティファイルと ResourceBundle サブクラスは、いくつかの方法で使用できます。

プロパティファイルのフォーマットとディレクトリ

プロパティファイルにローカライズするストリングプロパティを、ユーザー作成のロケールディレクトリに配置します。クラスごとにプロパティファイルを 1 つずつ作成することも、統合したキーと値を 1 つのファイルに配置することもできます。またこの 2 つの方法を組み合わせることもできます。

プロパティファイルに対応するクラスと同じ名前を指定すると、`@Resource` MXML ディレクティブの bundle パラメータを指定する必要はなくなりますが、bundle パラメータは指定することをお勧めします。bundle パラメータを指定しない場合、ルートディレクトリがロケール名と同一でない限り、プロパティファイルの場所はクラスと同じでなければなりません。たとえば、/myApp/MyAlert.as のプロパティファイルは、/fr_FR/myApp/MyAlert.properties に格納されています。この場合の fr_FR はロケールです。同様に、同一のプロパティファイルを、複数のロケールディレクトリに入れることもできます。compc の `locale` オプションを使用すると、ロケールを設定できます。実行時に Locale クラスを使用しても、ロケールは設定できません。

ロケールディレクトリは、compc の ActionScript ソースパスに組み込みますが、メインの SWF ファイルの構築時に mxmlc コマンドラインコンパイラが使用する ActionScript ソースパスには組み込みません。

ローカリゼーションプロパティファイルは、Java のプロパティファイルのフォーマットに従います。Java プロパティファイルは、キーと値を格納する単純なテキストフォーマットです。次の例は、プロパティファイルに格納されているキーと値のテキストストリングを示しています。

```
registration_title=Registration
submit=Submit Form
personname=Name
street_address=Street Address
city=City
state=State!
zip=ZIP Code
thanks=Thank you for registering!
subtext=[0] alphabetically comes before [1]
```

プロパティファイル内のストリングはすべて、Latin-1 または UTF-8 でエンコードされている必要があります。

ローカライズしたプロパティファイルとクラスの検索順序

ローカライズしたプロパティファイルと ResourceBundle サブクラスを、ロケール名に対応するディレクトリに追加します。ローカライズした ActionScript クラスの詳細については、890 ページの「ローカライズした ActionScript クラスの使用」を参照してください。

SWC ファイルをコンパイルする場合、mxmlc コンパイラまたは compc コンパイラは一般的なロケール規則に従ってファイルを検索します。つまり、正確なロケール名、変数と国名なしのロケール名、en_EN の順に検索します。

次の表では、mxmlc または compc のコマンドラインでロケールを fr_FR に設定した場合のプロパティファイルの検索順を示します。

ローカライズしたファイル	説明
/en_EN/Grape.as	"fr_FR" ディレクトリに同一ファイルの他のバージョンがあり、そちらの方が検索順序が高いため、このファイルは使用されません。
/en_EN/AppCommon.properties	このファイルよりも検索順序の高いバージョンはないため、このファイルが使用されます。
/fr/Grape.as	"fr_FR" ディレクトリに同一ファイルの他のバージョンがあり、そちらの方が検索順序が高いため、このファイルは使用されません。
/fr/Apple.as	このファイルよりも検索順序の高いバージョンはないため、このファイルが使用されます。

ローカライズしたファイル	説明
/fr_FR/Grape.as	このファイルが検索順位の最も上にあるので、このファイルが使用されます。
/fr_FR/Orange.as	このファイルが検索順位の最も上にあるので、このファイルが使用されます。

Flex ソフトウェア開発キットのリソースバンドルのローカライズ

独自のコードのローカライズと同じ方法で、Flex ソフトウェア開発キット (SDK) のリソースバンドルもローカライズできます。en_EN ロケールの Flex 2 SDK プロパティファイルは、Flex Builder インストールの "Flex Framework 2/frameworks/locale/en_EN" ディレクトリにあります。これらは、Adobe が Flex 2 SDK のデフォルト SWC ファイルである "framework_rb.swc" を作成するときに使用するプロパティファイルです。"en_EN" ディレクトリには、"SharedResources.properties" という名前のプロパティファイルが入っています。このファイルには、日付、バリデータ、フォーマッタに使用されるような Flex フレームワークの共有ストリングやシンボルのキーや値が入っています。フレームワークプロパティは、"framework_rb.swc" ファイルにコンパイルされます。独自の "framework_rb.swc" ファイルは、"build_framework.xml" Ant ファイルで作成できます。

アプリケーションの SWF ファイルの作成

アプリケーションの SWF ファイルは、mxmlc コンパイラで作成します。SWF ファイルはロケールごとに1つずつ作成する必要があります。次の例は、ローカライズした "HelloWorld.swf" ファイルを en_EN ロケールに作成する mxmlc のコマンドラインです。

```
mxmlc -locale en_EN -source-path locale/{locale} HelloWorld.mxml
```

Flex Data Services を使用してアプリケーションをコンパイルする場合は、次の例に示すように、flex-config.xml ファイル内の <locale> オプションのコメントを解除してロケールを指定します。

```
<!-- Specifies the locale for internationalization. -->
<locale>en_US</locale>
```

次に、ロケールトークンを使用して、ロケールファイルの場所を <source-path> に追加できます。

```
<!-- list of path elements that form the roots of ActionScript class hierarchies
   -->
<source-path>
  <path-element>./user_classes</path-element>
  <path-element>./locale/{locale}</path-element>
</source-path>
```

ローカライズした SWF ファイルを作成するには、次の mxmlc オプションを使用します。

mxmlc オプション	内容
`library-path` *`path-element [...]`*	生成されるアプリケーションの SWF ファイルに SWC ファイルをリンクします。コンパイラでは、SWC ファイル用の必須のクラスにしかリンクしません。このオプションは、ロケール名が入っている箇所に `{locale}` トークンを使用して指定します。 `library-path` オプションのデフォルト値には "libs" ディレクトリ内のすべての SWC ファイルと "framework_rb.swc" ファイルが含まれます。 SWC ファイル全体ではなく個々のクラスまたはパッケージを指定する場合は、`source-path` オプションを使用します。 `library-path` の値をコマンドラインコンパイラのオプションとして設定する場合は、"framework.swc " ファイルと "framework_rb.swc" ファイルも明示的に追加する必要があります。新しいエントリは `library-path` に追加されるのではなく、これで置き換えられます。 += 演算子を使用して、既存の SWC ファイルのリストに新しいパラメータを追加します。
`locale` *`string`*	SWF ファイルにパッケージ化されているロケールを指定します。ロケールが複数ある場合は、mxmlc コンパイラを複数回実行して SWF ファイルを作成し、`locale` および `output` オプションだけを変更します。
`output` *`string`*	生成するファイルの出力先パスとファイル名を指定します。このオプションを省略すると、ターゲットファイルの格納ディレクトリに SWF ファイルが保存されます。 デフォルトの SWF ファイル名はターゲットファイルの名前に SWF ファイル拡張子を付けたものです。 相対パスで *ファイル名*を定義する場合は、必ず、ターゲット MXML アプリケーションルートではなく現在の作業ディレクトリに相対させたパスを用います。 該当するディレクトリが存在しない場合は、コンパイラにより、指定されたファイル名に基づく新しいディレクトリが作成されます。
`source-path` *`path-element [...]`*	ユーザー作成のローカリゼーションディレクトリを指定します。このオプションは、ロケール名が入っている箇所に `{locale}` トークンを使用して指定します。たとえば、/myfiles/locale_dir/en_EN と /myfiles/locale_dir/ja_JP にファイルがある場合、`source-path` 値に /myfiles/locale_dir/{*`locale`*} と指定します。

mxmlc コンパイラの詳細については、『Flex 2 アプリケーションの構築とデプロイ』の第 9 章の「Flex コンパイラの使用」を参照してください。

ローカライズした SWC ファイルの作成

ローカライズしたプロパティファイルと ActionScript クラスのライブラリを、compc コンパイラで作成した SWC ファイルにパッケージ化できます。ただし、SWC ファイルはメインアプリケーションとは別になっています。ライブラリを構築していない場合は、ローカライズしたプロパティファイルに SWC ファイルを作成する必要はありません。

ローカライズした SWC ファイルを作成するには、次の compc オプションを使用します。

compc オプション	内容
`source-path`	ユーザー作成のローカリゼーションディレクトリを指定します。このオプションは、ロケール名が入っている箇所に {locale} トークンを使用して指定します。たとえば、/myfiles/locale_dir/en_EN と /myfiles/locale_dir/ja_JP にファイルがある場合、`source-path` 値に /myfiles/locale_dir/{locale} と指定します。
`include-resource-bundles`	ローカライズした SWC ファイルに含めるリソースバンドルを指定します。このリストには、プロパティファイルまたは ResourceBundle サブクラスの名前が入ります。ただしファイル名には接尾辞や基本のファイルパスは含まれません。compc でライブラリを構築するときに、`resource-bundle` オプションを使用すると、リソースバンドルの一覧を取得できます。
`library-path` *`path-element [...]`*	生成されるアプリケーションの SWF ファイルに SWC ファイルをリンクします。コンパイラでは、SWC ファイル用の必須のクラスにしかリンクしません。このオプションは、ロケール名が入っている箇所に {locale} トークンを使用して指定できます。`library-path` オプションのデフォルト値には "libs" ディレクトリ内のすべての SWC ファイルと "framework_rb.swc" ファイルが含まれます。 SWC ファイル全体ではなく個々のクラスまたはパッケージを指定する場合は、`source-path` オプションを使用します。 `library-path` の値をコマンドラインコンパイラのオプションとして設定する場合は、"framework.swc " ファイルと "framework_rb.swc" ファイルも明示的に追加する必要があります。新しいエントリは `library-path` に追加されるのではなく、これで置き換えられます。 += 演算子を使用すると、既存の SWC ファイルのリストに新しいパラメータを追加することができます。
`output` *`filename`*	SWC ファイルの完全修飾名を指定します。SWC ファイルを作成する場合は、このオプションを必ず指定する必要があります。

compc オプション	内容
`locale` *`string`*	SWC ファイルにパッケージ化されているロケールを指定します。ロケールが複数ある場合は、compc コンパイラを複数回実行して SWC ファイルを作成し、`locale` および `output` オプションだけを変更します。
`resource-bundle-list` *`filename`*	リソースバンドルの SWC ファイルを生成するために compc コンパイラに入力するリソースバンドルのリストを出力します。リソースバンドルの SWC ファイルを作成する場合は、このオプションを使用しないでください。*`filename`* パラメータはバンドルのリストが含まれたファイルの名前です。

次の例は、fr_FR リソースバンドルを使用して、ローカライズした SWC ファイルを作成する compc コマンドラインです。

```
compc -locale fr_FR -source-path locale/{locale} -include-resource-bundles
  HelloWorld -output locale/fr_FR/HelloWorld.swc
```

compc コンパイラの詳細については、『Flex 2 アプリケーションの構築とデプロイ』の第 9 章の「Flex コンパイラの使用」を参照してください。

第 4 部

Flex プログラミングに関するトピック

4

第IV部では、よりインタラクティブで表現力の高いアプリケーションを作成するための高度なプログラミング手法を紹介します。

次のトピックが含まれます。

第 26 章

コントロールおよびコンテナの動的な繰り返し

26

このトピックでは、Repeater コンポーネントの使用方法について説明します。Repeater コンポーネントを使用することによって、実行時に MXML コンテンツを動的に繰り返すことできます。

Flex では、リストコントロールもサポートしています。大量のデータを表示するときは、リストコントロールを使用するとパフォーマンスが向上します。HorizontalList コントロール、TileList コントロール、および List コントロールの詳細については、411 ページ、第 12 章の「データ駆動型コントロールの使用」を参照してください。

目次

Repeater コンポーネントについて

Repeater コンポーネントは、単純なユーザーインターフェイスコンポーネントの小さなセットを繰り返すのに便利です。たとえば、通常 Form コンテナで使用される RadioButton コントロールやその他のコントロールなどの繰り返しに使用します。通常、繰り返しは、Web サービスから返された Array オブジェクトなど、動的なデータの配列によって制御されますが、静的配列を使用して単純な `for` ループをエミュレートできます。

Repeater コンポーネントはコード内でコンテナのように見えますが、実際にはコンテナではなく、コンテナのような自動レイアウト機能はありません。Repeater コンポーネントを使用する唯一の目的は、指定されたデータプロバイダのコンテンツに基づいてアプリケーションに 1 回以上含める一連のサブコンポーネントを指定することです。Repeater が生成したアイテムを整列させる場合、またはその他のレイアウトタスクを実行する場合は、Repeater コンポーネントとそのコンテンツをコンテナ内に配置し、コンテナにレイアウトを適用します。

Flex では、HorizontalList コントロール、TileList コントロール、または List コントロールを使用して、大量のデータを表示するときのパフォーマンスを向上させることができます。Repeater コンポーネントが、繰り返されているすべてのオブジェクトをインスタンス化するのに対して、HorizontalList コントロール、TileList コントロール、および List コントロールは、リストに表示されているオブジェクトのみをインスタンス化します。データが 1 つの画面内に表示されない場合や、コンテナ内の表示スペースに収まらない場合は、リストコントロールのいずれかを使用してください。

HorizontalList コントロールは、HBox コンテナのようにデータを水平方向に表示するリストコントロールです。HorizontalList コントロールでは、アイテムは常に左から右に表示されます。詳細については、421 ページの「HorizontalList コントロール」を参照してください。

TileList コントロールは、Tile コンテナのようにデータをタイル状に並べて表示するリストコントロールです。TileList コントロールには、次のアイテムを下方向または右方向のどちらに配置するかを決定する `direction` プロパティがあります。詳細については、425 ページの「TileList コントロール」を参照してください。

List コントロールは、データを 1 列の垂直列に並べます。詳細については、412 ページの「List コントロール」を参照してください。

Repeater コンポーネントの使用

`<mx:Repeater>` タグは、Repeater コンポーネントを宣言するために使用します。Repeater コンポーネントは、実行時に動的または静的なデータ配列に基づいて 1 つまたは複数のユーザーインターフェイスコンポーネントの繰り返しを処理します。繰り返す対象のコンポーネントには、コントロールまたはコンテナを指定できます。Repeater コンポーネントを使用するには、データバインディングを行って、ランタイム固有の値を記述できるようにする必要があります。データバインディングの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

`<mx:Repeater>` タグは、コントロールまたはコンテナタグを使用できるあらゆる場所で使うことができます。ユーザーインターフェイスコンポーネントを繰り返すには、対応するタグを `<mx:Repeater>` タグ内に配置します。UIComponent クラスから派生したすべてのコンポーネントを繰り返すことができます (`<mx:Application>` コンテナタグを除く)。さらに、1 つの MXML ドキュメント内で複数の `<mx:Repeater>` タグを使用したり、`<mx:Repeater>` タグをネストすることもできます。

MXML での Repeater コンポーネントの宣言

Repeater コンポーネントは、<mx:Repeater> タグ内で宣言します。次の表に、Repeater コンポーネントのプロパティとその説明を示します。

次の表に、Repeater コンポーネントのイベントとその説明を示します。

プロパティ	説明
id	対応する Repeater コンポーネントのインスタンス名です。
dataProvider	ICollectionView インターフェイス、IList インターフェイス、または Array クラス (ArrayCollection オブジェクトなど) の実装です。 Repeater コンポーネントを実行するには、dataProvider の値を指定する必要があります。 dataProvider プロパティの値は実行時まで不明なので、通常はこの値をバインディング式として指定します。
startingIndex	繰り返しを開始するデータプロバイダのエレメントを指定する数値です。データプロバイダの配列は 0 から始まるので、配列の 2 番目のエレメントから開始する場合は、開始インデックスを 1 に指定します。startingIndex が dataProvider プロパティの範囲外の場合、繰り返しは発生しません。
count	繰り返しの回数を指定する数値です。dataProvider プロパティのアイテム数が count プロパティより少ない場合は、最後のアイテムで繰り返しが停止します。
currentIndex	現在処理中の dataProvider アイテムのエレメントを示す数値です。データプロバイダの配列は 0 から始まるので、配列の 3 番目のエレメントが処理中の場合、現在のインデックスは 2 です。このプロパティは、Repeater コンポーネントの実行に従って変化し、実行が完了した後の値は undefined になります。このプロパティは読み取り専用のプロパティであり、<mx:Repeater> タグ内で設定することはできません。
currentItem	dataProvider プロパティ内で処理中のアイテムへの参照です。このプロパティは、Repeater コンポーネントの実行に従って変化し、実行が完了した後の値は undefined になります。このプロパティは読み取り専用のプロパティであり、<mx:Repeater> タグ内で設定することはできません。 Repeater コンポーネントの繰り返しが完了した後は、currentItem プロパティを使用して現在のアイテムを取得することはありません。代わりに、繰り返されるコンポーネント自体の getRepeaterItem() メソッドを呼び出します。詳細については、914 ページの「Repeater コンポーネントのイベントハンドラ」を参照してください。
recycleChildren	ブール値です。true に設定した場合は、新しいデータアイテムが既存の Repeater の子にバインドされます。このとき、データアイテムの数が既存の子よりも多ければ、不足分の子が順番に新しく作成される一方で、必要なくなった余分な子は破棄されます。詳細については、922 ページの「Repeater コンポーネント内での子の再作成」を参照してください。

イベント	説明
repeat	アイテムが処理され、currentIndex および currentItem が更新されるたびに送出されます。
repeatEnd	Repeater のすべてのサブコンポーネントが作成された後に送出されます。
repeatStart	Flex が dataProvider プロパティの処理を開始し、指定されたサブコンポーネントの作成を開始したときに送出されます。

Repeater コンポーネントの基本原則

Repeater コンポーネントを使用して作成できる最も単純な繰り返しは、一定の回数実行される静的ループです。次の Repeater コンポーネントは、4 回実行される単純な for ループをエミュレートしています。短いテキストを印刷し、1回の実行ごとにカウンタがインクリメントします。

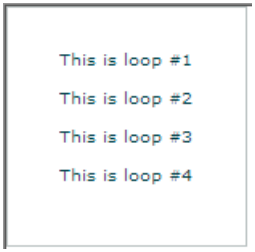
```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*">

  <mx:Script>
    <![CDATA[
      [Bindable]
      public var myArray:Array=[1,2,3,4];
    ]]>
  </mx:Script>

  <mx:ArrayCollection id="myAC" source="{myArray}"/>
  <mx:Repeater id="myrep" dataProvider="{myAC}">
    <mx:Label id="Label1" text="This is loop #{myrep.currentItem}"/>
  </mx:Repeater>
</mx:Application>
```

カウンタは、実際には配列内のデータではなく、配列内の位置です。配列内にエレメントが 4 つが存在していれば、配列内にどのようなデータでも含めることができ、Repeater コンポーネントが 4 回実行されます。この原則について、次の例で説明します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            [Bindable]
            public var myArray:Array=[10,20,30,40];
        ]]>
    </mx:Script>

    <mx:ArrayCollection id="myAC" source="{myArray}"/>

    <mx:Repeater id="myrep" dataProvider="{myAC}">
        <mx:Label id="Label1" text="This is loop #{myrep.currentIndex}+1"/>
  </mx:Repeater>
</mx:Application>
```

この例で印刷するのは、インデックスそのものではなく、現在のインデックスに 1 を加えた数値です。これは、配列の最初のエレメントのインデックス値に 0 が割り当てられているからです。

前の 2 つの例では、次のような同一の結果が得られます。



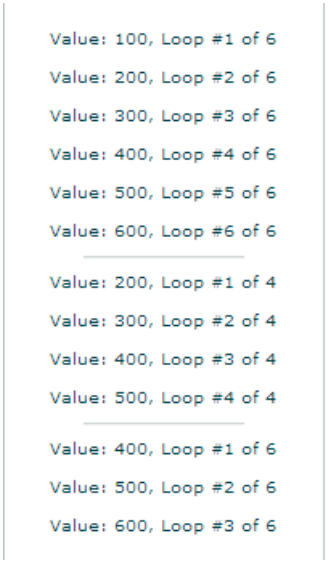
startingIndex プロパティと count プロパティを使用して、実行開始時点と実行回数を変更することもできます。count プロパティには Repeater コンポーネントの実行回数の上限を記述します。count プロパティは、多くの場合 Repeater コンポーネント内のコンテンツの実行回数と一致しますが、一致しない場合もあります。これは、count プロパティの値に関係なく、データプロバイダの最後のエレメントに到達すると Repeater コンポーネントが繰り返しを終了するからです。

次の例は、Repeater コンポーネントでの count プロパティと startingIndex プロパティの使用例を示します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            [Bindable]
            public var myArray:Array=[100,200,300,400,500,600];
        ]]>
    </mx:Script>

    <mx:ArrayCollection id="myAC" source="{myArray}"/>

    <mx:Repeater id="myrep" dataProvider="{myAC}" count="6">
        <mx:Label id="Label1"
            text="Value: {myrep.currentItem}, Loop #{myrep.currentIndex+1}
                of {myrep.count}"/>
    </mx:Repeater>

    <mx:HRule/>

    <mx:Repeater id="myrep2" dataProvider="{myAC}"
            count="4" startingIndex="1">
        <mx:Label id="Label2"
            text="Value: {myrep2.currentItem},
                Loop #{myrep2.currentIndex-myrep2.startingIndex+1}
                of {myrep2.count}"/>
    </mx:Repeater>
```

```
    <mx:HRule/>

    <mx:Repeater id="myrep3" dataProvider="{myAC}" count="6"
            startingIndex="3">
        <mx:Label id="Label3"
            text="Value: {myrep3.currentItem},
                Loop #{myrep3.currentIndex-myrep3.startingIndex+1}
                of {myrep3.count}"/>
     </mx:Repeater>
</mx:Application>
```

この例では、次の出力が生成されます。



最初の Repeater コンポーネントはデータプロバイダのすべてのエレメントを最初から最後までループし、最後のループ後に終了します。2 つ目の Repeater コンポーネントはデータプロバイダの 2 番目のエレメントから開始し、4 回繰り返して、5 番目のエレメントで終了します。3 つ目の Repeater コンポーネントはデータプロバイダの 4 番目のエレメントから開始し、データプロバイダの配列の最後に到達するまで繰り返して、終了します。count プロパティは 6 に設定されていますが、実際の繰り返し回数は 3 です。

Repeater コンポーネントを使用した動的ループの作成

906 ページの「Repeater コンポーネントの基本原則」で説明した原則は、動的データプロバイダにも適用されます。唯一の違いは、データ配列のソースが異なる点です。配列が、アプリケーションに直接書き込まれた静的配列ではなく、<mx:Model> タグで定義されて、XML ファイルや Web サービス、リモートオブジェクトなどのソースから渡されます。実行時にデータが収集および評価され、データプロバイダのエレメントの数と値、および、Repeater コンポーネントのビヘイビアが決定されます。

次の例の <mx:Repeater> タグは、XML ファイル内のすべての製品に対して RadioButton コントロールを繰り返します。

```
<?xml version="1.0"?>
<!-- repeater\DynamicLoop.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="catalogService.send()">
    <mx:HTTPService id="catalogService" url="assets/catalog.xml"
            resultFormat="e4x"/>
    <mx:XMLListCollection id="myXC"
            source="{catalogService.lastResult.product}"/>
    <mx:Repeater id="r" dataProvider="{myXC}">
        <mx:RadioButton id="Radio" label="{r.currentItem.name}"
                width="150"/>
    </mx:Repeater>
</mx:Application>
```

catalog.xml の内容が次のとおりであると仮定します。

```
<?xml version="1.0"?>
<products>
  <product>
    <name>Name</name>
    <price>Price</price>
    <freeship>Free Shipping?</freeship>
  </product>
  <product>
    <name>Whirlygig</name>
    <price>5</price>
    <freeship>false</freeship>
  </product>
  <product>
    <name>Tilty Thingy</name>
    <price>15</price>
    <freeship>true</freeship>
  </product>
<product>
    <name>Really Big Blocks</name>
    <price>25</price>
    <freeship>true</freeship>
  </product>
</products>
```

Flex によって次の出力が表示されます。

この場合も、count プロパティを使用して繰り返し回数の上限を指定できます。startingIndex プロパティを使用すると、データプロバイダの先頭でスキップするエントリを指定できます。

前の例では、XML ファイルの最初の製品エントリに、他のアプリケーションが列ヘッダを作成するために使用するメタデータが含まれています。このエントリにラジオボタンをつけて表示する必要はないので、Repeater コンポーネントをデータプロバイダの 2 番目のエレメントから開始します。

```
<?xml version="1.0"?>
<mx:Application     xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:HTTPService id="catalogService" url="assets/catalog.xml"
            resultFormat="e4x"/>
    <mx:XMLListCollection id="myXC"
            source="{catalogService.lastResult.product}"/>
    <mx:Repeater id="r" dataProvider="{myXC}"
            startingIndex="1">
        <mx:RadioButton id="Radio" label="{r.currentItem.name}" width="150"/>
    </mx:Repeater>
</mx:Application>
```

前の例では、次の出力が生成されます。

繰り返されるコンポーネントの参照

繰り返されるコンポーネントのそれぞれのインスタンスを参照するには、次の例に示すように、インデックス id 参照を使用します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<!-- repeater\RefRepeatedComponents.mxml -->
    <mx:Script>
        <![CDATA[
            public function labelTrace():void {
                for (var i:int = 0; i < nameLabel.length; i++)
                    trace(nameLabel[i].text);
            }
        ]]>
    </mx:Script>

    <mx:Model id="catalog" source="assets/catalog.xml"/>
    <mx:ArrayCollection id="myAC" source="{catalog.products.product}"/>
    <mx:Label id="title" text="Products:"/>
    <mx:Repeater id="r" dataProvider="{myAC}" startingIndex="1">
        <mx:Label id="nameLabel"
            text="{r.currentItem.name}: ${r.currentItem.price}" width="200"/>
    </mx:Repeater>
    <mx:Button label="Trace" click="labelTrace();"/>
</mx:Application>
```

この例では、繰り返される Label コントロールの id は nameLabel です。作成されるそれぞれの nameLabel インスタンスはこの id を持ちます。それぞれの Label インスタンスは、nameLabel[0]、nameLabel[1] などのように参照します。nameLabel インスタンスの数の合計は、nameLabel.length として参照します。for ループは、nameLabel Array オブジェクト内の各 Label コントロールの text プロパティをトレースします。mm.cfg ファイルでそのように定義している場合は、labelTrace() メソッドが各製品の名前と価格をシステムログにプリントします。

繰り返される子コンポーネントの参照

配列内でコンテナが繰り返されてインデックス付けされると、その子もインデックス付けされます。たとえば、次の MXML コードでは、VBox コンテナ vb[0] の子 Label コントロールを nameLabel[0] および shipLabel[0] として参照します。子を参照するためのシンタックスは、親を参照する場合のシンタックスと同じです。

```
<?xml version="1.0"?>
<!-- repeater\RefRepeatedChildComponents.mxml -->
<mx:Application borderStyle="solid" width="300" height="300"
xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function labelTrace():void {
                for (var i:int = 0; i < nameLabel.length; i++)
```

```
                    trace(nameLabel[i].text);
            }
        ]]>
    </mx:Script>

    <mx:Model id="catalog" source="assets/catalog.xml"/>
    <mx:Label id="title" text="Products:"/>

    <mx:Repeater id="r" dataProvider="{catalog.products.product}"
            startingIndex="1">
        <mx:VBox id="vb">
            <mx:Label id="nameLabel"
               text="{r.currentItem.name}: ${r.currentItem.price}" width="200"/>
            <mx:Label id="shipLabel"
                text="Free shipping: {r.currentItem.freeship}"/>
            <mx:Spacer/>
        </mx:VBox>
    </mx:Repeater>

    <mx:Button label="Trace" click="labelTrace();"/>
</mx:Application>
```

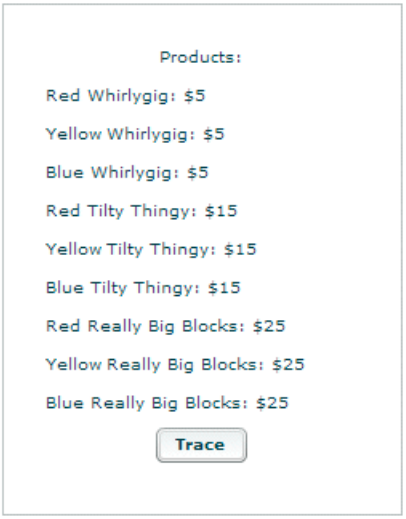
ネストされた Repeater コンポーネントの参照

<mx:Repeater> タグがネストされている場合、内側の <mx:Repeater> タグがインデックス付き Repeater コンポーネントとなります。たとえば、次の MXML コードでは、ネストされている Repeater コンポーネントに r2[0]、r2[1]、... のようにアクセスします。子を参照するためのシンタックスは、親を参照する場合のシンタックスと同じです。

```
<?xml version="1.0"?>
<!-- repeater\RefNestedComponents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
      public function labelTrace():void {
        for (var i:int = 0; i < nameLabel.length; i++)
         for (var j:int = 0; j < nameLabel[i].length; j++)
          trace(nameLabel[i][j].text);
      }
    ]]>
  </mx:Script>

  <mx:Model id="data">
    <color>
      <colorName>Red</colorName>
      <colorName>Yellow</colorName>
      <colorName>Blue</colorName>
    </color>
  </mx:Model>

  <mx:Model id="catalog" source="assets/catalog.xml"/>
```

```
  <mx:ArrayCollection id="myAC1" source="{catalog.products.product}"/>
  <mx:ArrayCollection id="myAC2" source="{data.colorName}"/>

  <mx:Label id="title" text="Products:"/>

  <mx:Repeater id="r" dataProvider="{myAC1}" startingIndex="1">
    <mx:Repeater id="r2" dataProvider="{myAC2}">
      <mx:Label id="nameLabel" text="{r2.currentItem}
          {r.currentItem.name}: ${r.currentItem.price}" width="200"/>
    </mx:Repeater>
  </mx:Repeater>

  <mx:Button label="Trace" click="labelTrace();"/>
</mx:Application>
```

このアプリケーションでは、ユーザーが Button コントロールをクリックすると、色および価格と共に製品名の一覧がシステムログに書き込まれ (mm.cfg ファイルでそのように定義している場合)、次のように画面に表示されます。



前に示した例では、Label コントロールのインスタンスに複数のインデックスが付けられています。これは、複数の Repeater コンポーネントにインスタンスが存在するからです。たとえば、インデックス `nameLabel[1][2]` は、`r` の 2 回目の繰り返しおよび `r2` の 3 回目の繰り返しによって生成された Label コントロールへの参照を含みます。

Repeater コンポーネントのイベントハンドラ

Repeater コンポーネントの繰り返しが行われているとき、作成された繰り返しオブジェクトは、Repeater コンポーネントのその時点の currentItem プロパティにバインドできます。このプロパティは、Repeater コンポーネントの繰り返しと共に変化します。

click="doSomething({r.currentItem})" のようなコードを書いて、それぞれのインスタンスに独自のイベントハンドラを持たせることはできません。これは、イベントハンドラ内ではバインディング式を使用できず、繰り返されるコンポーネントのすべてのインスタンスは同じイベントハンドラを共有する必要があります。

繰り返されるコンポーネントおよび繰り返される Repeater コンポーネントは、getRepeaterItem() メソッドを持ちます。このメソッドは、オブジェクトを生成するために使用された dataProvider プロパティ内のアイテムを返します。Repeater　コンポーネントの繰り返しが完了すると、getRepeaterItem() メソッドを使用して、currentItem プロパティに基づいてイベントハンドラが行うべき処理を判定できます。そのためには、event.currentTarget.getRepeaterItem() メソッドをイベントハンドラに渡します。getRepeaterItem() メソッドは、オプションのインデックスを受け取ります。このインデックスは、ネストされている Repeater コンポーネントがある場合に、どの Repeater コンポーネントを使用するかを指定します。インデックス 0 は、最も外側の Repeater コンポーネントです。インデックスの引数を指定しない場合は、最も内側の Repeater コンポーネントが暗黙的に指定されます。

メモ

Repeater コンポーネントの繰り返しが完了した後は、Repeater.currentItem プロパティを使用して現在のアイテムを取得することはありません。代わりに、繰り返されるコンポーネント自体の getRepeaterItem() メソッドを呼び出します。

次は、getRepeaterItem() メソッドの使用例です。ユーザーが個々の繰り返された Button コントロールをクリックすると、それに対応するデータモデルの colorName 値が Button コントロールのラベルに表示されます。

```
<?xml version="1.0"?>
<!-- repeater\GetItem.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
      public function clicker(cName:String):void {
         foolabel.text=cName;
       }
    ]]>
  </mx:Script>

  <mx:Label id="foolabel" text="foo"></mx:Label>

  <mx:Model id="data">
    <color>
      <colorName>Red</colorName>
      <colorName>Yellow</colorName>
```

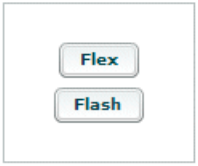
```
      <colorName>Blue</colorName>
    </color>
  </mx:Model>
  <mx:ArrayCollection id="myAC" source="{data.colorName}"/>

  <mx:Repeater id="myrep" dataProvider="{myAC}">
    <mx:Button click="clicker(event.currentTarget.getRepeaterItem());"
      label="{myrep.currentItem}"/>
  </mx:Repeater>
</mx:Application>
```

ユーザーが [Yellow] ボタンをクリックすると次のようになります。

次の例のコードでは、getRepeaterItem() メソッドを使用して、ユーザーによってクリックされた各 Button コントロールの特定の URL を表示します。イベントハンドラ内ではバインディング式を使用できないので、Button コントロールは共通のデータ駆動型クリックハンドラを共有しなければなりません。ただし、getRepeaterItem() メソッドを使用すると、各 Button コントロールの機能を変更できます。

```
<?xml version="1.0"?>
<!-- repeater\DisplayURL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
      [Bindable]
      public var dp:Array = [ { label: "Flex",
        url: "http://www.adobe.com/flex" },
        { label: "Flash", url: "http://www.adobe.com/flash" } ];
    ]]>
  </mx:Script>

  <mx:ArrayCollection id="myAC" source="{dp}"/>
  <mx:Repeater id="r" dataProvider="{myAC}">
    <mx:Button label="{r.currentItem.label}" click="navigateToURL(new
      URLRequest(event.currentTarget.getRepeaterItem().url));"/>
  </mx:Repeater>
</mx:Application>
```

実行すると、[Flex] と [Flash] の 2 つのボタンが表示されます。いずれかのボタンをクリックすると、選択した方の製品ページがブラウザの新しいウィンドウに表示されます。



繰り返されるコンポーネントの特定のインスタンスへのアクセス

繰り返されるコンポーネントおよび繰り返される Repeater コンポーネントは、3 つのプロパティを持ちます。これらのプロパティを使用することによって、繰り返されるオブジェクトの特定のインスタンスを動的に追跡したり、どの Repeater コンポーネントによってオブジェクトのインスタンスが生成されたかを調べたり、それぞれの Repeater コンポーネントによってどの `dataProvider` アイテムが使用されたかを調べることができます。これらのプロパティについて次の表で説明します。

プロパティ	説明
`instanceIndices`	ドキュメントからコンポーネントを参照するために必要なインデックスを含む Array です。コンポーネントが 1 つまたは複数の Repeater コンポーネント内に存在しない場合、この Array は空です。最初のエレメントが最も外側の Repeater コンポーネントを表します。たとえば、`id` が `b` で、`instanceIndices` が `[2,4]` の場合は、ドキュメント上でこれらのオブジェクトを `b[2][4]` として参照します。
`repeaters`	コンポーネントを生成した Repeater コンポーネントへの参照を含む Array です。コンポーネントが 1 つまたは複数の Repeater コンポーネント内に存在しない場合、この Array は空です。最初のエレメントが最も外側の Repeater コンポーネントを表します。
`repeaterIndices`	コンポーネントを生成した Repeater コンポーネントの `dataProvider` プロパティ内のアイテムのインデックスを含む Array です。コンポーネントが 1 つまたは複数の Repeater コンポーネント内に存在しない場合、この Array は空です。最初のエレメントが最も外側の Repeater コンポーネントを表します。たとえば、`repeaterIndices` が `[2,4]` の場合、外側の Repeater コンポーネントではデータアイテム `dataProvider[2]` が使用され、内側の Repeater コンポーネントではデータアイテム `dataProvider[4]` が使用されます。 このプロパティは、いずれかの Repeater コンポーネントの `startingIndex` が 0 以外の場合は、`instanceIndices` とは異なる値になります。たとえば、Repeater コンポーネントが `dataProvider` アイテム 4 で始まったとしても、最初の繰り返しコンポーネントのドキュメント参照は `b[4]` ではなく `b[0]` となります。

次に示すサンプルアプリケーションでは、repeaters プロパティを使用して、外側と内側の Repeater コンポーネントの各インデックス値のラベルがついた Button コントロールのいずれか 1 つをユーザーがクリックしたときに、Alert コントロールに Repeater コンポーネントの id 値を表示します。repeaters プロパティを使用して、内側の Repeater コンポーネントの id 値を取得しています。

```
<?xml version="1.0"?>
<!-- repeater\RepeaterProp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      [Bindable]
      public var myArray:Array=[1,2];
    ]]>
  </mx:Script>

  <mx:ArrayCollection id="myAC" source="{myArray}"/>
  <mx:Repeater  id="repeater1" dataProvider="{myAC}">
    <mx:Repeater  id="repeater2" dataProvider="{myAC}">
      <mx:Button
        label="[{repeater1.currentIndex},{repeater2.currentIndex}]"
        click="Alert.show(event.target.repeaters[1].id);"/>
    </mx:Repeater>
  </mx:Repeater>
</mx:Application>
```

前の例では、ユーザーが 3 つ目のボタンをクリックすると次の結果が生成されます。

次に示すサンプルアプリケーションでは、繰り返される Button および TextInput コントロールのセット内の対応する Button コントロールがユーザーによってクリックされたときに、instanceIndices プロパティを使用して、TextInput コントロールの text プロパティを設定します。この場合、適切なオブジェクトを動的に取得する必要があるので、instanceIndices プロパティを使用する必要があります。id 値を使用してオブジェクトを取得することはできません。

次の例に、Button コントロールがユーザーによってクリックされたときに instanceIndices プロパティを使用して TextInput コントロールの text プロパティを設定する方法を示します。引数 event.target.instanceIndices は、対応する TextInput コントロールのインデックスを取得します。

```
<?xml version="1.0"?>
<!-- repeater\InstanceIndicesProp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
      [Bindable]
      public var myArray:Array=[1,2,3,4,5,6,7,8];
    ]]>
  </mx:Script>

  <mx:ArrayCollection id="myAC" source="{myArray}"/>
  <mx:Repeater id="list" dataProvider="{myAC}" count="4" startingIndex="2">
    <mx:HBox>
      <mx:Button label="Click Me"
        click="myText[event.target.instanceIndices].text=
        event.target.instanceIndices.toString();"/>
      <mx:TextInput id="myText"/>
    </mx:HBox>
  </mx:Repeater>
</mx:Application>
```

ユーザーが 2 つ目と 4 つ目のボタンをクリックすると次のようになります。

次の例では、ユーザーが Button コントロールをクリックしたときに、instanceIndices プロパティの代わりに repeaterIndices プロパティを使用して TextInput コントロールの text プロパティを設定する方法を示します。event.target.repeaterIndices の値は、Repeater コンポーネントの現在のインデックスに基づいています。Repeater コンポーネントの startingIndex プロパティは 2 に設定されるので、常に 0 から始まる event.target.instanceIndices 値には一致しません。

```
<?xml version="1.0"?>
<!-- repeater\RepeaterIndicesProp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
```

```
      [Bindable]
      public var myArray:Array = [1,2,3,4,5,6,7,8];
    ]]>
  </mx:Script>

  <mx:Repeater id="list" dataProvider="{myArray}"
    startingIndex="2" count="4">
  <mx:HBox>
    <mx:Button id="b" label="Click Me"
      click="myText[event.target.repeaterIndices-list.startingIndex].text=
        event.target.repeaterIndices.toString();"/>
      <mx:TextInput id="myText"/>
    </mx:HBox>
  </mx:Repeater>
</mx:Application>
```

この場合、ボタンをクリックすると、ループの現在の繰り返しではなく、データプロバイダの現在のエレメントが印刷されます。

メモ

repeaterIndices プロパティの値は、Repeater コンポーネントの最初の繰り返しの startingIndex プロパティと等価です。startingIndex の値を repeaterIndices の値から引くと、常に instanceIndices の値になります。したがって、click イベントを次のように記述することもできます。

```
click="myText[event.target.instanceIndices].text=event.target.repeaterIndices.to
  String()"/>
```

いずれの場合も、ユーザーが 2 つ目と 4 つ目のボタンをクリックすると次のようになります。

カスタム MXML コンポーネント内での Repeater コンポーネントの使用

<mx:Repeater> タグは、アプリケーションファイル内で使用するのと同じ方法で、MXML コンポーネント定義内で使用できます。この MXML コンポーネントを他の MXML ファイル内でタグとして使用すると、繰り返されたアイテムが表示されます。個々の繰り返しアイテムには、アイテムの配列インデックス番号を使ってアクセスできます。これは、アプリケーションファイル内に定義された繰り返しアイテムに対するアクセス方法と同じです。

次の例では、childComp という名前の MXML コンポーネント内の Button コントロールが、dp という名前の Array オブジェクトのすべてのエレメントに対して繰り返されます。

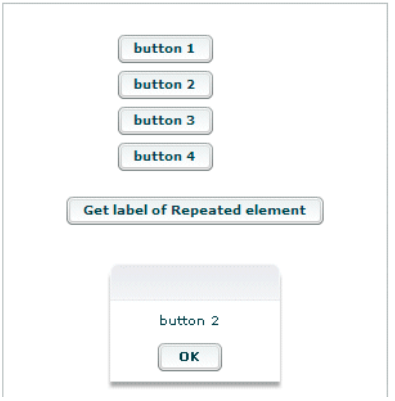
```
<?xml version="1.0"?>
<!-- repeater\myComponents\CustButton.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
    <![CDATA[
      [Bindable]
      public var dp:Array=[1,2,3,4];
    ]]>
  </mx:Script>

  <mx:ArrayCollection id="myAC" source="{dp}"/>

  <mx:Repeater id="r" dataProvider="{myAC}">
    <mx:Button id="repbutton" label="button {r.currentItem}"/>
  </mx:Repeater>
</mx:VBox>
```

次の例のアプリケーションファイルでは、childComp コンポーネントを使用して、配列内の各エレメントに対して 1 つずつ、合計 4 つの Button コントロールを表示します。getLabelRep() 関数は、配列の 2 番目の Button のラベルテキストを表示します。

```
<?xml version="1.0"?>
<!-- repeater\RepeatCustButton.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:MyComp="myComponents.*">

  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      public function getLabelRep():void {
        Alert.show(comp.repbutton[1].label);
      }
    ]]>
  </mx:Script>

  <MyComp:CustButton id="comp"/>
```

```
  <mx:Button label="Get label of Repeated element" width="200"
    click="getLabelRep();"/>
</mx:Application>
```

前の例では、ユーザーが最後のボタンをクリックすると次の出力が生成されます。



データ型に基づくコンポーネントの動的な作成

Repeater コンポーネントを使用すると、データのセット内の特定のアイテムに対して、異なる型のコンポーネントを動的に作成できます。Repeater コンポーネントは、実行時に repeat イベントをブロードキャストします。さらに、このイベントは、currentIndex プロパティと currentItem プロパティが設定された後にブロードキャストされます。このとき、repeat イベントのイベントハンドラ関数を呼び出すことによって、それぞれのデータアイテムに基づいて異なる型のコンポーネントを動的に作成できます。

Repeater コンポーネントの実行について

Repeater コンポーネントは、インスタンス化されたときに最初に実行されます。Repeater コンポーネントに dataProvider プロパティが存在する場合は、子、子の子というように再帰的に初期化が行われます。

Repeater コンポーネントは、dataProvider、startingIndex、または count プロパティが ActionScript 内で明示的に設定または変更されたとき、あるいはデータバインディングによって暗黙的に設定されたときに、再び実行されます。dataProvider プロパティが Web サービスの結果にバインドされている場合、Repeater コンポーネントは Web サービスから結果が返されたときに再び実行されます。また、Repeater コンポーネントは、startingIndex 値を徐々に増加させて dataProvider プロパティを変化させたときにも実行されます。次に例を示します。

```
r.startingIndex += r.count;
```

Repeater コンポーネントが再実行されると、以前に作成された子がすべて破棄された後 (recycleChildren プロパティが false に設定されている場合)、現在の dataProvider プロパティに基づいて子が再インスタンス化されます。このとき、コンテナ内の子の数が変更される場合があります。その場合、変更された子の数に合わせて、コンテナのレイアウトも変更されます。

Repeater コンポーネント内での子の再作成

recycleChildren プロパティは、Repeater コンポーネントが再実行されたときに、Repeater コンポーネントの子を再作成するかどうかを制御します。recycleChildren プロパティを false に設定すると、dataProvider を入れ替えたとき、または並べ替えたときに、Repeater コンポーネントがすべてのオブジェクトを再作成するので、パフォーマンスが低下します。dataProvider を変更しても Repeater コンポーネントの子が再作成されないことが確実な場合を除き、このプロパティは常に true に設定してください。

繰り返されるインスタンスに古い状態情報が保持されないよう、recycleChildren プロパティのデフォルトの値は false です。たとえば、Repeater コンポーネントを使用して写真イメージを表示するとします。それぞれの Image コントロールには、プリントの注文枚数を示す NumericStepper コントロールが関連付けられているとします。ここで、状態情報には、dataProvider から渡されるもの (イメージなど) と、ユーザーの操作によって設定されるもの (印刷枚数など) があります。recycleChildren プロパティを true に設定し、Repeater コンポーネントの startingIndex 値を徐々に増加させて写真間を移動すると、それに応じて Image コントロールが新しいイメージにバインドされますが、NumericStepper コントロールは古い情報を保持したままになります。

Repeater コンポーネントの使用に関する注意事項

Repeater コンポーネントを使用する場合は、次の点に注意してください。

- Repeater コンポーネントを使用して、プログラムによって生成された 2 次元 Array オブジェクトを繰り返すことはできません。これは、Array オブジェクトのエレメントは changeEvent イベントをトリガしないので、実行時にバインディング元として機能しないからです。バインディングにより、インスタンス化の実行中に初期値がコピーされます。これは、<mx:Script> タグで変数が宣言された後、initialize ハンドラが実行される前に行われます。
- データプロバイダとして使用された配列への実行時の変更は、Repeater コンポーネントに反映されません。実行時の変更が必要な場合は、コレクションを使用します。
- よくある間違いとして、Repeater コンポーネントを使用する際に、dataProvider プロパティに中括弧 ({ }) の記述を忘れてしまうことが挙げられます。Repeater コンポーネントが実行されない場合は、バインディングに誤りがないことを確認してください。

■ 繰り返されるオブジェクトが正しい順序で表示されず、Repeater コンポーネントを連続でまたはネストして使用している場合は、オブジェクトが正しく表示されない Repeater の直後にダミーの UIComponent を配置する必要があります。

次に示すコードには連続する <mx:Repeater> タグが含まれているので、<mx:Spacer> タグを使用してダミーの UIComponent を作成します。

```
<mx:VBox>
  <mx:Repeater id="r1">
    ...
  </mx:Repeater>
  <mx:Repeater id="r2">
    ...
  </mx:Repeater>
  <mx:Spacer height="0" id="dummy"/>
</mx:VBox>
```

次に示すコードにはネストされた <mx:Repeater> タグが含まれているので、<mx:Spacer> タグを使用してダミーの UIComponent を作成します。

```
<mx:VBox>
  <mx:Repeater id="outer">
    <mx:Repeater id="inner">
      ...
    </mx:Repeater>
    <mx:Spacer id="dummy" height="0">
  </mx:Repeater>
</mx:VBox>
```

第 27 章

ビューステートの使用

27

ビューステートでは、通常はユーザーの操作に対する応答として、コンポーネントの内容や外観を変更できます。ビューステートを使用するには、コンポーネントの基本ビューステートを 1 つ定義してから、この基本ビューステートへの変更を指定する追加のビューステートを定義します。追加のビューステートは複数定義できます。

このトピックでは、ビューステートを使用してコンポーネントやアプリケーションの表示を制御する方法について説明します。ビューステートの変化に応じてアプリケーションのビヘイビアを制御するトランジションの適用方法については説明しません。トランジションの詳細については、955 ページ、第 28 章の「トランジションの使用」を参照してください。

目次

ビューステートについて

多くの高度なインターネットアプリケーションでは、ユーザーの実行するタスクに応じてインターフェイスが変化します。簡単な例としては、ユーザーがマウスポインタをイメージの上に置くと別のイメージに変わるなどの変化が挙げられます。より複雑な例としては、ユーザーによるタスクの進捗状況に応じてブラウズビューから詳細ビューに変更するなどの、ユーザーインターフェイスの内容の変化が挙げられます。このようなインターフェイスでは、スムーズな開く - 閉じるエフェクトを使用して、ビューが切り替えられます。ビューステートを使用すると、複数の Macromedia Flash アプリケーションをロードすることなく、そのようなビヘイビアを簡単に実装できます。これにより、複雑なイベント処理コードを簡素化できます。

ビューステートでは、基本アプリケーションまたはコンポーネント (ベースステート) を定義し、このベースステートや別のステートからの変更のセットを定義できます。これにより、外観やビヘイビアが変化するアプリケーションを体系的に構成できます。変更のセットごとにビューステートが定義されます。これにより、子の追加と削除、スタイルやプロパティの設定と変更、状態固有のイベントリスナーの定義が可能です。

ビューステート間のトランジションを定義することにより、あるビューステートから別のビューステートへ移るとき、アプリケーションの外観がどのように変化するかを制御することもできます。トランジションの使用方法の詳細については、955 ページ、第 28 章の「トランジションの使用」を参照してください。

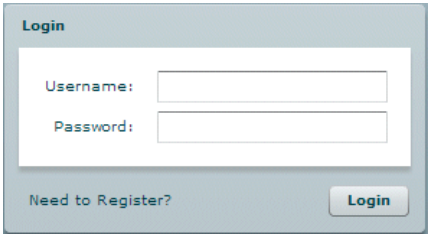
最も単純なビューステートは、コンポーネントのある特定のビューです。たとえば、ある製品サムネールには、最低限の情報だけを表示するベースステートと追加情報を含む詳細状態の 2 つのビューステートがあります。同様に、アプリケーションにも、アプリケーションの状況に応じて、複数のビューステートを持たせることができます。たとえば、ログイン状態、概要状態、検索結果状態などです。

ビューステートを定義しなくとも、インタラクティブ機能を持つ高度なインターネットアプリケーションは作成できますが、ビューステートは、動的なアプリケーションを論理的に構成するための強力なツールになります。

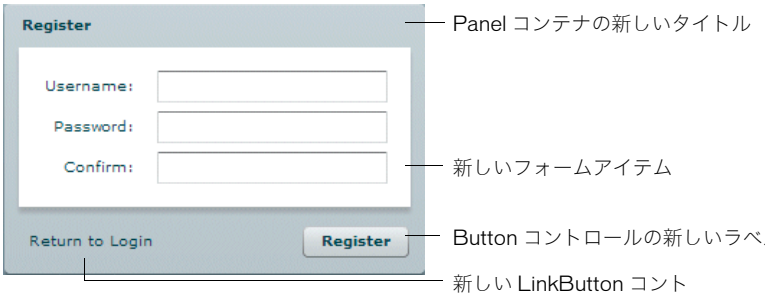
次に、ビューステートの使用例を示します。

使用例 : ログインインターフェイス

ビューステートを使用すると、ログインフォームや登録フォームを簡単に実装できます。この場合、初期ビューステートでユーザーにログインが促されます。また、必要に応じてユーザーが登録するためのリンクも表示されます。次の図を参照してください。



ユーザーが [Need to Register] リンクを選択すると、フォームのビューステートが変化して登録情報が表示されます。次の図を参照してください。



ユーザーが [Return to Login] リンクをクリックすると、ビューステートがログインフォームに戻ります。

この例のコードは、932 ページの「例 : ログインフォームアプリケーション」に記載されています。

使用例 : カスタムアイテムレンダラー

1つのページに複数のアイテムを表示するショッピングアプリケーションで、2 つのビューステートを持つサムネールのカスタムアイテムレンダラーを使用しています。基本ビューステートでは、アイテムのセルは次のように表示されます。

ユーザーがマウスポインタをアイテム上に置くと、ビューステートが変わります。在庫情報とお勧め度情報がサムネールに表示されなくなり、詳細を表示するためのボタンやアイテムをウィッシュリストやショッピングカートに追加するボタンが表示されます。新しいビューステートでは、セルの境界線とドロップシャドウが次のように表示されます。

この例のアプリケーションアイテムレンダラーには、異なる子コンポーネントとコンポーネントスタイルを持つ 2 つのビューステートがあります。たとえば、サマリーのビューステートには、在庫情報のラベルとお勧め度を示す星のイメージが表示され、境界線は表示されません。ポインタが置かれたビューステートには、在庫情報のラベルとお勧め度のコンポーネントに代わって 3 つのボタンが表示され、アウトセットの境界線が表示されます。

これに似た例のコードが、947 ページの「例 : カスタムアイテムレンダラーとビューステートの使用」に記載されています。アイテムレンダラーの詳細については、779 ページ、第 21 章の「アイテムレンダラーとアイテム エディタの使用」を参照してください。

ビューステートの定義と適用

このセクションでは、ビューステートを作成および適用するための基本概念を説明し、ルールの簡単な使用例を示します。後述するセクションでは、ビューステートのクラス、メソッド、プロパティの使用方法とビューステートに基づいたアプリケーションの作成方法について詳細に説明します。

ビューステートの適用

Application コンポーネントを含むあらゆる Flex コンポーネントのビューステートを設定できます。コンポーネントは、その都度異なるビューステートを持ちますが、一度に持てるビューステートは 1 つだけで、同時に複数のビューステートを持つことはできません。

- コンポーネントの currentState プロパティで、そのコンポーネントのビューステートを指定します。原則として、このプロパティはビューステートを変更するために使用してください。
- UIComponent クラスの setCurrentState() メソッドを呼び出しても、コンポーネントのビューステートを変更できます。2 つのビューステートの間に定義したトランジションを適用しない場合は、このメソッドを使用します。
- 基本ビューステートを指定する場合は、currentState プロパティを "" に設定します。
- 多くのアプリケーションでは、イベントに応じてビューステートが変わります。ログインしているかどうかなどのユーザー状態を基準にして、ビューステートを変更することもできます。

次の例は、Button コントロールのクリックに応じて 2 つのビューステートが切り替わります。ユーザーがボタン b1 をクリックすると、ビューステートが newButton になり、ボタン b2 をクリックすると、基本ビューステートになります。

```
<mx:Button id="b1" label="Add a Button" click="currentState='newButton';"/>
<mx:Button id="b2" label="Remove Added Button" click="currentState='';"/>
```

ビューステートの定義

あるコンポーネントに対して定義したプロパティ、スタイル、イベントリスナーおよび子コンポーネントによって、そのコンポーネントの基本あるいはデフォルトのビューステートが指定されます。<mx:State> タグまたは State クラスオブジェクトを使用して、他のビューステートを定義し、追加することもできます。それぞれのビューステートで、基本ビューステートまたは他のビューステートにどのように変化するかが指定されます。

ビューステートの定義は次のように行います。

- ビューステートの定義は、アプリケーションまたはカスタムコンポーネントのルート、つまり、アプリケーションファイルの <mx:Application> タグのプロパティまたは MXML コンポーネントのルートタグのプロパティでのみ行うことができます。
- コンポーネントの states プロパティを使用して、通常は <mx:states> 子タグとして状態を定義します。

- states プロパティに、1つまたは複数の States クラスの配列を指定します。ここで、各 State クラスインスタンスは1つのビューステートに対応します。
- State クラスの name プロパティを使用して、ビューステートの識別子を指定します。このビューステートに変更するには、コンポーネントの currentState プロパティを name プロパティの値に設定します。
- ビューステートを別のビューステートを基準に定義するには、<mx:State> basedOn プロパティに状態を指定します。そうしないと、基本ビューステートへのオーバーライドで構成されるビューステートになります (状態が変わるとき、まずベースステートに戻り、そこに State クラスの basedOn プロパティに指定された状態からの変更が適用されてから、新しい状態で定義された変更が適用されます)。
- 各 State クラスの overrides プロパティには、状態オーバーライドの配列が含まれます。overrides プロパティは、States クラスのデフォルトプロパティで、省略できます。MXML の <mx:Array> タグも同様です。
- 状態オーバーライドを定義するには、932 ページの「ビューステートのエレメント」に説明されたエレメントを使用します。

次の MXML コードに、この構造を示します。

```
<mx:Application> <!-- カスタムコンポーネントのルートタグにもできます。-->
  <mx:states>
    <mx:State>
      <mx:AddChild/>
      .
      <mx:SetStyle/>
      .
      <mx:SetProperty/>
      .
      <mx:SetEventHandler/>
      .
    </mx:State>
    <mx:State>
      .
      .
    </mx:State>
    .
    .
  </mx:states>
  .
  .
</mx:Application>
```

次の例では、State オブジェクトを1つ持つ単純な <mx:states> タグを示します。

```
<?xml version="1.0"?>
<!-- states\StatesSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:states>
        <mx:State name="NewButton">
            <mx:AddChild relativeTo="{v1}">
                <mx:Button id="b2" label="New Button"/>
            </mx:AddChild>
            <mx:SetProperty target="{b1}" name="enabled" value="false"/>
        </mx:State>
    </mx:states>

    <mx:VBox id="v1">
      <mx:Button label="Change State"
        click="currentState = currentState=='NewButton' ? '':'NewButton';"/>

      <mx:Button id="b1"/>
    </mx:VBox>
</mx:Application>
```

個々の子タグの target プロパティを変更するコンポーネントを指定することによって、1 つの State オブジェクトで複数のコンポーネントを変更できます。前の例では、<mx:AddChild> タグによって v1 VBox コントロールに子が追加され、<mx:SetProperty> タグによって、b1 Button コントロールのプロパティが変更されています。次のコードは、2 つの Button コントロールの enabled プロパティを、それぞれ false と true に設定しています。

```
<?xml version="1.0"?>
<!-- states\StatesSimple2Buttons.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:states>
        <mx:State name="NewButton">
            <mx:AddChild relativeTo="{v1}">
                <mx:Button id="b3" label="New Button"/>
            </mx:AddChild>
            <mx:SetProperty target="{b1}" name="enabled" value="false"/>
            <mx:SetProperty target="{b2}" name="enabled" value="true"/>
        </mx:State>
    </mx:states>

    <mx:VBox id="v1">
      <mx:Button label="Change State"
        click="currentState = currentState=='NewButton' ? '':'NewButton';"/>

      <mx:Button id="b1"/>
      <mx:Button id="b2" enabled="false"/>
    </mx:VBox>
</mx:Application>
```

ビューステートのエレメント

ビューステートは、ターゲットコンポーネントに対する変更の名前付きセット、または、オーバーライドの名前付きセットで構成されます。デフォルトでは、ビューステートを定義したアプリケーションまたはカスタムコンポーネントにオーバーライドが適用されます。オーバーライドを定義するには、次のクラスを使用します。

- 子オブジェクトを追加または削除するには、AddChild クラスまたは RemoveChild クラスを使用します。たとえば、次の例は v1 という VBox コントロールに Button 子コントロールを追加します。

```
<mx:AddChild relativeTo="{v1}">
  <mx:Button label="New Button"/>
</mx:AddChild>
```

- 以下のコンポーネント特性を設定または変更する方法は次のとおりです。

  プロパティを設定するには、SetProperty クラスを使用します。次の例は、button1 という Button コントロールを無効にします。

```
<mx:SetProperty target="{button1}" name="enabled" value="false"/>
```

  スタイルを設定するには、SetStyle クラスを使用します。次の例は、アプリケーションまたはカスタムコンポーネントの color プロパティを設定しています。

```
<mx:SetStyle name="color" value="0xAAAAAA"/>
```

  イベントリスナーを設定するには、SetEventHandler クラスを使用します。次の例は、button1 という Button コントロールの click イベントリスナーを設定しています。

```
<mx:SetEventHandler target="{button1}" name="click"
  handler="newClickHandler()"/>
```

- 独自に定義したカスタムクラスを使用するには、IOverride インターフェイスを実装します。カスタムオーバーライドの作成および使用の詳細は、952 ページの「カスタムオーバーライドクラスの作成」を参照してください。

例 : ログインフォームアプリケーション

次の例では、932 ページの「例 : ログインフォームアプリケーション」に示されたログインフォームおよび登録フォームを作成します。このアプリケーションには次の機能があります。

- ユーザーが [Need to Register] LinkButton コントロールをクリックすると、the event handler for the click イベントのイベントハンドラによってビューステートが Register に設定されます。
- Register 状態のコードによって、TextInput コントロールが追加され、Panel コンテナおよび Button コントロールのプロパティが変更され、既存の LinkButton コントロールが削除され、新しい LinkButton コントロールが追加されます。
- ユーザーが [Return to Login] LinkButton コントロールをクリックすると、the event handler for the click イベントのイベントハンドラによってビューステートが基本ビューステートにリセットされます。

```
<?xml version="1.0"?>
<!-- states\LoginExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalAlign="middle">

    <!-- The Application class states property defines the view states.-->
    <mx:states>
        <mx:State name="Register">
            <!-- Add a TextInput control to the form. -->
            <mx:AddChild relativeTo="{loginForm}"
                position="lastChild"
                creationPolicy="all">
                <mx:FormItem id="confirm" label="Confirm:">
                    <mx:TextInput/>
                </mx:FormItem>
            </mx:AddChild>

          <!-- Set properties on the Panel container and Button control.-->
            <mx:SetProperty target="{loginPanel}"
                name="title" value="Register"/>
            <mx:SetProperty target="{loginButton}"
                name="label" value="Register"/>

            <!-- Remove the existing LinkButton control.-->
            <mx:RemoveChild target="{registerLink}"/>

            <!-- Add a new LinkButton control to change view state
                back to the login form.-->
            <mx:AddChild relativeTo="{spacer1}" position="before">
                <mx:LinkButton label="Return to Login"
                    click="currentState=''"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <mx:Panel id="loginPanel"
        title="Login"
        horizontalScrollPolicy="off"
        verticalScrollPolicy="off">
        <mx:Form id="loginForm">
            <mx:FormItem label="Username:">
                <mx:TextInput/>
            </mx:FormItem>
            <mx:FormItem label="Password:">
                <mx:TextInput/>
            </mx:FormItem>
        </mx:Form>
        <mx:ControlBar>
          <!-- Use the LinkButton to change to the Register view state.-->
            <mx:LinkButton id="registerLink"
```

```
                label="Need to Register?"
                click="currentState='Register'"/>
            <mx:Spacer width="100%" id="spacer1"/>
            <mx:Button label="Login" id="loginButton"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

ActionScript でのビューステートの作成

ActionScript でビューステートを作成するには、次の手順を実行します。

1. mx.states パッケージ内のクラスを読み込みます。
2. それを使用するコードで利用できる State 変数を宣言します。
3. 次のことを実行する関数を作成します。
   a. 新しい State オブジェクトをインスタンス化します。
   b. 手順1で定義した変数にオブジェクトを割り当てます。
   c. 状態の名前を指定します。
   d. オーバーライドクラスオブジェクトの変数を宣言し、SetProperty などのクラスの新しいインスタンスにそれらを割り当てます。
   e. オーバーライドインスタンスのプロパティを指定します。
   f. State オブジェクトの `overrides` 配列にオーバーライドインスタンスを追加します。
   g. 状態を `states` 配列に追加します。
4. この関数を、通常は `initialize` イベントに対する応答として、アプリケーションまたはカスタムコンポーネントの初期化イベントリスナーの一部として呼び出します。

次の例では、ActionScript を使用して、HBox コンテナのスタイルプロパティを設定するビューステートを作成します。

```
<?xml version="1.0"?>
<!-- states\StatesAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="createState();" >

    <mx:Script>
        <![CDATA[

            import mx.states.*;

            // Variable for the ActionScript-defined view state.
            private var newState:State;

            // Initialization method and
            // creates a view state using ActionScript.
```

```
            private function createState():void {
                newState = new State();
                newState.name = "state1";
                var myProp:SetStyle = new SetStyle();
                myProp.name="backgroundColor";
                myProp.value= "#CCCCCC";
                myProp.target = myHB;
                newState.overrides[0] = myProp;
                states.push(newState);
            }
        ]]>
    </mx:Script>

    <mx:HBox id="myHB" height="50%" width="50%">
        <mx:Button label="Base state" click="currentState='';"/>
        <mx:Button label="Change state" click="currentState='state1';"/>
    </mx:HBox>
</mx:Application>
```

コンポーネントプロパティの設定

SetProperty クラスを使用すると、特定のビューステートでのみ有効になるプロパティ値を指定できます。たとえば、次のコードは、Register ビューステートに切り替えたときに Panel コンテナのプロパティと Button コントロールのプロパティを設定します。

```
<?xml version="1.0"?>
<!-- states\StatesSetProp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:states>
        <mx:State name="Register">
            <mx:SetProperty
                target="{loginPanel}"
                name="title"
                value="Register"/>
            <mx:SetProperty
                target="{loginButton}"
                name="label"
                value="Register"/>
        </mx:State>
    </mx:states>

    <mx:Panel id="loginPanel"
        title="Login"
        horizontalScrollPolicy="off"
        verticalScrollPolicy="off">

        <mx:Form id="loginForm">
            <mx:Button label="Login" id="loginButton"/>
```

```
        </mx:Form>

        <mx:ControlBar width="100%">
            <mx:Button label="Change State"
              click="currentState =
                currentState=='Register' ? '':'Register';"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

データバインディングを使用すると、value プロパティに情報を指定できます。次の例を参照してください。

```
<?xml version="1.0"?>
<!-- states\StatesSetProp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // Define a variable in ActionScript.
            [Bindable]
            public var registerValue:String="Register";
        ]]>
    </mx:Script>

    <mx:states>
        <mx:State name="Register">
            <mx:SetProperty
                target="{loginPanel}"
                name="title"
                value="{registerValue}"/>
            <mx:SetProperty
                target="{loginButton}"
                name="label"
                value="{registerValue}"/>
        </mx:State>
    </mx:states>

    <mx:Panel id="loginPanel"
        title="Login"
        horizontalScrollPolicy="off"
        verticalScrollPolicy="off">

        <mx:Form id="loginForm">
            <mx:Button label="Login" id="loginButton"/>
        </mx:Form>

        <mx:ControlBar width="100%">
            <mx:Button label="Change State"
              click="currentState =
                currentState=='Register' ? '':'Register';"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

Register 状態に切り替えると、SetProperty クラスの value プロパティが registerValue 変数によって決定されます。ただし、このバインディングは Register ビューステートに切り替えたときにのみ実行されます。Register ビューステートに切り替えた後に registerValue 変数の値を変更しても、ターゲットコンポーネントの title プロパティは更新されません。

コンポーネントの追加と削除

AddChild クラスと RemoveChild クラスを使用して、子コンポーネントを追加および削除できます。ActionScript を使用してコンポーネントを追加する場合は、AddChild クラスを使用する前にコンポーネントを作成する必要があります。子を削除しても、完全に削除するわけではありません。削除した子を再作成することなく、後で再表示できます。

AddChild クラスと RemoveChild クラスの使用

子を追加する場合、relativeTo プロパティを使用して新しい子を作成する対象のコンテナを指定します。デフォルト値は、ビューステートを定義するアプリケーションまたはカスタムコンポーネントです。たとえば、HBox コンテナに Button コントロールを追加する場合は、relativeTo プロパティの値としてこのコンテナを指定します。

position プロパティを使用すると、コンテナ内での子の位置を指定できます。有効な値は、before、after、firstChild、および lastChild です。デフォルト値は lastChild です。

次の例は、h1 という名前の HBox コンテナの最初の子として Button コントロールを追加します。

```
<?xml version="1.0"?>
<!-- states\StatesAddRelative.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:states>
        <mx:State name="NewButton">
            <mx:AddChild relativeTo="{h1}" position="firstChild">
                <mx:Button id="buttonNew" label="New Button"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <mx:HBox id="h1">
        <mx:Button label="Change State"
            click="currentState =
                currentState=='NewButton' ? '':'NewButton';"/>
    </mx:HBox>
</mx:Application>
```

ビューステートを使用して、コンポーネントの親を直接変更することはできません。たとえば、あるコンポーネントをあるコンテナから別のコンテナに移動する場合は、RemoveChild クラスを使用して子を親コンテナから削除してから、AddChild クラスを使用してこのコンポーネントを別のコンテナに追加する必要があります。次の例を参照してください。

```
<?xml version="1.0"?>
<!-- states\StatesAddRemove.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:states>
        <mx:State name="NewParent">
            <mx:RemoveChild target="{button1}"/>
            <mx:AddChild target="{button1}" relativeTo="{v2}"/>
        </mx:State>
    </mx:states>

    <mx:VBox id="v1" borderStyle="solid">
        <mx:Label text = "VBox v1"/>
        <mx:Button id="button1"/>
    </mx:VBox>

    <mx:VBox id="v2" borderStyle="solid">
        <mx:Label text = "VBox v2"/>
        <mx:Button label="Change Parent"
            click="currentState =
                currentState=='NewParent' ? '':'NewParent';"/>
    </mx:VBox>
</mx:Application>
```

追加された子を作成するタイミングの制御

Flex のデフォルトの設定では、アプリケーションを迅速にロードできるように、初めて子が必要になる時点で子が作成されます。しかし、子の作成に時間がかかると、状態が変わるときにユーザーが遅延に気づきます。それを避けるには、状態が変わる前に子を作成し、アプリケーションの外見上の速度や応答性を向上させます。Flex では 3 つの方法で子を作成できます。

- 最初に必要になる時点で自動的に子を作成する (デフォルト)。
- アプリケーションが最初にロードする時点で自動的に子を作成する。
- アプリケーションに明示的に子を作成させる。

子が作成されるタイミングは、作成ポリシーで指定します。次のセクションでは、ビューステートにより作成される子の作成ポリシーを決定する方法を説明します。作成ポリシーと子の作成の制御に関する一般的な情報については、『Flex 2 アプリケーションの構築および展開ガイド』の第 6 章の「起動時のパフォーマンスの向上」を参照してください。

AddChild の子を指定するプロパティについて

AddChild クラスには、追加する子クラスを指定できる 2 つの相互排他的なプロパティがあります。

target　追加するコンポーネントを直接的に指定します。このプロパティを使用すると、アプリケーションが起動した時点で子が作成されます。このプロパティと一緒に他の作成ポリシーを使用することはできません。このプロパティを使用することは、targetFactory プロパティとして Flex コンポーネントを指定し、all 作成ポリシーを使用することと同じです。

targetFactory　次のアイテムのいずれかを指定します。

- IDeferredInstance インターフェイスを実装し、子インスタンスを作成するファクトリクラス。
- Button コントロールなどの Flex コンポーネント (UIComponent クラスのサブクラスとなるあらゆるクラス)。Flex コンポーネントを使用する場合、Flex コンパイラではファクトリクラス内でコンポーネントを自動的にラップします。

targetFactory プロパティについて

targetFactory プロパティは AddChild クラスのデフォルトプロパティです。したがって、次の例のように、<mx:AddChild> タグ内で MXML タグを使用して子を指定する場合は、自動的に targetFactory プロパティが使用され、必要なファクトリクラスが生成されます。

```
<mx:AddChild relativeTo="{v1}">
  <mx:Button id="b0" label="New Button" />
</mx:AddChild>
```

targetFactory プロパティでは、子コンポーネントの作成ポリシーを指定できます。
targetFactory プロパティと共に次のプロパティと AddChild クラスのメソッドを使用できます。

creationPolicy property　どの時点で子を作成するかを、次の値を使用して指定します。

- auto　最初に追加された時点でインスタンスが作成されます。これがデフォルト値です。スクリプトでは、mx.core.ContainerCreationPolicy.AUTO 定数を使用してこのプロパティを設定してください。
- all　アプリケーションの起動時にインスタンスが作成されます。スクリプトでは、ContainerCreationPolicy.ALL 定数を使用してこのプロパティを設定してください。
- none　アプリケーションで AddChild クラスの createInstance メソッドを呼び出して、子のインスタンスを作成する必要があります。スクリプトでは、mx.core.ContainerCreationPolicy.NONE 定数を使用してこのプロパティを設定してください。

createInstance() メソッド　ファクトリクラスを使用して子のインスタンスを作成します。
creationPolicy の値を none に設定すると、このメソッドが呼び出されて、子インスタンスが作成されます。createInstance() メソッドは任意の作成ポリシーで呼び出すことができます。ただし、子が既に作成されている場合は、何も実行されません。たとえば、creationPolicy の値が auto であっても、その状態に移行する前に子にアクセスする必要がある場合は、createInstance() メソッドを呼び出すことにより、子が作成されていることを確認できます。

次の例では、子を追加し、アプリケーションの起動時に子インスタンスを作成します。AddChild タグの状態がアクティブになるまで、子は表示されません。

```
<mx:AddChild relativeTo="{v1}" position="lastChild" creationPolicy="all">
  <mx:Button id="b0" label="New Button"/>
</mx:AddChild>
```

例 : Button コントロールの動的な追加

次の例では、基本状態、静的に作成された Button コントロールを追加する状態、およびその状態に移行するまで作成されない Button コントロールを追加する状態の、3 つの状態を切り替えることができます。

```
<?xml version="1.0"?>
<!-- states\StatesDefInstan.mxml  -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="initializeApplication();">

    <mx:Script>
        <![CDATA[
            import mx.core.*;
            import mx.states.*;

            private function initializeApplication():void {
                //Code for adding a static button.
                // Create the Button to be added
                var newButton:Button = new Button();
                newButton.label = "New Button";

                // Create an AddChild object that adds the button.
                var addChild:AddChild = new AddChild();
                addChild.relativeTo = myPanel;
                addChild.target = newButton;

                // Create a State object
                var newState:State = new State();
                newState.name = "added";
                newState.overrides = new Array();
                newState.overrides.push(addChild);

                // Initialize the states property of the Application.
                // and add the new state.
                states = new Array ();
                states.push(newState);

                //Code for adding a dynamically created button.
                // Create an add AddChild object that adds the button.
                var addChildDyn:AddChild = new AddChild();
                addChildDyn.relativeTo = myPanel;

                // Explicitly set the AddChild targetFactory property
                // to a factory
```

```
                // that runs the createMyButton method to create the button.
                addChildDyn.targetFactory =
                    new DeferredInstanceFromFunction(createMyButton);

                // Create State object with the addChildDyn override.
                var newState2:State = new State();
                newState2.name = "dynAdded";
                newState2.overrides = new Array();
                newState2.overrides.push(addChildDyn);

                //Add the state to the Application's states property.
                states.push(newState2);
            }

            // Function to create a new Button control.
            public function createMyButton():Object {
                var newButton:Button = new Button();
                newButton.label = "New Dynamic Button";
                return newButton;
            }

        ]]>
    </mx:Script>
    <mx:Panel id="myPanel"
        title="Static and dynamic states"
        width="300" height="150">
        <!-- If the current state is not the added state, change to the
            added state; if the current state is the added state,
            change to the base state. -->
        <mx:Button id="myButton" label="Toggle Static Button"
            click="currentState = currentState == 'added' ? '' : 'added';"/>
        <!-- If the current state is not the dynAdded state, change to the
            dynAdded state; if current state is the dynAdded state,
            change to the base state. -->
        <mx:Button id="myButton2"
            label="Toggle Dynamic Button"
            click="currentState =
                currentState == 'dynAdded' ? '' : 'dynAdded';"/>
    </mx:Panel>
</mx:Application>
```

イベントリスナーの設定

Flex では、アプリケーション内で状態固有のイベントリスナーを定義できます。状態固有のイベントリスナーは、特定のビューステートでのみアクティブになります。たとえば、基本ビューステートではあるイベントリスナーを使用し、ビューステートを変更したときは別のイベントハンドラを使用する Button コントロールを定義できます。

状態固有のイベントリスナーを設定する方法は 2 通りあります。

- MXML では、SetEventHandler クラスの `handler` イベントタイプを使用して、イベントリスナーを指定します。
- ActionScript または MXML では、SetEventHandler クラスの `handlerFunction` プロパティを使用して、イベントリスナーを指定します。

handler イベントタイプの使用

`handlerFunction` プロパティを使用せずに `handler` イベントタイプを使用すると次の利点があります。

- イベントリスナーが複数のパラメータを取ることができます。`handlerFunction` プロパティを使用する場合、このハンドラ関数がパラメータに取ることができるイベントオブジェクトは 1 つだけです。
- `<mx:SetEventHandler>` タグ内に ActionScript コードを直接記述して指定できるので、別のイベントリスナー関数を定義する必要がありません。警告ダイアログボックスをポップアップ表示するなどの単純なイベントリスナーを使用する場合に便利です。

MXML でのみ `handler` イベントタイプを使用できます。`handler` イベントタイプを使う場合、`name` プロパティを使用してイベント名を指定します。

次の例は、`handler` プロパティを使用して、Button コントロールの `click` イベントの `<mx:SetEventHandler>` タグ内に、イベントリスナーコードを直接記述して指定する方法を示しています。

```
<mx:SetEventHandler target="{myButton1}" name="click" handler="Alert.show('Hello
  World.')"/>
```

次の例では、イベントとユーザー ID 変数の 2 つのパラメータを取るハンドラ関数を指定する方法を示します。

```
<mx:SetEventHandler target="{myButton1}" name="click" handler="myAlert(event,
  userID)"/>
```

handlerFunction プロパティの使用

ActionScript または MXML で handlerFunction プロパティを使用できます。このプロパティを使用して、1 つの flash.events.Event 属性を取るイベントリスナー関数を指定できます。次の例では、イベントリスナーを設定します。

```
<mx:SetEventHandler target="{myButton1}" name="click"
  handlerFunction="handler2"/>
```

次に示すように、handler2 イベントリスナーでは宣言が必要です。

```
private function handler2(theEvent:Event)
```

例 : イベントリスナーの設定

次の例では、基本ビューステートと 2 つの追加ビューステートを作成します。状態ごとに SetEventHandler クラスが使用され、b1 という名前の Button コントロールの click イベントに対して異なるイベントリスナーが定義されます。

テキスト領域にある Buttons コントロールで、ベースステートを含む現在の状態を選択できます。各ビューステートのイベントハンドラによって、その状態に関する情報と、イベントリスナーに渡された第 2 パラメータの値が表示されます。

```
<?xml version="1.0"?>
<!-- states\StatesEventHandlersSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    currentState="MXMLState2">

    <mx:Script>
        <![CDATA[
            import flash.events.Event;

            // Functions for setting click event listeners.
            private function handlerBase(theEvent:Event,
                theString:String):void {
                ta1.text = "Default click handler" +
                    "\nParameter 2: " + theString;
            }

            private function handlerMXMLState2(theString:String):void {
                ta1.text = "MXML state handler function" +
                    "\nParameter 2: " + theString;
            }
        ]]>
    </mx:Script>

    <!-- The Application's states property defines two view states. -->
    <mx:states>
        <mx:State name="MXMLState1">
            <mx:SetEventHandler name="click" target="{b1}"
                handler="ta1.text='MXML-inline event listener \n
                    No second parameter';"/>
```

```
        </mx:State>

        <!-- The mxmlState2 State specifies a click event listener
            that takes a second parameter. -->
        <mx:State name="MXMLState2">
            <mx:SetEventHandler name="click" target="{b1}"
              handler="handlerMXMLState2('MXML-defined event listener');"/>
        </mx:State>
    </mx:states>

    <mx:Button id="b1"
        label="Click Me"
        click="handlerBase(event, 'Hi there');"/>
    <mx:TextArea id="ta1" height="100" width ="50%"/>
    <mx:HBox>
        <mx:Button id="b2"
            label="MXML state, inline handler"
            click="currentState='MXMLState1';"/>
        <mx:Button id="b3"
            label="MXML state, handler function"
            click="currentState='MXMLState2';"/>
        <mx:Button id="b4"
            label="Base state"
            click="currentState='';"/>
    </mx:HBox>
</mx:Application>
```

ビューステートイベントの使用

コンポーネントの currentState プロパティが変更されると、移行および終了しようとしている状態に対応する State オブジェクトから次のイベントが送出されます。

enterState　あるビューステートに移行し、そのビューステートがまだ完全に適用されていないときに送出されます。ビューステートが入力された後に State オブジェクトから送出され、基本ビューステートに戻った後にコンポーネントから送出されます。

exitState　ビューステートが終了しようとしているときに送出されます。ビューステートが終了する前に State オブジェクトから送出され、基本ビューステートを終了する前にコンポーネントから送出されます。

状態を変更するために currentState プロパティを変更したコンポーネントによって、次のイベントが送出されます。

currentStateChanging　ビューステートが変更されようとしているときに送出されます。これは、currentState プロパティが変更された後、ビューステートが変化する前に送出されます。このイベントを使用すると、新しいビューステートに必要な任意のデータをサーバーに要求できます。

currentStateChange　ビューステートが完全に変更された後に送出されます。currentState プロパティが変更された後にコンポーネントから送出されます。このイベントを使用すると、ユーザーの現在のビューステートを示すデータをサーバーに返すことができます。

例 : 履歴管理とビューステートの使用

Flex History Manager を使用すると、ユーザーは、Web ブラウザの [戻る] および [進む] ナビゲーション機能を使用して、Flex アプリケーション内を移動できます。History Manager は、アプリケーションがある状態にいつ移行したかをトラッキングできます。これによりユーザーは、たとえばあるアプリケーションプロセス内の異なる段階に対応する複数の状態の間を、ブラウザを使用して移動できます。

履歴管理によるアプリケーション状態のトラッキングを有効にするには、アプリケーションによって以下が実行される必要があります。

- `loadState` および `saveState` メソッドを定義することによって、IHistoryState インターフェイスを実装します。
- アプリケーションを History Manager に登録します。
- 状態が変化するたびに、`HistoryManager.save` メソッドを呼び出します。

次のコードは、History Manager によって検索がトラッキングされるように検索インターフェイスを記述する方法を示しています。履歴管理の詳細については、1043 ページ、第 32 章の「History Manager の使用」を参照してください (状態のトラッキングに関する別の例も記載されています)。

```
<?xml version="1.0"?>
<!-- states\StatesHistoryManager.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="mx.managers.IHistoryManagerClient"
    creationComplete="initApp();">

    <mx:Script>
        <![CDATA[
        import mx.managers.HistoryManager;

        //////////////////////////
        // IHistoryState methods
        //////////////////////////
        // Restore the state and searchString value.
        public function loadState(state:Object):void {
            if (state) {
                currentState = state.currentState;
                searchString = searchInput.text = state.searchString;
            }
            else {
                currentState = '';
            }
        }
        // Save the current state and the searchString value.
        public function saveState():Object {
            var state:Object = {};
            state.currentState = currentState;
            state.searchString = searchString;
```

```
            return state;
        }

        //////////////////////////
        // App-specific scripts
        //////////////////////////
        // The search string value.
        [Bindable]
        public var searchString:String;

        // Register the application with the history manager
        // when the application is created.
        public function initApp():void {
            HistoryManager.register(this);
        }

        // The method for doing the search.
        // For the sake of simplicity it doesn't do any searching.
        // It does change the state to display the results box,
        // and save the new state in the history manager.
        public function doSearch():void {
            currentState = "results";
            searchString = searchInput.text;
            HistoryManager.save();
        }
        // Method to revert the state to the base state.
        // Saves the new state in the history manager.
        public function reset():void {
            currentState = '';
            searchInput.text = "";
            searchString = "";
            HistoryManager.save();
        }
        ]]>
    </mx:Script>

    <mx:states>
        <!-- The state for displaying the search results -->
        <mx:State name="results">
            <mx:SetProperty target="{p}" name="width" value="100%" />
            <mx:SetProperty target="{p}" name="height" value="100%" />
            <mx:SetProperty target="{p}" name="title" value="Results" />
            <mx:AddChild relativeTo="{searchFields}">
                <mx:Button label="Reset" click="reset()" />
            </mx:AddChild>
            <mx:AddChild relativeTo="{p}">
                <mx:Label text="Search results for {searchString}" />
            </mx:AddChild>
        </mx:State>
    </mx:states>
```

```
    <!-- In the base state, just show a panel
        with a search text input and button. -->
    <mx:Panel id="p" title="Search" resizeEffect="Resize">
        <mx:HBox id="searchFields" defaultButton="{b}">
            <mx:TextInput id="searchInput" />
            <mx:Button id="b" label="Go" click="doSearch();" />
        </mx:HBox>
    </mx:Panel>
</mx:Application>
```

例 : カスタムアイテムレンダラーとビューステートの使用

次のコードは、カスタムアイテムレンダラーを使用してカタログアイテムを表示するアプリケーションを示しています。ユーザーがマウスをアイテム上に移動すると、写真がわずかに拡大される状態にアイテムレンダラーが変化し、価格がセルに表示され、アプリケーションのテキストボックスにそのアイテムに関する情報が表示されます。すべての変化は、親アプリケーションの変化を含めて、アイテムレンダラーの状態によって行われます。

```
<?xml version="1.0"?>
<!-- states\StatesRendererMain.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="450" height="500">
    <mx:Script>
        <![CDATA[
            import mx.controls.listClasses.*;
            import mx.controls.Image;
            import mx.collections.ArrayCollection;

            //The data to display in the TileList control.
            [Bindable]
            public var catArray:ArrayCollection = new ArrayCollection([
                { name: "USB Watch",
                  data: "1",
                  price: "129.99",
                  image: "assets/usbwatch.jpg",
                  description: "So, you need to tell the time of course" },
                { name: "007 Digital Camera",
                  data: "1",
                  price: "99.99",
                  image: "assets/007camera.jpg",
                  description: "Just like 007 used" },
                { name: "2-Way Radio Watch",
                  data: "1",
                  price: "49.99",
                  image: "assets/radiowatch.jpg",
                  description: "Better than Dick Tracy's" },
                { name: "USB Desk Fan",
                  data: "1",
```

```
                price: "19.99",
                image: "assets/usbfan.jpg",
                description: "Computer-powered cool!!!"},
            ]);
        ]]>
    </mx:Script>

    <mx:TileList id="myList"
        dataProvider="{catArray}"
        columnWidth="150"
        rowHeight="150"
        width="310"
        height="310"
        itemRenderer="myComponents.ImageComp"/>

    <!-- The item renderer fills in the TextArea control. -->
    <mx:HBox>
        <mx:Label text="Description"/>
        <mx:TextArea id="t1"
            width="200" height="100"
            wordWrap="true"/>
    </mx:HBox>
</mx:Application>
```

次のコードは、ファイル "imagecomp.mxml" にアイテムレンダラーを定義します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- states\myComponents\ImageComp.mxml
    When the mouse pointer goes over a cell,
        this component changes its state to showdesc.
    When the mouse pointer goes out of the cell,
        the component returns to the base state. -->

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="center"
    verticalAlign="top"
    mouseOver="currentState='showdesc'"
    mouseOut="currentState=''">

    <!-- In the base state, display the image and the label. -->
    <mx:Image id="img1"
        source="{data.image}"
        width="75"
        height="75"/>
    <mx:Label text="{data.name}"/>

    <mx:states>
        <mx:State name="showdesc">
          <!-- In the showdesc state, add the price, make the image bigger,
             and put the description in the parent application's TextArea.-->
            <mx:AddChild>
```

```
                <mx:Text text="{data.price}"/>
            </mx:AddChild>
            <mx:SetProperty target="{img1}" name="width" value="85"/>
            <mx:SetProperty target="{img1}" name="height" value="85"/>
            <mx:SetProperty target="{parentApplication.t1}" name="text"
                value="{data.description}"/>
        </mx:State>
    </mx:states>
</mx:VBox>
```

ビューステートを使用したアプリケーションの構築

このセクションで説明する機能やテクニックを活用して、高効率で多機能のビューステートベースのコンポーネントやアプリケーションを構成および構築できます。

カスタムコンポーネントでのビューステートの使用

複数のビューステートを1つのコンポーネントまたは一連のコンポーネントのセットに適用する場合、そのコンポーネントで構成されるカスタムコンポーネントを定義し、そのコンポーネント定義でビューステートを指定します。Application タグレベルでは指定しません。たとえば、あるビューステートで HBox コンテナに Button コントロールが追加される場合は、HBox をカスタムコンポーネントにし、そのカスタムコンポーネントでボタンが追加されるビューステートを定義します。HBox コンテナの変更だけが実行される状態が複数ある場合は、特にこの方法をお勧めします。

同様に、1つのカスタムコンポーネントに複数のビューステートがある場合、メインアプリケーションではなく、コンポーネントコードでビューステートを定義します。たとえば、カスタムの TitleWindow コンポーネントに折り畳まれたビューステートと展開したビューステートがある場合、2 つのビューステートはメインアプリケーションファイルではなくパネルの MXML コンポーネントで定義します。

そうすることによって、ビューステートが適用される個々のコンポーネントだけにビューステート定義を指定することができます。アイテムレンダラーは、複数のビューステートを持つ MXML コンポーネントを巧みに使用した例です。

次の例では、折り畳まれた状態と展開した状態の 2 つのビューステートを持つ TitleWindow コンポーネントのカスタムコンポーネントを示します。

```
<?xml version="1.0"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    close="checkCollapse();" headerHeight="40" showCloseButton="true">

    <mx:Script>
        <![CDATA[

            // Skins for the close button when the
```

```
            // TitleWindow container is collapsed.
            [Embed(source="closeButtonUpSkin.jpg")]
            [Bindable]
            public var closeBUp:Class;

            [Embed(source="closeButtonDownSkin.jpg")]
            [Bindable]
            public var closeBDown:Class;

            [Embed(source="closeButtonOverSkin.jpg")]
            [Bindable]
            public var closeBOver:Class;

            private function checkCollapse():void {
                currentState =
                    currentState == "collapsed" ? "" : "collapsed";
            }
    ]]>
    </mx:Script>

    <mx:states>
        <mx:State name="collapsed">
            <mx:SetProperty
                name="height"
                value="{getStyle('headerHeight')}"/>
            <mx:SetStyle
                name="closeButtonUpSkin"
                value="{closeBUp}"/>
            <mx:SetStyle
                name="closeButtonDownSkin"
                value="{closeBDown}"/>
            <mx:SetStyle
                name="closeButtonOverSkin"
                value="{closeBOver}"/>
        </mx:State>
    </mx:states>
</mx:TitleWindow>
```

この例では、コンポーネントが折り畳まれた状態のときは、デフォルトの閉じるアイコンがTitleWindow に置き換えられます。

その後は、アプリケーションのどの場所でもこのコンポーネントを使用できます。次の例に示します。

```
<?xml version="1.0"?>
<!-- states\StatesTitleWindowMain.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:MyComp="myComponents.*">

    <MyComp:StateTitleWindow id="myP" title="My Application">
        <mx:HBox width="100%">
            <mx:Button/>
            <mx:TextArea/>
```

```
        </mx:HBox>

        <mx:ControlBar width="100%">
            <mx:Label text="Quantity"/>
            <mx:NumericStepper/>
            <!-- Use Spacer to push Button control to the right. -->
            <mx:Spacer width="100%"/>
            <mx:Button label="Add to Cart"/>
        </mx:ControlBar>
    </MyComp:StateTitleWindow>
</mx:Application>
```

基本以外の初期ビューステートの使用

デフォルトでは、Flex アプリケーションは基本ビューステートで起動します。基本ビューステートは、currentState プロパティが "" に設定された状態に対応します。

ただし、アプリケーションまたはコンポーネントの初期ビューステートを、基本以外のビューステートに設定することもできます。たとえば、検索インターフェイスコンポーネントで最も頻繁に使用されるインターフェイスは、最初の展開されたビューであって、折り畳まれたビューではありません。

基本ビューステート以外を初期ビューステートとして使用するには、currentState プロパティを特定のビューステートに設定します。たとえば、次のようなコードを使用して、コンポーネントの初期状態を折り畳まれた状態に指定します。

```
<mx:VBox currentState="collapsed">
```

ビューステートベースアプリケーションに関する追加の手法

ビューステートベースアプリケーションの構築と実装では次のようなテクニックも活用できます。

- ビューステートは、ユーザーの操作に応答してコントロールの外観を変える手段の 1 つです。複数のコンポーネントに影響する変更を実行するときは、Accordion、Tab Navigator、ViewStack コンテナなどのナビゲータコンテナも使用できます。たとえば、ユーザーが複雑なフォームを迷わず移動できるようにするために、ビューステートの代わりに Accordion コンテナを使用できます。ナビゲータコンテナの詳細については、583 ページ、第 16 章の「ナビゲータコンテナの使用」を参照してください。
- カスケーディングのビューステート定義を作成します。basedOn プロパティを使用して、あるビューステートの上に別のビューステートを明示的に配置します。このテクニックは、共通のエレメントをいくつか含む複数のビューステートがある場合に有効です。すべての共通エレメントを 1 つのビューステートに含ませ、その上に別のビューステートを配置します。927 ページの「使用例 : ログインインターフェイス」で紹介したドリルダウン形式の検索インターフェイスでは、この手法を活用しています。

- 表示画面の主要なセクションを置き換えるビューステートを使用します。たとえば、ショッピングアプリケーションでビューステートを使用して、製品セレクタパネルとチェックアウト Accordion コントロールのどちらを画面に大きく表示するかを制御することができます。
- ビューステート間の変化を制御するトランジションを使用します。たとえば、次のコードを使用すると、ビューステートの変更時にサイズが変更されるすべてのコンポーネントに Resize エフェクトを追加できます。トランジションの詳細については、955 ページ、第 28 章の「トランジションの使用」を参照してください。

カスタムオーバーライドクラスの作成

IOverride インターフェイスを実装するクラスを作成することによって、カスタムオーバーライドを定義できます。たとえば、オブジェクトにぼかし効果を加えるフィルタなどのビットマップフィルタを追加するクラスを作成できます。

IOverride インターフェイスには次のメソッドが含まれます。

メソッド	説明
`initialize()`	オーバーライドを初期化します。
`apply()`	元の値を保存し、オーバーライドに適用します。
`remove()`	元の値に戻します。

次の例は、Flash BlurFilter クラスに適用してターゲットコンポーネントにぼかし効果を加える AddBlur オーバーライドクラスを示します。

```
package myOverrides
{
   import flash.display.*;
   import flash.filters.*;
   import mx.core.*;
   import mx.states.*;

   /* コンポーネントに Blur エフェクトを加えるオーバーライドを宣言します。*/
   public class AddBlur implements IOverride
   {
      /* コンストラクタです。*/
      public function AddBlur(
         target:DisplayObject = null)
      {
         this.target = target;
      }

      /* プロパティ : ぼかしの対象オブジェクトです。*/
      public var target:DisplayObject;
```

```
    /* この例では initialize() メソッドは空です。*/
    public function initialize():void {
    }

    /* apply メソッドによってフィルタ配列に BlurFilter が追加されます。*/
    public function apply(parent:UIComponent):void {
      var obj:DisplayObject = target ? target : parent;
      var filters:Array = obj.filters;

      filters.push(new BlurFilter());
      obj.filters = filters;
    }

    /* remove メソッドによってフィルタ配列から BlurFilter が削除されます。*/
    public function remove(parent:UIComponent):void {
      var obj:DisplayObject = target ? target : parent;
      var filters:Array = obj.filters;

      filters.pop();
      obj.filters = filters;
    }
  }
}
```

第 28 章

トランジションの使用

28

ビューステートを使用すると、通常はユーザーの操作に対する応答として、アプリケーションの外観を変更できます。トランジションは、ビューステートの変化が画面上でどのように表示されるかを定義します。トランジションは、エフェクトクラスを使用して定義します。このとき、トランジションを処理するために明示的に設計された複数のエフェクトを組み合わせます。このトピックでは、トランジションを定義してビューステートを処理する方法について説明します。

トランジションを使用するには、エフェクトとビューステートの動作についての知識が必要です。エフェクトの詳細については、603 ページ、第 17 章の「ビヘイビアの使用」を参照してください。ビューステートの詳細については、925 ページ、第 27 章の「ビューステートの使用」を参照してください。

目次

トランジションについて

ビューステートでは、通常はユーザーの操作に対する応答として、アプリケーションの内容や外観を変更できます。ビューステートを使用するには、アプリケーションの基本ビューステートを1つ定義してから、この基本ビューステートへの変更を指定する追加のビューステートを定義します。追加のビューステートは複数定義できます。

ビューステートを変更すると、アプリケーションに対する視覚的な変更がすべて同時に実行されます。つまり、アプリケーションコンポーネントのサイズの変更、位置の移動、または外観の変更を行うと、コンポーネントがある状態から次の状態に瞬時に変化します。

これに対して、状態を視覚的にスムーズに変化させるため、変更を一定の時間をかけて行う必要がある場合があります。" トランジション " とは、ビューステートを変更するときに再生される、グループ化されたエフェクトのことです。

たとえば、アプリケーションでフォームが定義されているとします。このフォームでは、基本ビューステートでは数個のフィールドが表示され、展開されたビューステートではさらに多くのフィールドが表示されます。ここで、基本ビューステートから展開されたビューステートへの変化をスムーズに行うには、Resize エフェクトを使用してフォームを展開し、次に Fade エフェクトを使用して新しいフォームエレメントをゆっくりと画面に表示する Flex トランジションを定義します。

ユーザーがフォームを基本ビューステートに戻すことを選択すると、このトランジションにより、展開状態のフィールドが Fade エフェクトを使用して非表示にされ、フォームのサイズが Resize エフェクトを使用してゆっくりと元に戻されます。

トランジションを使用すると、フォームにも、フォームに追加されるフィールドにも、フォームから削除されるフィールドにも、エフェクトを適用できます。エフェクトは複数適用することもできます。複数のフィールドに同じエフェクトを適用することも、異なるエフェクトを適用することもできます。基本状態から展開状態に移行するときに再生されるエフェクトのセットと、展開状態から基本状態に移行するときに再生されるエフェクトのセットを、別々に定義することもできます。

トランジションとエフェクトの比較

トランジションはエフェクトに置き換わるものではありません。トランジションを使用している場合でも、単独のエフェクトをコンポーネントに適用し、エフェクトトリガまたは `playEffect()` メソッドを使用してこのエフェクトを呼び出すことができます。

トランジションを使用すると、ビューステートを変更したときに複数のエフェクトが同時にトリガされるように、エフェクトをグループ化できます。トランジションは、現在のビューステートに対する変更のみに作用するように設計されています。これは、ビューステートの変更が、通常は複数のコンポーネントの同時変更を意味するからです。

トランジションではエフェクトフィルタもサポートされます。" フィルタ " を使用すると、あるエフェクトターゲットが特定の方法で変更される場合のみ、そのエフェクトターゲットが再生されるように設定できます。たとえば、ビューステートへの変更の一部として、コンポーネントのサイズを変更する場合があります。フィルタを使用すると、サイズが変更されるコンポーネントのみに Blur エフェクトが適用されるように設定できます。

トランジションの定義

トランジションは、Transition クラスを使用して作成します。次の表は、Transition クラスのプロパティの定義です。

プロパティ	定義
`fromState`	このトランジションを適用するときの、変更元のビューステートを指定するストリングです。デフォルト値はアスタリスク ("*") です。これは任意のビューステートを意味します。
`toState`	このトランジションを適用するときの、変更後のビューステートを指定するストリングです。デフォルト値はアスタリスク ("*") です。これは任意のビューステートを意味します。
`effect`	このトランジションを適用するときに再生する Effect オブジェクトです。通常これは、Parallel エフェクトや Sequence エフェクトのように、複数のエフェクトを組み合わせたエフェクトです。

1 つのアプリケーションに複数の Transition オブジェクトを定義できます。UIComponent クラスに含まれる `transitions` プロパティを、Transition オブジェクトの配列に設定します。

次の例に示すように、3 つの Panel コンテナと 3 つのビューステートを持つアプリケーションを定義することができます。

基本ビュース

1 ビューステート

2 ビューステート

この例のトランジションを、次のように <mx:transitions> タグを使用して MXML で定義します。

```
<?xml version="1.0" ?>
<!-- transitions/DefiningTrans.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="400" >

    <!-- Define the two view states, in addition to the base state.-->
    <mx:states>
        <mx:State name="One">
            <mx:SetProperty target="{p1}" name="x" value="110"/>
            <mx:SetProperty target="{p1}" name="y" value="0"/>
            <mx:SetProperty target="{p1}" name="width" value="200"/>
            <mx:SetProperty target="{p1}" name="height" value="210"/>
            <mx:SetProperty target="{p2}" name="x" value="0"/>
            <mx:SetProperty target="{p2}" name="y" value="0"/>
            <mx:SetProperty target="{p2}" name="width" value="100"/>
            <mx:SetProperty target="{p2}" name="height" value="100"/>
            <mx:SetProperty target="{p3}" name="x" value="0"/>
            <mx:SetProperty target="{p3}" name="y" value="110"/>
            <mx:SetProperty target="{p3}" name="width" value="100"/>
            <mx:SetProperty target="{p3}" name="height" value="100"/>
        </mx:State>
        <mx:State name="Two">
            <mx:SetProperty target="{p2}" name="x" value="110"/>
            <mx:SetProperty target="{p2}" name="y" value="0"/>
            <mx:SetProperty target="{p2}" name="width" value="200"/>
            <mx:SetProperty target="{p2}" name="height" value="210"/>
            <mx:SetProperty target="{p3}" name="x" value="0"/>
            <mx:SetProperty target="{p3}" name="y" value="110"/>
            <mx:SetProperty target="{p3}" name="width" value="100"/>
            <mx:SetProperty target="{p3}" name="height" value="100"/>
        </mx:State>
    </mx:states>

    <!-- Define Transition array with one Transition object.-->
    <mx:transitions>
        <!-- Define a transition for changing from any state to any state.
        -->
        <mx:Transition id="myTransition" fromState="*" toState="*">
            <!-- Define a Parallel effect as the top-level effect.-->
            <mx:Parallel id="t1" targets="{[p1,p2,p3]}">
                <!-- Define a Move and Resize effect.-->
                <mx:Move  duration="400"/>
                <mx:Resize duration="400"/>
            </mx:Parallel>
        </mx:Transition>
    </mx:transitions>

    <!-- Define the Canvas container holdig the three Panel containers.-->
    <mx:Canvas id="pm" width="100%" height="100%" >
        <mx:Panel id="p1" title="One"
```

```
                x="0" y="0" width="100" height="100"
                click="currentState='One'" >
            <mx:Label fontSize="24" text="One"/>
        </mx:Panel>

        <mx:Panel id="p2" title="Two"
                x="0" y="110" width="100" height="100"
                click="currentState='Two'" >
            <mx:Label fontSize="24" text="Two"/>
        </mx:Panel>

        <mx:Panel id="p3" title="Three"
                x="110" y="0" width="200" height="210"
                click="currentState=''" >
            <mx:Label fontSize="24" text="Three"/>
        </mx:Panel>
    </mx:Canvas>
</mx:Application>
```

各 Panel コンテナの `click` イベントを使用して、ビューステートを変更します。アプリケーションのビューステートが変化すると、現在のビューステートと変更後のビューステートが一致する Transition オブジェクトが検索されます。この例では、`fromState` および `toState` プロパティを `"*"` に設定します。したがって、myTransition がすべての状態変更に適用されます。`fromState` および `toState` プロパティの設定の詳細については、960 ページの「複数のトランジションの定義」を参照してください。

ビューステートの変更に一致するトランジションが特定されると、そのトランジションによって定義される Move および Resize エフェクトが、エフェクトターゲットに適用されます。この例では、トランジションの最上位の Parallel エフェクト指定を使用して、ターゲットを 3 つの Panel コンテナとして指定します。エフェクトターゲットの設定の詳細については、961 ページの「エフェクトターゲットの定義」を参照してください。

Move および Resize エフェクトが、すべてのエフェクトターゲットに対してパラレル再生されます。これにより 3 つの Panel コンテナが新しい位置に移動し、同時にサイズが変更されます。最上位エフェクトを Sequence エフェクトとして定義することにより、Move および Resize エフェクトをパラレルにではなく順番に再生することもできます。

エフェクト、現在のビューステート、および変更後のビューステートに対して指定されたプロパティの情報を使用して、Move および Resize エフェクトのプロパティの開始値と終了値が特定されます。この例では、エフェクト定義のプロパティの指定を省略します。このため、`Move.xFrom`、`Move.yFrom`、`Resize.widthFrom`、および `Resize.heightFrom` プロパティの値は、各パネルコンテナの現在のサイズと位置を使用して特定されます。`Move.xTo`、`Move.yTo`、`Resize.widthTo`、および `Resize.heightTo` プロパティの値は、変更後のビューステートを使用して特定されます。詳細については、962 ページの「エフェクトの開始値および終了値の定義」を参照してください。

複数のトランジションの定義

1つのアプリケーション内に複数のトランジションを定義できます。これにより、特定のトランジションを、ビューステートの特定の変更に関連付けることができます。ビューステートの変更に関連付けてトランジションを指定するには、fromState および toState プロパティを使用します。

デフォルトでは、fromState プロパティも toState プロパティも "*" に設定されます。この設定では、トランジションがビューステートの任意の変更に適用されます。いずれのプロパティも、空のストリング ("") に設定できます。この設定は基本ビューステートに対応します。

変更元のビューステートを明示的に指定するには fromState プロパティを使用し、変更後のビューステートを明示的に指定するには toState プロパティを使用します。次の例を参照してください。

```
<mx:transitions>
  <!-- 任意のビューステートからログインビューステートへの変更の場合に再生します。 -->
  <mx:Transition id="toLoginFromAny" fromState="*" toState="login">
    ...
  </mx:Transition>

  <!-- 詳細ビューステートからログインビューステートへの変更の場合に再生します。 -->
  <mx:Transition id="toLoginFromDetails"  fromState="details"
  toState="login">
    ...
  </mx:Transition>

  <!-- 任意のビューステートからそれ以外の任意のビューステートへの変更の場合に再生します。 -->
  <mx:Transition id="toAnyFromAny" fromState="*" toState="*">
    ...
  </mx:Transition>
</mx:transitions>

<!-- ログインビューステートに移行し、トランジション toLoginFromAny を再生します。 -->
<mx:Button click="currentState="login";/>

<!-- 詳細ビューステートに移行し、トランジション toAnyFromAny を再生します。 -->
<mx:Button click="currentState="details";/>

<!-- ログインビューステートに移行し、トランジション toLoginFromDetails を再生します。これ
  は、詳細ビューステートからログインビューステートへの移行だったためです。 -->
<mx:Button click="currentState="login";/>

<!-- 基本ビューステートに移行し、トランジション toAnyFromAny を再生します。 -->
<mx:Button click="currentState='';/>
```

1つの状態変更が2つのトランジションに一致する場合は、toState プロパティが fromState プロパティよりも優先されます。複数のトランジションが一致する場合は、transition 配列の最初の定義が使用されます。

エフェクトターゲットの定義

957 ページの「トランジションの定義」のセクションで示した <mx:Transition> タグは、トランジションを構成するエフェクトを定義します。最上位エフェクトは、エフェクトに明示的にターゲットが定義されていないとき、トランジション内のエフェクトのターゲットコンポーネントを定義します。この例では、トランジションが、アプリケーションの 3 つの Panel コンテナすべてに対して実行されます。トランジションが最初の 2 つのパネルに対してのみ再生されるようにするには、Parallel エフェクトを次のように定義します。

```
<mx:Transition fromState="*" toState="*">
  <!-- Parallel エフェクトを最上位エフェクトとして定義します。-->
  <mx:Parallel id="t1" targets="{[p1,p2]}">
    <mx:Move duration="400"/>
    <mx:Resize duration="400"/>
  </mx:Parallel>
</mx:Transition>
```

3 番目のパネルをトランジションから削除したため、このパネルは Move および Resize エフェクトのターゲットではなくなります。したがって 3 番目のパネルは、ビューステートが変化するとき、画面上で新しい位置とサイズに瞬時に移行します。他の 2 つのパネルは、エフェクトの持続時間である 400 ミリ秒をかけて、サイズと位置がスムーズに変化します。

トランジション内のエフェクトの target または targets プロパティを使用して、次のようにエフェクトターゲットを明示的に指定することもできます。

```
<mx:Parallel id="t1" targets="{[p1,p2,p3]}">
  <mx:Move targets="{[p1,p2]}" duration="400"/>
  <mx:Resize duration="400"/>
</mx:Parallel>
```

この例では、Resize エフェクトはすべてのパネルに対して再生される一方、Move エフェクトは最初の 2 つのパネルに対してのみ再生されます。上のコードは次のように書き換えることもできます。

```
<mx:Parallel id="t1" targets="{[p1,p2]}">
  <mx:Move duration="400"/>
  <mx:Resize targets="{[p1,p2,p3]}" duration="400"/>
</mx:Parallel>
```

エフェクトの開始値および終了値の定義

他のエフェクトと同様、トランジション内のエフェクトにも、設定するために使用するプロパティがあります。たとえば、ほとんどのエフェクトには、Move エフェクトの xFrom、yFrom、xTo、yTo プロパティのような、対象のコンポーネントの開始および終了情報を定義するプロパティがあります。

トランジションで定義されているエフェクトは、実行するエフェクトのプロパティ値を決定する必要があります。トランジションのエフェクトプロパティの開始値および終了値は、次の規則に従って決定されます。

1. エフェクトでプロパティの値が明示的に定義されている場合は、次の例に示すように、トランジション内でそれらの値を使用します。

```
<mx:Transition fromState="*" toState="*">
  <mx:Sequence id="t1" targets="{[p1,p2,p3]}">
    <mx:Blur duration="100"  blurXFrom="0.0" blurXTo="10.0" blurYFrom="0.0"
  blurYTo="10.0"/>
   <mx:Parallel>
      <mx:Move duration="400"/>
      <mx:Resize duration="400"/>
    </mx:Parallel>
   <mx:Blur duration="100"  blurXFrom="10.0" blurXTo="0.0" blurYFrom="10.0"
  blurYTo="0.0"/>
  </mx:Sequence>
</mx:Transition>
```

   この例では、2 つの Blur フィルタがエフェクトのプロパティを明示的に定義します。

2. エフェクトで、エフェクトの開始値が明示的に定義されていない場合は、現在のビューステートでの定義に従い、コンポーネントの現在の設定から決定されます。

   ルール 1 の例では、Move および Resize エフェクトは開始値を定義していません。したがって開始値は、現在のビューステートにおけるエフェクトターゲットの、現在のサイズと位置から決定されます。

3. エフェクトで、エフェクトの終了値が明示的に定義されていない場合は、変更後のビューステートにおけるコンポーネントの設定から決定されます。

   ルール 1 の例では、Move および Resize エフェクトが、変更後のビューステートにおけるエフェクトターゲットのサイズと位置から終了値を決定します。場合によっては、変更後のビューステートでこれらの値が明示的に定義されています。変更後のビューステートでこれらの値が定義されていない場合は、基本ビューステートの設定から決定されます。

4. 明示的な値がなく、現在のビューステートまたは変更後のビューステートから値を決定できない場合、エフェクトはデフォルトのプロパティ値を使用します。

例 : トランジションの使用

このセクションの例では、ビューステートとトランジションを使用する完全なアプリケーションを示します。ここでは 957 ページの「トランジションの定義」のセクションで示した 3 パネルのアプリケーションを実装し、Blur エフェクトを追加します。

```
<?xml version="1.0"?>
<?xml version="1.0" ?>
<!-- transitions\DefiningTransWithBlurs.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="400" >

    <!-- Define the two view states, in addition to the base state.-->
    <mx:states>
        <mx:State name="One">
            <mx:SetProperty target="{p1}" name="x" value="110"/>
            <mx:SetProperty target="{p1}" name="y" value="0"/>
            <mx:SetProperty target="{p1}" name="width" value="200"/>
            <mx:SetProperty target="{p1}" name="height" value="210"/>
            <mx:SetProperty target="{p2}" name="x" value="0"/>
            <mx:SetProperty target="{p2}" name="y" value="0"/>
            <mx:SetProperty target="{p2}" name="width" value="100"/>
            <mx:SetProperty target="{p2}" name="height" value="100"/>
            <mx:SetProperty target="{p3}" name="x" value="0"/>
            <mx:SetProperty target="{p3}" name="y" value="110"/>
            <mx:SetProperty target="{p3}" name="width" value="100"/>
            <mx:SetProperty target="{p3}" name="height" value="100"/>
        </mx:State>
        <mx:State name="Two">
            <mx:SetProperty target="{p2}" name="x" value="110"/>
            <mx:SetProperty target="{p2}" name="y" value="0"/>
            <mx:SetProperty target="{p2}" name="width" value="200"/>
            <mx:SetProperty target="{p2}" name="height" value="210"/>
            <mx:SetProperty target="{p3}" name="x" value="0"/>
            <mx:SetProperty target="{p3}" name="y" value="110"/>
            <mx:SetProperty target="{p3}" name="width" value="100"/>
            <mx:SetProperty target="{p3}" name="height" value="100"/>
        </mx:State>
    </mx:states>

    <!-- Define the single transition for all view state changes.-->
    <mx:transitions>
        <mx:Transition fromState="*" toState="*">
            <mx:Sequence id="t1" targets="{[p1,p2,p3]}">
                <mx:Blur duration="100" blurXFrom="0.0" blurXTo="10.0"
                blurYFrom="0.0" blurYTo="10.0"/>
                <mx:Parallel>
                    <mx:Move  duration="400"/>
                    <mx:Resize duration="400"/>
                </mx:Parallel>
                <mx:Blur duration="100" blurXFrom="10.0" blurXTo="0.0"
```

```
                blurYFrom="10.0" blurYTo="0.0"/>
            </mx:Sequence>
        </mx:Transition>
    </mx:transitions>

    <!-- Define the Canvas container holding the three Panel containers.-->
    <mx:Canvas id="pm" width="100%" height="100%" >
        <mx:Panel id="p1" title="One"
                x="0" y="0" width="100" height="100"
                click="currentState='One'" >
            <mx:Label fontSize="24" text="One"/>
        </mx:Panel>

        <mx:Panel id="p2" title="Two"
                x="0" y="110" width="100" height="100"
                click="currentState='Two'" >
            <mx:Label fontSize="24" text="Two"/>
        </mx:Panel>

        <mx:Panel id="p3" title="Three"
                x="110" y="0" width="200" height="210"
                click="currentState='' " >
            <mx:Label fontSize="24" text="Three"/>
        </mx:Panel>
    </mx:Canvas>
</mx:Application>
```

トランジションを使用する場合のイベントの処理

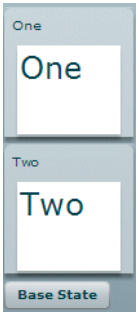
currentStateChange や currentStateChanging などのビューステートイベントは、トランジションを定義するアプリケーションの一部として処理できます。ビューステートへの変更、イベントの送出、およびトランジションエフェクトの再生は、次の順序で行われます。

1. currentState プロパティを変更後のビューステートに設定します。
2. Flex が currentStateChanging イベントを送出します。
3. Flex がトランジションのリストを調べ、ビューステートの変化に合致するトランジションを 1 つ決定します。
4. Flex がコンポーネントを調べ、エフェクトの開始値を決定します。
5. Flex が変更後のビューステートをアプリケーションに適用します。
6. Flex が currentStateChange イベントを送出します。
7. トランジションで定義したエフェクトを、Flex が再生します。

トランジション再生中にビューステートを再変更した場合は、新しい変更に関連付けられたトランジションが開始される前に、再生中のトランジションの末尾に Flex がジャンプします。

トランジション内でのアクションエフェクトの使用

次の例では、2 つのビューステートを持つアプリケーションを定義します。



基本ビューステート

OneOnly ビューステート

基本ビューステートから OneOnly ビューステートに移行するため、次のビューステート定義を作成します。

```
<mx:states>
  <mx:State name="OneOnly">
    <mx:SetProperty target="{p2}" name="visible" value="false"/>
    <mx:SetProperty target="{p2}"  name="includeInLayout" value="false"/>
  </mx:State>
</mx:states>
```

visible および includeInLayout プロパティの値を false に設定します。これにより、2 番目の Panel コンテナが非表示になり、アプリケーションのレイアウト時に無視されます。visible プロパティが false、includeInLayout プロパティが true の場合は、コンテナは非表示になりますが、アプリケーションは、このコンポーネントが表示されているかのようにレイアウトされます。

ビューステート定義では、ビューステートの変更方法を定義します。これに対してトランジションでは、視覚的な変更が行われる順序を定義します。前の図の例では、2 番目のパネルが非表示になるとき、および基本ビューステートに戻るトランジションで再表示されるとき、このパネルに Iris エフェクトを再生します。

基本ビューステートから OneOnly ビューステートへの変更に対して、toOneOnly トランジションを定義します。このトランジションは、Iris エフェクトを使用して 2 番目のパネルを非表示にしてから、このパネルの visible および includeInLayout プロパティを false に設定します。基本ビューステートに戻るトランジションに対して、toAnyFromAny トランジションを定義します。このトランジションは、2 番目のパネルの visible および includeInLayout プロパティを true に設定することによってこのパネルを表示可能にしてから、Iris エフェクトを使用してこのパネルを表示します。次の例を参照してください。

```
<mx:transitions>
  <mx:Transition id="toOneOnly" fromState="*" toState="OneOnly">
    <mx:Sequence id="t1" targets="{[p2]}">
      <mx:Iris showTarget="false" duration="350"/>
      <mx:SetPropertyAction name="visible"/>
      <mx:SetPropertyAction target="{p2}" name="includeInLayout"/>
    </mx:Sequence>
  </mx:Transition>

  <mx:Transition id="toAnyFromAny" fromState="*" toState="*">
    <mx:Sequence id="t2" targets="{[p2]}">
      <mx:SetPropertyAction target="{p2}" name="includeInLayout"/>
      <mx:SetPropertyAction name="visible"/>
      <mx:Iris showTarget="true" duration="350"/>
    </mx:Sequence>
  </mx:Transition>
</mx:transitions>
```

toOneOnly トランジションでは、ターゲットの visible プロパティを false に設定してから Iris エフェクトを再生しても、ターゲットが既に非表示になっているため、Iris エフェクトの再生は表示されません。

トランジション中にビューステートが変更される順序を制御するため、Flex ではいくつかの " アクションエフェクト " が定義されます。前の例では、トランジションで visible および includeInLayout プロパティを設定するタイミングを、SetPropertyAction アクションエフェクトを使用して制御しています。次の表に、これらのアクションエフェクトの説明を示します。

アクションエフェクト	対応するビューステートプロパティ	用途
SetPropertyAction	SetProperty	トランジションの一部としてプロパティ値を設定します。
SetStyleAction	SetStyle	トランジションの一部としてスタイルをある値に設定します。
AddChildAction	AddChild	トランジションの一部として子を追加します。
RemoveChildAction	RemoveChild	トランジションの一部として子を削除します。

ビューステートを定義するときは、SetProperty、SetStyle、AddChild、および RemoveChild プロパティを使用できます。ビューステートプロパティによって定義される変更が、トランジション内で実行されるタイミングを制御するには、対応するアクションエフェクトを使用します。アクションエフェクトを使用すると、状態の変更順序を制御できます。

前の例では、コンポーネントの visible プロパティの値が変更されるときに行われるアクションエフェクトを定義するため、次のステートメントを使用しました。

```
<mx:SetPropertyAction name="visible"/>
```

このアクションエフェクトは、visible プロパティの値が true または false に変更されると再生されます。次のように <mx:SetPropertyAction> タグの value プロパティを使用すると、さらにエフェクトを制御できます。

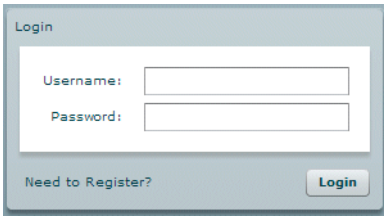
```
<mx:SetPropertyAction name="visible" value="true"/>
```

この例では、visible プロパティの値が true に変更されたときのみエフェクトを再生するように指定します。アクションエフェクトにこのタイプの情報を追加すると、トランジションにフィルタを使用する場合に役立ちます。詳細については、969 ページの「エフェクトへのフィルタ適用」を参照してください。

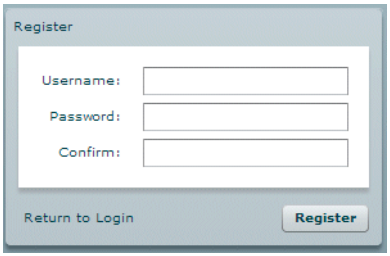
アクションエフェクトは、duration プロパティをサポートしません。アクションエフェクトは指定されたアクションのみを実行します。

例 : アクションエフェクトの使用

基本ログイン状態と登録状態の 2 つの状態を持つログインフォームを、次のように定義できます。



基本ビューステート



登録ビューステート

ビューステートでは、状態の変更方法を定義します。これに対してトランジションでは、アプリケーションが視覚的に変更される順序を定義します。たとえば、登録ビューステートへの視覚的な移行が、次の順序で行われるように定義します。

1. 登録フォームへの LinkButton コントロールを削除します。
2. フォームのタイトルを Register に変更します。
3. Button ラベルを Register に変更します。
4. パスワードを確認するための TextInput コントロールを追加します。
5. ログインフォームに戻るための LinkButton コントロールを追加します。

ユーザーがログインフォームに戻る場合、つまり基本ビューステートに戻る場合は、ログインフォームへの視覚的な移行が、次の順序で行われるように定義します。

1. ログインフォームへの LinkButton コントロールを削除します。
2. パスワードを確認するための TextInput コントロールを削除します。
3. フォームのタイトルを Login に変更します。
4. Button ラベルを Login に変更します。
5. 登録フォームに移行するための LinkButton コントロールを追加します。

次のトランジションでは、ログインフォームから登録フォームに状態を変更するとき、または登録フォームからログインフォームに状態を変更するときの視覚的な変更を、アクションエフェクトを使用して定義しています。この例では、どちらの状態変更にも同じトランジションを使用します。すべてのエフェクトは、トランジションの一部として順番に実行されます。

```
<?xml version="1.0" ?>
<!-- transitions\ActionTransitions.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalAlign="middle">

    <!-- Define one view state, in addition to the base state.-->
    <mx:states>
        <mx:State name="Register">
            <mx:AddChild relativeTo="{loginForm}" position="lastChild">
                <mx:target>
                    <mx:FormItem id="confirm" label="Confirm:">
                        <mx:TextInput/>
                    </mx:FormItem>
                </mx:target>
            </mx:AddChild>
            <mx:SetProperty target="{loginPanel}" name="title"
            value="Register"/>
            <mx:SetProperty target="{loginButton}" name="label"
            value="Register"/>
            <mx:RemoveChild target="{registerLink}"/>
            <mx:AddChild relativeTo="{spacer1}" position="before">
                <mx:target>
                    <mx:LinkButton id="loginLink"
                        label="Return to Login"
                        click="currentState=''"/>
                </mx:target>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <!-- Define Transition array with one Transition object.-->
    <mx:transitions>
        <mx:Transition id="toRegister" fromState="*" toState="Register">
            <mx:Sequence targets="{[loginPanel, registerLink, confirm,
```

```
            loginLink, spacer1]}">
                <mx:RemoveChildAction/>
                <mx:SetPropertyAction target="{loginPanel}" name="title"/>
                <mx:SetPropertyAction  target="{loginButton}" name="label"/>
                <mx:Resize target="{loginPanel}"/>
                <mx:AddChildAction/>
            </mx:Sequence>
        </mx:Transition>
    </mx:transitions>

    <!-- Define a Panel container that defines the login form.-->
    <mx:Panel title="Login" id="loginPanel"
            horizontalScrollPolicy="off"
            verticalScrollPolicy="off">
        <mx:Form id="loginForm" >
            <mx:FormItem label="Username:">
                <mx:TextInput/>
            </mx:FormItem>
            <mx:FormItem label="Password:">
                <mx:TextInput/>
            </mx:FormItem>
        </mx:Form>
        <mx:ControlBar>
            <mx:LinkButton id="registerLink"  label="Need to Register?"
                click="currentState='Register'"/>
            <mx:Spacer width="100%" id="spacer1"/>
            <mx:Button label="Login" id="loginButton"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

エフェクトへのフィルタ適用

デフォルトでは、トランジションで定義されているすべてのエフェクトが、そのトランジションのすべてのターゲットコンポーネントに適用されます。したがって、次の例では、Move および Resize エフェクトが 3 つのターゲットすべてに適用されます。

```
<mx:Transition fromState="*" toState="*">
  <!-- Parallel エフェクトを最上位エフェクトとして定義します。-->
  <mx:Parallel id="t1" targets="{[p1,p2,p3]}">
    <!-- Move および Resize エフェクトを定義します。-->
    <mx:Move duration="400"/>
    <mx:Resize duration="400"/>
  </mx:Parallel>
</mx:Transition>
```

あるエフェクトを、すべてのターゲットコンポーネントではなく、コンポーネントの特定のサブセットに適用する必要がある場合があります。3 つのビューステートを持つアプリケーションを、次のように定義します。

基本ビューステート

2 ビューステート

3 ビューステート

ビューステートを変更するたびに、最上部のパネルが削除され、最下部のパネルが最上部に移動し、画面最下部に次のパネルが追加されます。この例では、3 番目のパネルは基本ビューステートでは非表示です。

この例では、最上部のパネルが削除されるときにそのパネルに WipeUp エフェクトを適用し、最下部のパネルが最上部に移動するときにそのパネルに Move エフェクトを適用し、最下部に追加されるパネルに別の WipeUp エフェクトを適用するトランジションを 1 つ定義します。次の例を参照してください。

```
<mx:transitions>
  <mx:Transition fromState="*" toState="*">
    <mx:Sequence targets="{[p1,p2,p3]}">
      <mx:Sequence id="sequence1" filter="hide">
        <mx:WipeUp/>
        <mx:SetPropertyAction name="visible" value="false"/>
      </mx:Sequence>
      <mx:Move filter="move"/>
      <mx:Sequence id="sequence2" filter="show">
        <mx:SetPropertyAction name="visible" value="true"/>
        <mx:WipeUp/>
      </mx:Sequence>
    </mx:Sequence>
  </mx:Transition>
</mx:transitions>
```

sequence1 Sequence エフェクトでは、あるコンポーネントに対してこのエフェクトが再生されるために、そのコンポーネントに実行される必要がある変更が、filter プロパティを使用して指定されます。この例では、sequence1 エフェクトの filter プロパティに "hide" という値が指定されています。したがって、WipeUp および SetPropertyAction エフェクトは、visible プロパティが false に設定されることによって非表示に変更されるコンポーネントに対してのみ、再生されます。

2 番目の Sequence エフェクトでは、filter プロパティを show に設定します。したがって、WipeUp および SetPropertyAction エフェクトは、visible プロパティが true に設定されることによって表示可能に変更されるコンポーネントに対してのみ、再生されます。

Move エフェクトには、move という値の filter プロパティも指定されています。したがってこのエフェクトは、位置が変更されるすべてのターゲットコンポーネントに適用されます。

次の表に、filter プロパティに含まれる可能性がある値を示します。

値	説明
add	ビューステートの変更中に追加されるすべての子に対して、このエフェクトを再生します。
hide	ビューステートの変更中に visible プロパティが true から false に変更されるすべての子に対して、このエフェクトを再生します。
move	ビューステートの変更中に x または y プロパティが変更されるすべての子に対して、このエフェクトを再生します。
remove	ビューステートの変更中に削除されるすべての子に対して、このエフェクトを再生します。
resize	ビューステートの変更中に width または height プロパティが変更されるすべての子に対して、このエフェクトを再生します。
show	ビューステートの変更中に visible プロパティが false から true に変更されるすべての子に対して、このエフェクトを再生します。

例 : フィルタの使用

次の例では、969 ページの「エフェクトへのフィルタ適用」の例のすべてのコードを示します。

```
<?xml version="1.0" ?>
<!-- transitions\FilterShowHide.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  width="700" >

    <!-- Define two view state, in addition to the base state.-->
    <mx:states>
        <mx:State name="Two">
            <mx:SetProperty target="{p1}" name="visible" value="false"/>
            <mx:SetProperty target="{p2}" name="visible" value="true"/>
            <mx:SetProperty target="{p3}" name="visible" value="true"/>
            <mx:SetProperty target="{p2}" name="x" value="0"/>
            <mx:SetProperty target="{p2}" name="y" value="0"/>
            <mx:SetProperty target="{p3}" name="x" value="0"/>
            <mx:SetProperty target="{p3}" name="y" value="110"/>
        </mx:State>
        <mx:State name="Three">
            <mx:SetProperty target="{p1}" name="visible" value="true"/>
            <mx:SetProperty target="{p2}" name="visible" value="false"/>
            <mx:SetProperty target="{p3}" name="visible" value="true"/>
            <mx:SetProperty target="{p1}" name="x" value="0"/>
```

```
            <mx:SetProperty target="{p1}" name="y" value="110"/>
            <mx:SetProperty target="{p3}" name="x" value="0"/>
            <mx:SetProperty target="{p3}" name="y" value="0"/>
        </mx:State>
    </mx:states>

    <!-- Define a single transition for all state changes.-->
    <mx:transitions>
        <mx:Transition fromState="*" toState="*">
            <mx:Sequence targets="{[p1,p2,p3]}">
                <mx:Sequence id="sequence1" filter="hide" >
                    <mx:WipeUp/>
                    <mx:SetPropertyAction name="visible" value="false"/>
                </mx:Sequence>
                <mx:Move filter="move"/>
                <mx:Sequence id="sequence2" filter="show" >
                    <mx:SetPropertyAction name="visible" value="true"/>
                    <mx:WipeUp/>
                </mx:Sequence>
            </mx:Sequence>
        </mx:Transition>
    </mx:transitions>

    <mx:Canvas id="pm" width="100%" height="100%">
        <mx:Panel id="p1" title="One"
                x="0" y="0" width="100" height="100"
                click="currentState=''" >
            <mx:Label fontSize="24" text="One"/>
        </mx:Panel>

        <mx:Panel id="p2" title="Two"
                x="0" y="110" width="100" height="100"
                click="currentState='Two'" >
            <mx:Label fontSize="24" text="Two"/>
        </mx:Panel>

        <mx:Panel id="p3" title="Three"
                visible="false"
                width="100" height="100"
                click="currentState='Three'" >
            <mx:Label fontSize="24" text="Three"/>
        </mx:Panel>
    </mx:Canvas>
</mx:Application>
```

カスタムフィルタの定義

Flex では、EffectTargetFilter クラスを使用して、各トランジションエフェクトがそれぞれのターゲットに実行するカスタムフィルタを定義できます。次の表は、EffectTargetFilter クラスのプロパティの一覧です。

プロパティ	説明
filterProperties	コンポーネントプロパティを指定するストリングの配列です。この配列内のいずれかのプロパティがターゲットコンポーネントで変更された場合、ターゲットでエフェクトを再生します。
filterStyles	スタイルプロパティを指定するストリングの配列です。配列内のいずれかのスタイルプロパティがターゲットコンポーネントで変更された場合、ターゲットでエフェクトを再生します。
filterFunction	カスタムフィルタロジックを定義するコールバック関数への参照を含むプロパティ。エフェクトのすべてのターゲットに対してこのメソッドが呼び出されます。この関数によって true が返される場合、エフェクトはターゲットで再生されます。false が返される場合、ターゲットはエフェクトによりスキップされます。

このコールバックフィルタ関数のシグネチャは次のとおりです。

```
filterFunc(propChanges:Array, instanceTarget:Object):Boolean
{
  // instanceTarget でエフェクトを再生する場合は true を返します。
  // エフェクトを再生しない場合は false を返します。
}
```

この関数は、次のパラメータを受け取ります。

propChanges - PropertyChanges オブジェクトの配列で、エフェクトのターゲットコンポーネントごとに 1 つのオブジェクトを持ちます。ターゲットのプロパティがトランジションによって変更されない場合、このプロパティはこの配列に含まれません。

instanceTarget - フィルタ処理を実行するエフェクト特有のターゲットコンポーネント。

カスタムフィルタ関数を使用する例については、978 ページの「例 : カスタムエフェクトフィルタの使用」を参照してください。

フィルタを作成するには、EffectTargetFilter オブジェクトを定義してから、このオブジェクトをエフェクトの Effect.customFilter プロパティの値として指定します。次の例を参照してください。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="initFilter(event);" width="700">

  <mx:Script>
    <![CDATA[

      import mx.effects.EffectTargetFilter;
```

```
        // EffectTargetFilter オブジェクトを定義します。
        private var myBlurFilter:EffectTargetFilter;

        // EffectTargetFilter オブジェクトを初期化してから、
        // 2 つの Blur エフェクトに対して customFilter プロパティを設定します。
        private function initFilter(event:Event):void {
          myBlurFilter = new EffectTargetFilter();

          // x または width プロパティの値を変更する任意のターゲットに対して
          // このエフェクトを再生します。
          myBlurFilter.filterProperties=['x', 'width'];

          myBlur.customFilter=myBlurFilter;
          myUnBlur.customFilter=myBlurFilter;
        }
      ]]>
    </mx:Script>

    <mx:transitions>
      <mx:Transition fromState="*" toState="*">
        <mx:Sequence id="t1" targets="{[p1,p2,p3]}">
          <mx:Blur id="myBlur"/>
            <mx:Parallel>
              <mx:Move  duration="400"/>
              <mx:Resize duration="400"/>
            </mx:Parallel>
          <mx:Blur id="myUnBlur"/>
        </mx:Sequence>
      </mx:Transition>
    </mx:transitions>
    ...
</mx:Application>
```

カスタムフィルタは MXML で追加することもできます。次の例を参照してください。

```
<?xml version="1.0" ?>
<!-- transitions\EffectFilterExampleMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700">

    <mx:states>
        <mx:State name="One">
            <mx:SetProperty target="{p1}" name="x" value="110"/>
            <mx:SetProperty target="{p1}" name="y" value="0"/>
            <mx:SetProperty target="{p1}" name="width" value="500"/>
            <mx:SetProperty target="{p1}" name="height" value="210"/>
            <mx:SetProperty target="{p2}" name="x" value="0"/>
            <mx:SetProperty target="{p2}" name="y" value="0"/>
            <mx:SetProperty target="{p2}" name="width" value="100"/>
            <mx:SetProperty target="{p2}" name="height" value="100"/>
```

```
        <mx:SetProperty target="{p3}" name="x" value="0"/>
        <mx:SetProperty target="{p3}" name="y" value="110"/>
        <mx:SetProperty target="{p3}" name="width" value="100"/>
        <mx:SetProperty target="{p3}" name="height" value="100"/>
    </mx:State>
    <mx:State name="Two">
        <mx:SetProperty target="{p2}" name="x" value="110"/>
        <mx:SetProperty target="{p2}" name="y" value="0"/>
        <mx:SetProperty target="{p2}" name="width" value="500"/>
        <mx:SetProperty target="{p2}" name="height" value="210"/>
        <mx:SetProperty target="{p3}" name="x" value="0"/>
        <mx:SetProperty target="{p3}" name="y" value="110"/>
        <mx:SetProperty target="{p3}" name="width" value="100"/>
        <mx:SetProperty target="{p3}" name="height" value="100"/>
    </mx:State>
</mx:states>

<mx:transitions>
    <mx:Transition fromState="*" toState="*">
        <mx:Sequence id="t1" targets="{[p1,p2,p3]}">
            <mx:Blur id="myBlur" duration="100" blurXFrom="0.0"
            blurXTo="10.0" blurYFrom="0.0" blurYTo="10.0">
                <mx:customFilter>
                    <mx:EffectTargetFilter
                        filterProperties="['width','x']"/>
                </mx:customFilter>
            </mx:Blur>
            <mx:Parallel>
                <mx:Move  duration="400"/>
                <mx:Resize duration="400"/>
            </mx:Parallel>
            <mx:Blur id="myUnBlur" duration="100" blurXFrom="10.0"
            blurXTo="0.0" blurYFrom="10.0" blurYTo="0.0">
                <mx:customFilter>
                    <mx:EffectTargetFilter
                        filterProperties="['width','x']"/>
                </mx:customFilter>
            </mx:Blur>
        </mx:Sequence>
    </mx:Transition>
</mx:transitions>

<mx:Canvas id="pm" width="100%" height="100%">
    <mx:Panel id="p1" title="One"
            x="0" y="0" width="100" height="100"
            click="currentState='One'" >
        <mx:Label fontSize="24" text="One"/>
    </mx:Panel>

    <mx:Panel id="p2" title="Two"
```

```
                x="0" y="110" width="100" height="100"
                click="currentState='Two'" >
            <mx:Label fontSize="24" text="Two"/>
        </mx:Panel>

        <mx:Panel id="p3" title="Three"
                x="110" y="0" width="500" height="210"
                click="currentState='' " >
            <mx:Label fontSize="24" text="Three"/>
        </mx:Panel>
    </mx:Canvas>
</mx:Application>
```

フィルタ関数の記述

フィルタ関数を作成するには、EffectTargetFilter オブジェクトを定義してから、このオブジェクトをエフェクトの Effect.customFilter プロパティの値として指定します。次の例では、アプリケーションの creationComplete イベントを使用して EffectTargetFilter オブジェクトを初期化してから、このオブジェクトを 2 つある Blur エフェクトの customFilter プロパティの値として指定します。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="initFilter(event);" width="700">

  <mx:Script>
    <![CDATA[

      import mx.effects.EffectTargetFilter;
      import flash.events.Event;

      // この関数は、x=110 に移動する Panel に対して true を返します。
      public function filterFunc(propChanges:Array,
  instanceTarget:Object):Boolean
      {
        ...
      }

      // EffectTargetFilter オブジェクトを定義します。
      private var myBlurFilter:EffectTargetFilter;

      // EffectTargetFilter オブジェクトを初期化してから、
      // 2 つの Blur エフェクトに対して customFilter プロパティを設定します。
      private function initFilter(event:Event):void {
        myBlurFilter = new EffectTargetFilter();

        myBlurFilter.filterFunction=filterFunc;

        myBlur.customFilter=myBlurFilter;
        myUnBlur.customFilter=myBlurFilter;
      }
```

```
      ]]>
  </mx:Script>

  <mx:transitions>
    <mx:Transition fromState="*" toState="*">
      <mx:Sequence id="t1" targets="{[p1,p2,p3]}">
        <mx:Blur id="myBlur"/>
          <mx:Parallel>
            <mx:Move  duration="400"/>
            <mx:Resize duration="400"/>
          </mx:Parallel>
        <mx:Blur id="myUnBlur"/>
      </mx:Sequence>
    </mx:Transition>
  </mx:transitions>

  ...

</mx:Application>
```

フィルタ関数に渡される propChanges パラメータには、PropertyChanges オブジェクトの配列が含まれます。この配列には、エフェクトのターゲットコンポーネントにつき 1 つのオブジェクトが含まれます。次の表は、PropertyChanges クラスのプロパティの一覧です。

プロパティ	説明
target	エフェクトのターゲットコンポーネントです。PropertyChanges クラスの end および start プロパティは、ビューステートの変化によってターゲットコンポーネントがどのように変更されるかを定義します。
start	target コンポーネントの開始プロパティを含むオブジェクトです。現在のビューステートによって定義されます。たとえば、ビューステートの変化によって移動とサイズ変更の両方が行われる target コンポーネントの場合は、start プロパティにコンポーネントの開始位置とサイズが含まれます。次を参照してください。 {x:00, y:00, width:100, height:100}
end	target コンポーネントの終了プロパティを含むオブジェクトです。変更後のビューステートによって定義されます。たとえば、ビューステートの変化によって移動とサイズ変更の両方が行われる target コンポーネントの場合は、end プロパティにコンポーネントの終了位置とサイズが含まれます。次を参照してください。 {x:100, y:100, width:200, height:200}

カスタムフィルタ関数内では、instanceTarget 引数と propChanges.target プロパティを比較して、propChanges 配列で instanceTarget 引数と一致する PropertyChanges オブジェクトを最初に検索します。

次のフィルタ関数では、instanceTarget に対してエフェクトを再生するかどうかを判別するため、propChanges 配列を調べます。この例では、x プロパティが 110 の位置に移動するコンポーネントについてのみ、フィルタ関数が true を返します。次を参照してください。

```
// この関数は、x=110 に移動するターゲットに対して true を返します。
public function filterFunc(propChanges:Array, instanceTarget:Object):Boolean {

   // propChanges 配列の長さを判別します。
   for (var i:uint=0; i < propChanges.length; i++)
   {
      // エフェクトターゲットに一致する配列エレメントを判別します。
      if (propChanges[i].target == instanceTarget)
      {
         // end オブジェクトに x の値が含まれるかどうかをチェックします。
         if (propChanges[i].end["x"] != undefined)
         {
            // x の end 値が 110 の場合は true を返します。
            return (propChanges[i].end.x == 110);
         }
      }
   }
   // それ以外の場合は false を返します。
   return false;
}
```

例 : カスタムエフェクトフィルタの使用

次の例では、カスタムフィルタ関数を使用して、トランジションの 3 つのターゲットの 1 つに Blur エフェクトを適用します。他の 2 つのターゲットは Blur エフェクトによって変更されません。

Blur エフェクトのターゲットを判別するため、カスタムフィルタ関数によって各ターゲットの x プロパティが調べられます。Blur エフェクトは、x=110 に移動するコンポーネントに対してのみ再生されます。次を参照してください。

```
<?xml version="1.0" ?>
<!-- transitions\EffectFilterExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initFilter(event);"
    width="700">

    <mx:Script>
        <![CDATA[

            import mx.effects.EffectTargetFilter;
            import flash.events.Event;

            // This function returns true for the Panel moving to x=110.
            public function filterFunc(propChanges:Array,
                    instanceTarget:Object):Boolean {
                // Determine the length of the propChanges Array.
                for (var i:uint=0; i < propChanges.length; i++)
                {
                    // Determine the Array element
                    // that matches the effect target.
```

```
                if (propChanges[i].target == instanceTarget)
                {
                    // Check whether the end Object contains
                    // a value for x.
                    if (propChanges[i].end["x"] != undefined)
                    {
                        // Return true of the end value for x is 110.
                        return (propChanges[i].end.x == 110);
                    }
                }
            }

            return false;
        }

        // Define the EffectTargetFilter object.
        private var myBlurFilter:EffectTargetFilter;

        // Initialize the EffectTargetFilter object, and set the
        // customFilter property for two Blur effects.
        private function initFilter(event:Event):void {
            myBlurFilter = new EffectTargetFilter();

            myBlurFilter.filterFunction=filterFunc;

            myBlur.customFilter=myBlurFilter;
            myUnBlur.customFilter=myBlurFilter;
        }
      ]]>
</mx:Script>

<mx:states>
    <mx:State name="One">
        <mx:SetProperty target="{p1}" name="x" value="110"/>
        <mx:SetProperty target="{p1}" name="y" value="0"/>
        <mx:SetProperty target="{p1}" name="width" value="500"/>
        <mx:SetProperty target="{p1}" name="height" value="210"/>
        <mx:SetProperty target="{p2}" name="x" value="0"/>
        <mx:SetProperty target="{p2}" name="y" value="0"/>
        <mx:SetProperty target="{p2}" name="width" value="100"/>
        <mx:SetProperty target="{p2}" name="height" value="100"/>
        <mx:SetProperty target="{p3}" name="x" value="0"/>
        <mx:SetProperty target="{p3}" name="y" value="110"/>
        <mx:SetProperty target="{p3}" name="width" value="100"/>
        <mx:SetProperty target="{p3}" name="height" value="100"/>
    </mx:State>
    <mx:State name="Two">
        <mx:SetProperty target="{p2}" name="x" value="110"/>
        <mx:SetProperty target="{p2}" name="y" value="0"/>
        <mx:SetProperty target="{p2}" name="width" value="500"/>
```

```
            <mx:SetProperty target="{p2}" name="height" value="210"/>
            <mx:SetProperty target="{p3}" name="x" value="0"/>
            <mx:SetProperty target="{p3}" name="y" value="110"/>
            <mx:SetProperty target="{p3}" name="width" value="100"/>
            <mx:SetProperty target="{p3}" name="height" value="100"/>
        </mx:State>
    </mx:states>

    <mx:transitions>
        <mx:Transition fromState="*" toState="*">
            <mx:Sequence id="t1" targets="{[p1,p2,p3]}">
                <mx:Blur id="myBlur" duration="100" blurXFrom="0.0"
                blurXTo="10.0" blurYFrom="0.0" blurYTo="10.0"/>
                <mx:Parallel>
                    <mx:Move  duration="400"/>
                    <mx:Resize duration="400"/>
                </mx:Parallel>
                <mx:Blur id="myUnBlur" duration="100" blurXFrom="10.0"
                blurXTo="0.0" blurYFrom="10.0" blurYTo="0.0"/>
            </mx:Sequence>
        </mx:Transition>
    </mx:transitions>

    <mx:Canvas id="pm" width="100%" height="100%">
        <mx:Panel id="p1" title="One"
                x="0" y="0" width="100" height="100"
                click="currentState='One'" >
            <mx:Label fontSize="24" text="One"/>
        </mx:Panel>

        <mx:Panel id="p2" title="Two"
                x="0" y="110" width="100" height="100"
                click="currentState='Two'" >
            <mx:Label fontSize="24" text="Two"/>
        </mx:Panel>

        <mx:Panel id="p3" title="Three"
                x="110" y="0" width="500" height="210"
                click="currentState=''" >
            <mx:Label fontSize="24" text="Three"/>
        </mx:Panel>
    </mx:Canvas>
</mx:Application>
```

トランジションに関するヒントとトラブルシューティング

このセクションでは、トランジションの使用に関するヒントとトラブルシューティング情報を提供します。

ヒント

定義しているトランジションのタイプを判別してください。

- " 動的な " トランジションの場合は、再生されるエフェクトは既知ですが、エフェクトが再生されるターゲットは未知です。
- " 明示的な " トランジションの場合は、各ターゲットに何が再生されるかがすべて既知です。

動的な要素と明示的な要素の両方を含む、" 複合的な " トランジションが存在する場合もあります。

動的なトランジションに関するヒント

親にあたる組み合わせたエフェクトの中で、可能性があるターゲットをすべて挙げてください。

デフォルトでは、指定されたすべてのターゲットにすべてのエフェクトが再生されます。ターゲットを限定するには、動的なトランジションにフィルタを使用します。

動的なトランジションは、幅広い状態変化に使用できます。

明示的なトランジションに関するヒント

子エフェクトでターゲットを指定します。組み合わせたエフェクトのすべての子エフェクトが同じターゲットを持つ場合は、組み合わせたエフェクトでターゲットを指定します。

デフォルトでは、指定されたすべてのターゲットにすべてのエフェクトが再生されます。明示的なトランジションの場合は、ターゲットが正しく設定されていることを確認してください。

明示的なトランジションの場合は通常、ビューステートの変更ごとに異なるトランジションが必要です。

トラブルシューティング

トランジションエフェクトが再生されない場合のトラブルシューティング

エフェクトターゲットが非表示にされるか、または削除されますか？この場合は、ビューステートの定義に <mx:RemoveChild> プロパティが追加されているか、またはトランジションの定義に <mx:SetPropertyAction name="visible"> タグが追加されていることを確認してください。

ビューステートの変更で、トランジションエフェクトに関係する設定が定義されますか ？ たとえば、Resize エフェクトを使用する場合は、ビューステートの変化によってエフェクトがトリガされるときに、ターゲットの width または height プロパティが変更される必要があります。

トランジションに正しいターゲットが指定されていることを確認してください。

エフェクトまたは親エフェクトに関するフィルタ設定で、ターゲットが除外されていないかどうかを確認してください。

エフェクトが再生されるターゲットが多過ぎる場合のトラブルシューティング

動的なトランジションにフィルタを追加するか、または明示的なトランジションのターゲットを変更します。

エフェクトパラメータが誤っている場合のトラブルシューティング

エフェクトに明示的なパラメータを指定しましたか？それらは正しいですか？

マスクエフェクト (Iris エフェクトなど) とワイプエフェクトの showTarget プロパティが正しく設定されていることを確認してください。

ターゲットがちらつく場合や点滅する場合のトラブルシューティング

マスクエフェクト (Iris エフェクトなど) とワイプエフェクトの showTarget プロパティが正しく設定されていることを確認してください。

エフェクトターゲットが非表示にされるか、または削除されますか？この場合は、ターゲットを削除するためにビューステートの定義に <mx:RemoveChild> プロパティが追加されているか、またはターゲットを非表示にするためにトランジションの定義に <mx:SetPropertyAction name="visible"> タグが追加されていることを確認してください。

トランジション後にアプリケーションが正しく表示されなくなる場合のトラブルシューティング

エフェクトによっては、ターゲットのプロパティが変更されてそのままになる場合があります。たとえば、Fade エフェクトを実行すると、ターゲットの alpha プロパティが、エフェクトに指定された alphaTo 値のままになります。alpha プロパティを 0 に設定するターゲットに Fade エフェクトを使用した場合は、オブジェクトが再び表示される前に、alpha プロパティを 0 以外の値に設定する必要があります。

アプリケーションが正しく表示されるまで、トランジションからエフェクトを 1 つずつ削除してみてください。

第 29 章

Drag and Drop Manager の使用 29

Drag and Drop Manager を使用すると、Adobe Flex アプリケーション内のある場所から別の場所にデータを移動できます。この機能は、データがリスト内のアイテム、イメージ、Flex コンポーネントで表されるビジュアルアプリケーションで特に有効です。

このトピックでは、ドラッグ & ドロップ操作と、アプリケーションにこの機能を追加する方法について説明します。

目次

Drag and Drop Manager について

ビジュアル開発環境では一般的に、アプリケーション内のオブジェクトをマウスで選択し、画面上を移動させることで、オブジェクトを操作できます。Flex Drag and Drop Manager を利用すると、List コントロール内のアイテムなどのオブジェクトや、Image コントロールなどの Flex コントロールを選択して他のコンポーネントにドラッグし、そのコンポーネントに追加できます。

すべての Flex コンポーネントは、ドラッグ & ドロップ操作をサポートします。また Flex では、一部のコントロールのドラッグ & ドロップ操作に固有のサポートも行います。こうした一部のコントロールには、List、Tree、および DataGrid などが含まれます。

ドラッグ & ドロップ操作について

ドラッグ & ドロップ操作には、開始、ドラッグ、およびドロップの 3 つの段階があります。以下では、それぞれの段階を大まかに説明します。その後、各段階の詳細について説明します。

開始　ドラッグ & ドロップ操作を開始するには、マウスを使用して Flex コンポーネントを選択するか、Flex コンポーネント内のアイテムを選択し、マウスのボタンを押しながらコンポーネントまたはアイテムを移動させます。たとえば、マウスで List コントロール内のアイテムを選択し、マウスのボタンを押しながら、マウスを数ピクセル移動させます。選択されるコンポーネントが " ドラッグイニシエータ " です。

ドラッグ　マウスボタンを押しながら、ユーザーはマウスを Flex アプリケーション上で移動させます。ドラッグの間、Flex には画像が表示されます。この画像が " ドラッグプロキシ " です。" ドラッグソース "(DragSource オブジェクト) にはドラッグされているデータが含まれます。

ドロップ　ユーザーがドラッグプロキシを他の Flex コンポーネントの上に移動すると、コンポーネントはドロップ可能な " ドロップターゲット " になります。ドロップターゲットによってドラッグソースが調べられ、ターゲットで受け付けられるフォーマットのデータかどうか判断されます。ターゲットで受け付けられるフォーマットであれば、ユーザーはそこにデータをドロップすることができます。受け付けられないフォーマットのデータと判断された場合には、ドロップ操作は許可されません。

ドロップ操作が正常に行われると、データはターゲットに追加され、必要な場合は元の場所から削除されます。

リストコントロールでのドラッグ & ドロップの使用

次のコントロールでは、ドラッグ & ドロップ操作がビルトインでサポートされています。

- DataGrid
- HorizontalList
- List
- Menu
- PrintDataGrid
- TileList
- Tree

これらのコントロールをドラッグイニシエータにするには、`dragEnabled` プロパティを `true` に設定します。ドラッグ操作を開始するイベントリスナーを定義する必要はありません。Flex では、ドラッグが有効化されたコントロールから、ドロップが有効化されたコントロールへ、アイテムをドラッグして移動することができます。また、Ctrl キーを押したままドラッグするとアイテムをコピーすることができます。

これらのコントロールには、ドラッグ & ドロップ操作を管理するためのプロパティとメソッドがあります。次の表は、そのプロパティとメソッドの一覧です。

プロパティ / メソッド	説明
defaultDropIndicatorSkin	ドラッグデータの挿入位置を示すドロップ挿入インジケータに使用されるスキンの名前を指定します。ターゲットで showDropFeedback() メソッドを呼び出す場合にのみ実質的な効果があります。 デフォルト値は ListDropIndicator です。
dragEnabled	コントロールがドラッグイニシエータであるかどうかを指定するブール値です。デフォルト値は false です。true の場合は、ユーザーが選択したアイテムをコントロールからドラッグできます。ユーザーがアイテムをコントロールからドラッグすると、次のデータオブジェクトを含む DragSource オブジェクトが作成されます。 • コントロール内の選択されたアイテム (複数の場合あり) のコピーです。Tree コントロール以外のすべてのコントロールでは、フォーマットストリングは "items" で、アイテムが IDataProvider インターフェイスを実装します。Tree コントロールでは、フォーマットストリングが "treeItems" で、アイテムが ITreeDataProvider API インターフェイスを実装します。 • イニシエータのコピーとフォーマットストリング "source"。
dropEnabled	コントロールが、アイテムをデフォルト値で受け入れるドロップターゲットになれるかどうかを指定するブール値です。デフォルト値は false です。デフォルト値の場合は、ドラッグイベント用のイベントリスナーを記述する必要があります。この値が true の場合は、デフォルトのドロップビヘイビアを使用してコントロールにアイテムをドロップできます。
dragMoveEnabled	この値が true で、dragEnabled プロパティが true の場合は、ドラッグイニシエータからドロップターゲットにアイテムを移動またはコピーできるように指定します。アイテムを移動する場合は、アイテムがドロップターゲットに追加されたときにドラッグイニシエータから削除されます。 この値が false の場合は、アイテムのドロップターゲットへのコピーのみが可能です。コピーの場合は、ドラッグイニシエータ内のアイテムはドロップによる影響を受けません。 dragMoveEnabled プロパティが true のとき、コピーを実行するには、ドロップ操作中 Ctrl キーを押し続ける必要があります。 デフォルト値は、List および DataGrid コントロールの場合は false、Tree コントロールの場合は true です。
calculateDropIndex	アイテムがドロップされるドロップターゲット内のアイテムインデックスを返します。dragDrop イベントリスナーが正しい場所にアイテムを追加する際に使用されます。 TileList コントロールと HorizontalList コントロールでは使用できません。

プロパティ / メソッド	説明
hideDropFeedback()	ドロップターゲットのフィードバックを非表示にし、フォーカス矩形をターゲットから削除します。通常このメソッドは、dragExit イベントと dragDrop イベントのリスナー内で呼び出します。
showDropFeedback()	ターゲットコントロールの周囲にフォーカス矩形を表示し、ドロップ操作が発生する場所にドロップインジケータを表示します。コントロールにアクティブなスクロールバーがある場合、マウスポインタをコントロールの上端または下端に重ねると、コンテンツがスクロールします。 通常このメソッドは、dragOver イベントのリスナー内で呼び出します。

例 : List コントロール

次に、ある List コントロールから別の List コントロールにコピーできる簡単な例を示します。

```
<?xml version="1.0"?>
<!-- dragdrop\SimpleListToList.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="500" height="200"
    borderStyle="solid"
    creationComplete="initApp();">

    <mx:Script>
        <![CDATA[
            private function initApp():void {
               srclist.dataProvider = ['Reading', 'Television', 'Movies'];
                 destlist.dataProvider =[];
            }
        ]]>
    </mx:Script>

    <mx:HBox>
        <mx:VBox>
            <mx:Label text="Available Activities"/>
            <mx:List id="srclist"
                height="100"
                allowMultipleSelection="true"
                dragEnabled="true"/>
        </mx:VBox>

        <mx:VBox>
            <mx:Label text="Activities I Like"/>
            <mx:List id="destlist"
                height="100"
                dropEnabled="true"/>
        </mx:VBox>
    </mx:HBox>
</mx:Application>
```

最初の List コントロールの dragEnabled プロパティを true に設定し、2 番目の List コントロールの dropEnabled プロパティを true に設定することにより、ユーザーが最初のリストから次のリストへアイテムをドラッグできるようになります。背後で実行される処理をユーザーが気にすることはありません。

dragMoveEnabled プロパティのデフォルト値は false です。この場合は、あるコントロール List から別の List コントロールへのコピーのみが可能です。この例で、次のようにソース List コントロールの dragMoveEnabled を true に変更すると、エレメントの移動とコピーが可能になります。

```
<mx:VBox>
  <mx:Label text="Available Activities"/>
  <mx:List id="srclist"
    height="100"
    allowMultipleSelection="true" dragEnabled="true"
    dragMoveEnabled="true"/>
</mx:VBox>
```

デフォルトのアクションはエレメントの移動です。エレメントをコピーするには、ドロップ操作中 Ctrl キーを押したままにします。

次のように、両方の List コントロールの dragEnabled および dropEnabled プロパティを有効に設定すると、どちらの List コントロールからでもアイテムをドラッグし、ドロップできます。

```
<?xml version="1.0"?>
<!-- dragdrop\SimpleListToListBothWays.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="500" height="200"
    borderStyle="solid"
    creationComplete="initApp();">

    <mx:Script>
        <![CDATA[
            private function initApp():void {
               srclist.dataProvider = ['Reading', 'Television', 'Movies'];
                 destlist.dataProvider = [];
            }
        ]]>
    </mx:Script>

    <mx:HBox>
        <mx:VBox>
            <mx:Label text="Available Activities"/>
            <mx:List id="srclist"
                height="100"
                allowMultipleSelection="true"
                dragEnabled="true"
                dropEnabled="true"
                dragMoveEnabled="true"/>
        </mx:VBox>
```

```
        <mx:VBox>
            <mx:Label text="Activities I Like"/>
            <mx:List id="destlist"
                height="100"
                allowMultipleSelection="true"
                dragEnabled="true"
                dropEnabled="true"
                dragMoveEnabled="true"/>
        </mx:VBox>
    </mx:HBox>
</mx:Application>
```

例 : DataGrid コントロール

次の例は、List コントロールで使用したのと同じ手法を DataGrid コントロールで使用した例です。

```
<?xml version="1.0"?>
<!-- dragdrop\SimpleDGToDG.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700" height="250"
    borderStyle="solid"
    creationComplete="initApp();">

    <mx:Script>
        <![CDATA[
            private function initApp():void {
              srcgrid.dataProvider = [
                {Artist:'Carole King', Album:'Tapestry', Price:11.99},
                {Artist:'Paul Simon', Album:'Graceland', Price:10.99},
                {Artist:'Original Cast', Album:'Camelot', Price:12.99},
                {Artist:'The Beatles', Album:'The White Album', Price:11.99}
              ];

              destgrid.dataProvider =[];
            }
        ]]>
    </mx:Script>

    <mx:HBox>
        <mx:VBox>
            <mx:Label text="Available Albums"/>
            <mx:DataGrid id="srcgrid"
                allowMultipleSelection="true"
                dragEnabled="true"
                dropEnabled="true"
                dragMoveEnabled="true">
                <mx:columns>
                    <mx:DataGridColumn dataField="Artist"/>
                    <mx:DataGridColumn dataField="Album"/>
                    <mx:DataGridColumn dataField="Price"/>
```

```
            </mx:columns>
        </mx:DataGrid>
    </mx:VBox>

    <mx:VBox>
        <mx:Label text="Buy These Albums"/>
        <mx:DataGrid id="destgrid"
            allowMultipleSelection="true"
            dragEnabled="true"
            dropEnabled="true"
            dragMoveEnabled="true">
            <mx:columns>
                <mx:DataGridColumn dataField="Artist"/>
                <mx:DataGridColumn dataField="Album"/>
                <mx:DataGridColumn dataField="Price"/>
            </mx:columns>
        </mx:DataGrid>
    </mx:VBox>
</mx:HBox>
</mx:Application>
```

例 : Tree コントロール

次の例では、Tree コントロールを、コントロール内のツリーエレメントをドラッグ & ドロップ操作によって移動できるように定義しています。dropEnabled を false に設定した場合は、エレメントを他のコントロールにドラッグすることはできますが、同じコントロール内で移動することはできません。

```
<?xml version="1.0"?>
<!-- dragdrop\SimpleTreeSelf.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700" height="250"
    borderStyle="solid">

    <mx:Script>
        <![CDATA[
            // Initialize the data provider for the Tree.
            private function initApp():void {
                firstList.dataProvider = treeDP;
            }
        ]]>
    </mx:Script>

    <mx:XML id="treeDP">
        <node label="Mail">
            <node label="Inbox"/>
            <node label="Personal Folder">
                <node label="Demo"/>
                <node label="Personal"/>
```

```
                <node label="Saved Mail"/>
                <node label="bar"/>
            </node>
            <node label="Calendar"/>
            <node label="Sent"/>
            <node label="Trash"/>
        </node>
    </mx:XML>

    <mx:Tree id="firstList"
        height="200" width="200"
        showRoot="false"
        labelField="@label"
        dragEnabled="true"
        dropEnabled="true"
        dragMoveEnabled="true"
        allowMultipleSelection="true"
        creationComplete="initApp();"/>
</mx:Application>
```

Tree コントロールに組み込まれたドラッグ & ドロップ機能では、アイテムの移動が可能です。アイテムのコピーを可能にする場合は、dragDrop および dragComplete イベントに関する独自のイベントリスナーを実装する必要があります。これらのイベントの詳細については、991 ページの「ドラッグ & ドロップイベント」を参照してください。

ドラッグ & ドロップサポートの手動追加

リストコントロール以外やコンテナに対するドラッグ & ドロップ操作のサポートを実装する場合は、一連の特殊なクラスやイベントを使用して、サポートを明示的に追加する必要があります。

ドラッグ & ドロップ操作で使用されるクラス

リストクラス内のビルトイン機能を使用しない場合は、ドラッグ & ドロップ操作へのサポートを手動で追加する必要があります。次のクラスを使用してドラッグ & ドロップ操作を実装します。

クラス	関数
DragManager	ドラッグ & ドロップ操作を管理します。たとえば、doDrag() メソッドはドラッグ操作を開始します。
DragSource	ドラッグされるデータを識別し、格納します。他のドラッグ管理機能として、データが要求されたときに呼び出されるリスナーを追加する機能なども提供します。
DragEvent	ユーザーがドラッグターゲット上へドラッグするなどの、ドラッグ & ドロップイベントを表します。

ドラッグ & ドロップイベント

Flex アプリケーションでは、イベントを使用してドラッグ & ドロップ操作を制御します。ドラッグイニシエータとなるコントロールが、次のイベントを使用してドラッグ & ドロップ操作を管理します。

イベント	内容
mouseDown および mouseMove	mouseDown イベントは、ユーザーがマウスでコントロールを選択し、マウスボタンを押したままにすると送出されます。mouseMove イベントは、マウスが移動したときに送出されます。 ほとんどのコントロールでは、これらのイベントのいずれかに反応してドラッグ & ドロップ操作が開始されます。Tree コントロールや Grid コントロールなどの dragEnabled プロパティを持つコントロールでは、ドラッグ操作を開始するためのビルトインサポートが提供されているので、マウスイベントが使用されません。
dragComplete	ドラッグ操作が完了したとき (ドラッグデータをドロップターゲットにドロップするか、ドロップを実行せずにドラッグ & ドロップ操作を終了したとき) に送出されます。 このイベントは、ドラッグ & ドロップ操作の最終的なクリーンアップを行うために使用できます。たとえば、List コントロールのアイテムをリスト間でドラッグする場合、このイベントのリスナーを使用して、List コントロールアイテムをソースから削除することができます。

ドロップターゲットは次のイベントを使用します。

イベント	内容
dragEnter	ドラッグプロキシがターゲットの外側からターゲット上に移動したときに送出されます。 コンポーネントをドロップターゲットにするには、このイベントに対するイベントリスナーを定義する必要があります。リスナーでは、コンポーネントがドラッグ操作を受け付けることができることを示す視覚的なフィードバックをユーザーに提供するために、ドロップターゲットの外観を変更できます。たとえば、ドロップターゲットの周囲に境界線を表示したり、ドロップターゲットにフォーカスを移すことができます。
dragOver	dragEnter イベント後に、ユーザーがターゲット上でマウスを移動したときに送出されます。 このイベントは、ドロップ操作を許可する前に追加のロジックを実行する場合に使用します。追加のロジックの例としては、ドロップターゲット内のいろいろな場所にデータをドロップする操作、キーボード入力を読み取って、ドラッグ & ドロップ操作がドラッグデータの移動またはコピーのどちらであるかを判別する操作、ドラッグ & ドロップ操作のタイプに基づいて異なるタイプの視覚的なフィードバックを提供する操作などがあります。

イベント	内容
dragDrop	ユーザーがターゲット上でマウスボタンを離したときに送出されます。 このイベントリスナーは、ドラッグデータをドロップターゲットに追加するためにのみ使用できます。
dragExit	ユーザーがデータをドロップターゲットの外側にドラッグし、ターゲットにドロップしないときに送出されます。 このイベントは、dragEnter イベントや他のイベントに対応してドロップターゲットの外観を変更した場合に、ドロップターゲットを通常の外観に復元するのに使用できます。

ドラッグ & ドロップ操作

ドラッグ & ドロップ操作は、次の手順で定義されます。

1. 次のいずれかの方法でコンポーネントがドラッグ & ドロップイニシエータになります。

   dragEnabled プロパティがあるコンポーネント　(List、Tree、DataGrid、PrintDataGrid、Menu、HorizontalList、TileList など)dragEnabled プロパティが true に設定されている場合は、コンポーネント上でユーザーがマウスをクリックしたとき、およびマウスポインタを移動したときに、コンポーネントが自動的にイニシエータになります。

   dragEnabled プロパティがないコンポーネント　上記以外のコンポーネントは、ユーザーがドラッグ操作を開始したのを感知して、明示的にイニシエータになる必要があります。一般的には、mouseMove または mouseDown イベントを使用してドラッグ & ドロップ操作を開始します。

   a. コンポーネントによって、ドラッグされるデータを含む mx.core.DragSource クラスのインスタンスが作成され、データのフォーマットが指定されます。

   b. コンポーネントによって、mx.managers.DragManager.doDrag() メソッドが呼び出され、ドラッグ & ドロップ操作が開始されます。

2. マウスボタンが押されている間、ユーザーがアプリケーション上でマウスカーソルを移動すると、Flex によってアプリケーション上にドラッグプロキシイメージが表示されます。DragManager.defaultDragImageSkin プロパティでは、デフォルトのドラッグプロキシイメージを定義します。

> **メモ**
>
> ドラッグプロキシがターゲット上にないときにマウスボタンを離すと、ドラッグ & ドロップ操作は終了します。ドラッグイニシエータで DragComplete イベントが生成され、DragManager.getFeedback() メソッドが DragManager.NONE を返します。

3. ユーザーがドラッグプロキシを Flex コンポーネント上に移動すると、Flex によって dragEnter イベントがそのコンポーネントに送出されます。コンポーネントをドロップターゲットとするには、dragEnter イベントのイベントリスナーを定義する必要があります。

   dragEnter イベントリスナーによって DragSource オブジェクトが調べられ、ドラッグされるデータが受け付けられるデータフォーマットかどうか判断されます。ドロップを受け付ける場合は、イベントリスナーによって DragManager.acceptDragDrop() メソッドが呼び出されます。

   - ドロップターゲットでドロップが受け付けられない場合、ドロップターゲットコンポーネントの親チェーンが調べられ、そのドロップデータを受け付けるコンポーネントがチェーン内に存在するかどうか判断されます。
   - ドロップターゲットまたは親コンポーネントがドロップを受け付ける場合は、ユーザーがプロキシをターゲット上に移動したときに、dragOver イベントが送出されます。

4. (オプション) ドラッグターゲットが dragOver イベントを処理することもできます。たとえば、ターゲットがこのイベントリスナーを使用して、自分自身にフォーカスを設定する場合があります。
5. ユーザーがドロップターゲットにデータをドロップしないと決め、マウスボタンを離さずにドロップターゲットの外にドラッグプロキシを移動すると、dragExit イベントが送出されます。ドロップターゲットがこのイベントを処理します。たとえば、dragOver イベントリスナーで行われたアクションを元に戻します。
6. ユーザーがドロップターゲットの上でマウスボタンを離した場合、Flex によって dragDrop イベントが送出されます。ドロップターゲットの dragDrop イベントリスナーにより、ドラッグデータがターゲットに追加されます。
7. ドロップ操作が終了すると、dragComplete イベントが送出されます。ドラッグイニシエータがこのイベントを処理します。たとえば、データプロバイダからのデータを削除します。

例 : ドラッグデータへのアクセス

次の例では、dragDrop イベントリスナーを使用して、ある DataGrid コントロールから別の DataGrid コントロールにドラッグされたデータにアクセスします。この例では、ドラッグ & ドロップ操作が成功した後、Alert コントロールにアーティストの名前を表示するように、988 ページの「例 : DataGrid コントロール」にある例を変更しています。

```
<?xml version="1.0"?>
<!-- dragdrop\simpleDGToDGAlert.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700" height="250"
    borderStyle="solid"
    creationComplete="initApp();">

    <mx:Script>
        <![CDATA[
            import mx.events.DragEvent;
```

```
        import mx.controls.Alert;

        private function initApp():void {
          srcgrid.dataProvider =  [
            {Artist:'Carole King', Album:'Tapestry', Price:11.99},
            {Artist:'Paul Simon', Album:'Graceland', Price:10.99},
            {Artist:'Original Cast', Album:'Camelot', Price:12.99},
            {Artist:'The Beatles', Album:'The White Album', Price:11.99}
          ];

            destgrid.dataProvider =[];
        }

        // Define the event listener.
        public function dragDropHandler(event:DragEvent):void {
            // Access dragged data as an Array
            // in case multiple items were selected.
            var dragObj:Array=
                event.dragSource.dataForFormat("items") as Array;

            // Get the Artist for all dragged albums.
            var artistList:String='';
            for (var i:Number = 0; i < dragObj.length; i++) {
                artistList+='Artist: ' + dragObj[i].Artist + '\n';
            }

            Alert.show(artistList);
        }
    ]]>
</mx:Script>

<mx:HBox>
    <mx:VBox>
        <mx:Label text="Available Albums"/>
        <mx:DataGrid id="srcgrid"
            allowMultipleSelection="true"
            dragEnabled="true"
            dropEnabled="true"
            dragMoveEnabled="true">
            <mx:columns>
                <mx:DataGridColumn dataField="Artist"  />
                <mx:DataGridColumn dataField="Album" />
                <mx:DataGridColumn dataField="Price" />
            </mx:columns>
        </mx:DataGrid>
    </mx:VBox>

    <mx:VBox>
        <mx:Label text="Buy These Albums"/>
        <mx:DataGrid id="destgrid"
```

```
                allowMultipleSelection="true"
                dragEnabled="true"
                dropEnabled="true"
                dragMoveEnabled="true"
                dragDrop="dragDropHandler(event);">
                <mx:columns>
                    <mx:DataGridColumn dataField="Artist"  />
                    <mx:DataGridColumn dataField="Album" />
                    <mx:DataGridColumn dataField="Price" />
                </mx:columns>
            </mx:DataGrid>
        </mx:VBox>
    </mx:HBox>
</mx:Application>
```

例 : 単純なドラッグ & ドロップ操作

次の例では、2 つのサンプルカラーのいずれかをキャンバスにドロップして、キャンバスに色を付けます。ここでは、ドラッグ & ドロップ操作の基本的な要素を示します。その後、これらの要素について詳細に説明します。

```
<?xml version="1.0"?>
<!-- dragdrop\DandDCanvas.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
    <![CDATA[
        import mx.core.DragSource;
        import mx.managers.DragManager;
        import mx.events.*;
        import mx.containers.Canvas;

        // Called when the user clicks the mouse on either colored canvas.
        // Initializes the drag.
        private function dragIt(event:MouseEvent, text:String,
            format:String):void {

            // Get the drag initiator component from the event object.
            var dragInitiator:Canvas=Canvas(event.currentTarget);
            // Create a DragSource object.
            var ds:DragSource = new DragSource();
            // Add the data to the object.
            ds.addData(text, format);

            // Create a Canvas container to use as the drag proxy.
            // You must specify a size for the proxy image,
            // or else it will not appear.
            var canvasProxy:Canvas = new Canvas();
            canvasProxy.width=30;
```

```
            canvasProxy.height=30;
            canvasProxy.setStyle('backgroundColor',
                dragInitiator.getStyle('backgroundColor'));

            // Call the DragManager doDrag() method to start the drag.
            // For information on this method, see
            // the "Initiating the drag" section.
            DragManager.doDrag(dragInitiator, ds, event, canvasProxy);
        }

        // Called if the user dragged a proxy onto the drop target canvas.
        private function doDragEnter(event:DragEvent):void {
            // Get the drop target component from the event object.
            var dropTarget:Canvas=Canvas(event.currentTarget);

            // Accept the drag only if the user is dragging data
            // identified by the 'color' format value.
            if (event.dragSource.hasFormat('color')) {
                DragManager.acceptDragDrop(dropTarget);
            }
        }

        // Called if the target accepts the dragged object and the user
        // releases the mouse button while over the canvas.
        // Handles the dragDrop event for the List control.
        private function doDragDrop(event:DragEvent):void {
            // Get the data identified by the color format
            // from the drag source.
            var data:Object = event.dragSource.dataForFormat('color');
            // Set the canvas color.
            myCanvas.setStyle("backgroundColor", data);
        }

    ]]>
    </mx:Script>

    <!-- A horizontal box with red and green canvases the user can drag -->
    <mx:HBox>
        <mx:Canvas
            width="30" height="30"
            backgroundColor="red"
            borderStyle="solid"
            mouseMove="dragIt(event, 'red', 'color');"/>
        <mx:Canvas
            width="30" height="30"
            backgroundColor="green"
            borderStyle="solid"
            mouseMove="dragIt(event, 'green', 'color');"/>
    </mx:HBox>
```

```
    <mx:Label text="Drag the item into this canvas"/>

    <!-- Handles dragEnter and dragDrop events to allow dropping -->
    <mx:Canvas id="myCanvas"
        width="100" height="100"
        backgroundColor="#FFFFFF"
        borderStyle="solid"
        dragEnter="doDragEnter(event);"
        dragDrop="doDragDrop(event);"/>
</mx:Application>
```

ドラッグ & ドロップ操作のプログラミング

次のセクションでは、ドラッグ & ドロップ操作を使用するアプリケーションの作成方法について説明します。ドラッグ & ドロップ操作の開始、ドラッグプロキシイメージの指定、およびドラッグ & ドロップイベントの処理を行う方法について説明します。

ドラッグ & ドロップ操作の開始

リストコントロール以外のすべてのコントロールでは、mouseMove イベントまたは mouseDown イベントのイベントリスナーによって、ドラッグ & ドロップ操作が明示的に開始されます。原則として mouseMove イベントを使用します。この場合は、マウスボタンが押されマウスが移動している場合のみ、Drag and Drop Manager によってドラッグ & ドロップ操作が開始されます。1004 ページの「例 : すべてのドラッグ & ドロップイベント」のセクションで示したアプリケーションのように、ユーザーが最初にマウスを使用してテキストを選択するアプリケーションでは、mouseDown イベントをより容易に使用できます。

メモ

リストコントロールに対するドラッグ & ドロップ操作の開始については、984 ページの「リストコントロールでのドラッグ & ドロップの使用」を参照してください。

イニシエータのイベントリスナーの記述

ドラッグ & ドロップ操作を開始するイベントリスナーは次の引数を取ります。

引数	内容
event	イベントオブジェクト。オブジェクトの target プロパティには、イベントを送出するコントロール、つまり、ユーザーがマウスをクリックしたり移動したりするコントロールへの参照が含まれます。
data	ドラッグするデータを表すオブジェクトです。
format	データタイプを識別するアプリケーション固有の識別子を収めたストリングです。

format 引数は、データフォーマットに "color"、"list data"、"employee record" などのラベルを付けるために使用するテキストストリングです。このストリングを調べることで、そのデータフォーマットがドロップターゲットで受け付けられるデータのタイプと適合するかどうか判断されます。受け付けられるフォーマットの場合は、ターゲットにデータをドロップできます。そうでない場合には、ドロップ操作は許可されません。

メモ

リストコントロールには、フォーマットに関する定義済みの値があります。コントロール以外のコントロールの場合は、フォーマットストリングが "items" です。Tree コントロールの場合は、フォーマットストリングが "treeItems" です。詳細については、984 ページの「リストコントロールでのドラッグ & ドロップの使用」を参照してください。

ドラッグを開始するイベントリスナーは次の 2 つを必ず実行します。

1. DragSource オブジェクトを作成し、ドラッグデータとフォーマットを使用して初期化します。ユーザーがドラッグプロキシをコントロール上にドロップするとき、dragEnter や dragDrop などのコントロールのドラッグ & ドロップイベントの dragSource プロパティには、このオブジェクトへの参照が含まれています。

   次のコードは、995 ページの「例 : 単純なドラッグ & ドロップ操作」で紹介した mouseDown イベントに対するイベントリスナー関数の最初の数行です。このコードでは、DragSource オブジェクトを作成し、データとフォーマットをオブジェクトに格納しています。

```
// ユーザーが色付きキャンバス上でマウスをクリックすると呼び出されます。
// ドラッグを初期化します。
private function dragIt(event:MouseEvent, text:String,
  format:String):void {

  // イベントオブジェクトからイニシエータコンポーネントを取得します。
  var dragInitiator:Canvas=Canvas(event.currentTarget);
  // DragSource オブジェクトを作成します。
  var ds:DragSource = new DragSource();
  // オブジェクトにデータを追加します。
  ds.addData(text, format);
  ...
```

イベントリスナーは最初に、ドラッグイニシエータを含む event.currentTarget プロパティを Canvas コンテナにキャストします。event.currentTarget プロパティのデータ型が * なので、必ずキャストしなければなりません。キャスト後、Canvas コンテナのすべてのプロパティにアクセスできるようになります。

大量または複雑なデータアイテムをドラッグする場合は、データをコピーするリスナーを作成し、DragSource.addData() メソッドではなく、DragSource.addListener() メソッドを呼び出して、作成したリスナーを指定することができます。この場合、ユーザーがデータをドロップするまで、データがコピーされないので、ドラッグし始めたデータをユーザーがドロップしない場合に発生するデータコピー処理のオーバーヘッドが抑制されます。リストベースのクラスの実装には、この手法を使用します。

2. DragManager.doDrag() メソッドを呼び出し、ドラッグ & ドロップ操作を開始します。

doDrag() メソッドのシグネチャを次に示します。

```
doDrag( dragInitiator:IUIComponent,  dragSource:DragSource,
  mouseEvent:MouseEvent,  dragImage:IFlexDisplayObject = null,
  xOffset:Number = 0, yOffset:Number = 0,  imageAlpha:Number = 0.5,
  allowMove:Boolean = true):void
```

doDrag() メソッドには 3 つの引数、すなわち操作を開始するコンポーネントへの参照 (event.currentTarget オブジェクトにより識別)、手順 1 で作成した DragSource オブジェクト、およびドラッグを開始したマウスイベント (イベントリスナーに渡された event オブジェクト) が必要です。オプションの引数を使用すると、ドラッグプロキシイメージやイメージの特性を指定できます。

次のコードは、手順 1 で doDrag() メソッドを呼び出すことによって開始した mouseDown イベントリスナー関数の最後の部分です。ドラッグプロキシとして使用する Canvas コンテナを作成し、次に 4 番目の引数を使用してこのドラッグプロキシを DragManager に渡します。

```
    // 手順 1 からの続きです。
    ...
    // ドラッグプロキシとして使用する Canvas コンテナを作成します。
    // プロキシイメージのサイズを指定する必要があります。
    // 指定しない場合は表示されません。
    var canvasProxy:Canvas = new Canvas();
    canvasProxy.width=30;
    canvasProxy.height=30;
    canvasProxy.setStyle('backgroundColor',
    dragInitiator.getStyle('backgroundColor'));

    // DragManager doDrag() メソッドを呼び出してドラッグを開始します。
    // このメッソッドの詳細については
    // 「ドラッグの開始」を参照してください。
    DragManager.doDrag(dragInitiator, ds, event, canvasProxy);
}
```

ドラッグプロキシイメージの指定の詳細については、1000 ページの「ドラッグプロキシイメージの指定」を参照してください。

ドラッグプロキシイメージの指定

ドラッグプロキシイメージは、DragManager クラスの doDrag() メソッドで指定できます。イメージを指定しなかった場合は、Flex のデフォルトのプロキシイメージが使用されます。doDrag() メソッドは、イメージとイメージのプロパティを指定するため、次のオプションの引数を取ります。

引数	内容
dragImage	ドラッグプロキシイメージを指定するクラスまたはリンケージ識別子 (対象はイメージ) への参照です。 ユーザーが注文する製品の JPEG イメージなどのシンボルを指定する場合は、シンボル名を指定するストリングを使用します (例 : mylmage.jpg)。 Flex コンテナやコントロールなどのコンポーネントを指定するには、コントロールまたはコンテナのインスタンスを作成し、その構成とサイズを設定してから、それを引数として doDrag() メソッドに渡します。
xOffset	イニシエータコンポーネントの左上隅とドラッグプロキシイメージの左上隅の間の、水平方向のオフセット距離を指定する数値です。ほとんどの標準的な Flex コントロールでは、この引数を使用しません。
yOffset	イニシエータコンポーネントの左上隅とドラッグプロキシイメージの左上隅の間の、垂直方向のオフセット距離を指定する数値です。ほとんどの標準的な Flex コントロールでは、この引数を使用しません。
imageAlpha	ドラッグプロキシイメージに使用されるアルファ値を指定する数値です。省略した場合は、アルファ値 0.5 が使用されます。値 0 は透明に対応し、値 1.0 は完全な不透明に対応します。

ドラッグプロキシイメージのサイズを必ず指定します。指定しない場合、イメージが表示されません。次の例では、ドラッグプロキシとして Canvas オブジェクトを使用しています。ドラッグイニシエータの背景色に対する背景色を設定し、キャンバスのサイズを 30 x 30 ピクセルに指定します。*imageAlpha* 引数によってドラッグプロキシが不透明に指定されています。

```
// ドラッグプロキシとして使用する Canvas コンテナを作成します。
// プロキシイメージのサイズを指定する必要があります。
// 指定しない場合は表示されません。
var canvasProxy:Canvas = new Canvas();
canvasProxy.width=30;
canvasProxy.height=30;
canvasProxy.setStyle('backgroundColor',
  dragInitiator.getStyle('backgroundColor'));

// DragManager doDrag() メソッドを呼び出してドラッグを開始します。
DragManager.doDrag(dragInitiator, ds, event, canvasProxy);
```

写真やラベルと一緒に VBox コントロールを使用する場合など、特定のコンテンツと共にコントロールを使用するには、コントロールを含むカスタムコンポーネントを作成し、作成したコンポーネントの名前を *dragProxy* 引数として使用する必要があります。

ドラッグ & ドロップイベントの処理

以降のセクションでは、ドラッグ & ドロップ操作の開始後に発生するイベントの処理方法について説明します。各イベントのサンプルコードでは、選択したテキストをある List コントロールから別の List コントロールへドラッグする例を使用します。1004 ページの「例 : すべてのドラッグ & ドロップイベント」のコードでは、すべてのコードをまとめて 1 つのアプリケーションに構成しています。

dragEnter イベントの処理

ドラッグプロキシがコントロール内に入ると、dragEnter イベントが生成されます。コントロールをドロップターゲットとするには、dragEnter イベントに対するリスナーを次のように定義する必要があります。

- イベントリスナー内で DragSource オブジェクトのフォーマット情報を使用し、ドラッグデータがドロップターゲットで受け付けられるフォーマットを持っているかどうかを判断します。
- ユーザーがドロップターゲットにデータをドロップするためには、リスナーによって DragManager.acceptDragDrop() メソッドが呼び出される必要があります。
- ユーザーがドロップターゲットにドラッグプロキシを最初にドロップするときに発生させる他のアクションに対しても、このイベントリスナーを使用できます。
- イベントオブジェクトの action プロパティの値は、コピーを行う場合も DragManager.MOVE です。これは、コピーであることを示す Ctrl キーが押されたことをコントロールが認識する前に、dragEnter イベントが発生するためです。dragOver イベントのイベントオブジェクトの action プロパティには、ドラッグ操作のタイプを示す値が含まれます。この値は、DragManager.COPY、DragManager.LINK、DragManager.MOVE、または DragManager.NONE のいずれかになります。

次の dragEnter イベントリスナーでは、イベントリスナーが TextArea コントロールにイベントが発生したことを示すメッセージを書き込み、DragManager.acceptDragDrop() メソッドを呼び出してドラッグ & ドロップ操作を開始します。

```
private function doDragEnter(event:DragEvent):void {
   // イベントオブジェクトからドロップターゲットコンポーネントを取得します。
   var dragInitiator:List=List(event.currentTarget);

   // TextArea コントロールにメッセージを書き込みます。
   tiEnter.text += "dragEnter triggered" + "\n";
   DragManager.acceptDragDrop(dragInitiator);
}
```

このイベントリスナーを使用する詳しい例については、1004 ページの「例 : すべてのドラッグ & ドロップイベント」を参照してください。

dragOver イベントの処理

dragOver イベントは、ドラッグ & ドロップターゲットの dragEnter イベントリスナーが DragManager.acceptDragDrop() メソッドを呼び出し、そのターゲット上で、ユーザーがマウスを動かしたときに発生します。mouseMove イベントと同様に、このイベントは、ユーザーがターゲット上でマウスをドラッグするたびに何度でも送出されます。dragOver イベントリスナーはオプションです。このイベントリスナーを定義しなくてもドラッグ & ドロップ操作を実行できます。

dragOver イベントは、ドロップターゲット上にマウスがあるときにユーザーに提供する、視覚的なフィードバックを指定する場合に便利です。DragEvent オブジェクトには、イベント発生時に Ctrl キーが押されていたか、Alt キーが押されていたか、Shift キーが押されていたかを示す 3 つのブール型プロパティがあります。それぞれ、ctrlKey、altKey、および shiftKey です。この 3 つのプロパティを使用して、ユーザーがドラッグアイテムの移動、コピー、ドラッグアイテムへのリンクを要求しているかどうかを判断できます。また、要求された操作に基づくドラッグプロキシの外観を選択できます。

ドラッグプロキシに付随するドラッグフィードバックインジケータを指定するには、DragManager.showFeedback() メソッドを使用します。このメソッドは、メソッド引数として DragManager.COPY、DragManager.LINK、DragManager.MOVE、または DragManager.NONE のいずれかの定数値を使用します。たとえば、DragManager.showFeedBack(DragManager.NONE) と指定すると、ドラッグプロキシに赤色の円と白色の x が表示されます (ドラッグターゲットではないオブジェクト上にドラッグしたときに表示されるイメージと同じです)。

dragComplete イベントリスナーで DragManager.getFeedback() メソッドを使用して、要求された操作を判断することができます。たとえば、値が DragManager.MOVE の場合は、ドラッグされるオブジェクトの元のコピーを削除することが要求されています。詳細については、1003 ページの「dragDrop イベントの処理」および 1004 ページの「dragComplete イベントの処理」を参照してください。

次のコードでは、プロキシがターゲット上にドラッグされているときにユーザーがキーを押しているかどうかをリスナーが判断し、押されているキーに対応するフィードバック外観を設定しています。

```
private function doDragOver(event:DragEvent):void {
   tiOver.text += "dragOver triggered" + "\n";
   if (event.ctrlKey)
      DragManager.showFeedback(DragManager.COPY);
   else if (event.shiftKey)
      DragManager.showFeedback(DragManager.LINK);
   else
      DragManager.showFeedback(DragManager.MOVE);
}
```

このイベントリスナーを使用する詳しい例については、1004 ページの「例 : すべてのドラッグ & ドロップイベント」を参照してください。

dragDrop イベントの処理

dragDrop イベントは、ユーザーがターゲット上でマウスボタンを離し、データをドロップするときに発生します。ドロップ操作を処理するイベントのリスナーを定義し、ターゲットにデータを追加する必要があります。

```
private function doDragDrop(event:DragEvent):void {
   // ドロップターゲットを取得します。
   var dropTarget:List=List(event.currentTarget);
   // メッセージを TextArea コントロールに書き込みます。
   tiDrop.text += "dragDrop triggered" + "\n";

   // ドロップフィードバックを非表示にします。
   doDragExit(event);

   // ドラッグイニシエータからドラッグされるアイテムを取得します。
   var items:Array=event.dragSource.dataForFormat("items") as Array;

   // ドロップ先におけるドロップ位置を取得します。
   var dropLoc:int=dropTarget.calculateDropIndex(event);

   // ステータスメッセージを TextArea コントロールに書き込みます。
   tiDrop.text += "length: " + String(items.length) + "\n" + "dropLoc: " +
   String(dropLoc) + "\n" + "format: " + String(event.dragSource.formats[0])+
   "\n";

   // 各アイテムをドロップターゲットに追加します。
   for(var i:uint=0; i < items.length; i++) {
      tiDrop.text += "\n" + "label: " + items[i].label + "\n" + String(dropLoc);
      IList(dropTarget.dataProvider).addItemAt(items[i], dropLoc );
   }
}
```

このイベントリスナーを使用する詳しい例については、1004 ページの「例 : すべてのドラッグ & ドロップイベント」を参照してください。

dragExit イベントの処理

dragExit イベントは、ユーザーがドロップターゲットにデータをドロップせずに、マウスをターゲットの外に移動したときに発生します。リスナーを定義して、ドロップターゲットのクリーンアップを行うことができます。次の例では、リスナーがターゲットからフォーカスボックスを削除しています。

```
private function doDragExit(event:DragEvent):void {
   var dropTarget:List=List(event.currentTarget);
   tiExit.text += "dragExit triggered" + "\n";
   dropTarget.hideDropFeedback(event);
}
```

このイベントリスナーを使用する詳しい例については、1004 ページの「例 : すべてのドラッグ & ドロップイベント」を参照してください。

dragComplete イベントの処理

dragComplete イベントは、dragDrop イベントが終了した後に発生します。ドラッグイニシエータでリスナーを指定して、ドラッグが終了するとき、またはターゲットがドロップを受け付けないとき、クリーンアップを行うことができます。

dragComplete イベントリスナーを使用する目的の 1 つは、ターゲットに移動したオブジェクトをドラッグイニシエータから削除することです。コントロールからドラッグされるアイテムは、元のアイテムのコピーであり、アイテムそのものではありません。したがって、アイテムをドロップターゲットにドロップするときに、dragComplete イベントリスナーを使用してアイテムをドラッグイニシエータから削除します。

次の例では、リスナーによって TextArea コントロールに、イベントが発生したことを示すメッセージが書き込まれます。

```
private function doDragComplete(event:DragEvent):void {
   // ユーザーがテキストを移動した場合、ソーステキスト領域からそのテキストを削除します。
   if (DragManager.getFeedback() == DragManager.MOVE) {
      TextInput(event.currentTarget).text="";
}
```

このイベントリスナーを使用する詳しい例については、1004 ページの「例 : すべてのドラッグ & ドロップイベント」を参照してください。

例 : すべてのドラッグ & ドロップイベント

次の例は、List コントロールから別の List コントロールにアイテムをドラッグするときにトリガされる各イベントを示しています。

```
<?xml version="1.0"?>
<!-- dragdrop\DandDListToListAllEvents.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="500" height="620"
    borderStyle="solid"
    creationComplete="initApp();">

    <mx:Script>
        <![CDATA[
            import mx.events.DragEvent;
            import mx.controls.List;
            import mx.managers.DragManager;
            import mx.core.DragSource;
            import mx.collections.IList;

            private var placeHolder:Object = {label:"First", blah:"blah"};

            private function initApp():void {
                var dp:Array = ([
                    {label:"First", data:"1"},
```

```
            {label:"Second", data:"2"},
            {label:"Third", data:"3"},
            {label:"Fourth", data:"4"},
        ]);
        firstList.dataProvider = dp;
        secondList.dataProvider = [];
    }

    private function doDragEnter(event:DragEvent):void {
        var dragInitiator:List=List(event.currentTarget);
        tiEnter.text += "dragEnter triggered" + "\n";
        DragManager.acceptDragDrop(dragInitiator);
    }

    private function doDragOver(event:DragEvent):void {
        tiOver.text += "dragOver triggered" + "\n";
        if (event.ctrlKey)
            DragManager.showFeedback(DragManager.COPY);
        else if (event.shiftKey)
            DragManager.showFeedback(DragManager.LINK);
        else
            DragManager.showFeedback(DragManager.MOVE);
    }

    private function doDragDrop(event:DragEvent):void {
        // Get drop target.
        var dropTarget:List=List(event.currentTarget);
        // Write message to TextArea control.
        tiDrop.text += "dragDrop triggered" + "\n";

        // Hide drop feedback.
        doDragExit(event);

        // Get the dragged items from the drag initiator.
        var items:Array =
            event.dragSource.dataForFormat("items") as Array;
        // Get the drop location in the destination.
        var dropLoc:int = dropTarget.calculateDropIndex(event);

        // Write status message to TextArea control.
        tiDrop.text += "length: " + String(items.length) + "\n" +
            "dropLoc: " + String(dropLoc) + "\n" + "format: " +
            String(event.dragSource.formats[0])+ "\n";

        // Add each item to the drop target.
        for(var i:uint=0; i < items.length; i++)
        {
          tiDrop.text += "\n" + "label: " + items[i].label + "\n" +
            String(dropLoc);
          IList(dropTarget.dataProvider).addItemAt(items[i],
```

```
                    dropLoc);
                }
            }

            private function doDragComplete(event:DragEvent):void {
                tiComplete.text += "dragComplete triggered" + "\n";
            }

            private function doDragExit(event:DragEvent):void {
                var dropTarget:List=List(event.currentTarget);
                tiExit.text += "dragExit triggered" + "\n";
                dropTarget.hideDropFeedback(event);
            }
        ]]>
    </mx:Script>

    <mx:Label text="Drag items from left list to right list to see the triggered
events."/>
    <mx:HBox>
        <mx:List  id="firstList"
            dragEnabled="true"
            allowMultipleSelection="true"
            dragComplete="doDragComplete(event);"
            height="100"/>

        <mx:List  id="secondList"
            dragEnter="doDragEnter(event);"
            dragExit="doDragExit(event);"
            dragOver="doDragOver(event);"
            dragDrop="doDragDrop(event);"
            dragEnabled="true" height="100"/>
    </mx:HBox>

    <mx:VBox horizontalAlign="right">
        <mx:HBox>
            <mx:Label text="dragEnter: "/>
            <mx:TextArea id="tiEnter" width="200" height="80"/>
        </mx:HBox>

        <mx:HBox>
            <mx:Label text="dragDrop: "/>
            <mx:TextArea id="tiDrop" width="200" height="200"/>
        </mx:HBox>

        <mx:HBox>
            <mx:Label text="dragOver: "/>
            <mx:TextArea id="tiOver" width="200" height="80"/>
        </mx:HBox>

        <mx:HBox>
```

```
            <mx:Label text="dragExit: "/>
            <mx:TextArea id="tiExit" width="200" height="80"/>
        </mx:HBox>

        <mx:HBox>
            <mx:Label text="dragComplete: "/>
            <mx:TextArea id="tiComplete" width="200" height="80"/>
        </mx:HBox>
    </mx:VBox>
</mx:Application>
```

ドラッグ & ドロップに関するテクニックと注意事項

このセクションでは、有効なドラッグ & ドロップアプリケーションを作成するために役立つさまざまなテクニックと注意事項を説明します。

ドロップターゲットとしてのコンテナの使用

ドロップターゲットとしてコンテナを使用するには、コンテナの backgroundColor プロパティを使用して色を設定する必要があります。コンテナの背景色が透明な場合、Drag and Drop Manager ではマウスポインタが可能なドロップターゲットの上にあるかどうかを検知できません。

イメージのドラッグ

次の例では、<mx:Image> タグを使用してドラッグ可能イメージを Canvas コンテナにロードします。この例では、イメージは、ドラッグイニシエータとドラッグプロキシの両方です。イメージをドラッグプロキシに指定するには、Image コントロールを doDrag() メソッドへのドラッグプロキシとして指定します。

```
<?xml version="1.0"?>
<!-- dragdrop\DandDImageProxy.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            //Import classes so you don't have to use full names.
            import mx.managers.DragManager;
            import mx.core.DragSource;
            import mx.events.DragEvent;
            import flash.events.MouseEvent;
            import mx.controls.Image;
            import mx.containers.Canvas;
```

```
        //Variables used to hold the image's location
        public var xOff:Number;
        public var yOff:Number;

        // Embed icon image.
        [Embed(source='assets/globe.jpg')]
        public var globeImage:Class;

        // Drag initiator event handler, called by
        //  the image's mouseMove event.
        private function dragMe(event:MouseEvent, img1:Image,
            format:String):void {
            var dragInitiator:Image=Image(event.currentTarget);
            var ds:DragSource = new DragSource();
            ds.addData(img1, format);
            // The drag manager uses the image as the drag proxy
            // and sets the alpha to 100 (opaque),
            // so it appears to be dragged across the canvas.
            var imageProxy:Image = new Image();
            imageProxy.source = globeImage;
            imageProxy.height=10;
            imageProxy.width=10;
            DragManager.doDrag(dragInitiator, ds, event,
                imageProxy, 0, 0, 1.00);
        }

        //Function called by the canvas dragEnter event; enables dropping
        private function doDragEnter(event:DragEvent):void {
            DragManager.acceptDragDrop(Canvas(event.target));
        }

        // Function called by the canvas dragDrop event;
        // Sets the image object's position,
        // "dropping" it in its new location.
        private function doDragDrop(event:DragEvent,
            target1:Canvas, format:String):void {
            myimg.x = target1.mouseX - xOff
            myimg.y = target1.mouseY - yOff
        }

        // Helper function called by the dragged image's mouseMove event,
        // as the image drags across the canvas.
        // The function updates the xOff and yOff variables to show the
        // current mouse location.
        private function myoffset(img:Image):void {
            xOff = img.mouseX
            yOff = img.mouseY
        }
    ]]>
</mx:Script>
```

```
    <!-- The Canvas is the drag target -->
    <mx:Canvas id="v1"
        width="500" height="500"
        borderStyle="solid"
        backgroundColor="#DDDDDD"
        dragEnter="doDragEnter(event);"
        dragDrop="doDragDrop(event, v1, 'img');">

    <!-- The image is the drag initiator and the drag proxy. -->
        <mx:Image id="myimg"
            source="@Embed(source='assets/globe.jpg')"
            mouseMove="dragMe(event, myimg, 'img'); myoffset(myimg);"/>
    </mx:Canvas>
</mx:Application>
```

第 30 章

30 アセットの埋め込み

Adobe Flex アプリケーションの多くは、イメージ、サウンド、フォントなどの外部アセットを使用します。実行時にアセットを参照してロードすることもできますが、複数のアセットをアプリケーションにコンパイルすることもよくあります。アプリケーションにアセットをコンパイルするプロセスは、アセットの埋め込みと呼ばれます。Flex では、イメージファイル、ムービーファイル、MP3 ファイル、および TrueType フォントをアプリケーションに埋め込むことができます。

このトピックでは、イメージまたは SWF ファイルをアセットとしてアプリケーションに埋め込む手順の概要を説明します。フォントの埋め込みの詳細については、705 ページ、第 19 章の「フォントの使用」を参照してください。

目次

アセットの埋め込みについて

アセットを埋め込む場合は、アセットをアプリケーションの SWF ファイルにコンパイルします。アセットを埋め込む利点は、埋め込まれたアセットが SWF 内にあるため、実行時に離れた場所からアセットをロードするよりも短時間でアクセスできることです。アセットを埋め込むことの短所は、実行時にアセットをロードする場合と比べ SWF ファイルのサイズが大きくなることです。

アセット埋め込みの例

最も一般的な埋め込み方法は、MXML タグ定義内で @Embed( ) ディレクティブを使用して、Flex コントロールにイメージを読み込む方法です。たとえば、多くのコントロールは、アプリケーションへの埋め込みが可能なアイコンまたはスキンをサポートします。Button コントロールでは、ラベルテキストや、オプションのアイコンイメージを指定できます。次の例を参照してください。

```
<?xml version="1.0"?>
<!-- embed\ButtonIcon.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button label="Icon Button" icon="@Embed(source='logo.gif')"/>
</mx:Application>
```

もう 1 つの埋め込み方法は、[Embed] メタデータタグを使用して、埋め込みイメージを変数に関連付ける方法です。この方法では、次の例に示すように、アプリケーション内の複数の場所から埋め込まれたイメージを参照できます。

```
<?xml version="1.0"?>
<!-- embed\ButtonIconClass.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            [Embed(source="logo.gif")]
            [Bindable]
            public var imgCls:Class;
        ]]>
    </mx:Script>

    <mx:Button label="Icon Button 1" icon="{imgCls}"/>
    <mx:Button label="Icon Button 2" icon="{imgCls}"/>
</mx:Application>
```

スタイルプロパティの場合は、Embed( ) ディレクティブを使用して、スタイルシート定義の一部としてアセットを埋め込むことができます。スタイルプロパティに対する一般的な使用方法の 1 つは、コンポーネントのスキンを設定する方法です。次の例に示すように、overSkin、upSkin、および downSkin のスタイルプロパティを使用することにより、Button コントロールのスキンを設定できます。

```
<?xml version="1.0"?>
<!-- embed\ButtonIconCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Style>
        .myCustomButton {
            overSkin:Embed(source="overIconImage.gif");
            upSkin:Embed(source="upIconImage.gif");
            downSkin:Embed(source="downIconImage.gif");
        }
```

```
    </mx:Style>

    <mx:Button label="Icon Button Style Def" styleName="myCustomButton"/>
</mx:Application>
```

メモ

スタイルシートの等号 (=) は Flex 拡張子です。この拡張子は、一部の CSS プロセッサではサポートされない可能性があります。サポートされていないことが判明した場合は、`Embed(filename)` のシンタックスを使用してください。

アセットへの実行時のアクセス

アセットを埋め込むのではなく、実行時にアセットをロードすることもできます。アセットは、SWF ファイルが実行されるローカルなファイルシステムからロードすることもできます。また、リモートアセットにアクセスすることもできます。リモートアセットには、通常はネットワーク経由の HTTP 要求を使用してアクセスします。

アセットを埋め込んだ場合、埋め込まれたアセットは Flex SWF ファイルの一部となるため、ロードは直接的に行われます。ただし、アプリケーションのサイズが増えるので、アプリケーションの初期化プロセスは遅くなります。また、埋め込みアセットは、アセットに変更を加えるたびにアプリケーションを再コンパイルする必要があります。

実行時にアセットをロードした場合、対象のアセットは、Flex アプリケーションの一部としてコンパイルされるのではなく、独立したファイルとして Web サーバーなどに置かれます。アセットの参照により、アプリケーションの初期ロード時間に余分なオーバーヘッドが生じることはありません。ただし、Adobe Flash Player でアセットを使用し、ロードするときに遅延が生じる場合があります。アセットは、Flex アプリケーションとは独立しているため、アセットの名前さえ同じであれば、修正を加えたとしても再コンパイルの必要はありません。

実行時にアセットをロードする例については、301 ページの「Image コントロール」または 296 ページの「SWFLoader コントロール」を参照してください。

セキュリティのため、Flash Player のデフォルトでは、一部のタイプのリモートデータについては、実行時にアプリケーションの提供ドメイン以外のドメインからアクセスすることが許可されません。アクセスが許可されないリモートデータのタイプには、SWF ファイルも含まれます。このため、データをホストするサーバーがアプリケーションをホストしているサーバーと同じドメインにあるか、またはサーバーが "crossdomain.xml" ファイルを定義する必要があります。"crossdomain.xml" ファイルは XML ファイルで、サーバーのデータとドキュメントをどのドメインの SWF ファイルが利用できるかを示します。特定のドメインを指定することも、すべてのドメインを指定することもできます。アプリケーションのセキュリティの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 4 章の「Flex セキュリティの適用」を参照してください。

サポートされているファイルタイプ

Flex アプリケーションには、次のタイプのファイルを埋め込むことができます。

- GIF ファイル
- JPG および JPEG ファイル
- PNG ファイル
- SVG ファイル
- SWF ファイル
- SWF ファイルに保存されたシンボル
- MP3 ファイル
- TTF ファイル
- システムフォント

アセット埋め込みのシンタックス

アセットを埋め込む際に使用するシンタックスは、アプリケーションのどこにアセットを埋め込むかによって異なります。Flex では次のシンタックスをサポートします。

- `[Embed(parameter1, paramater2, ...)]` メタデータタグ

  ActionScript ファイルまたは MXML ファイルの `<mx:Script>` ブロック内にアセットを埋め込む場合に使用します。詳細については、1018 ページの「[Embed] メタデータタグの使用」を参照してください。

- `@Embed(parameter1, paramater2, ...)` ディレクティブ

  MXML タグ定義で使用して、アセットを埋め込みます。詳細については、1019 ページの「MXML での @Embed() ディレクティブの使用」を参照してください。

- `Embed(parameter1, paramater2, ...)` ディレクティブ

  MXML ファイルの `<mx:Style>` ブロック内で使用して、アセットを埋め込みます。詳細については、1019 ページの「スタイルシートでのアセットの埋め込み」を参照してください。

この 3 つの埋め込みシンタックスは、すべて同じアセットにアクセスします。アプリケーション内でシンタックスを使う場所が異なるだけです。

@ 文字のエスケープ

リテラル文字の @ を使用する場合は、バックスラッシュ文字 (\) を使用してアットマーク (@) をエスケープします。たとえば、“\@Embed(foo)” と表現されるストリングは、リテラルストリングの “@Embed(foo)” です。1 つの \ 文字をエスケープするには、バックスラッシュ文字を 2 つ使用します (\\)。たとえば、リテラルストリングの “\@” を指定するには、“\\@” と表現します。

埋め込みパラメータ

埋め込みシンタックスのすべてのフォームで、オプションのパラメータを 1 つ以上使用します。アセットの埋め込みに使用するシンタックスそのものは、埋め込む場所によって異なります。パラメータの中には、埋め込むアセットのタイプに関係なく使用できるものと、特定のタイプのメディアにだけ使用できるものがあります。たとえば、source パラメータと mimeType パラメータは、あらゆるタイプのメディアに対して使用できますが、scaleGridRight パラメータはイメージにのみ適用されます。

次の表は、あらゆるタイプの埋め込みアセットに対して使用できるパラメータを示します。詳細については、1016 ページの「source パラメータについて」および 1017 ページの「MIME タイプについて」を参照してください。

パラメータ	説明
source	埋め込むアセットの名前とパスを指定します。パスは、埋め込みステートメントを含むファイルへの絶対パスと相対パスのいずれでもかまいません。埋め込むアセットは、ローカルに格納されているアセットでなければなりません。したがって、埋め込むアセットの URL を指定することはできません。 パスの設定の詳細については、1016 ページの「埋め込みアセットへのパスの指定について」を参照してください。
mimeType	アセットの MIME タイプを指定します。

次の表は、イメージおよび Sprite オブジェクトにのみ使用できるパラメータを示します。詳細については、1025 ページの「埋め込みイメージでの Scale-9 フォーマットシステムの使用」を参照してください。

パラメータ	内容
scaleGridTop	Scale-9 フォーマットシステムを使用するイメージの上端から上グリッドまでの距離を、ピクセル単位で指定します。拡大・縮小されない元のイメージのサイズに対する相対値で距離を指定します。
scaleGridBottom	Scale-9 フォーマットシステムを使用するイメージの上端から下グリッドまでの距離を、ピクセル単位で指定します。拡大・縮小されない元のイメージのサイズに対する相対値で距離を指定します。
scaleGridLeft	Scale-9 フォーマットシステムを使用するイメージの左端から左グリッドまでの距離を、ピクセル単位で指定します。拡大・縮小されない元のイメージのサイズに対する相対値で距離を指定します。
scaleGridRight	Scale-9 フォーマットシステムを使用するイメージの左端から右グリッドまでの距離を、ピクセル単位で指定します。拡大・縮小されない元のイメージのサイズに対する相対値で距離を指定します。

次の表は、SWF ファイル固有のパラメータを示します。詳細については、1023 ページの「SWF ファイルの埋め込み」を参照してください。

パラメータ	内容
symbol	埋め込む SWF ファイル内のシンボルを指定します。アドビ システムズ社の Macromedia Flash Player 8 以前でのみ使用できます。

source パラメータについて

ほとんどの場合、source パラメータを指定しなければ、何かを埋め込むことはできません。

source パラメータは [Embed] メタデータタグのデフォルトパラメータなので、このパラメータ以外に指定するパラメータがない場合は、次の例のように、source パラメータ名を明示的に記述しなくても、値を指定することができます。

```
<mx:Style>
  .myCustomButton {
    overSkin:Embed("overIconImage.gif");
    upSkin:Embed(source="upIconImage.gif");
    downSkin:Embed(source="downIconImage.gif");
  }
</mx:Style>
```

埋め込みアセットへのパスの指定について

次の例のように、イメージまたは URL への完全修飾パスを指定できます。

```
<mx:Button label="Icon Button"
  icon="@Embed(source='c:¥myapp¥assets¥logo.gif')"/>
<mx:Button label="Icon Button"
  icon="@Embed(source='http://host.com/myapp/assets/logo.gif')"/>
```

メモ パスの区切り記号としてバックスラッシュ文字 (\) を使用しないでください。

パスがバックスラッシュ文字で始まっていない場合は、最初に、[Embed] メタデータタグを含むファイルとの関係でファイルが検索されます。たとえば、testEmbed.mxml ファイルには次のコードが含まれます。

```
<mx:Button label="Icon Button" icon="@Embed(source='assets/logo.gif')"/>
```

この例では、testEmbed.mxml ファイルを含むディレクトリ内で、assets という名前のサブディレクトリが検索されます。イメージが見つからない場合は、アプリケーションに関連付けられた SWC ファイル内のイメージが検索されます。

パスがバックスラッシュ文字で始まる場合は、MXML ファイルのディレクトリ内でアセットが検索され、次にソースパスが検索されます。Flex コンパイラにソースパスを指定するには、source-path コンパイラオプションを使用します。たとえば、source-path オプションを次のように使用します。

```
-source-path=a1,a2,a3
```

次に MXML ファイル a1/testEmbed.mxml が次のコードを使用します。

```
<mx:Button label="Icon Button" icon="@Embed(source='/assets/logo.gif')"/>
```

a1/assets、a2/assets、a3/assets の順にこのファイルが検索されます。イメージが見つからない場合は、アプリケーションに関連付けられた SWC ファイル内のイメージが検索されます。

MXML ファイルが a2/testEmbed.mxml のように a2 ディレクトリに存在する場合は、最初に a2 ディレクトリが検索され、次に source-path オプションで指定されたディレクトリが検索されます。

MIME タイプについて

オプションで mimeType パラメータを使用すると、読み込むアセットの MIME タイプを指定できます。mimeType パラメータを指定しなかった場合、Flex は、ファイルの拡張子に基づいて、読み込むファイルのタイプを推測します。mimeType パラメータを指定した場合、このパラメータはデフォルトで推測されるアセットの種類をオーバーライドします。

Flex では、次の MIME タイプがサポートされます。

- application/octet-stream
- application/x-font
- application/x-font-truetype
- application/x-shockwave-flash
- audio/mpeg
- image/gif
- image/jpeg
- image/png
- image/svg
- image/svg-xml

[Embed] メタデータタグの使用

[Embed] メタデータタグを使用して、JPEG、GIF、PNG、SVG、SWF、TTF、および MP3 ファイルをインポートできます。.

[Embed] メタデータタグは、必ず変数定義の前に挿入します。変数のタイプは Class 型です。次の例では、イメージファイルをロードし、それに imgCls 変数を割り当て、その変数を使用して Image コントロールの source プロパティに値を設定しています。

```
<?xml version="1.0"?>
<!-- embed\ImageClass.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100" height="80" borderStyle="solid">

    <mx:Script>
        <![CDATA[
            [Embed(source="logo.gif")]
            [Bindable]
            public var imgCls:Class;
        ]]>
    </mx:Script>

    <mx:Image source="{imgCls}"/>
</mx:Application>
```

Flex では、source プロパティに imgCls 変数を設定する際にデータバインディングを使用します。imgCls 変数定義の前に [Bindable] メタデータタグを記述しない場合は、アプリケーションの起動時に一度だけデータバインディングが実行されます。[Bindable] メタデータタグを記述した場合は、imgCls 変数へのすべての変更が認識され、変数の変更時に、この変数を使用するすべてのコンポーネントが更新されます。

一般に、アセットを埋め込むことは、それ以外の方法と比較すると、柔軟性に富むやり方です。その理由には、一度読み込んだアセットをアプリケーション内で何度も使用できることや、アセットを更新したときにはデータバインディングによって更新をアプリケーション全体へ反映できることなどが挙げられます。

MXML での @Embed() ディレクティブの使用

Button や TabNavigator など、Flex コンポーネントの多くは、icon プロパティなど、コントロールにイメージを指定するためのプロパティを備えています。MXML で @Embed( ) ディレクティブを使用してプロパティ値を指定することによって、イメージアセットを埋め込むことができます。SWF ファイルや SFW ファイル内のアセットなど、サポートされていればどのようなグラフィックファイルでも、@Embed( ) ディレクティブを使用して埋め込むことができます。

@Embed( ) ディレクティブを使用して、MXML タグプロパティを設定するか、または、子タグを使用してプロパティ値を設定できます。@Embed( ) ディレクティブは Class 型または String 型の値を返します。コンポーネントのプロパティが String 型の場合、@Embed( ) ディレクティブはストリングを返します。コンポーネントのプロパティが Class 型の場合、@Embed( ) ディレクティブはクラスを返します。他のデータ型の値を取るプロパティと共に @Embed( ) ディレクティブを使用するとエラーが発生します。

次の例では、Button コントロールを作成し、@Embed( ) ディレクティブを使用してこのコントロールの icon プロパティを設定します。

```
<?xml version="1.0"?>
<!-- embed\ButtonAtEmbed.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button label="Icon Button" icon="@Embed(source='logo.gif')"/>

</mx:Application>
```

スタイルシートでのアセットの埋め込み

Flex コンポーネントの多くのスタイルプロパティでは、読み込まれたアセットをサポートします。スタイルプロパティを使用してコンポーネントにスキンを設定することは頻繁に行われます。" スキニング " とは、コンポーネントのビジュアルエレメントを修正または置換することで外観を変更する処理です。これらのエレメントを構成するのは、イメージ、SWF ファイル、または描画 API メソッドが含まれるクラスファイルなどです。

スキニングとスタイルシートを使用したアセットの埋め込みの詳細については、741 ページ、第 20 章の「スキンの使用」を参照してください。

各種アセットの埋め込み

このセクションでは、イメージ、SWF ファイル、サウンドファイルなど、さまざまなタイプのメディアを読み込む方法について説明します。

JPEG、GIF、および PNG イメージの埋め込み

Flex では、JPEG、GIF、および PNG ファイルの埋め込みがサポートされます。これらのイメージは、アイコン、スキン、およびその他のアプリケーションアセットに使用できます。

埋め込まれたイメージを実行時に操作する必要がある場合があります。これを行うには、イメージを表すオブジェクトのデータ型を判別してから、オブジェクトの適切な ActionScript メソッドおよびプロパティを使用します。

たとえば、ActionScript で [Embed] メタデータタグを使用して GIF イメージを埋め込みます。次を参照してください。

```
[Embed(source="logo.gif")]
[Bindable]
public var imgCls:Class;
```

この例では、埋め込みイメージを表す imgCls という名前のクラスを定義しています。JPEG、GIF、および PNG ファイルを埋め込むときは、mx.core.BitmapAsset クラスのサブクラスへの参照として imgCls が定義されます。mx.core.BitmapAsset クラスは flash.display.BitmapData クラスのサブクラスです。

ActionScript では、埋め込みイメージを表すオブジェクトを作成し、コントロールに渡す前に操作することができます。そのためには、次の例に示すように、埋め込まれるクラス型を持つオブジェクトを作成し、操作してから、そのオブジェクトをコントロールへ渡します。

```
<?xml version="1.0"?>
<!-- embed/EmbedAccessClassObject.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[

            import mx.core.BitmapAsset;

            [Embed(source="logo.gif")]
            [Bindable]
            public var imgCls:Class;

            public var varOne:String = "This is a public variable";

            private function modImage():void {
                // Create an object from the embed class.
                // Since the embedded image is a GIF file,
                // the data type is BitmapAsset.
```

```
                var imgObj:BitmapAsset = new imgCls() as BitmapAsset;

                // Modify the object.
                imgObj.bitmapData.noise(4);

                // Write the modified object to the Image control.
                myImage.source=imgObj;
            }
        ]]>
    </mx:Script>

    <mx:HBox>
        <mx:Image id="myImageRaw" source="{imgCls}"/>
        <mx:Image id="myImage" creationComplete="modImage();"/>
        <mx:Text id="myText" text="foo"/>
    </mx:HBox>
</mx:Application>
```

この例では、最初の Image コントロールによって変更前のイメージが表示され、2 番目の Image コントロールによって変更されたイメージが表示されます。オブジェクトの変更には、mx.core.BitmapAsset クラスの bitmapData プロパティを使用します。bitmapData プロパティのタイプは flash.display.BitmapData なので、オブジェクトの操作に、BitmapData クラスのすべてのメソッドとプロパティを使用できます。

SVG イメージの埋め込み

Flex は、SVG (Scalable Vector Graphics) イメージや SVGZ ファイル (GZip 圧縮された SVG イメージ) をアプリケーションに読み込むことができます。このため、SVG イメージを読み込み、読み込みんだ SVG イメージを Flex コントロールのアイコンとして使用することができます。

Flex は SVG 1.1 仕様のサブセットをサポートしているため、計量可能な 2 次元のベクターグラフィックを読み込むことができます。サポートされる内容としては、基本的な SVG ドキュメント構造や CSS (Cascading Style Sheet : カスケーディングスタイルシート) のスタイル設定、変換、パス、基本シェイプ、色の他、テキスト、ペイント、グラデーション、フォントのサブセットなどが挙げられます。Flex は、SVG アニメーション、スクリプト、インタラクティブ機能を含む SVG イメージの読み込みはサポートしていません。

たとえば、ActionScript で [Embed] メタデータタグを使用して SVG イメージを埋め込みます。次を参照してください。

```
[Embed(source="logo.svg")]
[Bindable]
public var imgCls:Class;
```

この例では、埋め込みイメージを表す imgCls という名前のクラスを定義しています。SVG ファイルを埋め込むときは、mx.core.SpriteAsset クラスのサブクラスへの参照として imgCls が定義されます。mx.core.SpriteAsset クラスは flash.display.Sprite クラスのサブクラスです。したがって、SpriteAsset クラスのメソッドとプロパティを使用してイメージを操作できます。読み込んだイメージを操作する例については、1020 ページの「JPEG、GIF、および PNG イメージの埋め込み」を参照してください。

サウンドの埋め込み

Flex では、再生可能な MP3 サウンドファイルの埋め込みをサポートします。次の例では、[再生] および [停止] ボタンを持つ単純なメディアプレイヤーを作成しています。

```
<?xml version="1.0"?>
<!-- embed/EmbedSound.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import flash.media.*;

            [Embed(source="sample.mp3")]
            [Bindable]
            public var sndCls:Class;

            public var snd:Sound = new sndCls() as Sound;
            public var sndChannel:SoundChannel;

            public function playSound():void {
                sndChannel=snd.play();
            }

            public function stopSound():void {
                sndChannel.stop();
            }
        ]]>
    </mx:Script>

    <mx:HBox>
        <mx:Button label="play" click="playSound();"/>
        <mx:Button label="stop" click="stopSound();"/>
    </mx:HBox>
</mx:Application>
```

この例では、埋め込み MP3 ファイルを表す sndCls という名前のクラスを定義しています。MP3 ファイルを埋め込むときは、mx.core.SoundAsset のサブクラスへの参照として sndCls が定義されます。mx.core.SoundAsset は、flash.media.Sound クラスのサブクラスです。

Flex では、ファイル名にスペースや区切り記号が含まれていても、正当なファイル名であれば処理できます。MP3 のファイル名に二重引用符が含まれる場合は、ファイル名を一重引用符で囲んでください。

Flex でサウンドファイルを使用する方法は、サウンドファイルを埋め込む方法だけではありません。Sound クラスを使用して、実行時にサウンドファイルをロードすることもできます。詳細については、Adobe Flex 2 リファレンスガイド で Sound クラスを参照してください。

SWF ファイルの埋め込み

Flex は、Flash SWF ファイルの埋め込みを完全にサポートします。このセクションでは、さまざまなタイプの SWF ファイルの埋め込み方法について説明します。

Flash Player 8 以前用の SWF ファイルの埋め込み

Flash Player 8 以前用に作成された SWF ファイルを埋め込むことができます。埋め込んだ場合、Flex 2 アプリケーションは、埋め込み SWF ファイルとやり取りできません。つまり、Flash Player 8 以前用に作成された SWF ファイルのプロパティまたはメソッドには、Flex 2 アプリケーション内の ActionScript を使用してアクセスできません。

メモ

Flash Player 8 以前用に作成されたファイルと Flex 2 アプリケーションの間のやり取りには、flash.net.LocalConnection クラスを使用できます。

たとえば、ActionScript で [Embed] メタデータタグを使用して SWF ファイルを埋め込みます。次を参照してください。

```
[Embed(source="icon.swf")]
[Bindable]
public var imgCls:Class;
```

この例では、埋め込み SWF ファイルを表す imgCls という名前のクラスを定義します。imgCls は、mx.core.MovieClipLoaderAsset クラスのサブクラスへの参照として定義されます。mx.core.MovieClipLoaderAsset クラスは、flash.disaply.MovieClip クラスのサブクラスです。したがって、MovieClipLoaderAsset クラスのメソッドとプロパティを使用してイメージを操作できます。

SWF シンボルの埋め込み

Flex では、埋め込み SWF ファイルに書き出されているシンボルを参照できます。シンボルに依存ファイルが存在する場合、Flex はそれらの依存ファイルも埋め込みます。それ以外の場合は、指定されたシンボルだけが SWF ファイルから埋め込まれます。シンボルを参照するには、次のように symbol パラメータを指定します。

```
[Embed(source='SWFFileName.swf', symbol='symbolName')]
```

メモ

Flash で定義されるシンボルのタイプは、Button、MovieClip、および Graphic の 3 つです。Button および MovieClip シンボルは Flex アプリケーションに埋め込むことができますが、Graphic シンボルは埋め込むことができません。これは、Graphic シンボルを ActionScript 用に書き出すことができないためです。

この機能は、SWF ファイルに複数のシンボルが書き出されているときに、その一部だけを Flex アプリケーションにロードするような場合に使用します。こうして生成された Flex SWF ファイルは、アプリケーションに必要なシンボルだけがロードされているため、SWF ファイル全体を読み込んだ場合よりも、サイズを小さく抑えることができます。

Flex アプリケーションは、SWF ファイルをいくつでも読み込むことができます。ただし、2 つの SWF ファイルが同じファイル名であり、書き出されたシンボル名が同一である場合、SWF ファイルが別々のディレクトリに存在していたとしても、このように重複したシンボルを参照することはできません。

SWF ファイルに ActionScript コードが含まれる場合は、コンパイル時に警告が出力され、埋め込まれたシンボルから ActionScript が除去されます。これは、埋め込むことができるのはシンボル自体のみであることを意味します。

次の例は、各種シェイプのライブラリを含む SWF ファイルから緑色の四角形を読み込みます。

```
<?xml version="1.0"?>
<!-- embed\EmbedSWFSymbol.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >

    <mx:VBox id="vBox0" width="300" height="300" >
        <mx:Image id="image0"
            source="@Embed(source='circleSquare.swf', symbol='greenSquare')"
        />
    </mx:VBox>
</mx:Application>
```

ActionScript の [Embed] メタデータを使用してシンボルを埋め込む場合は、そのシンボルを表すオブジェクトにアクセスできます。次のコードを参照してください。

```
[Embed(source='shapes.swf', symbol='greenSquare')]
[Bindable]
public var imgCls:Class;
```

この例では、埋め込みシンボルを表す imgCls という名前のクラスを定義しています。内部では、次のいずれかのクラスのサブクラスへの参照として imgCls が定義されます。

SpriteAsset　シングルフレームの SWF ファイル用です。

MovieClipLoaderAsset　マルチフレームの SWF ファイル用です。

Flex 2 アプリケーションを表す SWF ファイルの埋め込み

Flex 2 アプリケーションを表す SWF ファイルを埋め込むことができます。たとえば、ActionScript で [Embed] メタデータタグを使用して SWF ファイルを埋め込みます。次を参照してください。

```
[Embed(source="flex2.swf")]
[Bindable]
public var flexAppCls:Class;
```

この例では、埋め込み SWF ファイルを表す flexAppCls という名前のクラスを定義します。flexAppCls は、mx.core.MovieClipLoaderAsset クラスのサブクラスへの参照として定義されます。mx.core.MovieClipLoaderAsset クラスは、flash.disaply.MovieClip クラスのサブクラスです。したがって、MovieClipLoaderAsset クラスのメソッドとプロパティを使用して埋め込み SWF ファイルを操作できます。

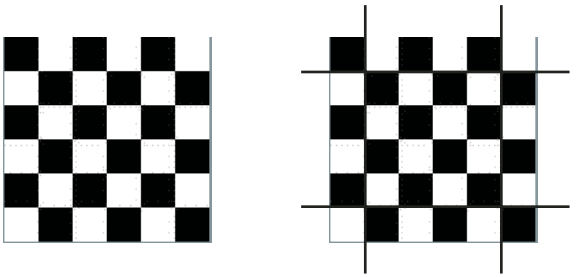
通常、Flex 2 アプリケーションを埋め込むのは、埋め込む側のアプリケーションと埋め込まれる側のアプリケーションがやり取りする必要がない場合です。埋め込みアプリケーションと埋め込みアプリケーションのやり取りが必要な場合は、独立したアプリケーションとしてではなく、カスタムコンポーネントとしてアプリケーションを実装することを検討する必要があります。

または、SWFLoader コントロールを使用して Flex 2 アプリケーションを実行時にロードする場合は、埋め込む側のアプリケーションが、ロードされる側のアプリケーションとやり取りして、そのアプリケーションのプロパティとメソッドにアクセスにアクセスできます。詳細および使用例については、298 ページの「ロードした Flex 2 アプリケーションの操作」を参照してください。

埋め込みイメージでの Scale-9 フォーマットシステムの使用

Flex は、埋め込みイメージの Scale-9 フォーマットをサポートします。Scale-9 フォーマットでは、1 つのイメージを、それぞれ独立して拡大・縮小する 9 つのセクションに分割して定義します。9 つの領域は、イメージ内に引かれた 2 本の水平グリッドと 2 本の垂直グリッドで定義されます。イメージの内側に 縦 3 x 横 3 のグリッドができます。イメージに囲み枠や四隅の飾りがある場合、グラフィック全体を拡大・縮小するよりも、Scale-9 フォーマットを使用する方が柔軟に操作できます。

次の例は、あるイメージと、そのイメージが Scale-9 のグリッドで定義された領域に分割された状態を示します。



Scale-9 フォーマットを使用する埋め込みイメージを拡大・縮小すると、すべてのテキストおよびグラデーションが通常どおり拡大・縮小します。ただし、他のタイプのオブジェクトには次のルールが適用されます。

- 中央領域のコンテンツは通常どおり拡大・縮小します。
- 左上、右上、左下、右下のコンテンツは拡大・縮小しません。
- 上側領域と下側領域にあるコンテンツは水平方向にのみ拡大・縮小します。左側領域と右側領域にあるコンテンツは垂直方向にのみ拡大・縮小します。
- すべての塗り (ビットマップ、ビデオ、グラデーションを含む) は、形状に収まるように伸縮されます。

イメージを回転した後で拡大・縮小を適用すると、すべて通常の拡大・縮小になり、Scale-9 フォーマットが定義されていない場合と同じ結果になります。

Scale-9 フォーマットを適用するには、埋め込みステートメント内で `scaleGridTop`、`scaleGridBottom`、`scaleGridLeft`、および `scaleGridRight` の 4 つのパラメータを定義します。これらのパラメータの詳細については、1015 ページの「埋め込みパラメータ」を参照してください。

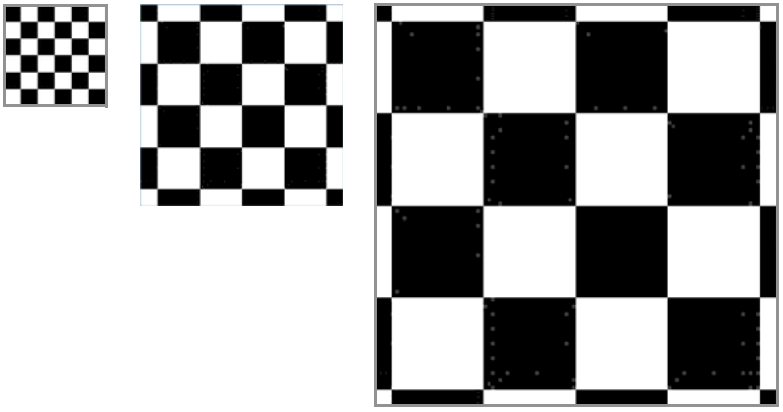
Flash Professional を使用して指定された Scale-9 情報が、埋め込み SWF ファイルに既に含まれる場合があります。こうした SWF ファイルでは、埋め込みステートメントで指定した Scale-9 パラメータは無視されます。

次の例は、枠線を持つ Flex ロゴを作成します。Scale-9 フォーマットを使用して、イメージのサイズを変更しても、枠線のサイズが変わらないようにしています。

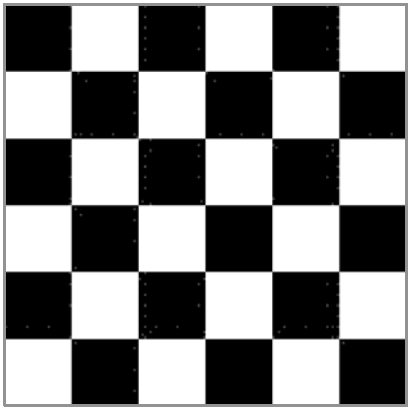
```
<?xml version="1.0"?>
<!-- embed\Embed9slice.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="1200" height="600">

    <mx:Script>
        <![CDATA[
            [Embed(source="slice_9_grid.gif",
```

```
                scaleGridTop="25", scaleGridBottom="125",
                scaleGridLeft="25", scaleGridRight="125")]
            [Bindable]
            public var imgCls:Class;
        ]]>
    </mx:Script>

    <mx:HBox>
        <mx:Image source="{imgCls}"/>
        <mx:Image source="{imgCls}" width="300" height="300"/>
        <mx:Image source="{imgCls}" width="450" height="450"/>
    </mx:HBox>
</mx:Application>
```

元のイメージは 30 x 30 ピクセルです。上記のコードでは、5 ピクセルの枠線のサイズが維持される、サイズ変更可能なイメージが生成されます。



Scale-9 フォーマットを省略すると、拡大・縮小されたイメージが元のイメージとまったく同じに表示されます。次の図を参照してください。



この例では、埋め込みイメージを表す imgCls という名前のクラスを定義しています。イメージが、Scale-9 フォーマットを使用する SWF ファイルの場合は、mx.core.SpriteAsset クラスのサブクラスとして imgCls が定義されます。mx.core.SpriteAsset は、flash.display.Sprite クラスのサブクラスです。したがって、そのオブジェクトの操作には、SpriteAsset クラスのすべてのプロパティとメソッドを使用できます。

第 31 章

モジュール化アプリケーションの作成

31

Flex アプリケーションに動的にロードするモジュールを作成できます。

内容

モジュール化アプリケーションの概要

このセクションでは、モジュールについて説明し、モジュールがモジュール化アプリケーションで使用される方法について説明します。

モジュールについて

" モジュール " とは、アプリケーションによりロードまたはアンロードされる SWF ファイルのことです。モジュールをアプリケーションから独立して実行されることはできませんが、任意の数のアプリケーションがモジュールを共有することはできます。

モジュールを使用すると、アプリケーションを複数の部分、つまりモジュールに分割できます。シェルと呼ばれるメインアプリケーションでは、必要に応じて他のモジュールを動的にロードすることができます。シェルの起動時にすべてのモジュールをロードする必要はありません。また、ユーザーが利用しないモジュールについては、ロードの必要はありません。アプリケーションでモジュールが必要なくなったら、モジュールをアンロードしてメモリとリソースを解放することができます。

モジュール化アプリケーションには次の利点があります。

- SWF ファイルの初期ダウンロードサイズが小さくなります。
- SWF ファイルサイズが小さいほど、ロード時間が短くなります。
- アプリケーションの関連側面のカプセル化が向上します。たとえば、“ レポート ” 機能を、個別に作業できるようにモジュールに分割することができます。

モジュールのメリット

タイム共有ライブラリ (RSL) と似ている点として、モジュールでは、コードがアプリケーションから分離されて、個別にロードされる SWF ファイルに含まれています。モジュールは実行時にロードまたはアンロードでき、アプリケーションがなくてもコンパイルできるので、RSL よりも柔軟性が高くなっています。

モジュールを使用するメリットがある 2 つの一般的なシナリオは、異なるユーザーパスを持つ大規模アプリケーションと、ポータルアプリケーションです。

1 番目のよくあるシナリオの例としては、生命保険、自動車保険、健康保険、歯科保険、旅行保険、ペット獣医保険のための数千の画面を持つ、大規模な保険用アプリケーションが挙げられます。

RIA (リッチインターネットアプリケーション) 設計の従来のアプローチでは、MXML クラスの階層ツリーを持つ、完全に統合されたアプリケーションを構築しました。アプリケーションのメモリ使用と起動時間は著しく大きくなり、新しい機能セットを追加するたびに SWF ファイルサイズが増大します。

ただし、このアプリケーションを使用する場合、ユーザーは 1 つの画面サブセットにのみアクセスします。画面をリファクタリングして、要求時にロードされるモジュールの小さなグループに分割することにより、メインアプリケーションの体感上のパフォーマンスを向上させ、メモリ使用を抑えることができます。また、アプリケーションをモジュールに分割した場合、設計のカプセル化が向上するため、開発者の生産性を高めることができます。アプリケーションを再構築するとき、開発者はアプリケーション全体を再コンパイルするのではなく、単一のモジュールのみ再コンパイルするだけですみます。

2 番目の一般的なシナリオの例は、多数のポートレット用のサービスを提供する、ActionScript 3 で記述されたメインポータルアプリケーションを使用したシステムです。ポートレットは、ユーザーごとにダウンロードされるデータに基づいて設定されます。従来のアプローチでは、既知のすべてのポートレットをコンパイルする 1 つのアプリケーションを構築しました。この方法は、デプロイと開発の両面で効率的ではありません。

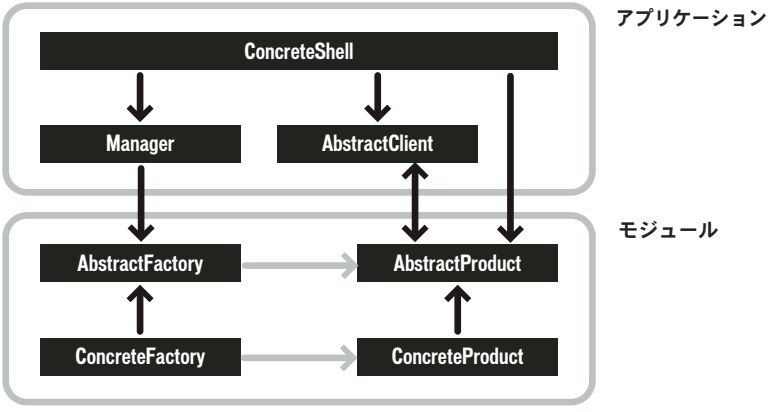
モジュールを使用することにより、ポータルサービスを含むインターフェイスと、汎用ポートレットインターフェイスを設定できます。XML データを使用して、特定のセッションでどのモジュールをロードする必要があるかを判別できます。モジュールがロードされると、そのモジュール内のクラスファクトリへのハンドラを取得し、そのハンドルを使用してポートレットインターフェイスを実装するクラスのインスタンスを作成します。このシナリオでは、完全な再コンパイルは、インターフェイスが変更された場合にのみ必要です。

モジュール API の詳細

モジュールは、標準のインターフェイスを使用してクラスファクトリを実装します。クラスファクトリの生成物がシェルにとって既知のインターフェイスを実装するか、シェルがモジュールにとって既知のインターフェイスを実装します。

それらの共有インターフェイスは共有インターフェイス定義を使用するので、シェルとモジュールの間のハード面での依存関係を軽減できます。これにより、タイプセーフな通信が提供され、SWF ファイルサイズを大幅に増大させることなく抽象レイヤーを適用することができます。

次の図は、シェルとモジュールインターフェイスの関係を示しています。



モジュール化アプリケーションの作成

モジュール化アプリケーションを作成するには、モジュールごとの別個のクラスと、モジュールをロードするアプリケーションを作成します。

モジュール化アプリケーションを作成するには :

1. 任意の数のモジュールを作成します。MXML ベースのモジュールファイルのルートタグは <mx:Module> です。ActionScript ベースのモジュールは、ModuleBase クラスを拡張したものです。
2. アプリケーションの場合と同じように、各モジュールをコンパイルします。そのためには、mxmlc コマンドラインコンパイラ、または Adobe Flex Builder に組み込まれたコンパイラを使用します。
3. Application クラスを作成します。通常、これは MXML ファイルで、そのルートタグは <mx:Application> ですが、ActionScript 専用アプリケーションの場合もあります。

4. Application ファイルでは、<mx:ModuleLoader> タグを使用して、各モジュールをロードします。mx.modules.ModuleLoader クラスの load() メソッドを使用することもできます。ModuleBase クラスを拡張したクラスの場合、ModuleManager クラスのメソッドを使用してロードする必要があります。

以降のセクションでは、これらの手順について詳しく説明します。

モジュールの作成

モジュールは、アプリケーションファイルと同じようにクラスです。ActionScript でモジュールを作成するには、mx.modules.ModuleBase クラスを拡張するファイルを作成します。MXML でモジュールを作成するには、ルートタグが <mx:Module> のファイルを作成することにより、mx.modules.Module クラスを拡張します。タグ内に、そのモジュール内で使用されている名前空間が追加されていることを確認してください。ファイルの先頭には、型宣言タグを指定する必要があります。

次の例は、Chart コントロールを含むモジュールです。

```
<?xml version="1.0"?>
<!-- modules/ColumnChartModule.mxml -->
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml" width="100%" height="100%"
>
    <mx:Script><![CDATA[
        import mx.collections.ArrayCollection;
        [Bindable]
        public var expenses:ArrayCollection = new ArrayCollection([
            {Month:"Jan", Profit:2000, Expenses:1500},
            {Month:"Feb", Profit:1000, Expenses:200},
            {Month:"Mar", Profit:1500, Expenses:500}
        ]);
    ]]></mx:Script>
    <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
```

```
                displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
</mx:Module>
```

MXML ベースのモジュールは、他のモジュールをロードすることができます。さらに、それらのモジュールが他のモジュールをロードし、それを続けることができます。

モジュールのコンパイル

モジュールのコンパイルには、他の Flex アプリケーションのコンパイルと同じく、mxmlc コマンドラインコンパイラまたは Flex Builder コンパイラを使用します。次のコマンドは、最も簡単な mxmlc コマンドです。

```
mxmlc MyModule.mxml
```

この結果として、モジュールとしてアプリケーションにロードする SWF ファイルが生成されます。モジュールベースの SWF ファイルをスタンドアローン Flash アプリケーションとして実行することや、ブラウザウィンドウにロードすることはできません。1 つのモジュールとしてアプリケーションにロードする必要があります。

モジュールサイズの制御

モジュールサイズは、モジュールで使用されるコンポーネントとクラスによって異なります。デフォルトでは、モジュールにはモジュールのコンポーネントが依存しているすべてのフレームワークコードが含まれます。そのため、アプリケーションのクラスと重複するクラスがリンクされるので、モジュールのサイズが大きくなる場合があります。

モジュールのサイズを削減するには、アプリケーションに含まれているクラスを外部化するようにモジュールに指示することができます。これには、カスタムクラスとフレームワーククラスが含まれます。この結果、モジュールには必要とするクラスのみ含まれ、フレームワークコードと他の依存関係はアプリケーションに含まれます。

フレームワーククラスを外部化するには、mxmlc コマンドを使用して、モジュールをロードするアプリケーションからリンカーレポートを生成します。このレポートを、モジュールの `load-externs` コンパイラオプションへの入力として使用します。

リンカーレポートを作成するには :

1. リンカーレポートを生成します。

```
mxmlc -link-report=report.xml MyApplication.mxml
```

2. アプリケーションの SWF ファイルをコンパイルします。

```
mxmlc MyApplication.mxml
```

3. モジュールをコンパイルします。

```
mxmlc -load-externs=report.xml MyModule.mxml
```

モジュールの再コンパイル

変更を加えた場合は、モジュールを再コンパイルする必要があります。メインアプリケーションを再コンパイルしても、モジュールの再コンパイルはトリガされません。同様に、アプリケーションファイルを変更した場合、リンカーレポートまたは共通コードに影響する可能性のある変更でない限り、モジュールを再コンパイルする必要はありません。

メモ

`load-externs` オプションを使用してモジュールの依存関係を外部化した場合、そのモジュールが、Adobe Flex の将来のバージョンと互換性を持たなくなる可能性があります。モジュールの再コンパイルが必要となる場合があります。将来の Flex アプリケーションでモジュールを確実に使用できるようにするには、モジュールとモジュールが必要とするすべてのクラスをコンパイルします。このことは、他のアプリケーション内でロードするアプリケーションにも当てはまります。

モジュールのロードとアンロード

モジュールをロードまたはアンロードするには、ModuleLoader クラスの `load()` および `unload()` メソッドを使用します。これらのメソッドはパラメータは受け取りません。ModuleLoader は、現在の `url` プロパティと値が一致するモジュールをロードまたはアンロードします。

次の例では、ボタンをクリックしたときに、モジュールをロードまたはアンロードします。

```
<?xml version="1.0"?>
<!-- modules/ASModuleLoaderApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import mx.modules.*;

        public function createModule(m:ModuleLoader, s:String):void {
            if (!m.url) {
                m.url = s;
                return;
            }
            m.loadModule();
```

```
        }

        public function removeModule(m:ModuleLoader):void {
            m.unloadModule();
        }
        ]]>
    </mx:Script>

    <mx:Panel title="Module Example"
        height="90%"
        width="90%"
        paddingTop="10"
        paddingLeft="10"
        paddingRight="10"
        paddingBottom="10"
    >
        <mx:TabNavigator id="tn"
            width="100%"
            height="100%"
            creationPolicy="auto"
        >
            <mx:VBox id="vb1" label="Column Chart Module">
                <mx:Button
                    label="Load"
                    click="createModule(chartModuleLoader, l1.text)"
                />
                <mx:Button
                    label="Unload"
                    click="removeModule(chartModuleLoader)"
                />
                <mx:Label id="l1" text="ColumnChartModule.swf"/>
              <mx:ModuleLoader id="chartModuleLoader"/>
            </mx:VBox>

            <mx:VBox id="vb2" label="Form Module">
                <mx:Button
                    label="Load"
                    click="createModule(formModuleLoader, l2.text)"
                />
                <mx:Button
                    label="Unload"
                    click="removeModule(formModuleLoader)"
                />
                <mx:Label id="l2" text="FormModule.swf"/>
                <mx:ModuleLoader id="formModuleLoader"/>
            </mx:VBox>
        </mx:TabNavigator>
    </mx:Panel>
</mx:Application>
```

ModuleLoader の場所を設定すると、loadModule() メソッドへの呼び出しもトリガされます。この呼び出しは、設定された url プロパティを使用して ModuleLoader を初めて作成したときに発生します。また、そのプロパティの値を変更したときにも発生します。

次の例では、<mx:ModuleLoader> タグで url プロパティが設定されているため、loadModule() メソッドの呼び出さないでモジュールをロードします。

```
<?xml version="1.0"?>
<!-- modules/URLModuleLoaderApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Panel
        title="Module Example"
        height="90%"
        width="90%"
        paddingTop="10"
        paddingLeft="10"
        paddingRight="10"
        paddingBottom="10"
    >

        <mx:Label width="100%" color="blue"
            text="Select the tabs to change the panel."/>

        <mx:TabNavigator id="tn"
            width="100%"
            height="100%"
            creationPolicy="auto"
        >
            <mx:VBox id="vb1" label="Column Chart Module">
                <mx:Label id="l1" text="ColumnChartModule.swf"/>
                <mx:ModuleLoader url="ColumnChartModule.swf"/>
            </mx:VBox>

            <mx:VBox id="vb2" label="Form Module">
                <mx:Label id="l2" text="FormModule.swf"/>
                <mx:ModuleLoader url="FormModule.swf"/>
            </mx:VBox>

        </mx:TabNavigator>
    </mx:Panel>
</mx:Application>
```

モジュールをロードするとき、そのモジュールへの load() メソッドを何回呼び出したかに関わらず、Flex では必ず 1 つのモジュールのコピーだけがロードされます。

モジュールは、現在のアプリケーションドメインの子にロードされます。異なるアプリケーションドメインを指定するには、ModuleLoader クラスの applicationDomain プロパティを使用します。

名前が同じだが、実装が異なる 2 つのクラスをロードする場合、最初にロードされたものが使用されるものとなります。

異なるサーバーからのモジュールのロード

あるサーバーから、異なるサーバー上で実行されているアプリケーションにモジュールをロードするには、モジュールとそのモジュールをロードするアプリケーションとの間に、信頼関係を確立する必要があります。

ドメイン間のアクセスを許可するには :

1. ロードするアプリケーションで allowDomain() メソッドを呼び出し、モジュールのロード元であるターゲットドメインを指定する必要があります。そのためアプリケーションの初期化前イベントハンドラでロード対象ドメインを指定し、モジュールがロードされる前に、確実にアプリケーションが設定されるようにします。
2. モジュールが置かれているリモートサーバーのクロスドメインファイルで、ロードアプリケーションを実行しているサーバーを指定するエントリを追加します。
3. ロードするアプリケーションの初期化前イベントハンドラで、リモートサーバー上のクロスドメインファイルをロードします。
4. ロードされたモジュールで allowDomain() メソッドを呼び出し、メソッドがローダーと通信できるようにします。

次の例は、ロード元のアプリケーションの init() メソッドを示しています。

```
public function setup():void {
  Security.allowDomain("remoteservername");
  Security.loadPolicyFile("http://remoteservername/crossdomain.xml");
  var request:URLRequest = new URLRequest("http://remoteservername
    /crossdomain.xml");
  var loader:URLLoader = new URLLoader();
  loader.load(request);
}
```

次の例は、ロードされたモジュールの init() メソッドを示しています。

```
public function initMod():void {
  Security.allowDomain("loaderservername");
}
```

次の例は、リモートサーバーに存在するクロスドメインファイルを示しています。

```
<!-- crossdomain.xml file located at the root of the server -->
<cross-domain-policy>
  <allow-access-from domain="loaderservername" to-ports="*"/>
</cross-domain-policy>
```

クロスドメインポリシーファイルの使用の詳細については、『Flex 2 アプリケーションの構築とデプロイ』の第 4 章の「Flex セキュリティの適用」を参照してください。

ModuleLoader イベントの使用

ModuleLoader クラスは、setup、ready、loading、unload、progress、error、および urlChanged を含むいくつかのイベントをトリガします。これらのイベントを使用して、ロードプロセスの状況を追跡し、モジュールがアンロードされたタイミングや ModuleLoader のターゲット URL が変更されたタイミングを判別することができます。

エラーイベントの使用

error イベントは、何らかの理由でモジュールがロードされなかったときに、安全な方法で失敗させるための機会を与えます。次の例では、Button コントロールを使用して、モジュールをロードおよびアンロードします。error イベントをトリガするには、TextInput コントロールの URL を、実在しないモジュールに変更します。エラーハンドラによりユーザーにメッセージが表示され、エラーメッセージがトレースログに書き込まれます。

```
<?xml version="1.0"?>
<!-- modules/ErrorEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import mx.events.ModuleEvent;
        import mx.modules.*;
        import mx.controls.Alert;

        private function errorHandler(e:ModuleEvent):void {
            Alert.show("There was an error loading the module." +
                " Please contact the Help Desk.");
            trace(e.errorText);
        }

        public function createModule():void {
            chartModuleLoader.url = ti1.text;
            chartModuleLoader.loadModule();
        }

        public function removeModule():void {
            chartModuleLoader.unloadModule();
        }

        ]]>
    </mx:Script>

    <mx:Panel title="Module Example"
        height="90%"
        width="90%"
        paddingTop="10"
        paddingLeft="10"
```

```
        paddingRight="10"
        paddingBottom="10"
    >
        <mx:HBox>
            <mx:Label text="URL:"/>
            <mx:TextInput width="200" id="ti1" text="ColumnChartModule.swf"/>
            <mx:Button label="Load" click="createModule()"/>
            <mx:Button label="Unload" click="removeModule()"/>
        </mx:HBox>
        <mx:ModuleLoader id="chartModuleLoader" error="errorHandler(event)"/>
    </mx:Panel>
</mx:Application>
```

progress イベントの使用

progress イベントを使用して、モジュールのロード状況を追跡できます。progress イベントのリスナーを追加すると、Flex は、モジュールのロードプロセスの間、リスナーを一定の間隔で呼び出します。リスナーが呼び出されるたびに、イベントの bytesLoaded プロパティを参照できます。この値と bytesTotal プロパティを比較することにより、完了のパーセント値を取得できます。

次の例では、モジュールのロードプロセスの間、完了の程度を報告します。また、ロードがどれほど完了に近づいたかをユーザーに示す、簡単なプログレスバーが作成されます。

```
<?xml version="1.0"?>
<!-- modules/SimpleProgressEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import mx.events.ModuleEvent;
        import flash.events.ProgressEvent;
        import mx.modules.*;

        [Bindable]
        public var progBar:String = "";
        [Bindable]
        public var progMessage:String = "";

        private function progressEventHandler(e:ProgressEvent):void {
            progBar += ".";
            progMessage =
                "Module " +
                Math.round((e.bytesLoaded/e.bytesTotal) * 100) +
                "% loaded";
        }

        public function createModule():void {
            chartModuleLoader.loadModule();
        }
```

```
        public function removeModule():void {
            chartModuleLoader.unloadModule();
            progBar = "";
            progMessage = "";
        }
        ]]>
    </mx:Script>

    <mx:Panel title="Module Example"
        height="90%"
        width="90%"
        paddingTop="10"
        paddingLeft="10"
        paddingRight="10"
        paddingBottom="10"
    >
        <mx:HBox>
            <mx:Label id="l2" text="{progMessage}"/>
            <mx:Label id="l1" text="{progBar}"/>
        </mx:HBox>

        <mx:Button label="Load" click="createModule()"/>
        <mx:Button label="Unload" click="removeModule()"/>

        <mx:ModuleLoader
            id="chartModuleLoader"
            url="ColumnChartModule.swf"
            progress="progressEventHandler(event)"
        />
    </mx:Panel>
</mx:Application>
```

モジュールローダーを ProgressBar コントロールに接続することもできます。次の例では、ProgressBar コントロールが含まれる ModuleLoader 用のカスタムコンポーネントを作成します。ProgressBar コントロールは、モジュールのロード状況を表示します。

```
<?xml version="1.0"?>
<!-- modules/MySimpleModuleLoader.mxml -->
<mx:ModuleLoader xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            private function clickHandler():void {
                if (!url) {
                    url="ColumnChartModule.swf";
                }
                loadModule();
            }
        ]]>
    </mx:Script>

    <mx:ProgressBar
```

```
        id="progress"
        width="100%"
        source="{this}"
    />
    <mx:HBox width="100%">
      <mx:Button
        id="load"
        label="Load"
        click="clickHandler()"
      />
      <mx:Button
        id="unload"
        label="Unload"
        click="unloadModule()"
      />
      <mx:Button
        id="reload"
        label="Reload"
        click="unloadModule();loadModule();"
      />
    </mx:HBox>
</mx:ModuleLoader>
```

次の例に示すように、単純なアプリケーション内でこのモジュールを使用できます。

```
<?xml version="1.0"?>
<!-- modules/ComplexProgressEventHandler.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">

    <mx:Panel title="Module Example"
        height="90%"
        width="90%"
        paddingTop="10"
        paddingLeft="10"
        paddingRight="10"
        paddingBottom="10"
    >
        <mx:Label text="Use the buttons below to load and unload
            the module."/>
        <local:MySimpleModuleLoader id="customLoader"/>
    </mx:Panel>

</mx:Application>
```

この例では、すべてのイベントについて ProgressBar の `label` プロパティを変更するわけではありません。たとえば、あるモジュールをロードしてからアンロードした場合、`label` プロパティは "LOADING 100%" のままです。label プロパティを調整するには、`unload` や `error` など、ModuleLoader イベント用の他のイベントハンドラを定義する必要があります。

第 32 章

History Manager の使用

32

Adobe Flex History Manager を使用すると、ユーザーは、Web ブラウザの [戻る] および [進む] ナビゲーション機能を使用して、Flex アプリケーション内を移動できます。

目次

履歴管理について

Flex History Manager を使用すると、ユーザーは、Web ブラウザの [戻る] および [進む] ナビゲーション機能を使用して、Flex アプリケーション内を移動できます。たとえば、ユーザーは、Flex アプリケーションのいくつかの Accordion コンテナペインを移動した後、ブラウザの [戻る] ボタンをクリックしてアプリケーションを前の状態に戻すことができます。

履歴管理は、アプリケーションのラッパー内で参照されるファイルのセットとして実装されます。デフォルトでは、Adobe Flex Builder は、履歴管理をサポートするラッパーを生成します。Flex データサービスに含まれる Web 層コンパイラも、デフォルトで履歴管理をサポートするラッパーを生成します。どちらの場合も、履歴管理サポートを無効にできます。

コマンドラインコンパイラを使用して Flex アプリケーションを作成する場合は、独自のラッパーを作成するか、またはラッパーテンプレートのいずれか 1 つを使用する必要があります。ラッパーは、履歴管理をサポートするようにカスタマイズできます。詳細については、1052 ページの「カスタムラッパー内での履歴管理の使用」を参照してください。カスタムラッパーの記述については、Flex 2 アプリケーションの構築および展開ガイドの第 16 章の「ラッパーの作成」を参照してください。

Flex コンポーネントの場合、履歴管理は、Accordion コンテナや TabNavigator コンテナなどのナビゲータコンテナによって自動的にサポートされます。また、ActionScript 内で HistoryManager クラスを使用して、アプリケーション内の他のオブジェクトのカスタム履歴管理を提供することや、HistoryManager クラスのメソッドを呼び出すことができます。

標準履歴管理の使用

Accordion および TabNavigator ナビゲータコンテナでは、履歴管理がデフォルトで有効になっています。これは、ユーザーが Accordion コントロールのいずれかのペインを選択したときに、ブラウザの [戻る] ボタンまたは [戻る] ナビゲーション機能を使用して、前のペインに戻ることができることを意味します。ViewStack ナビゲータコンテナでは、履歴管理がデフォルトで無効になっています。

履歴管理が有効になっている場合、ユーザーがアプリケーションの異なるナビゲータコンテナ内を移動するのに応じて、それぞれのナビゲーション状態が保存されます。Web ブラウザの [戻る] または [進む] ボタンを選択すると、保存されていた前のナビゲーション状態または次のナビゲーション状態が表示されます。履歴管理は、アプリケーション内の位置を追跡する機能であり、行った操作を記憶する取り消しおよびやり直し機能ではありません。

メモ

ナビゲータコンテナのような特定のコンポーネントに対して履歴管理が有効になっている場合は、ナビゲータコンテナの状態のみが保存されます。ナビゲータコンテナの子コンポーネントの状態は、特にそのコンポーネントに対して履歴管理が追加されていない限り、保存されません。

ナビゲーション状態がどのように保存および復元されるかについては、1051 ページの「HistoryManager クラスによる状態の保存とロード」を参照してください。

コンテナの historyManagementEnabled プロパティを false または true に設定することで、ナビゲータコンテナに対する履歴管理をそれぞれ無効または有効にすることができます。TabNavigator コンテナの履歴管理を無効にする例を次に示します。

```
<mx:TabNavigator historyManagementEnabled="false">
```

次の例では、ユーザーのパネル選択は、1 番目の Accordion コンテナに対して保存されます。これは、1 番目の Accordion コンテナにはデフォルト設定が使用されていて、2 番目の Accordion コンテナには historyManagementEnabled プロパティが明示的に false に設定されているためです。ユーザーが Web ブラウザの [戻る] または [進む] コマンドを選択すると、2 番目のコンテナではなく 1 番目のコンテナの前の状態または次の状態が表示されます。

```
<?xml version="1.0"?>
<!-- historymanager/DisableHistoryManagement.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="800">

  <!-- History management is enabled by default for this Accordion. -->
  <mx:Accordion width="100%" height="50%">
     <mx:VBox label="History management is ENABLED">
        <mx:TextInput text="View 1"/>
     </mx:VBox>
     <mx:VBox label="View 2">
        <mx:TextInput text="View 2"/>
     </mx:VBox>
  </mx:Accordion>

  <!-- History management is disabled for this Accordion. -->
  <mx:Accordion historyManagementEnabled="false" width="100%" height="50%">
     <mx:VBox label="History management is DISABLED.">
        <mx:TextInput text="View 1"/>
     </mx:VBox>
     <mx:VBox label="View 2">
        <mx:TextInput text="View 2"/>
     </mx:VBox>
  </mx:Accordion>
</mx:Application>
```

ラッパーから履歴管理サポートを削除すると、アプリケーション全体に対する履歴管理を無効にできます。詳細については、1052 ページの「カスタムラッパー内での履歴管理の使用」を参照してください。

カスタム履歴管理の使用

カスタムコンポーネントに次の操作を行うことで、コンポーネントを履歴管理対応にすることができます。

1. mx.managers.IHistoryManagerClient インターフェイスを実装します。
2. コンポーネントを HistoryManager の `register()` メソッドに登録します。
3. コンポーネントの状態が変化したときに状態を保存します。
4. IHistoryManagerClient インターフェイスの `saveState()` および `loadState()` メソッドを実装します。これらのメソッドのシグネチャは次のとおりです。

```
public function saveState():Object
public function loadState(state:Object):void;
```

以降のセクションで、これらの手順を詳しく説明します。

IHistoryManagerClient インターフェイスの実装

IHistoryManagerClient インターフェイスを実装するには、`implements` タグ属性を使用します。例を示します。

```
<mx:CheckBox
  xmlns:mx="http://www.adobe.com/2006/mxml"
  implements="mx.managers.IHistoryManagerClient"
  ...
>
```

コンポーネントを HistoryManager に登録する

HistoryManager クラスにコンポーネントを登録するには、HistoryManager クラスの `register()` メソッドを、IHistoryManagerClient インターフェイスを実装するコンポーネントインスタンスの参照と共に呼び出します。例を示します。

```
<mx:CheckBox
     xmlns:mx="http://www.adobe.com/2006/mxml"
     implements="mx.managers.IHistoryManagerClient"
     creationComplete="mx.managers.HistoryManager.register(this);"
>
```

通常は、コンポーネントの初期化が終了したときに呼び出します。

コンポーネントの状態の保存

コンポーネントの状態が変化したときは状態を保存して、History Manager が状態を返せるようにします。通常は、イベントハンドラで静的な HistoryManager.save() メソッドを呼び出して保存します。

次の例では、change イベントハンドラで boxChanged() メソッドを呼び出します。

```
<mx:CheckBox
   xmlns:mx="http://www.adobe.com/2006/mxml"
   implements="mx.managers.IHistoryManagerClient"
   creationComplete="mx.managers.HistoryManager.register(this);"
   change="boxChanged(event)">
<mx:Script><![CDATA[
   import mx.managers.HistoryManager;
   ...

   // ボックスの現在の状態を保存する
   private function boxChanged(e:Event):void {
      HistoryManager.save();
   }
]]></mx:Script>
```

loadState() メソッドと saveState() メソッドの実装

登録されているコンポーネントに対して HistoryManager クラスを使用するには、IHistoryManagerClient インターフェイスを実装し、目的の状態情報を保存およびロードするための saveState() メソッドと loadState() メソッドを含める必要があります。saveState() メソッドは、コンポーネントの現在のナビゲーション状態を表す " プロパティ : 値 " ペアを含むオブジェクトを返します。

HistoryManager クラスには、load() メソッドがあります。load() メソッドは、saveState() メソッドが返すオブジェクトと同じオブジェクトを持つそれぞれの登録済みコンポーネントに対して loadState() メソッドを呼び出します。

IHistoryManagerClient を実装するコンポーネントは、loadState() メソッドも実装する必要があります。UIComponent を拡張するコンポーネントは、自動的に loadState() メソッドを継承します。

次の例では、カスタム CheckBox コントロールの loadState() および saveState() メソッドを実装します。

```
<?xml version="1.0"?>
<!-- historymanager/MyCheckBox.mxml -->
<mx:CheckBox
     xmlns:mx="http://www.adobe.com/2006/mxml"
     label="Check me"
     selected="false"
     implements="mx.managers.IHistoryManagerClient"
     creationComplete="mx.managers.HistoryManager.register(this);"
     change="boxChanged(event)">

  <mx:Script><![CDATA[
        import mx.managers.HistoryManager;

        // Returns an object that contains property:value pairs that
        // represent the current navigation state of a component.
        public function saveState():Object {
           return {selected:selected};
        }

        // Sets the selected property, depending on the state.
        public function loadState(state:Object):void {
           var newState:Boolean = state;

           if (newState != selected) {
            selected = newState;
           } else {
            if (newState) {
                selected = false;
            } else {
                selected = true;
            }
           }
        }

        // Saves the box's current state.
        private function boxChanged(e:Event):void {
           HistoryManager.save();
        }
  ]]></mx:Script>

</mx:CheckBox>
```

このコントロールを使用するアプリケーションは、次のようなものになります (MXML ファイルが同じディレクトリにある場合)。

```
<?xml version="1.0"?>
<!-- historymanager/CheckBoxApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:local="*">

  <local:MyCheckBox/>

</mx:Application>
```

ユーザーがカスタム CheckBox コントロールをオン / オフしたとき、ブラウザの [進む] および [戻る] ボタンを使用して、コントロールの前の状態に戻ることができます。

アプリケーションのナビゲーション状態の合計は、ユーザーの Web ブラウザでサポートされる最大 URL サイズに制限されます。したがって、できる限り小さな量のデータを保存するようにコンポーネントの saveState() メソッドを記述する必要があります。たとえば、List コントロールに対して、selectedIndex プロパティだけを保存する saveState() メソッドを記述できます。

リストベースのコントロールでの履歴管理の使用

通常、リストベースのコントロールで履歴管理を使用するときは、コントロールの選択されたインデックスを保存し、そのインデックスを使用して状態を判定します。

次の例は、MXML コンポーネントの HistoryList.mxml です。このコンポーネントは、HistoryManager に登録されて saveState() メソッドと loadState() メソッドを実装します。このコンポーネントによって、ユーザーは List コントロールを閲覧できるようになります。

```
<?xml version="1.0"?>
<!-- historymanager/HistoryList.mxml -->
<mx:List xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="initList()"
  change="listChanged()"
  implements="mx.managers.IHistoryManagerClient"
>
  <mx:Script><![CDATA[
     import mx.managers.HistoryManager;

     // Register with the HistoryManager.
     private function initList():void {
        HistoryManager.register(this);
     }

     // Saves the application's current state.
     private function listChanged():void {
        HistoryManager.save()
     }

     // Save the List index.
```

```
    public function saveState():Object {
       return { selectedIndex: selectedIndex };
    }

    // Load the List index.
    public function loadState(state:Object):void {
       var newIndex:int = state ? int(state.selectedIndex) : -1;
       if (newIndex != selectedIndex)
       selectedIndex = newIndex;
    }
  ]]></mx:Script>
</mx:List>
```

次に示すのは、HistoryList.mxml コンポーネントを使用するアプリケーションファイルの例です。

```
<?xml version="1.0"?>
<!-- historymanager/HistoryListApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:local="*" width="400" height="400" verticalGap="20">

<mx:Script>
<![CDATA[

[Bindable]
private var listData:Array = ["Flex", "Dreamweaver",
"Flash", "Breeze", "Contribute"];
]]>
</mx:Script>

<local:HistoryList id="list1" dataProvider="{listData}"
width="120" height="120"/>

<mx:TextInput id="text1"
text="{list1.selectedItem? list1.selectedItem : ''}"/>
</mx:Application>
```

この例では、HistoryList コントロールにストリングの配列を設定します。HistoryList コントロールのアイテムを 1 つ選択すると、選択したアイテムが TextInput コントロールにも表示されます。ブラウザの [戻る] ボタンを使用すると、選択したアイテムを順に戻ることができます。

HistoryManager クラスの静的メソッドの呼び出し

ViewStack、Accordion、または TabNavigator コンテナを使用した場合、ユーザーがアプリケーション内を移動すると HistoryManager の `save()` メソッドが自動的に呼び出されます。HistoryManager クラスにコンポーネントを登録するときは、HistoryManager `save()` メソッドを明示的に呼び出してコンポーネントの状態を保存する必要があります。

`save()` メソッドと、コンポーネントの登録と登録解除を行うための `register()` メソッドおよび `unregister()` メソッドは、ActionScript コード内から呼び出すことができる静的メソッドです。これらのメソッドについて次の表で説明します。

メソッド	説明
`register(component)`	HistoryManager クラスにコンポーネントを登録します。例 : `HistoryManager.register(myList);`
`save()`	HistoryManager クラスに登録されているすべてのコンポーネントの現在のナビゲーション状態を保存します。例 : `HistoryManager.save();`
`unregister(component)`	HistoryManager クラスからコンポーネントを登録解除します。例 : `HistoryManager.unregister(myList);`

HistoryManager クラスによる状態の保存とロード

履歴管理機能は、`navigateToURL()` グローバル関数を使用して、不可視の HTML フレームを現在の Web ブラウザウィンドウにロードします。次に、Flex アプリケーションのナビゲーション状態を、不可視のフレームの URL クエリパラメータにエンコードします。次に、不可視フレーム内の SWF ファイル history.swf がクエリパラメータをデコードし、ナビゲーション状態を HistoryManager クラスに送ります。このセクションでは、HistoryManager クラスがどのようにナビゲーションデータを URL クエリ文字列にエンコードし、またどのようにそのナビゲーションデータをデコードしてナビゲーション状態を復元するかについて説明します。

ナビゲーション状態のデータのエンコード

HistoryManager クラスの `save()` メソッドは、登録されているそれぞれのコンポーネントについて、`saveState()` メソッドから返された状態オブジェクトを収集します。`save()` メソッドは、各オブジェクトの各プロパティを、標準 `prop1=value1&prop2=value2` 形式を使用するクエリ文字列にエンコードします。それぞれのプロパティが属するコンポーネントを識別するため、登録されている適切なコンポーネントの状態 ID が、ダッシュと共にそれぞれのプロパティ名に追加されます。

状態 ID は、`3da7` のような 4 文字の 16 進数ストリングです。これは登録されているコンポーネントの、アプリケーションのビジュアル階層内での完全パス名をハッシュ化したものです。

たとえば、状態 ID が 3da7 で、次の状態オブジェクトを返す TabNavigator コンテナがあるとします。

```
{selectedIndex:5}
```

この場合のクエリ文字列は、次のようになります。

```
3da7-selectedIndex=5
```

ナビゲーション状態データのデコードと復元

history.swf ファイルは、格納されている状態プロパティを 1 つのオブジェクトとして Flex アプリケーションに渡します。HistoryManager クラスは、このオブジェクトのプロパティから状態 ID を抽出し、登録されている各コンポーネントの状態オブジェクトを再構築します。状態オブジェクトは、Application オブジェクトとその子で構築されます。たとえば、Accordion コンテナが ViewStack コンテナ内の 3 番目の項目である場合、Accordion コンテナのナビゲーション状態を復元する前に、ViewStack コンテナをその 3 番目の項目に設定する必要があります。

カスタムラッパー内での履歴管理の使用

ラッパーを自動的に生成する代わりに Flex アプリケーションをカスタムラッパー内に配置する場合、履歴管理を使用するには、履歴管理をサポートするようにラッパーを設定する必要があります。履歴管理をサポートするラッパーは、Flex Builder を設定することによって生成できます。Adobe Flex データサービスに含まれる Web 層コンパイラも、履歴管理をサポートするラッパーを生成できます。

また、履歴管理をサポートするラッパーテンプレートが、/resources/html-templates ディレクトリ内の次のディレクトリにあります。

- /client-side-detection-with-history
- /express-installation-with-history
- /no-player-detection-with-history

ラッパーのカスタマイズの詳細については、Flex 2 アプリケーションの構築および展開ガイドの第 16 章の「ラッパーの作成」を参照してください。

Flex データサービスを使用しない場合の履歴管理の使用

Flex データサービスを使用せずに履歴管理をサポートするには、アプリケーション内に次のファイルをデプロイする必要があります。

- history.js
- history.htm
- history.swf

これらのファイルは、任意の /resources/html-templates/directory_name-with-history ディレクトリに存在します。履歴管理を使用するためにこれらのファイルを変更する必要はありませんが、これらのファイルを参照するようにアプリケーションのラッパーを編集する必要があります。

Flex データサービスを使用せずに履歴管理をサポートするためには、次の手順を行う必要があります。

1. テキストエディタでラッパーを開きます。
2. 次の例に示すように、history.js ファイルを読み込みます。

   ```
   <script language="JavaScript" type="text/JavaScript" src="history.js"/>
   ```

3. 次のように、ページの下部にある iframe エレメントに history.htm ファイルを挿入します。

   ```
   <iframe name="_history.htm" src="history.htm" width="22" height="0"/>
   ```

   iframe の `name` プロパティを _history.htm (アンダースコア付き) に設定する必要があります。`height` プロパティは任意の値に設定できますが、Internet Explorer で動作させるには、`width` プロパティは 22 以上にする必要があります。history.htm ページはユーザーが使用しないので、ページに表示する必要はありません。
4. 次の例に示すように、`historyUrl` パラメータおよび `lconid` パラメータを `object` タグおよび `embed` タグの `flashVars` 変数に追加します。

   履歴管理では lc_id という JavaScript 変数が使用されるので、これらのパラメータを JavaScript で追加する必要があります。

   ```
   document.write("flashvars='historyUrl=history%5Fhtml&lconid=" + lc_id'");
   ```

5. history.js および history.html ファイルを Web サーバーにデプロイし、これらのファイルの位置がラッパーの設定と一致することを確認します。
6. history.swf ファイルを Web サーバーにデプロイし、このファイルが history.htm ファイルと同じ位置にあることを確認します。

   history.swf ファイルの位置を変更する場合は、history.htm ファイル内の history.swf ファイルへのパスを必ず更新してください。

ラッパーが履歴管理をサポートしていても、プレーヤーのバージョン検出ロジックまたは高速インストールのサポートを含んでいない場合は、history.js ファイルと myscript.js ファイルの内容を合成する必要があります。

Flex データサービスを使用する場合の履歴管理の使用

Flex データサービスを使用している場合は、特別な Flex 内部アクションを呼び出して、履歴管理をサポートするファイルを生成できます。Flex データサービスのデフォルト設定では、これらの特別なアクションがラッパーに追加されます。このラッパーは、クライアントが MXML ファイルのみを要求したときに生成されます。一般にカスタムラッパーは、履歴管理サポートを追加するために修正する必要があります。

Flex データサービスを使用して履歴管理をサポートするためには、次の手順を行う必要があります。

1. Web アプリケーションの正しいコンテキストルートを使用して、次のテキストを HTML ドキュメントの先頭に記述します。

```
<script language='javascript' charset='utf-8'
  src='/flex/flex-internal?action=history_js'></script>
```

2. 次の例に示すように、historyUrl パラメータおよび lconid パラメータを object タグおよび embed タグの flashVars 変数に追加します。

   履歴管理では lc_id という JavaScript 変数が使用されるので、これらのパラメータを JavaScript で追加する必要があります。

```
document.write("
  flashvars='historyUrl=%2Fflex%2Fflex%2Dinternal%3Faction%3Dhistory%5Fhtml&
  lconid=" + lc_id'");
```

3. 次の例に示すように、_history iframe エレメントを追加します。height プロパティの値は任意ですが、width プロパティの値は 22 以上である必要があります。

```
<iframe src='/flex/flex-internal?action=history_html' name='_history'
  frameborder='0' scrolling='no' width='22' height='0'></iframe>
```

第 33 章

プリント

33

多くの Adobe Flex アプリケーションでは、アプリケーションの中からプリントすることができます。たとえば、ユーザーが買い物をした後に確認情報を表示するアプリケーションがあるとします。アプリケーションでは、ページ上の情報を記録用にプリントすることができます。

このトピックでは、mx.printing.FlexPrintJob および mx.printing.PrintDataGrid クラスを使用するときの、プリントのオプションについて説明します。

メモ

プリントは、Adobe Flash Player 9 のコンテキストメニュー、または Flash ActionScript PrintJob クラスを使用して行うこともできます。このクラスについては、ActionScript 3.0 リファレンスガイドを参照してください。

目次

Flex クラスを使用したプリントについて

Flex mx.printing パッケージには、Flex アプリケーションからのプリント出力の作成を容易にする 3 つのクラスがあります。

FlexPrintJob　1 つあるいは複数のオブジェクトをプリントするクラスです。大きなオブジェクトを自動的に分割して複数のページでプリントするようにし、ページのサイズに収まるよう出力を拡大および縮小します。

PrintDataGrid　プリント用にカスタマイズされたデフォルトの外観を持つ、DataGrid コントロールのサブクラスです。このクラスには、追加のサイズ変更機能やプリント機能を提供するプロパティやメソッドがあります。

FlexPrintJobScaleType　FlexPrintJob `addObject()` メソッドで使用される定数を定義します。

これらのクラスを使用することにより、アプリケーションからの情報をプリントする方法を制御できます。たとえば、アプリケーションで、画面上の情報の選択されたサブセットのみをプリントすることや、表示されていない情報をプリントすることができます。また、この情報を再フォーマットしたり、レイアウトや外観をプリント用に最適化したりすることもできます。

ユーザーは、PostScript プリンタでも非 PostScript プリンタでもプリントできます。また、Adobe® の Adobe PDF および Macromedia® FlashPaper™ プリンタドライバによる出力も可能です。

FlexPrintJob クラスの使用

FlexPrintJob クラスを使用すると、Form や VBox コンテナなど、1 つあるいは複数の Flex オブジェクトをプリントできます。指定したオブジェクトごとに、オブジェクト自体とそのオブジェクトに含まれる全オブジェクトがプリントされます。オブジェクトは、表示されているインターフェイスの全部または一部にすることも、データをプリント用に特別にフォーマットするコンポーネントにすることもできます。FlexPrintJob クラスを使用すると、ページに収まるように出力を拡大および縮小することや、1 つのページに収まらないオブジェクトを自動的に複数のページにプリントすることができます。

FlexPrintJob クラスを使用すると、プリント用に特別にフォーマットした、動的にレンダリングされる文書をプリントできます。この機能は、領収書や旅行プランなどの情報や、データベースコンテンツやダイナミックテキストなどの外部ダイアミックコンテンツを含むその他の表示をレンダリングおよびプリントするときに役立ちます。

FlexPrintJob クラスは、イベントリスナー内で使用する場合がよくあります。たとえば、Button コントロールは、アプリケーションの一部あるいはすべてをプリントするイベントリスナーに対して使用できます。

メモ

FlexPrintJob クラスにより、オペレーティングシステムでは [印刷] ダイアログボックスが表示されます。ユーザーのアクションなしでは、プリントできません。

プリントジョブの構築と送信

次の手順で説明するとおり、プリントジョブを構築および送信することで出力をプリントします。

プリントジョブを構築および送信するには :

1. FlexPrintJob クラスのインスタンスを作成します。

   ```
   var printJob:FlexPrintJob = new FlexPrintJob();
   ```

2. プリントジョブを開始します。

   ```
   printJob.start();
   ```

   オペレーティングシステムにより、[印刷] ダイアログボックスが表示されます。

3. プリントジョブに 1 つまたは複数のオブジェクトを追加し、拡大および縮小する方法を指定します。

   ```
   printJob.addObject(myObject, FlexPrintJobScaleType.MATCH_WIDTH);
   ```

   新規のページでオブジェクトが開始されます。

4. プリントジョブをプリンタに送信します。

   ```
   printJob.send();
   ```

5. 必要でないオブジェクトをすべて解放します。

メモ

`start()` メソッドを呼び出してから `send()` メソッドを呼び出すまでの間、プリントジョブはユーザーのオペレーティングシステム上にスプールされるので、その間はコードをプリント固有のアクティビティに限定する必要があります。たとえば、`start()` メソッドから `send()` メソッドまでの間は、Flash コンテンツでユーザーとの対話操作を実行しないでください。

次のセクションでは、これらの手順で使用するプロシージャについて詳しく説明します。

プリントジョブの開始

プリントジョブを開始するには、FlexPrintJob クラスのインスタンスを作成して、このインスタンスの `start()` メソッドを呼び出します。このメソッドを呼び出すと、プリントジョブをユーザーのオペレーティングシステム上にスプールするように Flash Player に指示が出され、ユーザーのオペレーティングシステムにより [印刷] ダイアログボックスが表示されます。

[印刷] ダイアログボックスでユーザーがプリントを開始するオプションを選択すると、`start()` メソッドは `true` の値を返します。ユーザーがプリントジョブをキャンセルすると、戻り値は `false` になります。ユーザーがオペレーティングシステムの [印刷] ダイアログボックスを閉じると、`start()` メソッドはプリンタ情報を使用して FlexPrintJob オブジェクトの `pageHeight` および `pageWidth` プロパティの値を設定します。これらのプロパティは、プリントするページ領域のサイズを表します。

メモ

ユーザーのオペレーティングシステムによっては、スプールが完了し、アプリケーションが `send()` を呼び出すまで、追加のダイアログボックスが表示される場合があります。

プリントジョブは一度に 1 つだけアクティブにすることができます。直前のプリントジョブに対して次のいずれかの処理が行われない限り、新しいプリントジョブを開始することはできません。

- `start()` メソッドが `false` を返します (ジョブが失敗しました)。
- `addObject()` メソッドの呼び出しが成功した後に `send()` メソッドの実行が完了します。`send()` メソッドは同期しているため、それ以降のコードでは、この呼び出しが無事に完了したものと想定されます。

プリントジョブへのオブジェクトの追加

FlexPrintJob クラスの `addObject()` メソッドを使用して、プリントジョブにオブジェクトを追加します。各オブジェクトは新規のページで開始されます。そのため、次のコードでは DataGrid コントロールと Button コントロールが個別のページにプリントされます。

```
printJob.addObject(myDataGrid);
printJob.addObject(myButton);
```

プリントジョブの拡大および縮小

`addObject()` メソッドの *`scaleType`* パラメータで、出力の拡大および縮小を決定します。次の FlexPrintJobScaleType クラス定数を使用して、拡大および縮小の方法を指定します。

定数	アクション
MATCH_WIDTH	(デフォルト) 使用可能なページの幅に合わせてオブジェクトを拡大および縮小します。結果としてオブジェクトの高さがページの高さを上回る場合は、複数のページに出力されます。
MATCH_HEIGHT	使用可能なページの高さ領域に合わせてオブジェクトを拡大および縮小します。結果としてオブジェクトの幅がページの幅を上回る場合は、複数のページに出力されます。
SHOW_ALL	1 ページに収まるように、1 つのサイズに合わせてオブジェクトを拡大および縮小します。つまり、MATCH_WIDTH または MATCH_HEIGHT のいずれか小さい方のタイプが選択されます。
FILL_PAGE	少なくとも 1 ページに完全に収まるようにオブジェクトを拡大および縮小します。つまり、MATCH_WIDTH または MATCH_HEIGHT のいずれか大きい方のタイプが選択されます。
NONE (なし)	出力は拡大および縮小されません。プリントされるページは、画面上のオブジェクトと同じサイズになります。オブジェクトの高さ、幅、または両方のサイズが、ページの幅または高さを上回ると、複数のページで出力されます。

1つのオブジェクトが複数のページを必要とする場合、出力はページの境界で分割されます。これにより、テキストが読み取り不能になったり、画像が不適切に分割されたりする場合があります。この問題を避けるようにプリントジョブを設定する方法の詳細については、1065 ページの「複数ページ出力のプリント」を参照してください。

FlexPrintJob クラスには、アプリケーションがプリントジョブの拡大および縮小を決定するのに役立つ 2 つのプロパティがあります。これらのプロパティは読み取り専用で、初期値は 0 です。アプリケーションが start() メソッドを呼び出し、ユーザーがオペレーティングシステムの [印刷] ダイアログボックスで [印刷] オプションを選択した場合、Flash Player はオペレーティングシステムからプリント設定を取得します。start() メソッドは、次のプロパティを設定します。

プロパティ	型	単位	説明
pageHeight	数値	ポイント	用紙上のプリント可能領域の高さです。ユーザーが設定した余白は含まれません。
pageWidth	数値	ポイント	用紙上のプリント可能領域の幅です。ユーザーが設定した余白は含まれません。

メモ

" ポイント " は長さのプリント単位で、1インチ (25.4 mm) の 72 分の 1 に相当します。Flex のプリント出力では、プリンタの設定を基にして、1インチ (72 ポイント) が 72 ピクセルにマッピングされます。

プリント操作の完了

FlexPrintJob addObject() メソッドの呼び出し後にプリントジョブをプリンタに送信するには、send() メソッドを使用します。このメソッドにより、Flash Player はプリントジョブのスプールを停止して、プリントを開始します。

プリンタにプリントジョブを送信した後、プリント専用のコンポーネントを使用してプリントの出力をフォーマットした場合は、removeChild() を呼び出してこのプリント固有コンポーネントを削除します。詳細については、1061 ページの「プリント固有の出力フォーマットの使用」を参照してください。

例 : 簡易なプリントジョブ

次の例では、DataGrid を、拡大および縮小しないで画面で表示されているとおりにプリントします。

```
<?xml version="1.0"?>
<!-- printing\DGPrint.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            import mx.printing.*;

            // Create a PrintJob instance.
            private function doPrint():void {
                // Create an instance of the FlexPrintJob class.
                var printJob:FlexPrintJob = new FlexPrintJob();

                // Start the print job.
                if (printJob.start() != true) return;

                // Add the object to print. Do not scale it.
                printJob.addObject(myDataGrid, FlexPrintJobScaleType.NONE);

                // Send the job to the printer.
                printJob.send();
            }
        ]]>
    </mx:Script>

    <mx:VBox id="myVBox">
        <mx:DataGrid id="myDataGrid" width="300">
            <mx:dataProvider>
                <mx:Object Product="Flash" Code="1000"/>
                <mx:Object Product="Flex" Code="2000"/>
                <mx:Object Product="ColdFusion" Code="3000"/>
                <mx:Object Product="JRun" Code="4000"/>
            </mx:dataProvider>
        </mx:DataGrid>
        <mx:Button id="myButton"
            label="Print"
            click="doPrint();"/>
    </mx:VBox>
</mx:Application>
```

この例では、Button コントロールが選択されると doPrint() イベントリスナーが呼び出されます。このイベントリスナーでは、DataGrid コントロールをプリントする FlexPrintJob クラスのインスタンスを作成し、`addObject()` メソッドを使用してプリントジョブに DataGrid コントロールを追加した後、`send()` メソッドを使用してページをプリントします。

ページ上に DataGrid コントロールと Button コントロールをプリントするには、addObject() メソッドの *object* パラメータで myVBox を指定します。myVBox は、両方のコントロールを含むオブジェクトの ID です。DataGrid コントロールと Button コントロールを個別のページにプリントする場合は、それぞれのオブジェクトを個別の addObject() メソッドで指定します。

ページ幅に合わせて DataGrid コントロールをプリントするには、addObject() メソッドの 2 番目のパラメータを省略するか、FlexPrintJobScaleType.MATCH_WIDTH を指定します。1 ページに収まる最大サイズで DataGrid コントロールをプリントするには、FlexPrintJobScaleType.SHOW_ALL を指定します。前の例では、DataGrid が短いので、FlexPrintJobScaleType.SHOW_ALL の結果は FlexPrintJobScaleType.MATCH_WIDTH と同じです。

プリント固有の出力フォーマットの使用

多くの場合、プリント出力は画面の表示とは異なります。画面上で使用されている水平方向のレイアウトが、用紙へのプリントには適していない場合があります。画面上の表示要素によっては、印刷結果の妨げになるものもあります。プリント内容と表示内容を変える必要がある場合は、他にもあります。たとえば、パスワードフィールドを省略する必要がある場合などです。

プリント固有の内容と外観で出力をプリントするには、画面レイアウトとプリントレイアウトで別のオブジェクトを使用し、画面レイアウトで使用されたデータを使用してプリントレイアウトを行います。次に、FlexPrintJob addObject() メソッドへの呼び出しで、このプリントレイアウトを使用します。

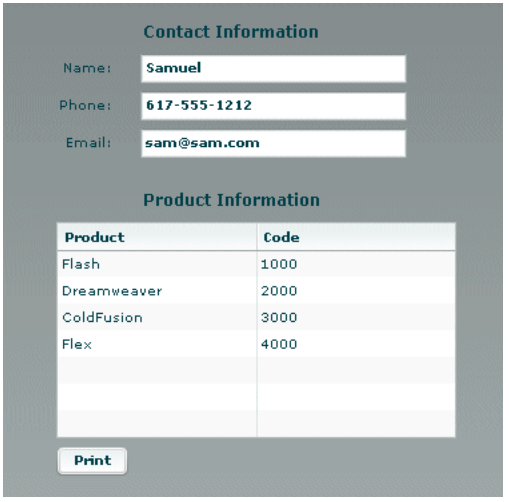
フォームに DataGrid コントロールが含まれている場合は、プリントレイアウトで PrintDataGrid コントロールを使用します。プリント出力に関して、PrintDataGrid コントロールには DataGrid コントロールに勝る 2 つの利点があります。

- プリント専用にデザインされたデフォルトの外観があります。
- 複数のデータページが含まれるプリントグリッドをサポートする、プロパティやメソッドがあります。

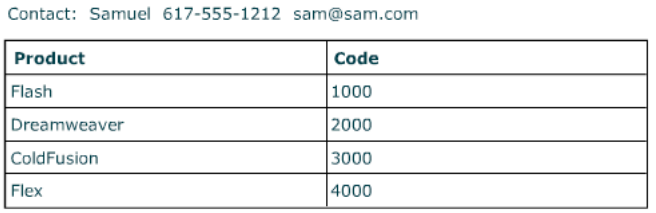
1062 ページの「例 : 簡易なプリント固有の出力フォーマット」のコードでは、簡単な PrintDataGrid コントロールを使用します。PrintDataGrid コントロールの使用の詳細については、1066 ページの「複数ページのグリッドに対する PrintDataGrid コントロールの使用」を参照してください。

例 : 簡易なプリント固有の出力フォーマット

このセクションの例では、連絡先情報を入力するための 3 つのテキストボックスと、数行に渡る製品情報を表示するためのデータグリッドを作成します。次の図は、このアプリケーションの出力が画面に表示された状態を示しています。



次のイメージは、ユーザーが [印刷] ボタンをクリックしてデータをプリントしたときの、出力の外観を示しています。



この例では、MXML アプリケーションファイルは画面を表示してプリントを制御します。個別のカスタム MXML コンポーネントにより、プリント出力の外観が決定されます。

ユーザーが [印刷] ボタンをクリックすると、アプリケーションの `doPrint()` メソッドは以下の操作を実行します。

1. プリントジョブを作成および開始して、オペレーティングシステムの [印刷] ダイアログボックスを表示します。
2. ユーザーが [印刷] ダイアログボックスでプリント操作を開始すると、myPrintView コンポーネントを使用して子コントロールを作成します。

3. フォームデータから MyPrintView コントロールのデータを設定します。

4. プリントジョブをプリンタに送信します

5. プリント固有の子コンポーネントを削除して、メモリをクリーンアップします。

プリント出力には、画面からのラベルは含まれていません。アプリケーションでは、画面の 3 つの入力ボックスのテキストが 1 つにまとめられて、Label コントロール内のプリント用文字列に挿入されます。

次のコードでは、アプリケーションファイルのコンテンツが表示されます。

```
<?xml version="1.0"?>
<!-- printing\DGPrintCustomComp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    height="450"
    width="550">

    <mx:Script>
        <![CDATA[
            import mx.printing.FlexPrintJob;
            import myComponents.MyPrintView;

            public function doPrint():void {
                // Create a FlexPrintJob instance.
                var printJob:FlexPrintJob = new FlexPrintJob();

                // Start the print job.
                if(printJob.start()) {
                    // Create a MyPrintView control as a child
                    // of the current view.
                    var formPrintView:MyPrintView = new MyPrintView();
                    addChild(formPrintView);

                    // Populate the print control's contact label
                    // with the text from the form's name,
                    // phone, and e-mail controls.
                    formPrintView.contact.text =
                        "Contact: " + custName.text + "  " +
                        custPhone.text + "  " + custEmail.text;

                    // Set the print control's data grid data provider to be
                    // the displayed data grid's data provider.
                    formPrintView.myDataGrid.dataProvider =
                        myDataGrid.dataProvider;

                    // Add the SimplePrintview control to the print job.
                    // For comparison, try setting the
                    // second parameter to "none".
                    printJob.addObject(formPrintView);

                    // Send the job to the printer.
                    printJob.send();
```

```
                    // Remove the print-specific control to free memory.
                    removeChild(formPrintView);
                }
            }
        ]]>
    </mx:Script>

    <!-- The form to display-->
    <mx:Form id="myForm">
        <mx:FormHeading label="Contact Information"/>
        <mx:FormItem label="Name: ">
            <mx:TextInput id="custName"
                width="200"
                text="Samuel Smith"
                fontWeight="bold"/>
        </mx:FormItem>
        <mx:FormItem label="Phone: ">
            <mx:TextInput id="custPhone"
                width="200"
                text="617-555-1212"
                fontWeight="bold"/>
        </mx:FormItem>
        <mx:FormItem label="Email: ">
            <mx:TextInput id="custEmail"
                width="200"
                text="sam@sam.com"
                fontWeight="bold"/>
        </mx:FormItem>

        <mx:FormHeading label="Product Information"/>
        <mx:DataGrid id="myDataGrid" width="300">
            <mx:dataProvider>
                <mx:Object Product="Flash" Code="1000"/>
                <mx:Object Product="Flex" Code="2000"/>
                <mx:Object Product="ColdFusion" Code="3000"/>
                <mx:Object Product="JRun" Code="4000"/>
            </mx:dataProvider>
        </mx:DataGrid>
        <mx:Button id="myButton"
            label="Print"
            click="doPrint();"/>
    </mx:Form>
</mx:Application>
```

次の "MyPrintView.mxml" ファイルでは、アプリケーションの doPrint() メソッドが使用するコンポーネントを定義します。このコンポーネントは VBox コンテナです。このコンポーネントには、フォームの最初の 3 フィールドの連絡先情報を書き込むための Label コントロールと、画面ビューの DataGrid コントロールのデータソースからのデータを表示するための PrintDataGrid コントロールが含まれます。PrintDataGrid コントロールとプリントでの利点の詳細については、1066 ページの「複数ページのグリッドに対する PrintDataGrid コントロールの使用」を参照してください。

```
<?xml version="1.0"?>
```

```
<!-- printing\myComponents\MyPrintView.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    backgroundColor="#FFFFFF"
    height="250" width="450"
    paddingTop="50" paddingLeft="50" paddingRight="50">

    <!-- The controls to print, a label and a PrintDataGrid control. -->
    <mx:Label id="contact"/>
    <mx:PrintDataGrid id="myDataGrid" width="100%">
        <mx:columns>
            <mx:DataGridColumn dataField="Product"/>
            <mx:DataGridColumn dataField="Code"/>
        </mx:columns>
    </mx:PrintDataGrid>
</mx:VBox>
```

複数ページ出力のプリント

このセクションのトピックでは、次のような複数ページ出力を、正しいフォーマットでプリントする方法について説明します。

- 各コントロールが、単一のプリントページ以内のサイズに収まる場合。こうしたジョブは、固定長フィールドを持つフォームをプリントするときなどによく発生します。
- プリント出力に、単一ページにはプリントできない長さの PrintDataGrid コントロールが含まれる場合。特に、コントロールの高さがデータによって異なる場合。この種の出力の良い例としては、顧客の注文書が挙げられます。注文書には、最初に顧客情報、次に未確定の注文品目、最後に合計情報が記載されます。

長さが既知の複数ページ出力のプリント

複数ページに及ぶ文書の各コンポーネントの長さが分かっている場合、プリントするページごとにプリントレイアウトコンポーネントを作成し、各レイアウトページを個別の addObject() メソッドで指定することができます。次のようになります。

```
printJob.addObject(introPrintView, "ShowAll");
printJob.addObject(finDetailPrintView, "ShowAll");
printJob.addObject(hrDetailPrintView, "ShowAll");
printJob.addObject(summaryPrintView, "ShowAll");
```

複数ページのグリッドに対する PrintDataGrid コントロールの使用

複数行の DataGrid コントロールがアプリケーションの 1 つの画面に収まらない場合、通常は、すべてのデータを表示するために使用できるスクロールバーが表示されます。DataGrid コントロールをプリントするときは、出力が画面の表示と同一になります。つまり、DataGrid コントロールに、直ちに見えない行や列がある場合、それらの行や列はプリントされません。DataGrid コントロールを、高さが指定されていない (または高さが大きい) PrintDataGrid コントロールに置き換える場合、すべての行がプリントされますが、HTML 印刷結果で一般的に見られるように、一部の行が 1 ページの最下部に印刷され、一部の行が別のページの最上部に印刷される場合があります。

これらの問題は、PrintDataGrid コントロールの次の機能を使用して解決できます。これらの機能を使用すると、複数のページにまたがるデータを含むグリッドを、ページ間で行を分割することなく正しくプリントできます。

sizeToPage プロパティ　プリントするデータグリッドに、完全な行のみが含まれるようにします。

nextPage() メソッド　プリント可能な次のデータページを取得します。

validNextPage プロパティ　データをプリントするために追加のページが必要な場合は `true` です。

sizeToPage 属性を使用したページのフォーマット

PrintDataGrid ページは、コントロールの現在のビューに表示される行で構成されます。たとえば、PrintDataGrid コントロールの高さが 130 ピクセルだとします。各行とヘッダーの高さの合計は 30 ピクセルで、コントロールのデータプロバイダは 10 行のデータを提供します。この場合、プリントされる PrintDataGrid ページには 3 つの完全な行とヘッダーが含まれることになります。`sizeToPage` プロパティでは、4 番目の行を部分的に含めるかどうかを指定します。

`sizeToPage` プロパティのデフォルトは `true` です。この設定では、部分的に見えている行や空白の行が PrintDataGrid コントロールにより削除され、現在のビュー内の完全な行のみが含まれるようにコントロールのサイズが変更されます。前の段落で説明したデータグリッドで、このプロパティが `true` の場合、DataGrid は、3 つの完全なデータ行を表示するように縮小されます。不完全な行は表示されません。属性が `false` の場合、このグリッドの最下部には行の一部が表示されます。

次のプロパティは、`sizeToPage` プロパティの影響を受けるページサイズについての情報を提供します。

プロパティ	説明
`currentPageHeight`	`sizeToPage` プロパティが `true` の場合のグリッドの高さを、ピクセル単位で指定します。`sizeToPage` プロパティが `true` の場合、`currentPageHeight` プロパティは `height` プロパティと等しくなります。
`originalHeight`	`sizeToPage` プロパティが `false` の場合のグリッドの高さを、ピクセル単位で指定します。`sizeToPage` プロパティが `false` の場合、`originalHeight` プロパティは `height` プロパティと等しくなります。

通常のアプリケーションでは、sizeToPage 属性をデフォルト (true) のまま使用し、height プロパティを使用してグリッドの高さを決定します。

1 つの PrintDataGrid コントロールページがプリントページより長い場合、sizeToPage プロパティはページの分割には影響ありません。行を分割せずに複数のページのデータグリッドをプリントするには、nextPage() メソッドを使用してグリッドの項目を複数のビューに分割する必要があります。1067 ページの「nextPage() メソッドと validNextPage プロパティを使用した複数ページのプリント」を参照してください。

nextPage() メソッドと validNextPage プロパティを使用した複数ページのプリント

validNextPage プロパティは、現在のプリントページに収まる行を超えるデータが PrintDataGrid コントロールにある場合、true になります。このプロパティは、追加のページをフォーマットしてプリントする必要があるかどうかを判別するために使用します。

The nextPage() メソッドを使用すると、PrintDataGrid コントロールの最初の行を、直前の PrintDataGrid ページの最後の行に続くデータプロバイダ行として設定することにより、データプロバイダコンテンツのページ間を移動できます。言い換えると、nextPage() メソッドを使用すると、グリッドの verticalScrollPosition プロパティが、グリッドの rowCount プロパティの値の分だけ増加します。

次のコードは、行が複数ページにまたがることなく、複数のページにグリッドをプリントするループを示しています。

```
// 最初のページをキューに入れる
printJob.addObject(thePrintView);
// ページがある間、プリントを繰り返す
while (thePrintView.myDataGrid.validNextPage) {
   // 次のページのデータをビューに送る
   thePrintView.myDataGrid.nextPage();
   // 追加のページをキューに入れる
   printJob.addObject(thePrintView);
}
```

1068 ページの「例 : 複数ページの PrintDataGrid コントロールを使用したプリント」のセクションでは、nextPage() メソッドを使用して、複数ページのデータグリッドを持つレポートをプリントする方法を説明しています。

PrintDataGrid レイアウトの更新

PrintDataGrid コントロールを使用して複数ページにまたがる単一のデータグリッドをプリントするときは、このグリッドの各ページを個別にキューに入れます。ただ単に nextPage() メソッドを使用して PrintDataGrid のページを移動するだけでなく、各ページのカスタマイズも行う場合は、各ページをプリントする前に、validateNow() メソッドを呼び出してページレイアウトを更新する必要があります。1072 ページの「出力コンポーネントのプリント」を参照してください。

例 : 複数ページの PrintDataGrid コントロールを使用したプリント

次の例では、データプロバイダに項目数を指定できるデータグリッドをプリントします。これにより、DataGrid コントロールのコンテンツを 1 ページ、2 ページ、または複数のページにプリントするように設定し、印刷結果における、サイズが異なるデータセットの影響を見ることができます。

この例は、納品書や領収書のように、グリッドの前にヘッダー情報を配置し、グリッドの後にフッター情報を配置する方法も示しています。また、プリントするページに応じて、ヘッダーとフッターを表示するか非表示にするかを選択する機能を使用しています。コードをできるだけ短くするため、この例では簡単なプレースホルダー情報のみを使用しています。

アプリケーションは、次のファイルで構成されています。

- ユーザーにフォームを表示するアプリケーションファイル。フォームには、行数を設定する TextArea および Button コントロールや、[印刷] ボタンが含まれます。このファイルには、ビューを初期化し、データを取得し、ユーザーのプリント要求を処理するためのコードが含まれています。プリント出力用テンプレートとして、FormPrintView MXML コンポーネントを使用します。
- プリント出力をフォーマットするための "FormPrintView.mxml" ファイル。これには 2 つの主な要素があります。

  プリント出力テンプレート　PrintDataGrid コントロールが含まれ、2 つの MXML コンポーネントを使用してヘッダーとフッターのコンテンツをフォーマットします。

  showPage() 関数　ページのタイプに応じて、出力の特定のページに含めるテンプレートの部分を決定します。ページのタイプは、先頭、中間、最終、または単一のいずれかです。複数ページ出力の先頭ページでは、`showPage()` 関数はフッターを非表示にし、中間ページと最終ページではヘッダーを非表示にします。単一ページの場合、ヘッダーとフッターの両方が表示されます。
- 出力の最初と最後のコンテンツを指定する "FormPrintHeader.mxml" および "formPrintFooter.mxml" ファイル。アプリケーションを簡潔に保つため、ヘッダーには単一の静的 Label コントロールがあります。ヘッダーには、Quantity 列の数値の合計が表示されます。完全なアプリケーションでは、ヘッダーページに発送先住所などの情報を表示し、フッターページに発送明細を表示することもできます。

これらのファイルには、コードの目的を説明した詳しいコメントが含まれます。

複数ページプリントのアプリケーションファイル

次のコードは、複数ページプリントのアプリケーションファイルを示しています。

```
<?xml version="1.0"?>
<!-- printing\MultiPagePrint.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="initData();">

    <mx:Script>
    <![CDATA[
        import mx.printing.*;
        import mx.collections.ArrayCollection;
        // Import the MXML custom print view control.
        import myComponents.FormPrintView;

        // Declare variables and initialize simple variables.
        // The dgProvider ArrayCollection is the DataGrid data provider.
        // It must be bindable because you change its contents dynamically.
        [Bindable]
        public var dgProvider:ArrayCollection;
        public var footerHeight:Number = 20;
        public var prodIndex:Number;
        public var prodTotal:Number = 0;

        // Data initialization, called when the application initializes.
        public function initData():void {
            // Create the data provider for the DataGrid control.
            dgProvider = new ArrayCollection;
        }

        // Fill the dgProvider ArrayCollection with the specified items.
        public function setdgProvider(items:int):void {
            // First initialize the index and clear any existing data.
            prodIndex=1;
            dgProvider.removeAll();

            // Fill the ArrayCollection, and calculate a product total.
            // For simplicity, it increases the Index field value by
            // 1, and the Qty field by 7 for each item.
            for (var z:int=0; z<items; z++)
            {
                var prod1:Object = {};
                prod1.Qty = prodIndex * 7;
                prod1.Index = prodIndex++;
                prodTotal += prod1.Qty;
                dgProvider.addItem(prod1);
            }
        }

        // The function to print the output.
        public function doPrint():void {
            // Create a FlexPrintJob instance.
```

```
var printJob:FlexPrintJob = new FlexPrintJob();

// Start the print job.
if (printJob.start()) {
    // Create a FormPrintView control
    // as a child of the application.
    var thePrintView:FormPrintView = new FormPrintView();
    addChild(thePrintView);

    // Set the print view properties.
    thePrintView.width=printJob.pageWidth;
    thePrintView.height=printJob.pageHeight;
    thePrintView.prodTotal = prodTotal;

    // Set the data provider of the FormPrintView
    // component's DataGrid to be the data provider of
    // the displayed DataGrid.
    thePrintView.myDataGrid.dataProvider =
        myDataGrid.dataProvider;

    // Create a single-page image.
    thePrintView.showPage("single");

    // If the print image's DataGrid can hold all the
    // data provider's rows, add the page to the print job.
    if(!thePrintView.myDataGrid.validNextPage)
    {
        printJob.addObject(thePrintView);
    }
    // Otherwise, the job requires multiple pages.
    else
    {
        // Create the first page and add it to the print job.
        thePrintView.showPage("first");
        printJob.addObject(thePrintView);
        thePrintView.pageNumber++;

        // Loop through the following code
        // until all pages are queued.
        while(true)
        {
            // Move the next page of data to the top of
            // the PrintDataGrid.
            thePrintView.myDataGrid.nextPage();

            // Try creating a last page.
            thePrintView.showPage("last");

            // If the page holds the remaining data, or if
            // the last page was completely filled by the last
            // grid data, queue it for printing.
            // Test if there is data for another
            // PrintDataGrid page.
```

```
                        if(!thePrintView.myDataGrid.validNextPage)
                        {
                            // This is the last page;
                            // queue it and exit the print loop.
                            printJob.addObject(thePrintView);
                            break;
                        }
                        else
                        // This is not the last page. Queue a middle page.
                        {
                            thePrintView.showPage("middle");
                            printJob.addObject(thePrintView);
                            thePrintView.pageNumber++;
                        }
                    }
                }
                // All pages are queued; remove the FormPrintView
                // control to free memory.
                removeChild(thePrintView);
            }
            // Send the job to the printer.
            printJob.send();
        }
    ]]>
    </mx:Script>

    <!-- The form that appears on the user's system.-->
    <mx:Form id="myForm" width="80%">
        <mx:FormHeading label="Product Information"/>
                <mx:DataGrid id="myDataGrid" dataProvider="{dgProvider}">
                <mx:columns>
                    <mx:DataGridColumn dataField="Index"/>
                    <mx:DataGridColumn dataField="Qty"/>
                </mx:columns>
            </mx:DataGrid>
        <mx:Text width="100%"
            text="Specify the number of lines and click Fill Grid first.
            Then you can click Print."/>
        <mx:TextInput id="dataItems" text="35"/>
        <mx:HBox>
            <mx:Button id="setDP"
                label="Fill Grid"
                click="setdgProvider(int(dataItems.text));"/>
            <mx:Button id="printDG"
                label="Print"
                click="doPrint();"/>
        </mx:HBox>
    </mx:Form>
</mx:Application>
```

出力コンポーネントのプリント

次のコードでは、"FormPrintView.mxml" カスタムコンポーネントのファイルを表示します。

```
<?xml version="1.0"?>
<!-- printing\myComponents\FormPrintView.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:MyComp="myComponents.*"
    backgroundColor="#FFFFFF"
    paddingTop="50" paddingBottom="50" paddingLeft="50">

    <mx:Script>
        <![CDATA[
            import mx.core.*

            // Declare and initialize the variables used in the component.
            // The application sets the actual prodTotal value.
            [Bindable]
            public var pageNumber:Number = 1;
            [Bindable]
            public var prodTotal:Number = 0;

            // Control the page contents by selectively hiding the header and
            // footer based on the page type.
            public function showPage(pageType:String):void {
                if(pageType == "first" || pageType == "middle") {
                    // Hide the footer.
                    footer.includeInLayout=false;
                    footer.visible = false;
                }
                if(pageType == "middle" || pageType == "last") {
                    // The header won't be used again; hide it.
                    header.includeInLayout=false;
                    header.visible = false;
                }
                if(pageType == "last") {
                    // Show the footer.
                    footer.includeInLayout=true;
                    footer.visible = true;
                }
                //Update the DataGrid layout to reflect the results.
                validateNow();
            }
        ]]>
    </mx:Script>

    <!-- The template for the printed page,
        with the contents for all pages. -->
    <mx:VBox width="80%" horizontalAlign="left">
        <mx:Label text="Page {pageNumber}"/>
    </mx:VBox>
    <MyComp:FormPrintHeader id="header"/>
```

```
    <!-- The sizeToPage property is true by default, so the last
        page has only as many grid rows as are needed for the data. -->
    <mx:PrintDataGrid id="myDataGrid" width="60%" height="100%">
    <!-- Specify the columns to ensure that their order is correct. -->
        <mx:columns>
            <mx:DataGridColumn dataField="Index" />
            <mx:DataGridColumn dataField="Qty" />
        </mx:columns>
    </mx:PrintDataGrid>

    <!-- Create a FormPrintFooter control
        and set its prodTotal variable. -->
    <MyComp:FormPrintFooter id="footer" pTotal="{prodTotal}"/>
</mx:VBox>
```

ヘッダーとフッターのファイル

次のコードは、"FormPrintHeader.mxml" ファイルを表示します。

```
<?xml version="1.0"?>
<!-- printing\myComponents\FormPrintHeader.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    width="60%"
    horizontalAlign="right" >

    <mx:Label text="This is a placeholder for first page contents"/>
</mx:VBox>
```

次のコードは、"FormPrintFooter.mxml" ファイルを表示します。

```
<?xml version="1.0"?>
<!-- printing\myComponents\FormPrintFooter.mxml -->
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    width="60%"
    horizontalAlign="right">

    <!-- Declare and initialize the product total variable. -->
    <mx:Script>
        <![CDATA[
            [Bindable]
            public var pTotal:Number = 0;
        ]]>
        </mx:Script>

    <mx:Label text="Product Total: {pTotal}"/>
</mx:VBox>
```

第 34 章

ラッパーとの通信

34

このトピックでは、Adobe Flex アプリケーションとこのアプリケーションを組み込んだ HTML ページ間でデータを交換する方法について説明します。

目次

Flex アプリケーションとのデータの交換について

一般に Flex アプリケーションは、より大きな Web アプリケーションの内部に存在します。この Web アプリケーションでは、セキュリティや状態管理から Web サイト全体の外観と操作性までのすべてを制御します。このシナリオでは、Flex アプリケーションが周囲の環境と通信し、より大きな Web アプリケーションと緊密に統合されることが重要です。Flex アプリケーションが周囲の環境と通信できるようにすることにより、AJAX などの他の技術との統合も可能になります。

多くの場合、Flex アプリケーションは、ブラウザのラッパー内でロードされます。このラッパーは多くの場合、Flex アプリケーションとのやり取りが可能な JavaScript またはその他のクライアント側ロジックを含むことができる、HTML ページです。ラッパーの詳細については、Flex 2 アプリケーションの構築および展開ガイドの第 16 章の「ラッパーの作成」を参照してください。

Flex アプリケーションが周囲の環境と通信する方法は複数あります。必要な統合のタイプに応じて、`flashVars` プロパティ、クエリ文字列パラメータ、`navigateToURL()` メソッド、および ExternalInterface クラスの任意の組み合わせを使用できます。

要求データを Flex アプリケーションに渡すには、クエリ文字列パラメータを使用し、Application.application.parameters または LoaderConfig.parameters オブジェクトを使用して値にアクセスします。データをラッパー内の `flashVars` プロパティとして渡し、Application および LoaderConfig オブジェクトの `parameters` プロパティを使用して値にアクセスすることもできます。詳細については、1082 ページの「flashVars の使用」を参照してください。これらの方法を使用すると、再コンパイルをトリガせずに Flex アプリケーションをパーソナライズできます。

Flex アプリケーションのメソッドを呼び出すには、ExternalInterface API のメソッドを使用します。また、その逆も可能です。`addCallback()` メソッドは、Flex アプリケーションのメソッドをラッパーに対して公開します。`call()` メソッドは、ラッパー内のメソッドを呼び出し、結果を返します。ラッパーが HTML の場合は、`addCallback()` および `call()` メソッドにより、Flex アプリケーションとホストされブラウザ内で実行されている JavaScript の間で、相互の呼び出しが可能になります。詳細については、1078 ページの「ExternalInterface API について」を参照してください。

新しいブラウザウィンドウを開いたり、新しい場所に移動したりする場合には、`navigateToURL()` グローバル関数を使用します。このメソッドは、ExternalInterface API の一部ではありませんが、柔軟性があるので、JavaScript を内部に記述したり、結果の HTML ページで JavaScript 関数を呼び出すことができます。`navigateToURL()` メソッドは、flash.net パッケージ内のグローバル関数です。

アクセス環境に関する情報

Flex には、ブラウザおよびアプリケーションの動作環境に関する情報を取得するための、いくつかの簡単なメカニズムがあります。Flex アプリケーション内から SWF ファイルへの URL を取得できます。コンテキストルートを取得することもできます。次の例では、SWF ファイルの URL を取得し、URL からホスト名を作成し、コンテキストルートを表示します。コンテキストルートには、@ContextRoot() トークンを使用してアクセスできます。

```
<?xml version="1.0"?>
<!-- wrapper/GetURLInfo.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="getHostName()">
  <mx:Script><![CDATA[

  [Bindable]
  public var g_HostString:String;
  [Bindable]
  public var g_ContextRoot:String;
  [Bindable]
  public var g_BaseURL:String;

  private function getHostName():void {
     g_BaseURL = Application.application.url;
     var pattern1:RegExp = new RegExp("http://[^/]*/");
     if (pattern1.test(g_BaseURL) == true) {
        g_HostString = pattern1.exec(g_BaseURL).toString();
     } else{
        g_HostString = "http://localhost/"
     }
  }
  ]]></mx:Script>

  <mx:Form>
     <mx:FormItem label="Base URL:">
        <mx:Label text="{g_BaseURL}"/>
     </mx:FormItem>
     <mx:FormItem label="Host Name:">
        <mx:Label text="{g_HostString}"/>
     </mx:FormItem>
     <mx:FormItem label="Context Root:">
        <mx:Label text="@ContextRoot()"/>
     </mx:FormItem>
  </mx:Form>
</mx:Application>
```

flash.system.Capabilities クラスを使用して、オペレーティングシステム、Player バージョン、および言語などのクライアント情報にアクセスすることもできます。詳細については、Flash ActionScript の使用を参照してください。

ブラウザおよびアプリケーションの環境に関する詳細情報にアクセスするには、ExternalInterface を使用します。詳細については、1078 ページの「ExternalInterface API について」を参照してください。

Netscape 接続の有効化

Netscape ブラウザでは、ブラウザと Flex アプリケーションの間で通信できるようにするために、ラッパーの <embed> タグに swliveconnect=true を記述する必要があります。これにより、Flex アプリケーションはページのスクリプト言語 (通常は JavaScript) に接続できます。このパラメータは、Flex アプリケーションのラッパーの <embed> タグに追加します。次を参照してください。

```
<embed pluginspage='http://www.macromedia.com/go/getflashplayer'
  width='300'
  height='100'
  flashvars=''
  src='TitleTest.mxml.swf'
  name='MyApp'
  SWLIVECONNECT='true'
/>
```

<object> タグは Netscape ブラウザではなく、Microsoft Internet Explorer で使用するため、<object> タグ内で swliveconnect の値を true に設定する必要はありません。

ExternalInterface API について

ExternalInterface API を使用すると、Flex アプリケーションはラッパー内のメソッドを呼び出せるようになり、ラッパーは Flex アプリケーション内の関数を呼び出せるようになります。ExternalInterface API は主に、flash.external パッケージの `call()` メソッドと `addCallback()` メソッドで構成されます。

ExternalInterface API は、次のブラウザでサポートされています。

- Windows 用 Internet Explorer のすべてのバージョン (5.0 以降)。
- Adobe Flash Player ActiveX コントロールが組み込まれたデスクトップアプリケーションなどの、埋め込みカスタム ActiveX コンテナ。
- NPRuntime インターフェイスをサポートするすべてのブラウザ。現時点では、以下のブラウザが含まれます。
  - Firefox 1.0 以降
  - Mozilla 1.7.5 以降
  - Netscape 8.0 以降
  - Safari 1.3 以降

ExternalInterface API を使用したコードを実行する前に、API がブラウザでサポートされていることを確認してください。確認するには、Flex アプリケーションに含まれる ExternalInterface オブジェクトの available プロパティを使用します。available プロパティはブール値であり、ブラウザが ExternalInterface API をサポートする場合は true、サポートしていない場合は false になります。このプロパティは読み取り専用です。

次の例では、available を使用して、ExternalInterface API がサポートされているかを検出してから、クラスを使用するメソッドを実行します。

```
<?xml version="1.0"?>
<!-- wrapper/CheckExternalInterface.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="checkEI()">
  <mx:Script><![CDATA[
     [Bindable]
     public var eiStatus:String;

     private function checkEI():void {
           eiStatus = ExternalInterface.available.toString();
     }

  ]]></mx:Script>

  <mx:Label id="l1" text="{eiStatus}"/>

</mx:Application>
```

Flex アプリケーションの ExternalInterface API の使用例については、1085 ページの「ExternalInterface API を使用した Flex から JavaScript へのアクセス」と 1096 ページの「JavaScript からの Flex へのアクセス」を参照してください。

ExternalInterface API を使用するには、ブラウザが一定のバージョン要件を満たす必要があるだけでなく、ブラウザで JavaScript が有効になっている必要もあります。JavaScript が無効になっているブラウザは、HTML ページで <noscript> タグを使用して操作できます。詳細については、1100 ページの「JavaScript が無効なブラウザの処理」を参照してください。

available プロパティは、ブラウザのバージョンと製造元に基づいて、ブラウザが ExternalInterface API をサポートするかどうかのみを判別します。ブラウザで JavaScript が無効になっている場合も、available プロパティは true を返します。

ExternalInterface API は、SWF ファイルが実行されているセキュリティサンドボックスによって制限されます。この API を使用できるかどうかは、allowScriptAccess および allowNetworking パラメータで定義されるドメインベースのセキュリティ制限に依存します。allowScriptAccess および allowNetworking パラメータの値は、SWF ファイルのラッパーで設定します。

これらのパラメータの詳細については、Flex 2 アプリケーションの構築および展開ガイドの第 16 章の「ラッパーの作成」を参照してください。

セキュリティ制限の詳細については、1101 ページの「Flex における ExternalInterface API セキュリティについて」を参照してください。また、Flex 2 アプリケーションの構築および展開ガイドの第 4 章の「Flex セキュリティの適用」も参照してください。

Flex アプリケーションへの要求データの受け渡し

要求データを Flex アプリケーションに渡すには、ラッパーで flashVars プロパティを定義します。これについては、1082 ページの「flashVars の使用」を参照してください。ユーザーが MXML ファイルを Adobe Flex データサービスサーバーから直接要求すると、Flex はクエリ文字列パラメータを flashVars 変数に変換します。これについては、1084 ページの「クエリ文字列パラメータの使用」を参照してください。

Flex データサービスサーバーを使用している場合、クエリ文字列パラメータや flashVars プロパティを変更しても Flex アプリケーションは再コンパイルされません。

flashVars プロパティやクエリ文字列パラメータを MXML ファイル内で使用するには、Application.application.parameters オブジェクトにアクセスします。詳細については、1080 ページの「Application.application.parameters オブジェクトの使用」を参照してください。

Application.application.parameters オブジェクトの使用

Application オブジェクトの parameters プロパティは、パラメータを名前と値のペアとして保存する動的オブジェクトを指しています。parameters オブジェクトの変数にアクセスするには、parameters.< 変数名 > を指定します。

次の例では、myName および myHometown パラメータを定義し、Label コントロールのテキストにこれらの変数をバインドします。

```
<?xml version="1.0"?>
<!-- wrapper/ApplicationParameters.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initVars()">
  <mx:Script><![CDATA[
     // Declare bindable properties in Application scope.
     [Bindable]
     public var myName:String;
     [Bindable]
     public var myHometown:String;

     // Assign values to new properties.
     private function initVars():void {
        myName = Application.application.parameters.myName;
        myHometown = Application.application.parameters.myHometown;
     }
  ]]></mx:Script>

  <mx:VBox>
  <mx:HBox>
     <mx:Label text="Name: "/>
```

```
    <mx:Label text="{myName}" fontWeight="bold"/>
  </mx:HBox>
  <mx:HBox>
    <mx:Label text="Hometown: "/>
    <mx:Label text="{myHometown}" fontWeight="bold"/>
  </mx:HBox>
  </mx:VBox>
</mx:Application>
```

ユーザーが myName および myHometown パラメータを flashVars 変数として定義してこのアプリケーションを要求すると、それらの値が Label コントロールに表示されます。

flashVars プロパティをすべて表示するには、次の例に示すように、parameters オブジェクトを繰り返します。

```
<?xml version="1.0"?>
<!-- wrapper/IterateOverApplicationParameters.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
    private function init():void {
      for (var i:String in Application.application.parameters) {
        ta1.text += i + ":" + Application.application.parameters[i] + "\n";
      }
    }
  ]]></mx:Script>

  <mx:TextArea id="ta1" width="300" height="200"/>

</mx:Application>
```

要求変数を変更しても、Flex はアプリケーションを再コンパイルしません。したがって、flashVars プロパティまたはクエリ文字列パラメータの値を動的に設定した場合に、再コンパイルは強制されません。

クエリ文字列パラメータは、URL エンコードされている必要があります。ストリング形式は、アンパサンド (&) で区切られた名前と変数のペアです。特殊文字およびプリント不可の文字をエスケープするには、パーセント記号 (%) に続けて 2 桁の 16 進数値を入力します。プラス記号 (+) を使用して 1 つのスペースを表現できます。

flashVars プロパティおよびクエリ文字列パラメータのエンコードは、ページのエンコードと同じです。Windows プラットフォーム上の Internet Explorer は、UTF-16 準拠のストリングを提供します。Netscape は、UTF-8 エンコードされたストリングを Adobe Flash Player に送ります。

ほとんどのブラウザは、最大 64 KB (65535 バイト) 長の flashVars ストリングまたはクエリ文字列をサポートします。名前と値のペアをいくつでも含めることができます。

flashVars の使用

ラッパーの <object> および <embed> タグの flashVars プロパティを使用して、Flex アプリケーションに変数を渡すことができます。

次の例では、単純なラッパー内の <object> タグ内部で firstname、middlename、および lastname flashVars プロパティの値を設定します。

```
<html>
<head>
<title>/flex2/code/wrapper/SimplestFlashVarTestWrapper.html</title>
<style>
body { margin: 0px;
 overflow:hidden }
</style>
</head>
<body scroll='no'>
<table width='100%' height='100%' cellspacing='0' cellpadding='0'><tr><td
valign='top'>

<h1>Simplest FlashVarTest Wrapper</h1>

<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='100%' width='100%'>
<param name='src' value='FlashVarTest.swf'/>
<param name='flashVars' value='firstName=Nick&lastName=Danger'/>
<embed name='mySwf' src='FlashVarTest.swf' pluginspage='http://
www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash'
height='100%' width='100%' flashVars='firstName=Nick&lastName=Danger'/>
</object>

</td></tr></table>
</body>
</html>
```

Flex データサービスによって生成されたラッパー、または "/resources/html-templates" ディレクトリに含まれるラッパーを使用している場合、実際のラッパーが同じように見えないことがありますが、flashVars 変数に渡す基本的な方法は同じです。たとえば、次の例に示すように、flashVars 変数を関数パラメータに追加することにより、挿入することができます。

```
"flashvars","historyUrl=%2Fflex%2Fflex%2Dinternal%3Faction%3Dhistory%5Fhtml&lcon
  id=" + lc_id + "&firstName=Nick&lastName=Danger",
```

flashVars 変数の値は必ずしも静的である必要はありません。たとえば JSP を使用してラッパーを返す場合は、flashVars 変数の値に、ストリングとして評価可能な任意の JSP 式を使用できます。

次の例では、HttpServletRequest オブジェクトに保存された値を使用します (この場合は、フォームまたはクエリ文字列パラメータを使用できます)。

```
<html>
<head>
<title>/flex2/code/wrapper/DynamicFlashVarTestWrapper.jsp</title>
<style>
body { margin: 0px;
 overflow:hidden }
</style>
</head>

<%
String fName = (String) request.getParameter("firstname");
String mName = (String) request.getParameter("middlename");
String lName = (String) request.getParameter("lastname");
%>

<body scroll='no'>
<table width='100%' height='100%' cellspacing='0' cellpadding='0'><tr><td
valign='top'>

<script>
<h1>Dynamic FlashVarTest Wrapper</h1>
</script>


<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='100%' width='100%'>
<param name='src' value='../assets/FlashVarTest.swf'/>
<param name='flashVars' value='firstname=<%= fName %>&lastname=<%= lName %>'/>
<embed name='mySwf' src='../assets/FlashVarTest.swf' pluginspage='http://
www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash'
height='100%' width='100%' flashVars='firstname=<%= fName %>&lastname=<%= lName
%>'/>
</object>

</td></tr></table>
</body>
</html>
```

クエリ文字列パラメータの使用

Flex データサービスを使用するときは、任意の Flex アプリケーションで、クライアントの要求ストリングにクエリ文字列パラメータを追加してパラメータの値を使用できます。つまり、Flex は URL に名前と値のペアとして渡された GET 要求変数をサポートします。

ラッパーを作成するとき、クエリ文字列パラメータは Flex データサービスによって flashVars 変数に変換されます。このため、Flex データサービスを使用している場合、またはラッパの生成を処理するサーバーサイドロジックを実装している場合は、Flex アプリケーション内のクエリ文字列パラメータへのアクセスが可能です。たとえば、Web 層コンパイラを使用している場合に、ブラウザのアドレスバーで次のクエリ文字列を要求するとします。

```
http://localhost:8101/flex/charts/PieChart1.mxml?fname=Nick&lname=Danger
```

Flex は、それらの値を flashVars 変数としてラッパーに追加します。

メモ	スタンドアローンの Flash Player 内部で動作している Flex アプリケーションにクエリ文字列パラメータを渡すことはできません。

クエリ文字列パラメータをラッパー内の <object> タグと <embed> タグの src プロパティに追加することもできます。src プロパティは、Flex アプリケーションの SWF ファイルの場所を識別します。これらのパラメータにアクセスする方法は、ブラウザのアドレスバーでクエリ文字列パラメータにアクセスする場合と同じです。

次の例では、カスタムラッパー内で src プロパティにクエリ文字列パラメータを追加します。

```
<object ... >
  <param name='src' value='TitleTest.mxml.swf?myName=Danger'>
  ...
  <embed src='TitleTest.mxml.swf?myName=Danger' ... />
</object>
```

flashVars プロパティと同様、parameters オブジェクトで Flex アプリケーションのクエリ文字列パラメータの値にアクセスします。

```
var myName:String = Application.application.parameters.myName;
```

ユーザーがラッパーを使用せずに直接 SWF ファイルを要求した場合は、クエリ文字列の変数にアクセスするために追加のコードを提供する必要はありません。次の URL は、Nick という名前と居住地である San Francisco を Flex アプリケーションに渡します。

```
http://localhost:8100/flex/myApp.swf?myName=Nick&myHometown=San%20Francisco
```

Flex からの JavaScript 関数へのアクセス

HTML ページに含まれる JavaScript 関数は、Flex アプリケーションから呼び出すことができます。Flex とブラウザとの通信を可能にすることにより、スタイル設定の変更、リモートプロセスの起動や、ページのスクリプト内部から通常実行できる他の処理を実行できます。

Flex アプリケーションから、JavaScript 関数を含む HTML ページにデータを渡し、データを処理してから Flex アプリケーションにデータを戻すこともできます。これを行うには、ExternalInterface API または `navigateToURL()` メソッドを使用します。詳細については、1085 ページの「ExternalInterface API を使用した Flex から JavaScript へのアクセス」および 1090 ページの「Flex での navigateToURL() メソッドの使用」を参照してください。

関数を含むページと通信する際には、実行する動作すべてをブラウザが処理できるかどうかを判別する必要があります。したがって、使用するオブジェクトがブラウザでサポートされているかどうかを最初に判別する必要があります。ブラウザによるサポートを判別するための一般的なガイドラインについては、1085 ページの「ExternalInterface API を使用した Flex から JavaScript へのアクセス」を参照してください。

ExternalInterface API を使用した Flex から JavaScript へのアクセス

Flex アプリケーションから JavaScript 関数を呼び出す最も簡単な方法は、ExternalInterface API を使用することです。この API を使用すると、ラッパーで任意の JavaScript メソッドを呼び出し、パラメータを渡し、戻り値を取得することができます。メソッドの呼び出しが失敗した場合は、例外が返されます。

ExternalInterface API では、ブラウザサポートの確認がカプセル化されており、メソッドを使用するときにサポートを確認する必要はありません。ただし、`available` プロパティを使用すれば、ブラウザがインターフェイスをサポートしているかどうかを確認することができます。詳細については、1078 ページの「ExternalInterface API について」を参照してください。

ExternalInterface API は、単一のクラス flash.external.ExternalInterface で構成されています。このクラスには静的な `call()` メソッドがあり、このメソッドを使用して JavaScript と Flash との通信が容易になります。ExternalInterface API の `addCallback()` メソッドを使用して、ラッパーから Flex アプリケーション内のメソッドを呼び出すこともできます。詳細については、1102 ページの「addCallback() メソッドについて」を参照してください。

ExternalInterface を使用するには、HTML ページで `id` プロパティと `name` プロパティを呼び出し可能にする必要があります。詳細については、1099 ページの「Flex アプリケーションの id プロパティと name プロパティの編集」を参照してください。

Flex アプリケーションからの JavaScript メソッドの呼び出し

ExternalInterface API を使用すると、ラッパー内でそのメソッドを呼び出すことが非常に簡単になります。使用できる静的な `call()` メソッドのシグネチャを次に示します。

```
flash.external.ExternalInterface.call(function_name:
  String[, arg1, ...]):Object;
```

function_name は、HTML ページの JavaScript で使用されている関数名です。引数は、JavaScript 関数に渡す引数です。カンマで区切る従来の方法を使用して 1 つ以上の引数を渡すことも、ブラウザで非直列化されるオブジェクトを渡すこともできます。この引数はオプションです。

次のスクリプトブロックの例では、`call()` メソッドを使用して、JavaScript の `changeDocumentTitle()` 関数を、それが含まれるラッパー内で呼び出します。

```
<?xml version="1.0"?>
<!-- wrapper/WrapperCaller.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
     import flash.external.*;

     public function callWrapper():void {
        var s:String;
        if (ExternalInterface.available) {
           var wrapperFunction:String = "changeDocumentTitle";
           s = ExternalInterface.call(wrapperFunction,ti1.text);
        } else {
           s = "Wrapper not available";
        }
        trace(s);
     }
  </mx:Script>

  <mx:Form>
     <mx:FormItem label="New Title:">
        <mx:TextInput id="ti1"/>
     </mx:FormItem>
  </mx:Form>

  <mx:Button label="Change Document Title" click="callWrapper()"/>
</mx:Application>
```

HTML ページで、他の JavaScript 関数と同じように関数を定義します。次に示す例のように、値を返すことができます。

```
<html><head>
<title>wrapper/WrapperBeingCalled.html</title>
</head>
<body scroll='no'>

<SCRIPT LANGUAGE="JavaScript">
function changeDocumentTitle(a) {
```

```
window.document.title=a;
alert(a);
return "successful";
}
</SCRIPT>

<h1>Wrapper Being Called</h1>
<table width='100%' height='100%' cellspacing='0' cellpadding='0'>
<tr>
<td valign='top'>
<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='200' width='400'>
<param name='src' value='WrapperCaller.swf'/>
<param name='flashVars' value=''/>
<embed name='mySwf' src='WrapperCaller.swf' pluginspage='http://
www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash'
height='100%' width='100%' flashVars=''/>
</object>
</td>
</tr>
</table>
</body></html>
```

この機能を使用するには、埋め込みムービーファイルに id 属性が設定されている必要があります。設定されていない場合、Flex アプリケーションからの呼び出しは成功しません。

call() メソッドは 0 個以上の引数を受け入れます。ActionScript 型も、この引数として可能です。ActionScript 型は、Flex によって JavaScript の数値およびストリングとして直列化されます。オブジェクトを渡す場合、JavaScript 内の非直列化オブジェクトのプロパティにアクセスできます。次に例を示します。

```
<?xml version="1.0"?>
<!-- wrapper/DataTypeSender.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import flash.external.*;

     public function callWrapper():void {
        var s:String;
        if (ExternalInterface.available) {
           var o:Object = new Object();
           o.fname = "Nick";
           o.lname = "Danger";
           var wrapperFunction:String = "receiveComplexDataTypes";
           s = ExternalInterface.call(wrapperFunction, o);
        } else {
           s = "Wrapper not available";
        }
        trace(s);
```

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

  <mx:Button label="Send" click="callWrapper()"/>

</mx:Application>
```

Flex で直列化されるのは、静的ではないパブリック変数と、ActionScript オブジェクトの読み取りおよび書き込み可能なプロパティだけです。数値およびストリングは、オブジェクトのプロパティとして渡すことも、プリミティブ型や配列のような単純オブジェクトとして渡すことも、単純オブジェクトの配列として渡すこともできます。

JavaScript コードは、次の例に示すように、オブジェクトのプロパティにアクセスできます。

```
<html><head>
<title>wrapper/DataTypeWrapper.html</title>
</head>
<body scroll='no'>

<SCRIPT LANGUAGE="JavaScript">
function receiveComplexDataTypes(o) {
alert("Welcome " + o.fname + " " + o.lname + "!");
return "successful";
}
</SCRIPT>

<h1>Data Type Wrapper</h1>

<table width='100%' height='100%' cellspacing='0' cellpadding='0'>
<tr>
<td valign='top'>
<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='200' width='400'>
<param name='src' value='DataTypeSender.swf'/>
<param name='flashVars' value=''/>
<embed name='mySwf' src='DataTypeSender.swf' pluginspage='http://
www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash'
height='100%' width='100%' flashVars=''/>
</object>
</td>
</tr>
</table>

</body></html>
```

次の例のように、オブジェクト内にオブジェクトを埋め込むこともできます。たとえば、オブジェクトを、オブジェクト内の配列として埋め込むことができます。Flex アプリケーションの <mx:Script> ブロックに次のコードを追加します。

```
<?xml version="1.0"?>
<!-- wrapper/ComplexDataTypeSender.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import flash.external.*;

     public function callWrapper():void {
        var s:String;
        if (ExternalInterface.available) {
           var o:Object = new Object();
           o.fname = "Nick";
           o.lname = "Danger";
           o.b = new Array("DdW","E&T","LotR:TS");
           var wrapperFunction:String = "receiveComplexDataTypes";
           s = ExternalInterface.call(wrapperFunction, o);
        } else {
           s = "Wrapper not available";
        }
        trace(s);
     }
  ]]></mx:Script>

  <mx:Button label="Send" click="callWrapper()"/>

</mx:Application>
```

このコードは、次のラッパー内で JavaScript 関数をトリガします。

```
<html><head>
<title>wrapper/ComplexDataTypeWrapper.html</title>
</head>
<body scroll='no'>

<SCRIPT LANGUAGE="JavaScript">
function receiveComplexDataTypes(o) {
// Get value of fname and lname properties.
var s = ("Welcome " + o.fname + " " + o.lname + "!\n");
// Iterate over embedded object's properties.
for (i=0; i<o.b.length; i++) {
s +=  o.b[i] + "\n";
}
alert(s);
}
</SCRIPT>

<h1>Complex Data Type Wrapper</h1>
```

```
<table width='100%' height='100%' cellspacing='0' cellpadding='0'>
<tr>
<td valign='top'>
<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='200' width='400'>
<param name='src' value='ComplexDataTypeSender.swf'/>
<param name='flashVars' value=''/>
<embed name='mySwf' src='ComplexDataTypeSender.swf' pluginspage='http://
www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash'
height='100%' width='100%' flashVars=''/>
</object>
</td>
</tr>
</table>

</body></html>
```

Flex と Flash Player には、強力なセキュリティ機能が内蔵されており、サイト間のスクリプティングを防ぐことができます。デフォルトでは、HTML ページが Flex アプリケーションと同じドメインにない場合、HTML ページ上のスクリプトを呼び出すことができません。ただし、スクリプトが呼び出されるソースを展開できます。詳細については、1101 ページの「Flex における ExternalInterface API セキュリティについて」を参照してください。

循環参照が含まれるオブジェクトや配列を渡すことはできません。たとえば、次のオブジェクトを渡すことはできません。

```
var obj = new Object();
obj.prop = obj; // 循環参照です。
```

循環参照は、ActionScript と JavaScript の両方で無限ループになります。

Flex での navigateToURL() メソッドの使用

`navigateToURL()` メソッドは特定の URL からウィンドウにドキュメントをロードしたり、定義済みの URL にある別のアプリケーションに変数を渡します。このメソッドを使用して、Flex アプリケーションが記述されている HTML ページで JavaScript 関数を呼び出すことができます。

`navigateToURL()` メソッドの機能を、URLLoader クラスの `load()` メソッドと混同しないでください。URLLoader クラスは、指定された URL を ActionScript による操作のためにオブジェクトにロードします。`navigateToURL()` メソッドは、指定された URL にブラウザで移動します。

多くの場合、Flex とラッパーの間で通信を行うには ExternalInterface API を使用する必要がありますが、`navigateToURL()` メソッドは ExternalInterface API の一部ではないので、このメソッドに対するブラウザサポートの厳密な要件はありません。それらの要件については、1078 ページの「ExternalInterface API について」で説明されています。

navigateToURL() メソッドは、SWF ファイルが実行されているセキュリティサンドボックスによって制限されます。この API を使用できるかどうかは、allowScriptAccess および allowNetworking パラメータで定義されるドメインベースのセキュリティ制限に依存します。allowScriptAccess および allowNetworking パラメータの値を SWF ファイルのラッパーで設定します。

これらのパラメータの詳細については、Flex 2 アプリケーションの構築および展開ガイドの第 16 章の「ラッパーの作成」を参照してください。セキュリティ制限の詳細については、Flex 2 アプリケーションの構築および展開ガイドの第 4 章の「Flex セキュリティの適用」を参照してください。

navigateToURL() メソッドシンタックス

navigateToURL() メソッドは、flash.net パッケージにあります。シグネチャは次のとおりです。

navigateToURL(*request*:URLRequest, *window*:String):void

request パラメータは、宛先を指定する URLRequest オブジェクトです。*window* 引数には、新規ブラウザウィンドウを起動するか、新規 URL を現在のウィンドウにロードするかを指定します。次の表で、*window* 引数の有効な値について説明します。

値	説明
_self	現在のウィンドウ内の現在のフレームを指定します。
_blank	新規ウィンドウを指定します。この新規ウィンドウは、クライアントのブラウザ内でポップアップウィンドウとして機能します。したがって、ポップアップブロッカーによりロードできない場合があることに注意してください。
_parent	現在のフレームの親を指定します。
_top	現在のウィンドウ内の最上位のフレームを指定します。

URLRequest オブジェクトを navigateToURL() メソッドに渡します。このオブジェクトは、要求のメソッド URL ターゲット、メソッド (POST または GET)、ウィンドウ、およびヘッダを定義します。次の例では、移動先の単純な URL を定義しています。

```
<?xml version="1.0"?>
<!-- wrapper/SimplestNavigateToURL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import flash.net.*;
     public function openNewWindow(event:MouseEvent):void {
        var u:URLRequest = new URLRequest("http://www.adobe.com/flex");
        navigateToURL(u,"_blank");
     }
  ]]></mx:Script>
  <mx:Button label="Open New Window" click="openNewWindow(event)"/>
</mx:Application>
```

navigateToURL() メソッドは、url パラメータの値を URL エンコードします。

URLRequest オブジェクトと共にデータを送信するために、要求ストリングに変数を追加できます。次の例では、新規ウィンドウを起動し、検索語句を URL に渡します。

```
<?xml version="1.0"?>
<!-- wrapper/SimpleNavigateToURL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import flash.net.*;
     public function executeSearch(event:MouseEvent):void {
       var u:URLRequest = new URLRequest("http://www.google.com/search?hl=en&q="
+ ta1.text);
        navigateToURL(u,"_blank");
     }
  ]]></mx:Script>
  <mx:TextArea id="ta1"/>
  <mx:Button label="Search" click="executeSearch(event)"/>
</mx:Application>
```

ストリングを追加する以外に、URLRequest の data プロパティを使用して GET または POST 要求に URL 変数を追加することができます。このクエリ文字列パラメータのタイプは URLVariables です。Flex は、アンパサンド区切り文字を URL 要求ストリングに追加します。次の例では、name=fred を URLRequest に追加します。

```
<?xml version="1.0"?>
<!-- wrapper/NavigateWithGetMethod.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import flash.net.*;
     public function openNewWindow(event:MouseEvent):void {
        var url:URLRequest = new URLRequest("http://mysite.com/index.jsp");
        var uv:URLVariables = new URLVariables();
        url.method = "GET";
        uv.name = "fred";
        url.data = uv;
        navigateToURL(url,"_blank");
     }
  ]]></mx:Script>
  <mx:Button label="Open New Window" click="openNewWindow(event)"/>
</mx:Application>
```

URLRequest オブジェクトと共に POST データを使用するには、URLRequest オブジェクトの method プロパティの値を POST に設定します。

navigateToURL() メソッドを使用して複数ウィンドウを開く

navigateToURL() メソッドを呼び出すことにより、任意の数の新規ブラウザウィンドウを開くことができます。ただし、ActionScript は非同期なので、このメソッドの単純なループを試みたときに Flash Player でブラウザウィンドウが開かれるのは、最後の呼び出しに対してのみです。これを回避するには、callLater() メソッドを使用します。このメソッドは、アプリケーションの新しいフレームごとに新規ブラウザウィンドウを開くよう Flash Player に指示します。次の例では、callLater() メソッドを使用して複数のブラウザウィンドウを開き、その際に navigateToURL() メソッドを指定します。

```
<?xml version="1.0"?>
<!-- wrapper/NavigateToMultipleURLS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="openWindows(0)">
  <mx:Script><![CDATA[
     import flash.net.navigateToURL;
     private var listingData:Array=[
        "http://www.google.com",
        "http://www.adobe.com",
        "http://www.usatoday.com"
     ];

     private function openWindows(n: Number):void {
        if (n < listingData.length) {
           navigateToURL(new URLRequest(listingData[n]), '_blank');
           callLater(callLater, [openWindows, [n+1]]);
        }
     }
  ]]></mx:Script>
</mx:Application>
```

URLRequest オブジェクトを使用した JavaScript 関数の呼び出し

JavaScript 関数をメソッドの 1 番目のパラメータに埋め込むことにより、URLRequest を使用してこの関数を呼び出すことができます。次に例を示します。

```
public var u:URLRequest = new URLRequest("javascript:window.close()");
```

上記のコードがすべてのブラウザで動作するとは限りません。ブラウザのタイプを検出し、そのタイプに応じてウィンドウを閉じるコードを追加する必要があります。また、Internet Explorer などの一部のブラウザでは、JavaScript を含む URLRequest を呼び出す際に予測しない動作が起こることがあります。次に例を示します。

- JavaScript URL を非同期で実行します。つまり、JavaScript メソッドの実行を試みる navigateToURL() メソッドを複数呼び出すことが可能ですが、実行されるのは最後の呼び出しのみです。各呼び出しは次の呼び出しで上書きされます。
- JavaScript URL を呼び出すときに、イメージや IFRAME のロードなど、他のすべてのナビゲーションを停止します。
- URL にアンパサンド (&) や疑問符 (?) 記号が含まれている場合に、セキュリティ警告ダイアログボックスが表示されます。

navigateToURL() メソッドを使用した JavaScript の呼び出し

navigateToURL() メソッドを使用して、Flex アプリケーションが実行されている HTML ページ上の JavaScript 関数を呼び出すことができます。ただし、navigateToURL() メソッドを使用してパラメータを渡す場合、各パラメータを引用符で囲む必要があります。次に例を示します。

```
<?xml version="1.0"?>
<!-- wrapper/CallingJavaScriptWithNavigateToURL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
     import flash.net.*;
     public function callWrapper(e:Event):void {
        var eType:String = String(e.type);
        var eName:String = String(e.currentTarget.id);
        var u:URLRequest = new URLRequest("javascript:catchClick('" + eName +
"','" + eType + "')");
        navigateToURL(u,"_self");
     }
  </mx:Script>
  <mx:Button id="b1" click="callWrapper(event)" label="Call Wrapper"/>
</mx:Application>
```

次に示す例のように、周りを囲む HTML ラッパーにはこの呼び出しを処理するための JavaScript が含まれています。

```
<html><head>
<title>wrapper/NavigateToURLWrapper.html</title>
</head>
<body scroll='no'>

<SCRIPT LANGUAGE="JavaScript">
function catchClick(name, type) {
alert(name + " triggered the " + type + " event.");
}
</SCRIPT>

<h1>Navigate To URL Wrapper</h1>

<table width='100%' height='100%' cellspacing='0' cellpadding='0'>
<tr>
```

```
<td valign='top'>
<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='200' width='400'>
<param name='src' value='CallingJavaScriptWithNavigateToURL.swf'/>
<param name='flashVars' value=''/>
<embed name='mySwf' src='CallingJavaScriptWithNavigateToURL.swf'
pluginspage='http://www.macromedia.com/shockwave/download/
index.cgi?P1_Prod_Version=ShockwaveFlash' height='100%' width='100%'
flashVars=''/>
</object>
</td>
</tr>
</table>

</body></html>
```

navigateToURL() メソッドは、URLRequest 自体の基本 JavaScript 関数に対して使用することもできます。たとえば、次の1行のコードでデフォルトの電子メールクライアントを起動できます。

```
<?xml version="1.0"?>
<!-- wrapper/EmailWithNavigateToURL.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
     import flash.net.*;
     public function sendMail(e:Event):void {
        var u:URLRequest = new URLRequest("mailto:" + ti1.text);
        navigateToURL(u,"_self");
     }
  </mx:Script>
  <mx:Button id="b1" click="sendMail(event)" label="Send Mail"/>
  <mx:Form>
     <mx:FormItem>
        <mx:Label text="Email Address: "/>
     </mx:FormItem>
     <mx:FormItem>
        <mx:TextInput id="ti1"/>
     </mx:FormItem>
  </mx:Form>
</mx:Application>
```

navigateToURL() メソッドに対する javascript プロトコルの呼び出しが、すべてのブラウザでサポートされているとは限りません。可能な場合は、ExternalInterface API の call() メソッドを使用して、HTML ページ内の JavaScript メソッドを呼び出してください。詳細については、1085 ページの「ExternalInterface API を使用した Flex から JavaScript へのアクセス」を参照してください。

JavaScript からの Flex へのアクセス

ExternalInterface API を使用すると、ラッパーから Flex メソッドを呼び出すことができます。そのためには、Flex アプリケーションのパブリックメソッドを、呼び出し可能なメソッドのリストに追加します。Flex アプリケーションでは、ExternalInterface API の `addCallback()` メソッドを使用してローカル Flex 関数をリストに追加します。このメソッドにより、ActionScript メソッドは、ラッパーの JavaScript または VBScript から呼び出し可能なメソッドとして登録されます。

メモ

この機能を使用するには、クライアントで特定のブラウザが実行されている必要があります。詳細については、1078 ページの「ExternalInterface API について」を参照してください。

`addCallback()` メソッドのシグネチャを次に示します。

```
addCallback(function_name:String, closure:Function):void
```

function_name パラメータは、HTML ページのスクリプトから Flex 関数を呼び出すときに使用する名前です。*closure* パラメータは、呼び出す関数のローカル名です。このパラメータには、アプリケーションのメソッド、またはオブジェクトインスタンスを指定できます。

次の例では、ラッパーで呼び出し可能にする `myFunc()` 関数を宣言します。

```
<?xml version="1.0"?>
<!-- wrapper/AddCallbackExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script>
     import flash.external.*;

     public function initApp():void {
        ExternalInterface.addCallback("myFlexFunction",myFunc);
     }

     public function myFunc(s:String):void {
        l1.text = s;
     }

  </mx:Script>

  <mx:Label id="l1"/>

</mx:Application>
```

Flex 関数を HTML ページから呼び出すには、ムービーオブジェクトへの参照を取得します。この値は、`<object>` および `<embed>` タグの `id` および `name` プロパティと同じ値です。この場合は mySwf です。次に、必要なパラメータをすべて渡して、そのオブジェクトでメソッドを呼び出します。次に例を示します。

```
<html><head>
<title>wrapper/AddCallbackWrapper.html</title>
</head>
```

```
<body scroll='no'>

<SCRIPT LANGUAGE="JavaScript">
function callApp() {
window.document.title = document.getElementById("newTitle").value;
mySwf.myFlexFunction(window.document.title);
}
</SCRIPT>

<h1>AddCallback Wrapper</h1>

<form id="f1">
Enter a new title: <input type="text" size="30" id="newTitle"
onchange="callApp()">
</form>

<table width='100%' height='100%' cellspacing='0' cellpadding='0'>
<tr>
<td valign='top'>
<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='200' width='400'>
<param name='src' value='AddCallbackExample.swf'/>
<param name='flashVars' value=''/>
<embed name='mySwf' src='AddCallbackExample.swf' pluginspage='http://
www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash'
height='100%' width='100%' flashVars=''/>
</object>
</td>
</tr>
</table>

</body></html>
```

Flex データサービスによって生成されるラッパー内の id および name プロパティのデフォルト値は、‹MXML ファイル名›.mxml.swf です。HTML ページのスクリプト内で Flex アプリケーションにアクセスするために使用する名前には、ピリオドを含めることができないので、ラッパーではデフォルト値を変更する必要があります。詳細については、1099 ページの「Flex アプリケーションの id プロパティと name プロパティの編集」を参照してください。

ユーザーが Flex アプリケーションを要求する際に使用するブラウザが不明な場合は、ラッパーのスクリプトをブラウザに依存しないようにする必要があります。詳細については、1098 ページの「複数のブラウザタイプの操作」を参照してください。

Flex アプリケーションに適切な名前の付いた関数がない場合、またはその関数が呼び出し可能になっていない場合、ブラウザは JavaScript エラーをスローします。

Flex と Flash Player には、強力なセキュリティ機能が内蔵されており、サイト間のスクリプティングを防ぐことができます。デフォルトでは、Flex 関数を HTML スクリプトから呼び出すことはできません。関数を呼び出し可能として明示的に指定する必要があります。また、HTML ページがアプリケーションとして同じドメインにない場合、Flex 関数を HTML ページから呼び出すことはできません。ただし、Flex 関数の呼び出し元のソースを展開することは可能です。詳細については、1102 ページの「addCallback() メソッドについて」を参照してください。

複数のブラウザタイプの操作

通常、複数のブラウザタイプに対応させるには、HTML ページのスクリプトで記述する必要があります。Flash アプリケーションオブジェクトへの参照を取得する方法は、Internet Explorer と FireFox などの Netscape ベースブラウザでは異なります。

ブラウザに依存しないスクリプトを記述するには、ナビゲータオブジェクトの名前を確認し、その情報に基づいて Macromedia Flash アプリケーションへの参照を返します。次のコードでは、主要なブラウザすべてで Flash アプリケーションへの参照を取得します。

```
<html><head>
<title>wrapper/BrowserAwareAddCallbackWrapper.html</title>
</head>
<body scroll='no'>

<SCRIPT LANGUAGE="JavaScript">
// Internet Explorer and Mozilla-based browsers refer to the Flash application
// object differently.
// This function returns the appropriate reference, depending on the browser.
function getMyApp(appName) {
if (navigator.appName.indexOf ("Microsoft") !=-1) {
return window[appName];
} else {
return document[appName];
}
}

function callApp() {
window.document.title = document.getElementById("newTitle").value;
getMyApp("mySwf").myFlexFunction(window.document.title);
}
</SCRIPT>

<h1>AddCallBack Wrapper</h1>

<form id="f1">
Enter a new title: <input type="text" size="30" id="newTitle"
onchange="callApp()">
</form>
```

```
<table width='100%' height='100%' cellspacing='0' cellpadding='0'>
<tr>
<td valign='top'>
<object id='mySwf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=9,0,0,0' height='200' width='400'>
<param name='src' value='AddCallbackExample.swf'/>
<param name='flashVars' value=''/>
<embed name='mySwf' src='AddCallbackExample.swf' pluginspage='http://
www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash'
height='100%' width='100%' flashVars=''/>
</object>
</td>
</tr>
</table>

</body></html>
```

Flex アプリケーションの id プロパティと name プロパティの編集

デフォルトでは、Flex アプリケーションの Flash アプリケーションオブジェクトの名前は、ラッパー内の <MXML ファイル名 >.mxml.swf です。ただし、JavaScript メソッド名、VBScript メソッド名、またはオブジェクト名にはピリオドを使用できません。したがって、ExternalInterface API を使用するには、ラッパーで Flex アプリケーションに割り当てられた id プロパティを編集する必要があります。

ページ上のオブジェクトの名前を変更するには、<embed> タグの name プロパティと <object> タグの id プロパティの値を変更します。

次の例はラッパー内でアプリケーション名を編集する場所を示しています。

```
<noscript>
  // <object> タグに Flash アプリケーションの id を設定します。
  <object
    classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
    codebase='http://download.macromedia.com/pub/shockwave/cabs/flash/
      swflash.cab#version=9,0,0,0'
    width='300'
    height='100'
    id='MyApp'>
    <param name='flashvars' value=''>
    <param name='src' value='MyApp.mxml.swf'>

    // <embed> タグに Flash アプリケーションの名前を設定します。
    <embed
      pluginspage='http://www.macromedia.com/go/getflashplayer'
      width='300'
      height='100'
      flashvars=''
```

```
        src='MyApp.mxml.swf'
        name='MyApp'
      />
  </object>
</noscript>

<script language='javascript' charset='utf-8'>
  // スクリプトブロックでも id プロパティと name プロパティを変更します。
  ...
</script>
```

この例では、ラッパーの <noscript> ブロックの <object> および <embed> タグを示します。<script> ブロック内の <object> および <embed> タグの name と id プロパティも変更する必要があります。これらのよく似た 2 つのブロックは、共に Flex アプリケーションの全機能を実行するために必要です。

Flex アプリケーションオブジェクトの名前を変更した後は、名前を指定して JavaScript からアプリケーションを参照できます。

JavaScript が無効なブラウザの処理

場合によっては、クライアントのブラウザが JavaScript をサポートしないことや、ユーザーが JavaScript を意図的に無効にしていることがあります。このユーザーが Flex アプリケーションの実行を試みたときに行われることを、ラッパーの <noscript> タグを使用して定義できます。

解決方法の 1 つは、JavaScript が無効なためアプリケーションの機能を使用できないことをユーザーに伝えるメッセージを挿入することです。次の例では、JavaScript を無効にしているユーザーが Flex アプリケーションの実行を試みると、ユーザーに警告が表示されます。

```
<html>
  <body>
    <noscript>
      <EM>Your browser either does not support JavaScript or has disabled
        it. Some features of this application require JavaScript. Click
        <A HREF="...">here</A> to continue anyway, Otherwise, please
        enable JavaScript and reload this page.
      </EM>
    </noscript>
    <script src="mysource.js"></script>
  </body>
</html>
```

Flex における ExternalInterface API セキュリティについて

Flex アプリケーションが HTML ページに組み込まれたスクリプトを呼び出すこと、またはその逆を許可する場合、厳密なセキュリティ制限に従う必要があります。デフォルトでは、HTML ページと Flex アプリケーションが同一ドメインにある場合、HTML ページ上のスクリプトは Flex アプリケーションの ActionScript のみと通信できます。ドメイン外部のアプリケーションを含めるためにこの制限を拡張できます。

call() メソッドについて

`call()` メソッドの呼び出しが成功するかどうかは、HTML ページで `allowScriptAccess` パラメータを使用するかどうかによって決まります。このパラメータは、ActionScript のメカニズムではなく、HTML パラメータです。この値は、Flex アプリケーションが HTML ページの JavaScript を呼び出すことができるかどうかを決定し、ページ上のすべての関数に適用されます。`allowScriptAccess` のデフォルト値では、Flex アプリケーションと HTML ページが同じドメインにある場合のみに通信が許可されます。

HTML ページの `<object>` および `<embed>` タグの `allowScriptAccess` プロパティを設定します。`<object>` タグで、次のようにプロパティを設定します。

```
<object id='SendComplexDataTypes' classid='clsid:D27CDB6E-AE6D-11cf-96B8-
  444553540000' codebase='http://download.macromedia.com/pub/shockwave/cabs/
  flash/swflash.cab#version=9,0,0,0' allowScriptAccess='always' height='100%'
  width='100%'>
```

`<embed>` タグで、次のようにプロパティを設定します。

```
<embed name='SendComplexDataTypes.mxml.swf' pluginspage='http://
  www.macromedia.com/go/getflashplayer' src='SendComplexDataTypes.mxml.swf'
  allowScriptAccess='always' height='100%' width='100%' flashvars=''/>
```

次の表で、`allowScriptAccess` パラメータの有効な値について説明します。

値	説明
`never`	`call()` メソッドは失敗します。
`sameDomain`	呼び出しアプリケーションが HTML ページと同じドメインにある場合、`call()` メソッドは成功します。これがデフォルト値です。
`always`	呼び出しアプリケーションが HTML ページと同じドメインであるかに関わらず、`call()` メソッドは成功します。

addCallback() メソッドについて

Flex では、明示的にメソッドを呼び出し可能として設定することが必要であり、これにより、JavaScript メソッドがアプリケーションの任意のメソッドを呼び出せないようになっています。デフォルトでは、すべてのメソッドは、JavaScript から呼び出し可能ではありません。ExternalInterface API により、JavaScript による呼び出しを可能にする特定のインターフェイスに SWF ファイルを公開できます。

デフォルトでは、同一ドメインから起動された場合、HTML ページは Flex アプリケーションの ActionScript のみと通信できます。Flex アプリケーションのドメインの外部にある HTML ページがアプリケーションを呼び出せるようにするには、`allowDomain()` メソッドを使用します。詳細については、Adobe Flex 2 リファレンスガイドを参照してください。

第 35 章

共有オブジェクトの使用

35

SharedObject クラスは、クライアントコンピュータに少量のデータを保存するために使用します。このトピックでは、Adobe Flex 2 で共有オブジェクトを使用する方法について説明します。

目次

共有オブジェクトについて

共有オブジェクトは、ブラウザの Cookie のように機能します。SharedObject クラスを使用して、ユーザーのローカルマシンにあるハードディスクにデータを格納し、同じセッション中や後のセッション中にそのデータを呼び出すことができます。アプリケーションからアクセスできる SharedObject データは、そのアプリケーション自体が作成したもので、しかも、アプリケーションと同じドメイン内で動作しているものに限られます。データがサーバーに送信されたり、他のドメインで動作する別の Flex アプリケーションによってアクセスされることはありませんが、同じドメインで動作するアプリケーションからアクセス可能にすることができます。

共有オブジェクトと Cookie の比較

Cookie と共有オブジェクトは非常によく似ています。ほとんどの Web プログラマは Cookie の機能を理解しているので、Cookie とローカル SharedObject を比較すると役に立つかもしれません。

RFC 2109 規格に準拠した Cookie には、一般的に次の特性があります。

- 有効期限を設定できます。多くの場合、デフォルトではセッションの終了時に期限切れになります。
- サイト固有の条件により、クライアントから無効にできます。
- Cookie の数は、合計で 300 個、サイトあたり最大 20 個に制限されています。
- 通常は、1 個あたり 4 KB 以内のサイズに制限されています。

- セキュリティ上の脅威と考えられることがあり、場合によってはクライアントで無効にされます。
- クライアントのブラウザで指定した場所に格納されます。
- HTTP を通じてクライアントからサーバーに転送されます。

一方、共有オブジェクトには次の特性があります。

- デフォルトでは、有効期限がありません。
- デフォルトでは、1 個あたり 100 KB 以内のサイズに制限されています。
- 単純なデータ型 (String、Array、Date など) を格納できます。
- アプリケーションで指定した場所 (ユーザーのホームディレクトリ内) に格納されます。
- クライアントとサーバー間で転送されることはありません。

SharedObject クラスについて

SharedObject クラスを使用すると、共有オブジェクトを作成および削除できます。また、使用している SharedObject オブジェクトの現在のサイズを検出できます。SharedObject クラスは、次のメソッドで構成されます。

メソッド	内容
`clear()`	SharedObject オブジェクトのすべてのデータを消去し、その SharedObject ファイルをディスクから削除します。
`flush()`	SharedObject ファイルをクライアント上のファイルに直ちに書き込みます。
`getLocal()`	クライアントのドメイン固有のローカル SharedObject オブジェクトへの参照を返します。存在しない場合、このメソッドはクライアント上に新規の共有オブジェクトを作成します。
`getSize()`	SharedObject ファイルのサイズをバイト単位で返します。デフォルトのサイズ制限は 100KB ですが、クライアントの許可によってはそれより大きい場合もあります。

これらのメソッドに加えて、SharedObject オブジェクトには次のプロパティがあります。

プロパティ	内容
`data`	読み取り専用プロパティ。共有オブジェクトが格納する属性の集合を表しています。
`onStatus`	共有オブジェクトのイベントハンドラ。すべての警告、エラー、情報通知に対して呼び出されます。

共有オブジェクトの作成

SharedObject オブジェクトを作成するには、SharedObject.getLocal() メソッドを使用します。このメソッドのシンタックスは次のとおりです。

```
SharedObject.getLocal("objectName" [, pathname]): SharedObject
```

次の例では、mySO という共有オブジェクトを作成します。

```
public var mySO:SharedObject;
mySO = SharedObject.getLocal("preferences");
```

これにより、クライアントコンピュータ上に preferences.sol という名前のファイルが作成されます。

" ローカル " という用語は、共有オブジェクトの場所を指しています。この場合、SharedObject ファイルは、Adobe Flash Player によりクライアントのホームディレクトリにローカルに格納されます。

共有オブジェクトを作成すると、Flash Player によってアプリケーションおよびドメイン用に新しいディレクトリが作成されます。また、SharedObject データを保存する空の *.sol ファイルも作成されます。このファイルのデフォルトの作成場所は、ユーザーのホームディレクトリです。Windows では、デフォルトで次のディレクトリになります。

```
c:¥Documents and Settings¥username¥user_domain¥Application Data¥Macromedia¥
  Flash Player¥web_domain¥path_to_application¥ApplicationName¥objectName.sol
```

Flex コンテキストの "\sos" サブディレクトリ内で、ローカルホスト上に MyApp.mxml という名前のアプリケーションを要求した場合、Flash Player により Windows の次の場所に *.sol ファイルが格納されます。

```
c:¥Documents and Settings¥username¥user_domain¥Application Data¥Macromedia¥
  Flash Player¥#localhost¥flex¥sos¥MyApp.mxml.swf¥objectName.sol
```

メモ

SharedObject.getLocal() メソッドで名前を指定しない場合、Flash Player により、ファイルに undefined.sol という名前が付けられます。

SharedObject ファイルは、多くの場合には予測可能な場所にありますが、Flash Player がアクセス可能なサンドボックス内の場所であればどこにでも作成することもでき、ファイル名も Flash Player によって任意の名前が割り当てられます。

デフォルトでは、Flash がローカルに保存できる SharedObject オブジェクトのサイズは、ドメインあたり最大 100KB です。アプリケーションで、100 KB よりも大きい共有オブジェクトにデータを保存しようとすると、[ローカル記憶領域] ダイアログボックスが表示されます。ユーザーは、アクセスを要求するドメインに対して、ローカル記憶域を許可または禁止できます。

パスの指定

オプションの pathname パラメータを使用して、SharedObject ファイルの場所を指定できます。このファイルは、ドメインの SharedObject ディレクトリのサブディレクトリです。たとえば、ローカルホストでアプリケーションを要求する場合、次のように指定します。

```
mySO = SharedObject.getLocal("myObjectFile","/");
```

Flash Player は、SharedObject ファイルを "\#localhost" ディレクトリに書き込みます。この方法は、クライアント上で複数のアプリケーションが同一の共有オブジェクトにアクセスできるようにする場合に便利です。この場合、クライアントで 2 つの Flex アプリケーションを実行できます。両方のアプリケーションでドメインのルートである共有オブジェクトへのパスを指定すると、クライアントで、両方のアプリケーションから同一の共有オブジェクトにアクセスできます。一時的に複数のアプリケーションでデータを共有するには、LocalConnection オブジェクトを使用できます。

存在しないディレクトリを指定した場合、Flash Player では、SharedObject ファイルが作成されません。

共有オブジェクトへのデータの追加

SharedObject オブジェクトの data プロパティを使用して、SharedObject の *.sol ファイルにデータを追加します。共有オブジェクトに新しいデータを追加するには、次のシンタックスを使用します。

```
sharedObject_name.data.variable = value;
```

次の例では、userName、itemNumbers、adminPrivileges の各プロパティとその値を SharedObject に追加します。

```
public var currentUserName:String = "Reiner";
public var itemsArray:Array = new Array(101,346,483);
public var currentUserIsAdmin:Boolean = true;
mySO.data.userName = currentUserName;
mySO.data.itemNumbers = itemsArray;
mySO.data.adminPrivileges = currentUserIsAdmin;
```

data プロパティに値を割り当てた後は、その値を SharedObject のファイルに書き込むよう Flash Player に指示する必要があります。強制的に Flash Player が値を SharedObject ファイルに書き込むようにするには、次のように SharedObject.flush() メソッドを使用します。

```
mySO.flush();
```

SharedObject.flush() メソッドを呼び出さない場合、アプリケーションの終了時に、Flash Player によって値がファイルに書き込まれます。ただし、この方法では、データのサイズがデフォルト設定よりも大きい場合、データを格納できるように領域を拡大できません。したがって、SharedObject.flush() は必ず呼び出すようにするのが望ましいと言えます。

共有オブジェクトに、型指定された ActionScript インスタンスを保存できます。これを行うには、flash.net.registerClassAlias() メソッドを呼び出してクラスを登録します。クラスのインスタンスを作成して共有オブジェクトのデータメンバーに保存してから、後でオブジェクトを読み出すと、型指定されたインスタンスを取得できます。デフォルトでは、SharedObject objectEncoding プロパティは AMF3 エンコーディングをサポートし、保存されたインスタンスを SharedObject オブジェクトから展開します。保存されたインスタンスは、registerClassAlias() メソッドを呼び出したときに指定した型と同じ型を保持します。

複数の共有オブジェクトの作成

共有オブジェクトは、1 つの Flex アプリケーションについて複数作成することもできます。これを行うには、個々の共有オブジェクトにそれぞれ異なるインスタンス名を付けます。次を参照してください。

```
public var mySO:SharedObject = SharedObject.getLocal("preferences");
public var mySO2:SharedObject = SharedObject.getLocal("history");
```

これにより、"preferences.sol" ファイルと "history.sol" ファイルが Flex アプリケーションのローカルディレクトリに作成されます。

共有オブジェクトの破棄

クライアントで SharedObject を破棄するには、SharedObject.clear() メソッドを使用します。この方法では、アプリケーションの共有オブジェクトのデフォルトパス内のディレクトリは削除されません。

次の例では、クライアントから SharedObject ファイルを削除します。

```
public function destroySharedObject():void {
   mySO.clear();
}
```

SharedObject の例

次の例は、Date オブジェクトなどの単純なオブジェクトを、手動で直列化および非直列化することなく、SharedObject オブジェクトに格納できることを示しています。

次の例では、最初に初めて訪れたユーザーに歓迎メッセージを表示します。[Log Out] をクリックすると、アプリケーションにより共有オブジェクトに現在の日付が格納されます。次回このアプリケーションを起動したとき、またはページを更新したときに、ユーザーが前回ログアウトした時刻と共に歓迎メッセージを表示します。

実行中のアプリケーションを確認するには、アプリケーションを起動し、[Log Out] をクリックした後にページを更新します。アプリケーションにより、前回訪問したときに [Log Out] ボタンをクリックした日時が表示されます。[Delete LSO] ボタンをクリックすれば、保存した情報をいつでも削除できます。

```
<?xml version="1.0"?>
<!-- lsos/WelcomeMessage.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initApp()">
  <mx:Script><![CDATA[
  public var mySO:SharedObject;
  [Bindable]
  public var welcomeMessage:String;

  public function initApp():void {
     mySO = SharedObject.getLocal("mydata");
     if (mySO.data.visitDate==null) {
        welcomeMessage = "Hello first-timer!"
     } else {
        welcomeMessage = "Welcome back. You last visited on " +
           getVisitDate();
     }
  }

  private function getVisitDate():Date {
     return mySO.data.visitDate;
  }

  private function storeDate():void {
     mySO.data.visitDate = new Date();
     mySO.flush();
  }

  private function deleteLSO():void {
     // Deletes the SharedObject from the client machine.
     // Next time they log in, they will be a 'first-timer'.
     mySO.clear();
  }

  ]]></mx:Script>
  <mx:Label id="label1" text="{welcomeMessage}"/>
  <mx:Button label="Log Out" click="storeDate()"/>
  <mx:Button label="Delete LSO" click="deleteLSO()"/>
</mx:Application>
```

共有オブジェクトの他の使用例については、"samples.war" ファイルに含まれる Flex サンプルアプリケーションを参照してください。

第 36 章

36 アクセシビリティアプリケーションの作成

Adobe Flex 2 にはアクセシビリティ機能が用意されています。この機能を使用すると、障害を持つユーザーがアクセスしやすいアプリケーションを作成できます。アクセシビリティアプリケーションを設計する場合は、ユーザーがどのようにコンテンツを利用するかについて考慮する必要があります。たとえば視覚障害のあるユーザーには、画面の内容を音声で出力するスクリーンリーダーや、画面のごく一部を拡大し、可視スクリーン領域を効果的に制限するスクリーンマグニファイアなどの補助技術が必要です。聴覚障害のあるユーザーには、オーディオコンテンツの代わりになるテキストやキャプションが必要です。身体障害または認知障害のあるユーザーについても考慮する必要があります。

このトピックでは、Flex のアクセシビリティ機能について説明します。

目次

アクセシビリティの概要

Flex に用意されているアクセシビリティ機能を使ってアクセシビリティコンテンツを作成する際は、アクセシビリティを実装するために設計された ActionScript の利点を活用し、推奨される設計および開発方法に従います。次に示す推奨方法の一覧はすべてを網羅していませんが、考慮すべき一般的な事項について記載しています。対象ユーザーのニーズによっては、要件の追加が必要な場合があります。

視覚障害のあるユーザー　視覚障害のあるユーザーに配慮し、次のような設計をお勧めします。

- タブには論理的なタブ順序を設計して実装します。
- コンテンツが頻繁に変更されても、スクリーンリーダーの更新が必要最小限となるようにドキュメントを設計します。たとえば、ループエレメントをグループ化するか、非表示にします。
- ナレーションオーディオのキャプションを表示します。ドキュメント内のオーディオが、スクリーンリーダーを聞くユーザーとのやり取りに使用されることに注意してください。
- 小さな画面サイズでもアプリケーションが適切に拡大および縮小されるように、パーセントによるサイズ指定を使用します。これにより、スクリーンマグニファイアのユーザーが、一度にアプリケーションのより多くの部分を表示できます。視覚障害のあるユーザーの多くは、他のユーザーよりも低い画面解像度でアプリケーションを実行していることにも考慮が必要です。
- 視覚障害のあるユーザーがテキストを読めるように、テキストと背景のコントラストを明確にします。
- マウスやトラックボールといった特定のポインティングデバイスを使用しなくても、アプリケーションを制御できるようにします。
- キーボードでコメントにアクセスでるようにします。アクセシビリティコンポーネントとして定義されている Flex コンポーネントは、すべてキーボードナビゲーションを含んでいます。該当するコンポーネントの一覧と、各コンポーネントに使用できるキーボードコマンドについては、1115 ページの「アクセシビリティコンポーネントとコンテナ」を参照してください。

色覚異常のあるユーザー　色覚異常のあるユーザーに配慮し、情報を色だけで伝える部分をなくします。

身体障害のあるユーザー　身体障害のあるユーザーに配慮し、次のような設計をお勧めします。

- 特定のポインティングデバイスを使用しなくても、アプリケーションを制御できるようにします。
- キーボードでコメントにアクセスでるようにします。アクセシビリティコンポーネントとして定義されている Flex コンポーネントは、すべてキーボードナビゲーションを含んでいます。該当するコンポーネントの一覧と、各コンポーネントに使用できるキーボードコマンドについては、1115 ページの「アクセシビリティコンポーネントとコンテナ」を参照してください。

聴覚障害のあるユーザー　聴覚障害のあるユーザーに配慮し、オーディオコンテンツにはキャプションを追加します。

認知障害のあるユーザー　読書障害などの認知障害のあるユーザーに配慮し、次のような設計をお勧めします。

- 読み進めやすい、整然としたデザインにします。
- アプリケーションの目的とメッセージを伝えるのに役立つグラフィックを提供します。こうしたグラフィックによってテキストやオーディオコンテンツを置き換えるのではなく、テキストやオーディオコンテンツを補強するようにします。
- 一般的な作業は、複数の方法で実行できるようにします。

国際的なアクセシビリティ標準

米国、オーストラリア、カナダ、日本、および EU の多くの国で、World Wide Web Consortium (W3C) により作成された技術基準に基づくアクセシビリティ標準が適用されています。W3C が発行している "Web Content Accessibility Guidelines" には、Web アクセシビリティコンテンツを作成するときにデザイナーが行うべき作業の優先順位が記載されています。Web Accessibility Initiative の詳細については、W3C Web サイト www.w3.org/WAI を参照してください。

アクセシビリティについて定めた米国の法律は、米国リハビリテーション法に追加された Section 508 として知られています。Section 508 では、障害を持つユーザーが利用できない電子テクノロジーの連邦政府機関による購入、開発、維持、または使用を禁じています。また、Section 508 に準拠しない州政府機関を公務員および一般市民が連邦裁判所に訴える権利を認めています。

Section 508 の詳細については、米国政府の Web サイト (www.section508.gov.) を参照してください。

Flex アクセシビリティ Web ページの参照

このトピックでは、Flex に用意されているアクセシビリティ機能、およびアクセシビリティアプリケーション開発の概要を説明します。サポートされるプラットフォーム、既知の問題、スクリーンリーダーの互換性、関連記事、アクセシビリティのサンプルなど、Flex コンテンツの作成と表示の最新情報については、Flex アクセシビリティ Web ページ (www.adobe.com/go/flex_accessibility_jp) を参照してください。

スクリーンリーダー技術について

スクリーンリーダーは、Web サイト内を移動して Web コンテンツを読み上げるために設計されたソフトウェアです。一般に、視覚に障害のあるユーザーに利用される技術です。スクリーンリーダー用に設計されたコンテンツを作成できるプラットフォームは、Microsoft® Windows® に限られています。コンテンツを表示するには、Windows 2000 または Windows XP 以降と、Adobe Flash Player 9 以降および Internet Explorer が必要です。

Freedom Scientific の JAWS は、スクリーンリーダーソフトウェアの 1 つの例です。詳細については、Freedom Scientific Web サイトの JAWS ページ (www.hj.com/fs_products/software_jaws.asp) を参照してください。GW Micro の Window-Eyes も、よく使われるスクリーンリーダープログラムです。Window-Eyes の最新情報については、GW Micro Web サイト (www.gwmicro.com) を参照してください。

メモ

Flex が最も包括的なサポートを提供しているのは、JAWS スクリーンリーダーです。JAWS バージョン 6.10.1006 以降をインストールする必要があります。

スクリーンリーダーを使用すると、Web ページや Flex アプリケーションの内容を知ることができます。定義済みのキーボードショートカットを使用して、スクリーンリーダーでアプリケーション内を簡単に移動することもできます。

情報を音声に変換する方法はスクリーンリーダーアプリケーションによって異なるので、ユーザーに対するコンテンツの表現方法に違いが生じます。アクセシビリティアプリケーションを設計するときには、スクリーンリーダーの動作を制御できないということを考慮する必要があります。アプリケーションのコンテンツをマークアップすることにより、テキストを公開し、スクリーンリーダーのユーザーがコントロールをアクティブにできるようにすることはできます。つまり、アプリケーション内のどのオブジェクトをスクリーンリーダーに公開するか決定し、それらのオブジェクトに説明を加え、オブジェクトが公開される順序を決めることはできます。しかし、特定のテキストが読まれる回数や、コンテンツがどのように読まれるかをスクリーンリーダーに指定することはできません。

Flex アプリケーションで JAWS スクリーンリーダーを使用するには、Flex アプリケーションを起動する前にユーザーが一連のスクリプトをダウンロードする必要があります。詳細については、1114 ページの「Flex アプリケーション用の JAWS スクリーンリーダーの設定」を参照してください。

Flash Player と Microsoft Active Accessibility

Adobe Flash Player は、Microsoft Active Accessibility (MSAA) を採用しています。MSAA は、アプリケーションとスクリーンリーダーがやり取りするために記述された標準的な方法です。MSAA は、Windows オペレーティングシステムでのみ利用可能です。Microsoft Accessibility Technology の詳細については、Microsoft アクセシビリティ Web サイト (www.microsoft.com/enable/default.aspx) を参照してください。

Flash Player 9 の Windows ActiveX (Internet Explorer プラグイン) バージョンは MSAA をサポートしますが、Windows Netscape および Windows スタンドアローンプレーヤーは現時点ではサポートしません。

Flex でサポートされる Flash Player のデバッガバージョンでは、ランタイムのデバッグ情報を表示し、プロファイル情報を生成することができるため、アプリケーションの開発に役立ちます。ただし、Flash Player のデバッガバージョンではアクセシビリティがサポートされません。

注意	MSAA は、現在、不透明表示モードおよび透明表示モードではサポートされていません (これらのモードは [HTML パブリッシュ設定] パネルのオプションであり、Internet Explorer 4.0 以降の Windows バージョンで Flash ActiveX コントロールを使った場合に利用可能です)。Flash コンテンツをスクリーンリーダーで利用できるようにするには、これらのモードを使用しないでください。

Flex アプリケーションのアクセシビリティの設定

このセクションでは、Flex のアクセシビリティ機能を有効にする方法、および Flex アプリケーションで使用するスクリーンリーダーの設定方法について説明します。

Flex のアクセシビリティの有効化

デフォルトでは、Flex アクセシビリティ機能は有効になっていません。アクセシビリティを有効にするには、アプリケーションがスクリーンリーダーと通信できるようにします。

アクセシビリティを有効にするには、次のいずれかの方法を使用します。

- すべての要求に対してアクセシビリティコンテンツが返されるように、デフォルトですべての Flex アプリケーションのアクセシビリティを有効にする。

  すべての Flex アプリケーションのアクセシビリティを有効にするには、"flex-config.xml" ファイルを編集し、次の例のように `<accessible>` プロパティを `true` に設定します。

```
<compiler>
   ...
   <accessible>true</accessible>
   ...
<compiler>
```

- mxmlc コマンドラインコンパイラを使用する場合にアクセシビリティを有効にする。

  mxmlc コマンドラインコンパイラを使用してファイルをコンパイルする場合は、次に示すように、-accessible オプションを使用してアクセシビリティを有効にすることができます。

  ```
  mxmlc -accessible c:¥dev¥myapps¥mywar.war¥app1.mxml
  ```

  コマンドラインコンパイラの詳細については、Flex 2 アプリケーションの構築および展開ガイドの第 9 章の「Flex コンパイラの使用」を参照してください。

Flex データサービス用のアプリケーションを作成し、MXML ファイルとしてデプロイする場合は、accessible クエリパラメータを true に設定することにより、個別の要求でアクセシビリティを有効にできます。次を参照してください。

```
http://www.mycompany.com/myflexapp/app1.mxml?accessible=true
```

"flex-config.xml" ファイルを編集してアクセシビリティをデフォルトで有効にしてある場合は、accessible クエリパラメータを false に設定することにより、個別の要求に対してアクセシビリティを無効にできます。次を参照してください。

```
http://www.mycompany.com/myflexapp/app1.mxml?accessible=false
```

コマンドラインコンパイラの詳細については、Flex 2 アプリケーションの構築および展開ガイドの第 9 章の「Flex コンパイラの使用」を参照してください。

Flex アプリケーション用の JAWS スクリーンリーダーの設定

JAWS スクリーンリーダーを Flex アプリケーションで使用するには、Flex アプリケーションを起動する前に、ユーザーが Adobe のアクセシビリティ Web サイトからスクリプトをダウンロードする必要があります。スクリーンリーダーが最適に動作するのは Forms モードです。このモードでは、ユーザーが Flex アプリケーションと直接やり取りできます。これらのスクリプトを使用すると、ユーザーは Flex アプリケーションのほぼどこからでも、Enter キーを使用して Virtual Cursor モードと Forms モードを切り替えることができます。必要な場合、ユーザーは標準の JAWS キーストロークを使用して Forms モードを終了できます。

ユーザーは、これらのスクリプトとインストールの手順を Adobe Web サイト (www.adobe.com/go/flex_accessibility) からダウンロードできます。

JAWS 用の Flex スクリプトが正しくインストールされたかどうかを検証するには、JAWS が実行されているときに Ins + Q キーを押します。スクリプトが正しくインストールされている場合は、このキーストロークに対して音声により "Updated for Adobe Flex 1.5 and 2" という応答があります。

視覚障害のあるユーザーが JAWS を使用できるように、スクリプトのダウンロードページに適切に誘導することが重要です。

アクセシビリティコンポーネントとコンテナ

アクセシビリティアプリケーションを迅速に構築できるように、アクセシビリティのサポートが Flex コンポーネントおよびコンテナに組み込まれています。これらのコンポーネントおよびコンテナを使用すると、ラベル機能、キーボードアクセス、およびテストに関連する一般的なアクセシビリティ実践の多くが自動化されます。また、複数のインターネットアプリケーション間でユーザーの操作感を統一するのにも役立ちます。

Flex には、次のアクセシビリティコンポーネントおよびコンテナが組み込まれています。

コンポーネント	スクリーンリーダーの動作
Accordion コンテナ	矢印キーを押すとフォーカスが別のパネルに移動します。次にスペースバーまたは Enter キーを押すとこのパネルが選択されます。PageUp および PageDown キーを押すと、コンテナの各パネル間を移動します。 スクリーンリーダーは Accordion コンテナを認識すると、各パネルを "tab" という語で示します。現在のペインは、"active" という語で示されます。 キーボードナビゲーションの詳細については、596 ページの「Accordion コンテナのキーボード操作」を参照してください。
Alert コントロール	Forms モードでは、Alert コントロール内のテキストが読み上げられます。また、デフォルトボタンのラベルが読み上げられます。 Forms モード以外では、下矢印を押すと Alert コントロール内のテキストが 2 回読み上げられます。
Button コントロール	スペースバーを押すと、Button コントロールがアクティブになります。ボタンのアクティブ化をキャンセルするには、スペースバーを離す前に Tab キーを押して Button コントロールからフォーカスを移動します。 スクリーンリーダーが Button コントロールを認識したとき、どのようにアクティブ化されるかは、スクリーンリーダーによって異なります。JAWS 6.10 では、Forms モードがアクティブの場合は、スペースバーによって Button コントロールがアクティブになります。Forms モードが非アクティブの場合は、スペースバーまたは Enter キーを使用して Button コントロールをアクティブにできます。 キーボードナビゲーションの詳細については、253 ページの「Button コントロールのユーザー操作」を参照してください。
CheckBox コントロール	スペースバーを押すと、チェックボックス項目がアクティブになります。 キーボードナビゲーションの詳細については、269 ページの「CheckBox コントロールのユーザー操作」を参照してください。
ComboBox コントロール	キーボードナビゲーションの詳細については、436 ページの「ComboBox コントロールのユーザー操作」を参照してください。

コンポーネント	スクリーンリーダーの動作
DataGrid コントロール	矢印キーを押すと、コンテンツがハイライト表示され、次にそのフィールド内の各文字の間を移動します。 スクリーンリーダーを Forms モードで使用しているときは、Tab キーを押すと DataGrid コントロール内の編集可能な TextInput フィールドの間を移動します。 キーボードナビゲーションの詳細については、448 ページの「DataGrid コントロールのユーザー操作」を参照してください。
DateChooser コントロール	上矢印、下矢印、左矢印、および右矢印キーを押すと、選択した日付が変更されます。Home キーを押すと月内の最初の有効な日付に移動し、End キーを押すと月内の最後の有効な日付に移動します。PageUp キーおよび PageDown キーを押すと、前の月および次の月に移動します。キーボードナビゲーションの詳細については、285 ページの「ユーザーの操作」を参照してください。
DateField コントロール	Ctrl + 下矢印キーを押すと、DateChooser コントロールが開き、適切な日付が選択されます。スクリーンリーダーを Forms モードで使用しているときは、同じキーストロークがキーボードナビゲーションに使用されます。 スクリーンリーダーが Forms モードで DateChooser コントロールを認識すると、このコントロールが “DropDown Calendar currentDate, to open press ControlDown, ComboBox” と読み上げられます。ここで currentDate は現在選択されている日付です。 キーボードナビゲーションの詳細については、285 ページの「ユーザーの操作」を参照してください。
Form コンテナ	キーボードナビゲーションの詳細については、540 ページの「デフォルトボタンの定義」を参照してください。
Image コントロール	Forms モードが非アクティブの場合のみ、ToolTip が定義された Image コントロールがスクリーンリーダーによって読み上げられます。Image コントロールは Forms モードではフォーカス不可能です。また、キーボードでもフォーカス不可能です。
Label コントロール	Label コントロールは、他のコントロールに関連付けられているとき、または Forms モードが非アクティブのときに、スクリーンリーダーによって読み上げられます。Label コントロールは Forms モードではフォーカス不可能です。また、キーボードでもフォーカス不可能です。
LinkButton コントロール	スクリーンリーダーを使用しているとき、LinkButton コントロールがどのようにアクティブ化されるかは、スクリーンリーダーによって異なります。JAWS 6.10 では、Forms モードのときにスペースバーを押すと LinkButton コントロールがアクティブになります。Forms モードが非アクティブのときは、スペースバーまたは Enter キーを押してこのコントロールをアクティブにします。 キーボードナビゲーションの詳細については、288 ページの「LinkButton コントロールのユーザー操作」を参照してください。

コンポーネント	スクリーンリーダーの動作
List コントロール	スクリーンリーダーナビゲーションは、キーボードナビゲーションと同じです。 キーボードナビゲーションの詳細については、425 ページの「キーボード操作」を参照してください。 データヒント (個々のリストエレメントのツールヒント) の作成については、415 ページの「データヒントの表示」を参照してください。スクロールヒント (ユーザーがリストをスクロールしているときに情報を提供するツールヒント) の作成については、416 ページの「スクロールヒントの表示」を参照してください。
Menu コントロール	スクリーンリーダーナビゲーションは、キーボードナビゲーションと同じです。 キーボードナビゲーションの詳細については、402 ページの「Menu コントロールのユーザー操作」を参照してください。
MenuBar コントロール	スクリーンリーダーナビゲーションは、キーボードナビゲーションと同じです。 キーボードナビゲーションの詳細については、402 ページの「Menu コントロールのユーザー操作」を参照してください。
Panel コンテナ	Forms モードが非アクティブのときのみ、スクリーンリーダーによってパネルタイトルが読み上げられます。
RadioButton コントロール	グループ内のラジオボタンを 1 つ選択した状態で Enter キーを押すと、そのグループに入ります。矢印キーを押すと、グループ内の項目間を移動します。下矢印キーおよび右矢印キーを押すとグループ内の次の項目に移動し、上矢印キーおよび左矢印キーを押すとグループ内の前の項目に移動します。 スクリーンリーダーを使用しているときは、スペースバーキーを使ってラジオボタンを選択します。 キーボードナビゲーションの詳細については、272 ページの「RadioButton のユーザーの操作」を参照してください。
RadioButtonGroup コントロール	スクリーンリーダーナビゲーションは、RadioButton コントロールと同じです。
TabNavigator コンテナ	スクリーンリーダーは TabNavigator コンテナペインを認識すると、各ペインを "tab" という語で示します。現在のペインは、"active" という語で示されます。ペインが選択された状態で Enter キーを押すと、そのパネルに移動します。 矢印キーを押すとフォーカスが別のパネルに移動します。次にスペースバーまたは Enter キーを押すとこのパネルが選択されます。PageUp および PageDown キーを押すと、コンテナの各パネル間を移動します。 キーボードナビゲーションの詳細については、593 ページの「TabNavigator コンテナのキーボード操作」を参照してください。
Text コントロール	Text コントロールはフォーカス不可能です。Forms モードが非アクティブのときのみ、スクリーンリーダーによって読み上げられます。

コンポーネント	スクリーンリーダーの動作
TextArea コントロール	Home または Page Down キーを押すと、行頭に移動します。End または Page Up キーを押すと、行末に移動します。
TextInput コントロール	Home または Page Down キーを押すと、行頭に移動します。End または Page Up キーを押すと、行末に移動します。
TitleWindow コンテナ	Forms モードが非アクティブのときのみ、スクリーンリーダーによって TitleWindow が読み上げられます。
ToolTipManager	スクリーンリーダーを使用しているときは、ツールヒントが付いた項目がフォーカスを取得すると、ツールヒントの内容が読み上げられます。非アクセシビリティコンポーネント (Image コントロール以外) に付いたツールヒントは読み上げられません。
Tree コントロール	上矢印キーおよび下矢印キーを押すと、Tree コントロール内の項目間を移動します。グループを開くには、右矢印キーまたはスペースバーを押します。グループを閉じるには、左矢印キーまたはスペースバーを押します。キーボードナビゲーションの詳細については、457 ページの「実行時のノードラベル編集」および 457 ページの「ユーザーによるツリーの操作」を参照してください。

タブ順序と読み取り順序の作成

タブインデックス順序には、ユーザーが Web コンテンツ内を移動する " タブ順序 " と、オブジェクトがスクリーンリーダーによって読み上げられる " 読み取り順序 " の 2 つの面があります。

Flash Player のデフォルトのタブインデックス順序は、左から右、上から下です。ただし、デフォルトのタブ順序を使用しない場合は、`InteractiveObject.tabIndex` プロパティを使用してタブ順序と読み取り順序をカスタマイズできます (ActionScript では、`tabIndex` プロパティは読み取り順序と同義です)。

タブ順序　各コンポーネントの `tabIndex` プロパティを使用してタブ順序を作成できます。タブ順序により、ユーザーが Tab キーを押した場合にオブジェクトが入力フォーカスを受け取る順序が決まります。

読み取り順序　`tabIndex` プロパティを使用すると、スクリーンリーダーがオブジェクトに関する情報を読み上げる順序を制御できます。読み取り順序を作成するには、アプリケーション内の各コンポーネントの `tabIndex` プロパティに値を割り当てます。フォーカス可能なオブジェクトだけでなく、アクセス可能なすべてのオブジェクトに `tabIndex` プロパティを作成します。たとえば、ユーザーが Tab キーで操作できない Text コントロールにも `tabIndex` プロパティを設定する必要があります。アクセス可能なオブジェクトに `tabIndex` プロパティが設定されていない場合、このオブジェクトは適切なタブ順序位置ではなく、タブ順序の末尾に置かれます。

Tab キーで移動したコンポーネントへのスクロール

原則としてアプリケーションは、利用可能なスクリーン領域にすべてのコンポーネントが収まるように構成する必要があります。収まらない場合は、必要に応じて垂直または水平スクロールバーが追加されます。スクロールバーを使用すると、ユーザーはスクリーン領域外のコンポーネントにアクセスできます。

アプリケーションがスクロールバーを使用していても、ユーザーが Tab キーを押してアプリケーションコンポーネント間を移動したときに、アプリケーションが自動スクロールし、現在選択されているコンポーネントが表示されるわけではありません。現在選択されているコンポーネントに自動的にスクロールするようにするには、次のようにアプリケーションにロジックを追加します。

```
<?xml version="1.0"?>
<!-- accessibility\ScrollComp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="setupFocusViewportWatcher();">

    <mx:Script>
        <![CDATA[

        import mx.core.Container;
        import mx.core.EdgeMetrics;

        [Bindable]
        public var cards: Array = [
            {label:"Visa", data:1},
            {label:"Master Card", data:2},
            {label:"American Express", data:3} ];

        [Bindable]
        public var selectedItem:Object;

        [Bindable]
        public var forListDP:Array = [
            {label:'Apple', data:10.00},
            {label:'Banana', data:15.00},
            {label:'Melon', data:3.50},
            {label:'Kiwi', data:7.65},
            {label:'123', data:12.35 },
            {label:'some', data:10.01 }];

        // Set up the event listener for the focusIn event.
        public function setupFocusViewportWatcher():void {
            addEventListener("focusIn", makeFocusedItemVisible);
        }

        public function makeFocusedItemVisible(event:FocusEvent):void {
            // Target is the actual object that has focus.
```

```
    var target:InteractiveObject = InteractiveObject(event.target);

    // OriginalTarget is the component that has focus as some
    // component actually delegate true focus to an internal object.
    var originalTarget:InteractiveObject =
InteractiveObject(focusManager.findFocusManagerComponent(target));

    // The viewable portion of a container
    var viewport:Rectangle = new Rectangle();
    do {
        // Cycle through all parents looking for containers.
        if (target.parent is Container) {
            var viewportChanged:Boolean = false;
            var c:Container = target.parent as Container;

            // Get the viewable area in the container.
            var vm:EdgeMetrics = c.viewMetrics;
            viewport.x = vm.left;
            viewport.y = vm.top;
            viewport.width =
                c.width / c.scaleX - vm.left - vm.right;
            viewport.height =
                c.height / c.scaleY - vm.top - vm.bottom;

            // Calculate the position of the target in the container.
            var topLeft:Point = new Point(0, 0);
            var bottomRight:Point =
        new Point(originalTarget.width, originalTarget.height);
            topLeft = originalTarget.localToGlobal(topLeft);
            topLeft = c.globalToLocal(topLeft);
            bottomRight = originalTarget.localToGlobal(bottomRight);
            bottomRight = c.globalToLocal(bottomRight);

            // Figure out if we have to move the scroll bars.
            // If the scroll bar moves, the position of the component
            // moves as well. This algorithm makes sure the top
            // left of the component is visible if the component is
            // bigger than the viewport.
            var delta:Number;

            if (bottomRight.x > viewport.right) {
                delta = bottomRight.x - viewport.right;
                c.horizontalScrollPosition += delta;
                topLeft.x -= delta;
                viewportChanged = true;
            }

            if (topLeft.x < viewport.left) {
                // leave it a few pixels in from the left
                c.horizontalScrollPosition -=
```

```
                    viewport.left - topLeft.x + 2;
                viewportChanged = true;
            }

            if (bottomRight.y > viewport.bottom) {
                delta = bottomRight.y - viewport.bottom;
                c.verticalScrollPosition += delta;
                topLeft.y -= delta;
                viewportChanged = true;
            }

            if (topLeft.y < viewport.top) {
                // leave it a few pixels down from the top
                c.verticalScrollPosition -=
                    viewport.top - topLeft.y + 2;
                viewportChanged = true;
            }

            // You must the validateNow() method to get the
            // container to move the component before working
            // on the next parent.
            // Otherwise, your calculations will be incorrect.
            if (viewportChanged) {
                c.validateNow();
            }
        }

        target = target.parent;
    }

    while (target != this);
}
]]>
</mx:Script>

<mx:Model id="statesModel" source="assets/states.xml"/>
<mx:Panel
    x="58" y="48"
    width="442" height="201"
    layout="absolute"
    title="Tab through controls to see if focus stays in view">

<mx:VBox x="10" y="10" verticalScrollPolicy="off">
    <mx:TextInput/>
    <mx:TextInput/>
    <mx:TextArea width="328" height="64"/>
    <mx:ComboBox dataProvider="{cards}" width="150"/>
    <mx:DataGrid dataProvider="{forListDP}" />
    <mx:DateChooser yearNavigationEnabled="true"/>
    <mx:List id="source"
```

```
                width="75"
                dataProvider="{statesModel.state}"/>
        </mx:VBox>
    </mx:Panel>
</mx:Application>
```

ActionScript でのアクセシビリティの作成

ドキュメント全体に適用されるアクセシビリティプロパティには、flash.accessibility.AccessibilityProperties クラスを使用します。このクラスの詳細については、Adobe Flex 2 リファレンスガイドを参照してください。

次の表に、AccessibilityProperties クラスの関連プロパティの一部を示します。

プロパティ	データ型	説明
説明	String	スクリーンリーダーに読み取られるコンポーネントの説明を指定します。
forceSimple	Boolean	true に設定した場合は、コンポーネントの子がスクリーンリーダーから隠されます。デフォルト値は false です。
name	String	スクリーンリーダーに読み取られるコンポーネントの説明を指定します。指定された名前がアクセス可能なオブジェクトにない場合、スクリーンリーダーは "Button" などの一般的な語を使用します。
shortcut	String	この表示オブジェクトに関連したキーボードショートカットを示します。
silent	Boolean	true に設定した場合は、コンポーネントがスクリーンリーダーから隠されます。デフォルト値は false です。

これらのプロパティを変更しても、それ自体では影響はありません。Flash Player コンテンツの変更をスクリーンリーダーのユーザーに知らせるには、Accessibility.updateProperties メソッドも使用する必要があります。このメソッドを呼び出すと、Flash Player はすべてのアクセシビリティプロパティを再検証し、スクリーンリーダーに対するプロパティの説明を更新して、必要であれば変更を通知するイベントをスクリーンリーダーに送ります。

複数のオブジェクトのアクセシビリティプロパティを一度に更新する場合は、Accessiblity.updateProperties() メソッドの呼び出しを 1 つだけ含める必要があります (スクリーンリーダーへの過剰な更新によって、スクリーンリーダーが冗長になってしまうことがあります)。

Accessibility.isActive() メソッドによるスクリーンリーダー検出の実装

スクリーンリーダーがアクティブである場合に固有の動作を行うコンテンツを作成する場合は、ActionScript の `Accessibility.active` メソッドを使用できます。このメソッドは、スクリーンリーダーが存在する場合は値 `true` に設定し、スクリーンリーダーが存在しない場合は `false` に設定します。子エレメントをスクリーンリーダーから隠すなど、固有の用途に適した方法で動作するようにコンテンツを作成できます。

たとえば、`Accessibility.active` プロパティを使用して、応答不要のアニメーションを含めるかどうかを決定できます。応答不要のアニメーションは、スクリーンリーダーが何もしなくても動作します。したがって、スクリーンリーダーを混乱させる可能性があります。

Accessibility クラスの詳細については、Adobe Flex 2 リファレンスガイドを参照してください。

聴覚障害のあるユーザー向けのアクセシビリティ

聴覚障害のあるユーザーに対するアクセシビリティの場合は、提供するコンテンツ全体の総合的な理解に不可欠なオーディオコンテンツのキャプションを設定します。たとえば、スピーチのビデオにはアクセシビリティを考慮したキャプションが必要ですが、ボタンを押すと素早く音が出るような場合にキャプションは必要はありません。

アクセシビリティコンテンツのテスト

アクセシビリティアプリケーションをテストする場合は、次の推奨事項に従ってください。

- アクセシビリティが有効であることを確認してください。詳細については、1113 ページの「Flex アプリケーションのアクセシビリティの設定」を参照してください。
- インタラクティブなコンテンツを、ユーザーがキーボードのみを使用して効果的に使用できるかどうか、テストして検証してください。各スクリーンリーダーはそれぞれ異なる方法でキーボードからの入力を処理するので、コンテンツに対して予想どおりのキーストロークが使われるとは限りません。すべてのキーボードショートカットをテストしてください。
- すべてのイメージのツールヒントを確認し、グラフィックコンテンツには必ず代替表現が用意されるようにしてください。
- コンテンツのサイズを変更可能にして、コンテンツにスクリーンマグニファイアを適用しやすくしてください。すべての素材が、小さなサイズでもはっきり表示されるようにしてください。
- 重要な音にはキャプションを付けてください。コンテンツ消失について調べるには、音を消すか、または無関係な音楽をヘッドフォンで聴きながらアプリケーションを使用してみてください。

- スクリーンリーダーや認知障害のあるユーザーが迷わないように、整理された単純なインターフェイスを使用してください。そのアプリケーションを一度も見たことがない人に、手助けやマニュアルを提供せずにアプリケーションのキータスクを実行してもらい、その様子を観察してください。うまく実行できないようであれば、インターフェイスを修正してください。
- スクリーンリーダーを使ってアプリケーションをテストしてください。または、スクリーンリーダーのユーザーにアプリケーションのテストを依頼してください。

第 5 部

Flex データ機能

5

第 V 部では、Adobe Flex 2 データ表現およびデータ機能の使用方法について説明します。データアクセスの詳細については、第 6 部の「データアクセスと相互接続性」を参照してください。

次のトピックが含まれます。

第 37 章

データの表現と操作

37

このトピックでは、データの表現と操作について説明します。データの表現と操作は、データの検証、データのフォーマット、およびオブジェクト間のデータの格納と受け渡しの機能を組み合わせて行います。

目次

データの表現と操作について

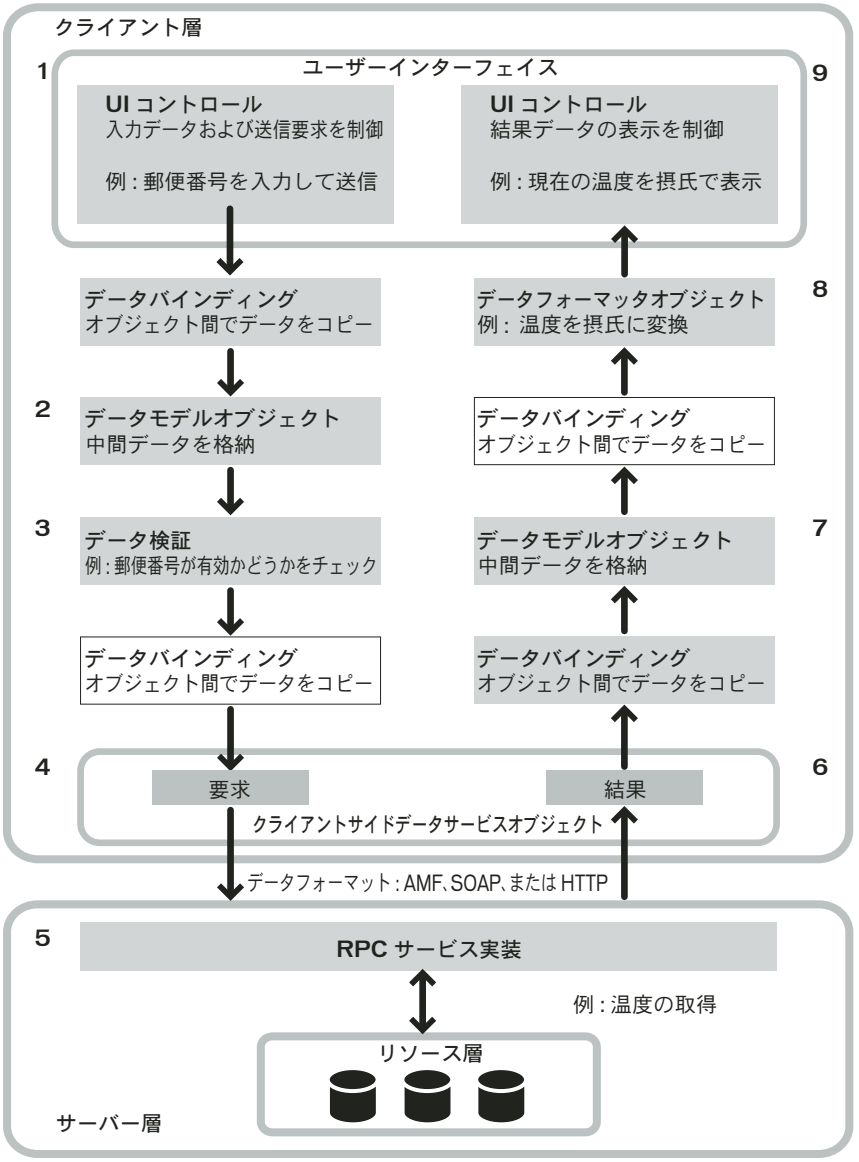
Adobe Flex 2 には、アプリケーションでデータを表現し操作するために、データバインディング、データ検証、データフォーマット機能が用意されています。これらの機能は、Adobe Flex データサービス機能と共に使用できます。これらが一体となって、次の作業を行うことができます。

- クライアントサイドオブジェクト間でのデータ受け渡し
- クライアントサイドのオブジェクトへのデータの格納
- クライアントサイドのオブジェクト間でのデータ受け渡し前のデータ検証
- データを表示する前のデータのフォーマット

次の手順は、ユーザーがデータを入力し、Adobe Flex アプリケーションの情報を要求する簡単なシナリオです。

1. ユーザーが入力フィールドにデータを入力し、Button コントロールをクリックして要求を送信します。
2. (オプション) " データバインディング " によって、中間データを格納するためのデータモデルオブジェクトにデータが渡されます。これにより、データを操作して、アプリケーションの他のオブジェクトに渡すことができるようになります。
3. (オプション) " データバリデータ " オブジェクトによって、要求データが検証されます。バリデータオブジェクトによって、データが特定の条件を満たしているかどうかがチェックされます。
4. データがサーバーサイドオブジェクトに渡されます。
5. サーバーサイドオブジェクトが要求を処理し、データを返します。有効な結果を返すことができない場合は、失敗オブジェクトを返します。
6. (オプション) データバインディングによって、中間データを格納するためのデータモデルオブジェクトにデータが渡されます。これにより、アプリケーションでデータを操作し、他のオブジェクトに渡すことができるようになります。
7. (オプション) " データフォーマッタ " オブジェクトによって、ユーザーインターフェイスで表示するためにデータがフォーマットされます。
8. データバインディングによって、データを表示するユーザーインターフェイスコントロールにデータが渡されます。

次の図は、2 つの異なるユーザー入力例の場合に Flex でどのような処理が行われるのかを示しています。一方の例では、ユーザーは郵便番号データを入力し、Flex によって形式が検証されます。他方の例では、ユーザーは現在の温度を摂氏で要求します。



データバインディング

データバインディング機能は、実行時に自動的にクライアントサイドオブジェクトのプロパティの値を別のオブジェクトのプロパティにコピーするシンタックスを提供します。データバインディングは通常、ソースのプロパティ値が変化するとトリガされます。データバインディングを使用して、ユーザーの入力データを、ユーザーインターフェイスコントロールからデータサービスに渡すことができます。また、データサービスから返された結果をユーザーインターフェイスコントロールに渡すこともできます。

次の例では、Slider コントロールの value プロパティのデータが Text コントロールに取り込まれます。中括弧 ({}) 内のプロパティ名がバインディング式で、この式によってソースプロパティ mySlider.value の値が Text コントロールの text プロパティにコピーされます。

```
<mx:Slider id="mySlider"/>
<mx:Text text="{mySlider.value}"/>
```

詳細については、1133 ページ、第 38 章の「データバインディング」を参照してください。

データモデル

データモデル機能によって、クライアントサイドオブジェクトにデータが格納されます。" データモデル " は、データを格納するプロパティを持ち、オプションで機能を追加するメソッドを持つ場合もある、ActionScript オブジェクトです。データモデルは、アプリケーションでユーザーインターフェイスとデータを分離するために使用します。

データバインディング機能を使用すると、ユーザーインターフェイスデータをデータモデルにバインドできます。また、データバインディング機能によって、データサービスからデータモデルにデータをバインドすることもできます。

単純なデータモデルは MXML タグを使用して定義できます。型指定のないデータの格納以外の機能が必要な場合には、データモデルとして ActionScript クラスを使用します。

次の例は MXML ベースのデータモデルで、TextInput コントロールのプロパティがデータモデルのフィールドにバインドされます。

```
<?xml version="1.0"?>
<!-- datarep\Model.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
     <mx:Model id="reg">
       <registration>
          <name>{nme.text}</name>
          <email>{email.text}</email>
          <phone>{phone.text}</phone>
          <zip>{zip.text}</zip>
          <ssn>{ssn.text}</ssn>
       </registration>
    </mx:Model>
    <mx:TextInput id="nme"/>
```

```
    <mx:TextInput id="email"/>
    <mx:TextInput id="phone"/>
    <mx:TextInput id="zip"/>
    <mx:TextInput id="ssn"/>

</mx:Application>
```

データモデルの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

データ検証

データ検証機能によって、アプリケーションでデータを使用する前に、データが特定の条件を満たしているかどうかが確認されます。" データバリデータ " は、コンポーネント内のデータが正しくフォーマットされているかどうかをチェックする ActionScript オブジェクトです。あらゆるコンポーネントのプロパティにデータバリデータを適用できます。RPC (Remote Procedure Call : リモートプロシージャコール) コンポーネントの宣言で宣言したモデルに対しては、要求が RPC サービスの宛先に送信される直前に、バリデータコンポーネントが適用されるプロパティが検証され、有効な要求だけが送信されます。

次の例は、標準 ZipCodeValidator コンポーネントを使用する MXML コードで、<mx:ZipCodeValidator> タグで記述されています。このコードでは、ユーザーが入力した郵便番号の形式を検証します。ZipCodeValidator バリデータの source プロパティで、検証するプロパティを指定します。

```
<?xml version="1.0"?>
<!-- datarep\Validate.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:TextInput id="input" text="enter zip" width="80"/>

    <mx:Model id="zipModel">
        <root>
            <zip>{input.text}</zip>
        </root>
    </mx:Model>

    <mx:ZipCodeValidator source="{zipModel}" property="zip"
        listener="{input}" trigger="{input}"/>
</mx:Application>
```

バリデータコンポーネントの詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

データフォーマット

データフォーマット機能によって、ユーザーインターフェイスコントロールで表示する前に、データの形式を変更できます。たとえば、データサービスから返されるストリングを (xxx)xxx-xxxx という電話番号形式で表示したい場合に、フォーマッタコンポーネントを使用して表示前にストリングを再フォーマットすることができます。

" データフォーマッタコンポーネント " は、生のデータをカスタマイズしたストリング形式にフォーマットするオブジェクトです。データフォーマッタコンポーネントを使用すると、データサービスから返されたデータを再フォーマットしてバインドできます。

次の例では、DateFormatter コンポーネントを宣言し、MM/DD/YYYY という日付形式を指定しています。Web サービスから返された Date オブジェクトがこの形式にフォーマットされ、TextInput コントロールの text プロパティにバインドされます。

```
<?xml version="1.0"?>
<!-- datarep\Validate.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:WebService id="myService" destination="Shop"/>

    <!-- Declare a formatter and specify formatting properties. -->
    <mx:DateFormatter id="StandardDateFormat" formatString="MM/DD/YYYY"/>

    <!-- Trigger the formatter while populating a string with data. -->
    <mx:TextInput
        text="Your order shipped on
{StandardDateFormat.format(myService.purchase.result.date)}"/>
</mx:Application>
```

データフォーマッタの詳細については、1207 ページ、第 41 章の「データのフォーマット」を参照してください。

第 38 章

データバインディング

38

このトピックでは、アプリケーションのクライアントサイドのオブジェクト間でデータの受け渡しを可能にするデータバインディングについて説明します。バインディングでは、ソースプロパティが変更されるとき、ソースオブジェクトのプロパティの値が、宛先オブジェクトのプロパティに自動的にコピーされます。バインディングを使用すると、ユーザーインターフェイス、データモデル、データサービスなどのアプリケーションの異なるレイヤー間でデータを受け渡しできます。

目次

データバインディングについて

" データバインディング " は、あるオブジェクトのデータを別のオブジェクトに結び付けるプロセスです。データバインディングによって、アプリケーション内でデータの受け渡しを簡単に行うことができます。Adobe Flex 2 でデータバインディングを指定するには、3 つの方法があります。中括弧 ({ }) シンタックスを使用する方法、MXML で <mx:Binding> タグを使用する方法、および ActionScript で BindingUtils メソッドを使用する方法です。

データバインディングの一般的な用途は以下のとおりです。

- ユーザーインターフェイスコントロールのプロパティをデータサービス要求にバインドします。
- データサービスの結果をユーザーインターフェイスコントロールのプロパティにバインドします。
- データサービスの結果を中間層データモデルにバインドし、そのデータモデルのフィールドをユーザーインターフェイスコントロールにバインドします。データモデルの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。
- ユーザーインターフェイスコントロールのプロパティを中間層データモデルにバインドし、そのデータモデルのフィールドをデータサービス要求にバインドします (3 層システム)。
- ArrayCollection オブジェクトまたは XMLListCollection オブジェクトをリストコントロールの dataProvider プロパティにバインドします。
- 複雑なプロパティの個別の部分をユーザーインターフェイスコントロールのプロパティにバインドします。たとえば、List コントロールのアイテムをクリックすると他のいくつかのコントロールのデータが表示されるマスター詳細シナリオが考えられます。
- バインディング式の中で、E4X (ECMAScript for XML) 式を使用して XML データをユーザーインターフェイスコントロールにバインドします。

バインディングは強力なメカニズムですが、あらゆる状況に適しているわけではありません。たとえば、個別の設定を厳密なタイミングで更新する必要がある複雑なユーザーインターフェイスでは、プロパティを順番に割り当てるメソッドを用いたほうが好ましいと考えられます。また、バインディングはプロパティが変更されるたびに実行されるので、毎回ではなく特定の変更時にのみ変更を認識させる場合、最良のソリューションではありません。

データバインディングには、ソースプロパティ、宛先プロパティ、およびソースから宛先にデータをいつコピーするかを指定するトリガが必要です。次の例では、HSlider コントロールの value プロパティのデータが Text コントロールに取り込まれます。中括弧内のプロパティ名がバインディング式のソースプロパティです。ソースプロパティの値が変更されると、Flex によって、ソースプロパティ mySlider.value の現在の値が、宛先プロパティである Text コントロールの text プロパティにコピーされます。

```
<?xml version="1.0"?>
<!-- binding/BasicBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:HSlider id="mySlider"/>
```

```
    <mx:Text text="{mySlider.value}"/>
</mx:Application>
```

コンポーネントのプロパティはすべて、データバインディング式の宛先として使用できます。ただし、プロパティをデータバインディング式のソースとして使用するためには、データバインディングをサポートするようにコンポーネントを実装する必要があります。つまり、プロパティの値が変更されてバインディングがトリガされると、コンポーネントからイベントが送出されます。データバインディング式のソースとして使用できるコンポーネントプロパティの作成に関する詳細については、『Flex 2 コンポーネントの作成と拡張』の「Bindable メタデータタグ」を参照してください。

『Adobe Flex 2 リファレンスガイド』では、データバインディング式のソースとして使用できるプロパティについて、次の文章が説明の中に記載されています。

“ このプロパティはデータバインディングのソースとして使用できます。”

バインディングは、以下の場合に実行されます。

- バインディング元であるオブジェクトからイベントが送出される場合。
- アプリケーションコードからサービスを呼び出す場合。

プロパティだけでなく ActionScript 関数もバインディング式のソースとして使用することができます。ActionScript 関数をバインディング式のソースとして使用するのは、通常、関数の引数としてバインド可能プロパティを使用している場合です。プロパティが変更されると関数が実行され、結果がバインディング先で使用されます。

バインディングのソースまたは宛先として、Function 型のプロパティも使用できます。Function 型プロパティは、関数への参照を格納する変数です。

中括弧を使用したデータバインディング

中括弧シンタックスの使用は、アプリケーションでオブジェクト間のデータの受け渡しを行うための最も簡単な方法です。このシンタックスを使用する場合、宛先プロパティの値となるバインディングソースを中括弧 ({ }) で囲みます。

次の例では、あるデータモデルに対するバインディングのソースとして、コントロールの複数のプロパティを使用しています。データモデルの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

```
<?xml version="1.0"?>
<!-- binding/BindingBraces.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Data model stores registration data that user enters. -->
    <mx:Model id="reg">
        <registration>
            <name>{fullname.text}</name>
            <email>{email.text}</email>
            <phone>{phone.text}</phone>
```

```
            <zip>{zip.text}</zip>
            <ssn>{ssn.text}</ssn>
        </registration>
    </mx:Model>

    <!-- Form contains user input controls. -->
    <mx:Form>
        <mx:FormItem label="Name" required="true">
            <mx:TextInput id="fullname" width="200"/>
        </mx:FormItem>

        <mx:FormItem label="Email" required="true">
            <mx:TextInput id="email" width="200"/>
        </mx:FormItem>

        <mx:FormItem label="Phone" required="true">
            <mx:TextInput id="phone" width="200"/>
        </mx:FormItem>

        <mx:FormItem label="Zip" required="true">
            <mx:TextInput id="zip" width="60"/>
        </mx:FormItem>

        <mx:FormItem label="Social Security" required="true">
            <mx:TextInput id="ssn" width="200"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

中括弧内での ActionScript 式の使用

中括弧内のバインディング式には、値を返す ActionScript 式を含めることができます。たとえば、次のタイプのバインディングで中括弧のシンタックスを使用できます。

- 中括弧内の 1 つのバインド可能プロパティ
- 中括弧内にバインド可能プロパティを含むストリングの連結
- 中括弧内のバインド可能プロパティの計算
- バインド可能プロパティ値を評価する条件演算

次の例は、各タイプのバインディング式を使用するデータモデルを示しています。

```
<?xml version="1.0"?>
<!-- binding/AsInBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Model id="myModel">
        <myModel>
          <!-- Perform simple property binding. -->
          <a>{nameInput.text}</a>
```

```
            <!-- Perform string concatenation. -->
            <b>This is {nameInput.text}</b>
            <!-- Perform a calculation. -->
            <c>{(Number(numberInput.text) as Number) * 6 / 7}</c>
            <!-- Perform a conditional operation using a ternary operator. -->
            <d>{(isMale.selected) ? "Mr." : "Ms."} {nameInput.text}</d>
        </myModel>
    </mx:Model>

    <mx:Form>
        <mx:FormItem label="Last Name:">
            <mx:TextInput id="nameInput"/>
        </mx:FormItem>
        <mx:FormItem label="Select sex:">
            <mx:RadioButton id="isMale"
                label="Male"
                groupName="gender"
                selected="true"/>
            <mx:RadioButton id="isFemale"
                label="Female"
                groupName="gender"/>
        </mx:FormItem>
        <mx:FormItem label="Enter a number:">
            <mx:TextInput id="numberInput" text="0"/>
        </mx:FormItem>
    </mx:Form>

    <mx:Text
       text="{'Calculation: '+numberInput.text+' * 6 / 7 = '+myModel.c}"/>
    <mx:Text text="{'Conditional: '+myModel.d}"/>
</mx:Application>
```

中括弧内での E4X 式の使用

中括弧または <mx:Binding> タグ内のバインディング式には、バインディングのソースが XML 型のバインド可能プロパティである場合に、ECMAScript for XML (E4X) 式を含めることができます。バインディング先が String プロパティの場合は、中括弧のバインディング式から toString() メソッドが自動的に呼び出されます。詳細については、1143 ページの「<mx:Binding> タグ内での E4X 式の使用」を参照してください。

次の例のコードには、XML オブジェクトからのデータをバインドする 3 つのバインディング式が中括弧内にあります。1 つ目では . (1 つのドット表記) を、2 つ目では .. (2 つのドット表記) を、3 つ目では || (OR 表記) を、それぞれ使用しています。

```
<?xml version="1.0"?>
<!-- binding/E4XInBraces.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:Script>
        <![CDATA[

            [Bindable]
            public var xdata:XML = <order>
                <item id = "3456">
                    <description>Big Screen Television</description>
                    <price>1299.99</price><quantity>1</quantity>
                </item>
                <item id = "56789">
                    <description>DVD Player</description>
                    <price>399.99</price>
                    <quantity>1</quantity>
                </item>
            </order>;
        ]]>
    </mx:Script>

    <mx:Label text="Using .. notation."/>
    <!-- Inline databinding will automatically call the
        toString() method when the binding destination is a string. -->
    <mx:List width="25%"
        dataProvider="{xdata..description}"/>

    <mx:Label text="Using . notation."/>
    <mx:List width="25%"
        dataProvider="{xdata.item.description}"/>

    <mx:Label text="Using || (or) notation."/>
    <mx:List width="25%"
        dataProvider="{xdata.item.(@id=='3456'||@id=='56789').description}"/>
</mx:Application>
```

中括弧内での ActionScript 関数の使用

ActionScript 関数をバインディング式のソースとして使用できます。ActionScript 関数をバインディング式のソースとして使用するのは、通常、関数の引数としてバインド可能プロパティを使用している場合です。バインド可能プロパティが変更されると関数が実行され、結果がバインディング先で使用されます。例を次に示します。

```
<?xml version="1.0"?>
<!-- binding/ASInBraces.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            [Bindable]
            public var inString:String;
```

```
            public function getNewText(sourceString:String):String {
                return sourceString.toUpperCase();
            }
        ]]>
    </mx:Script>

    <mx:TextInput id="myTI" text="Enter text here"/>
    <mx:Button label="Trigger Binding" click="inString=myTI.text;"/>

    <mx:TextArea text="{getNewText(inString)}"/>
</mx:Application>
```

この例では、Flex によって getNewText() 関数が呼び出され、inString プロパティが更新されるたびに TextArea コントロールが更新されます。

inString プロパティを引数として渡さず、関数の内部から参照している場合は、inString プロパティが変更されても関数は呼び出されません。次の例では、getNewText() から inString プロパティを参照しています。

```
public function getNewText():String {
   if(inString == "")
   ...
}
```

次に、この関数をバインディング式の中で使用します。

```
<mx:TextArea text="{getNewText()}"/>
```

getNewText() 関数はアプリケーションの起動時に 1 回呼び出されますが、inString プロパティが変更されてもデータバインディングはトリガされず、TextArea コントロールは変更されません。

<mx:binding> タグを使用したデータバインディング

中括弧シンタックスの代わりに、<mx:Binding> タグを使用できます。<mx:Binding> タグを使用する場合は、<mx:Binding> タグの source プロパティにソースプロパティを指定し、destination プロパティに宛先プロパティを指定します。この方法は、中括弧シンタックスを使用する場合と同じです。

中括弧シンタックスとは対照的に、<mx:Binding> タグではビュー (ユーザーインターフェイス) をモデルから完全に分離することができます。さらに、複数の <mx:Binding> タグに同一の宛先を指定できるので、異なるソースのプロパティを同じ宛先のプロパティにバインドすることができます。

次の例では、<mx:Binding> タグを使用して、ユーザーインターフェイスコントロールのプロパティを myEmployee データモデルにバインドしています。

```
<?xml version="1.0"?>
<!-- binding/BindingTags.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- The myEmployee data model. -->
    <mx:Model id="myEmployee">
        <Employee>
            <name>
                <first/>
                <last/>
            </name>
            <department/>
            <email/>
        </Employee>
    </mx:Model>

    <!-- Properties of user interface controls are bound to the
        myEmployee data model using <mx:Binding> tags. -->
    <mx:Binding source="firstName.text"
        destination="myEmployee.name.first"/>
    <mx:Binding source="lastName.text"
        destination="myEmployee.name.last"/>
    <mx:Binding source="department.text"
        destination="myEmployee.department"/>
    <mx:Binding source="email.text"
        destination="myEmployee.email"/>

    <!-- Form contains user input controls. -->
    <mx:Form label="Employee Information">
        <mx:FormItem label="First Name">
            <mx:TextInput id="firstName"/>
        </mx:FormItem>
        <mx:FormItem label="Last Name">
            <mx:TextInput id="lastName"/>
        </mx:FormItem>
        <mx:FormItem label="Department">
            <mx:TextInput id="department"/>
        </mx:FormItem>
        <mx:FormItem label="Email Address">
            <mx:TextInput id="email"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

バインディングタグ内での ActionScript 式の使用

<mx:Binding> タグの source プロパティには、中括弧を含めることができます。source プロパティに中括弧が含まれていない場合、その値は単一の ActionScript 式として扱われます。source プロパティに中括弧が含まれている場合、その値は連結された ActionScript 式として扱われます。次の例の <mx:Binding> タグは互いに同等であり、有効です。

```
<?xml version="1.0"?>
<!-- binding/ASInBindingTags.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function whatDogAte():String {
                return "homework";
            }
        ]]>
    </mx:Script>

    <mx:Binding
        source="'The dog ate my '+ whatDogAte()"
        destination="field1.text"/>
    <mx:Binding
        source="{'The dog ate my '+ whatDogAte()}"
        destination="field2.text"/>
    <mx:Binding
        source="The dog ate my {whatDogAte()}"
        destination="field3.text"/>

    <mx:TextArea id="field1"/>
    <mx:TextArea id="field2"/>
    <mx:TextArea id="field3"/>
</mx:Application>
```

次の例の source プロパティは、ActionScript 式ではないので無効です。

```
<mx:Binding source="The dog ate my homework" destination="field1.text"/>
```

複数のソースプロパティから1つの宛先プロパティへのバインド

複数のソースプロパティを1つの宛先プロパティにバインドするには、複数の <mx:Binding> タグを使用して複数のソースを1つの宛先に指定するか、中括弧内のバインディング式と <mx:Binding> タグを組み合わせます。この操作は、中括弧シンタックスのみでは行えません。

次の例では、データモデルフィールド thing1.part が宛先になり、input1.text と input2.text がソースになります。input1.text または input2.text が更新されると、更新された値が thing1.part に格納されます。

```
<?xml version="1.0"?>
<!-- binding/BindMultSources.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Model id="thing1">
        <data>
            <part>{input1.text}</part>
        </data>
    </mx:Model>

    <mx:Binding source="input2.text" destination="thing1.part"/>

    <mx:TextInput id="input1"/>
    <mx:TextInput id="input2"/>

    <mx:TextArea text="{thing1.part}"/>
</mx:Application>
```

1つのソースプロパティから複数の宛先プロパティへのバインド

1つのソースプロパティから複数の宛先プロパティへバインドできます。次の例では、TextInput コントロールの text プロパティが、2 つのデータモデルのプロパティにバインドされ、データモデルのプロパティが 2 つの Label コントロールの text プロパティにバインドされています。

```
<?xml version="1.0"?>
<!-- binding/BindMultDestinations.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Model id="mod1">
        <data>
            <part>{input1.text}</part>
        </data>
    </mx:Model>

    <mx:Model id="mod2">
```

```
        <data>
            <part>{input1.text}</part>
        </data>
    </mx:Model>

    <mx:TextInput id="input1" text="Hello" />

    <mx:Label text="{mod1.part}"/>
    <mx:Label text="{mod2.part}"/>
</mx:Application>
```

<mx:Binding> タグ内での E4X 式の使用

中括弧または <mx:Binding> タグ内のバインディング式には、バインディングのソースが XML 型のバインド可能プロパティである場合に、E4X 式を含めることができます。中括弧内の E4X 式とは異なり、<mx:Binding> タグ内で E4X 式を使用する際は、バインディング先が String プロパティの場合に toString() メソッドを明示的に呼び出す必要があります。

次の例のコードには、XML オブジェクトからのデータをバインドする 3 つのバインディング式が中括弧内にあります。1 つ目では . (1 つのドット表記) を、2 つ目では .. (2 つのドット表記) を、3 つ目では || (OR 表記) を、それぞれ使用しています。

```
<?xml version="1.0"?>
<!-- binding/E4XInBindingTag.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="600" height="900">

    <mx:Script>
        <![CDATA[
            [Bindable]
            public var xdata:XML =
                <order>
                    <item id = "3456">
                        <description>Big Screen Television</description>
                        <price>1299.99</price><quantity>1</quantity>
                    </item>
                    <item id = "56789">
                        <description>DVD Player</description>
                        <price>399.99</price>
                        <quantity>1</quantity>
                    </item>
                </order>;
        ]]>
    </mx:Script>

    <mx:Label text="Using .. notation."/>

    <!-- This will update because what is
        binded is actually the String and XMLList. -->
```

```
    <mx:List width="75%" id="txts"/>
    <mx:Binding
        source="xdata..description"
        destination="txts.dataProvider"/>

    <mx:Label text="Using . notation."/>
    <mx:List width="75%" id="txt2s"/>
    <mx:Binding
        source="xdata.item.description"
        destination="txt2s.dataProvider"/>

    <mx:Label text="Using || (or) notation."/>
    <mx:List width="75%" id="txt3s"/>
    <mx:Binding
        source="xdata.item.(@id=='3456'||@id=='56789').description"
        destination="txt3s.dataProvider"/>
</mx:Application>
```

バインディングメカニズムについて

コンパイル時に MXML コンパイラが、MXML ドキュメントに含まれるバインディングタグや式に対応する ActionScript の Watcher オブジェクトおよび Binding オブジェクトを作成するためのコードを生成します。実行時に、バインディングソース式の構成要素に基づく変更イベントによって Watcher オブジェクトがトリガされ、その Watcher オブジェクトから Binding オブジェクトがトリガされて、バインディングが実行されます。

可能な限り最良のバインディング結果を得るためには、変数の型指定を厳密に行う必要があります。List クラスなど、Flex の標準クラスの 1 つを使用する場合は、`selectedItem` プロパティが Object として型指定されていることを知っておく必要があります。`selectedItem` が、モデルまたは RPC サービスの結果ではなく実際のカスタムクラスである場合は、次に示すバインディング式のようにキャストする必要があります。

```
{MyClass(myList.selectedItem).someProp}
```

バインド可能プロパティチェーンの使用

データバインディングのソースとしてプロパティを指定すると、そのプロパティの変更だけでなく、そのプロパティが含まれるプロパティチェーンも監視されます。宛先プロパティを含むプロパティのチェーン全体は、" バインド可能プロパティチェーン " と呼ばれます。次の例の firstName.text は、firstName オブジェクトとその text プロパティを含むバインド可能プロパティチェーンです。

```
<first>{firstName.text}</first>
```

バインド可能プロパティチェーン内の指定したプロパティが変更された場合には、イベントを生成する必要があります。プロパティが [Bindable] メタデータタグによってマークされている場合は、Flex コンパイラによって自動的にイベントが生成されます。[Bindable] メタデータタグの使用例の詳細については、『Flex 2 コンポーネントの作成と拡張』の「Bindable メタデータタグ」を参照してください。

次の例は、変数および getter プロパティに対して [Bindable] メタデータタグを使用し、dispatchEvent() 関数を呼び出す方法を示します。

```
[Bindable]
public var minFontSize:Number = 5;


[Bindable("textChanged")]
public function get text():String {
    return myText;
}

public function set text(t : String):void {
    myText = t;
   dispatchEvent( new Event( "textChanged" ) );}
```

[Bindable] メタデータタグでイベント名を省略した場合は、Flex コンパイラによって propertyChange というイベントが自動的に生成されて送出されるため、プロパティをデータバインディング式のソースとして使用できます。

また、オブジェクトを既知の型にキャストすることによって、オブジェクトに関する詳しい情報をコンパイラに提供する必要があります。次の例では、List コントロールの myList に Customer オブジェクトが含まれているので、selectedItem プロパティは Customer オブジェクトにキャストされます。

```
<mx:Model id="selectedCustomer">
  <customer>
    <name>{Customer(myList.selectedItem).name}</name>
    <address>{Customer(myList.selectedItem).address}</address>
    ...
  </customer>
</mx:Model>
```

場合によっては、バインディングが意図したとおりに自動的に実行されないことがあります。dataProvider プロパティのアイテム全体を変更した場合には、バインディングは自動的には実行されません。

また、次の例のように、プロパティのサブプロパティが [Bindable] メタデータを持つ場合にも、バインディングは自動的には実行されません。

```
...
[Bindable]
var temp;
// バインディングはトリガされます。
temp = new Object();
// ラベルは Object のバインド可能プロパティではないため、
// バインディングはトリガされません。
temp.label = foo;
...
```

このコード例では、{temp.label} に関して、temp が Object であることに問題があります。次のいずれかの方法で問題を解決できます。

- Object の初期化前処理を行います。
- ObjectProxy を temp に割り当てます。ObjectProxy のプロパティはすべてバインド可能です。
- temp を、バインド可能な label プロパティを使用して、厳密に型指定されたオブジェクトにします。

mouseX プロパティなど、Flash Player によって自動的に更新されるプロパティにデータをバインディングする場合にも、バインディングは自動的には実行されません。

UIComponent クラスの executeBindings() メソッドは、UIComponent オブジェクトを宛先とするすべてのバインディングを実行します。Repeater コンポーネントを含むすべてのコンテナおよびコントロールは、UIComponent クラスを拡張したものです。Container および Repeater クラスの executeChildBindings() メソッドは、Container または Repeater クラスの子 UIComponent コンポーネントを宛先とするすべてのバインディングを実行します。コンテナはすべて、Container クラスを拡張したものです。

これらのメソッドにより、意図したとおりに行われないバインディングを実行できます。変更を加えてもバインディングが実行されない場合にユーザーインターフェイスを更新するには、executeChildBindings() メソッドへの呼び出しなどのコードを 1 行追加します。ただし、executeBindings() メソッドは、バインディングが自動的に実行されないことがわかっている場合にのみ使用します。

ActionScript でのバインディングの定義

Help ID:

通常、MXML でデータバインディングを定義するには、中括弧 ({ }) <mx:Binding> タグを使用します。ActionScript でバインディングを定義する場合は、mx.binding.utils.BindingUtils クラスを使用することもできます。このクラスでは静的なメソッドが定義されています。bindProperty() メソッドを使用すると、変数として実装されているプロパティへのバインディングを作成できます。または、bindSetter() メソッドを使用して、setter メソッドとして実装されているプロパティへのバインディングを作成できます。

次の例では、bindProperty() メソッドを使用し、TextInput コントロールと TextArea コントロール間でバインディングを作成しています。

```
<?xml version="1.0"?>
<!-- binding/BindInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="initBinding();">

    <mx:Script>
      <![CDATA[

         import mx.binding.utils.*;

         // Define data binding.
         public function initBinding():void {
          BindingUtils.bindProperty(textarea, "text", textinput, "text");
         }
      ]]>
    </mx:Script>

    <mx:TextInput id="textinput" text="Hello"/>
    <mx:TextArea id="textarea"/>
    <mx:Button label="Submit" click="textinput.text='Goodbye';"/>
</mx:Application>
```

次の例では、bindSetter() メソッドを使用して 2 つのバインディングを設定しています。TextInput コントロールにテキストを入力するにつれて、対応する TextArea コントロールにテキストがコピーされます。

```
<?xml version="1.0"?>
<!-- binding/BindSetterAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.binding.utils.*;
            import mx.events.FlexEvent;

            // Set method.
```

```
            public function setMyString(val:String):void {
                taSetter2.text = val;
            }

            <!-- Event listener to configure binding with a setter. -->
            public function mySetterBindingInline(event:FlexEvent):void {
                var watcherSetter:ChangeWatcher =
                    BindingUtils.bindSetter(
                        function(v:String):void {
                            taSetter1.text = v}, tiSetter1, "text");
            }

            <!-- Event listener to configure binding with a setter. -->
            public function mySetterBinding(event:FlexEvent):void {
                var watcherSetter:ChangeWatcher =
                    BindingUtils.bindSetter(setMyString, tiSetter2, "text");
            }
        ]]>
    </mx:Script>

    <mx:Label text="Bind Setter using inline setter"/>
    <mx:TextInput id="tiSetter1" text="Hello Setter" />
    <mx:TextArea id="taSetter1"
        initialize="mySetterBindingInline(event);"/>

    <mx:Label text="Bind Setter using setter method"/>
    <mx:TextInput id="tiSetter2" text="Hello Setter" />
    <mx:TextArea id="taSetter2"
        initialize="mySetterBinding(event);"/>
</mx:Application>
```

バインディング Watcher の定義

バインディングがアプリケーションの中でいつ発生したのかを検出することができます。Flex に組み込まれている mx.binding.utils.ChangeWatcher クラスを使用すると、データバインディングの Watcher を定義できます。通常、バインディングが発生すると、イベント Watcher からイベントリスナーが呼び出されます。

バインディング Watcher を設定するには、次の例のように ChangeWatcher クラスの静的メソッド `watch()` を使用します。

```
<?xml version="1.0"?>
<!-- binding/DetectWatcher.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="initWatcher();">

    <mx:Script>
      <![CDATA[
        import mx.utils.*;
```

```
    import mx.binding.utils.*;
    import mx.events.FlexEvent;
    import mx.events.PropertyChangeEvent;

    // Define a bindable property.
    [Bindable]
    public var propChain:Object;

    public var myWatcher:ChangeWatcher;

    // Define binding watcher.
    public function initWatcher():void {
        // Define a watcher for the text binding.
        ChangeWatcher.watch(textarea, "text", watcherListener);

        // Initialize the Object.
        propChain = new ObjectProxy();
        propChain.sub1 = new ObjectProxy();
        propChain.sub1.sub2 = new String("first sub2");

        // Define a watcher for the Object binding.
        myWatcher=
            ChangeWatcher.watch(this, ["propChain","sub1","sub2"],
                watcherListenerProp);
    }

    // Event listener when binding occurs.
    public function watcherListener(event:Event):void {
        myTA1.text="binding occurred";
    }

    // Event listener when binding occurs.
    public function watcherListenerProp(event:PropertyChangeEvent):void
    {
        myTA2.text="prop chain binding occurred";
        myWatcher.unwatch();
    }
  ]]>
</mx:Script>

<!-- Define a binding expression to watch. -->
<mx:TextInput id="textinput" text="Hello"/>
<mx:TextArea id="textarea" text="{textinput.text}"/>

<!-- Trigger a binding. -->
<mx:Button label="Submit" click="textinput.text='Goodbye';"/>
<mx:TextArea id="myTA1"/>

<!-- Trigger a binding to a bindable property chain. -->
<mx:Button label="Submit prop chain"
```

```
        click="propChain.sub1.sub2 = 'second sub2';"/>
    <mx:TextArea id="myTA2"/>
</mx:Application>
```

この例では、ObjectProxy クラスを使用して propChain 変数を操作しています。このクラスはアイテムへの変更を追跡する機能を備えているためです。この例では、propChain Object のイベントリスナーで unwatch() メソッドを使用し、最初のバインディングが発生した後に Watcher を削除しています。

イベントリスナーはバインディング Watcher ごとに定義しますが、各イベントリスナーはイベントオブジェクトを格納する引数を 1 つ受け取ります。イベントオブジェクトのデータ型は、監視対象のプロパティによって決まります。バインド可能プロパティからはそれぞれ、異なるイベントタイプと関連するイベントオブジェクトを送出できます。プロパティをバインド可能にするために使用する [Bindable] メタデータタグで、プロパティが変更されたときに送出されるイベントのイベントタイプを指定します。[Bindable] メタデータタグの詳細については、『Flex 2 コンポーネントの作成と拡張』の第 5 章の「カスタムコンポーネントでのメタデータタグの使用」を参照してください。

watcherListener() イベントリスナーは FlexEvent 型のイベントオブジェクトを受け取ります。これは、TextArea コントロールの text プロパティが変更されたときにそのプロパティによって送出されるイベントのデータ型であるためです。

watcherListenerProp() イベントリスナーは PropertyChangeEvent 型のイベントオブジェクトを受け取ります。これは、propChain Object によって送出されるイベントのデータ型であるためです。このデータ型は propertyChange イベントに関連付けられたイベントタイプでもあります。このイベントは、[Bindable] メタデータタグでイベントタイプを指定しない場合にプロパティから送出されるデフォルトイベントです。

どちらのイベントリスナーでも、すべての Flex イベントの基本クラスであるイベントタイプ flash.events.Event を指定できます。

バインディング機能を使用する際の注意事項

バインディング機能を使用するときは、以下の点に注意してください。

- SharedObject オブジェクトは Adobe Flash Player に組み込まれているので、このオブジェクトの動作をバインディングメカニズムで変更することはできません。SharedObject でバインディングを使用するには、SharedObject オブジェクトを内部的に使用し、アプリケーションの残りの部分で利用可能な getter または setter プロパティを持つ、ラッパーオブジェクトを記述する必要があります。ラッパーオブジェクトを使用しないと、SharedObject でデータが失われる可能性があります。特に、バインディング式で使用されたデータが失われる場合があります。
- オブジェクトへのバインディングにおいて変数が Object として型指定される場合、オブジェクトが getter または setter プロパティを持っているときはキャストを使用する必要があります。キャストを使用すると、Repeater オブジェクトなど、`dataProvider` プロパティを持つオブジェクトが影響を受けます。キャストの詳細については、1145 ページの「バインド可能プロパティチェーンの使用」を参照してください。
- List クラスの `selectedItem` プロパティと Repeater クラスの `currentItem` プロパティは、Object として明示的に型指定されています。プロパティが名前と値のペアを含む単純なオブジェクトである場合は、バインディングが正しく動作します。しかし、プロパティが型指定されたオブジェクトで、バインドされたプロパティが getter または setter プロパティである場合は、バインディングが失敗します。変数が Object として明示的に型指定されているので、警告は出ません。`selectedItem` または `currentItem` プロパティにバインディングするときに、バインディング先のプロパティが getter または setter として実装されている場合は、適切なクラスにキャストしてください。次に例を示します。

  ```
  {MyGetterClass(theList.selectedItem).myGetterProp}
  ```

- コンポーネントでバインディング式が使用されている場合、バインディングはコンポーネントの変更イベントが実行される前に実行され、バインディング先が更新されてから、そのコンポーネントの変更イベントが実行されます。ただし、まだ作成していないオブジェクトにコンポーネントをバインドすることもできます。すべてのバインディングを実行してからコードが実行されるようにするには、ルートタグのイベントを使用します。
- スタイルにバインドすることはできません。
- モデルをコンポーネントの `dataProvider` プロパティにバインドする場合は、モデルのアイテムを直接変更するのではなく、Collections API を使用してアイテムを変更してください。Collections API を使用しない場合、モデルをバインドしたコンポーネントは再描画されないので、モデルへの変更が表示されません。たとえば、次のコードはお勧めしません。

  ```
  myGrid.getItemAt(itemIndex).myField = 1;
  ```

  次のコードをお勧めします。

  ```
  myGrid.dataProvider.editField(itemIndex, "myField", 1);
  ```

- Array のエレメントを実行時のバインディングソースとして使用することはできません。ソースの Array の個別のフィールドが変更された場合、バインドされた Array は更新されません。バインディングにより、インスタンス化の実行中に値がコピーされます。これは、`<mx:Script>` タグで変数が宣言された後、イベントリスナーが実行される前に行われます。

データバインディングのデバッグ

場合によっては、データバインディングが正常に機能せず、デバッグが必要になることがあります。データバインディングの問題の解決法を以下に示します。

- 警告に注意してください。

  バインディングが開始時に正常に動作していると、警告が表示されても問題がないと判断しがちですが、警告は重要です。

  getter プロパティまたは setter プロパティの `[Bindable]` が存在しないという警告が表示された場合、バインディングが開始時に正常に行われていても、プロパティに対するそれ以降の変更は認識されません。

  静的変数またはビルトインプロパティに関する警告が表示された場合、変更は認識されません。

  コンパイラによって不明の型であると警告されたプロパティが、明らかにバインド可能プロパティである場合は、`[Bindable]` メタデータなどの適切な型情報をコンパイラに提供できるように、バインディング式でオブジェクトをキャストする必要があります。キャストの詳細については、1145 ページの「バインド可能プロパティチェーンの使用」を参照してください。

- バインディングのソースが実際に変更されていることを確認します。

  バインディングのソースがさらに大きな処理の一部である場合には、ソースを割り当てていないことを忘れがちです。

- バインド可能イベントが送出されていることを確認します。

  Flex のコマンドラインデバッガ (fdb)、または Adobe Flex Builder のデバッガを使用して、`dispatchEvent()` メソッドが呼び出されているかどうかを確認できます。また、そのクラスに通常のイベントリスナーを追加して、確実に呼び出されるようにすることもできます。イベントリスナーをタグ属性として追加するには、`[Event('myEvent')]` メタデータをクラス定義の先頭に配置するか、MXML の `<mx:Metadata>` タグに配置する必要があります。

- setter 関数を作成し、`<mx:Binding>` タグを使用して割り当てます。

  そして、割り当てられる値を指定して、trace や alert などのデバッグコードを setter に配置することができます。この方法により、バインディング自体の正常な動作が保証されます。setter が適切な情報で呼び出されていれば、失敗した場合にバインディング先に問題があると判断できるので、そのバインディング先からデバッグを開始できます。

バインディングを使用した関連するデータの移動

アプリケーションの多くは、関連するデータをオブジェクト間で移動するためにメッセージを伝達する手段が必要です。メッセージングオブジェクトは、アプリケーション内で関連データを移動するために使用するものであり、メッセージングオブジェクトに対してバインディングを使用できます。

たとえば、図書目録カード検索システムを構築し、ユーザーが書籍データベースを検索できるようにするとします。この場合、最初に行う作業は、ユーザーに検索フォームを用意することです。同時に、ActionScript クラスに単純な Query オブジェクトを作成します。Query オブジェクトには、著者、タイトル、テーマのフィールドが含まれます。Query オブジェクトで保持するのは型指定されたデータだけなので、多数の機能は必要ありません。

ユーザーが作成するクエリは一度に 1 つだけなので、Query オブジェクトを MXML コンポーネント内の ActionScript のカスタムコンポーネントとして宣言します。続いて、次の例のように、検索フォームエレメントを Query オブジェクトのプロパティにバインドします。

```
<?xml version="1.0"?>
<!-- QueryForm.mxml -->

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:library="*">
  <library:Query id="myQuery">
    <library:author>{authorInput.text}</library:author>
    <library:title>{titleInput.text}</library:title>
    <library:subject>{subjectInput.text}</library:subject>
  </library:Query>

  <mx:Form>
    <mx:FormItem label="Author">
      <mx:TextInput id="authorInput"/>
    </mx:FormItem>
  ...
  </mx:Form>
    <mx:Button label="Submit Query"
  click="queryBizDelegate.sendQuery(myQuery)"/>
</mx:VBox>
```

また、必要に応じて、検証機能や他のユーザーインターフェイスロジックをフォームに含めることもできます。重要なのは、sendQuery() メソッドのパラメータを知らなくても、バインディングにより Query オブジェクトに入力してフォームに渡すことができるということです。

渡されたフォームに対して、書籍のクエリサービスから、複数の書籍を含む結果を返したとします。別のロジックによって結果ページに移動でき、結果ページにある DataGrid コントロールによって返された書籍をすべて表示できます。サービスは、検索した書籍を既知の場所にドロップするだけで、結果を表示するビューについて知る必要はありません。サービス結果リスナーの例を次に示します。

```
public function queryResult(event):void {
  queryBizDelegate.acceptResult(event.result);
}
```

queryBizDelegate.acceptResult() メソッドは、queryResults プロパティなど、わかりやすい場所に結果を格納します。次の例に示すように、queryResults プロパティを DataGrid コントロールの dataProvider プロパティにバインドすると、新しい結果が生成されるたびにビューが自動的に更新されます。

```
<mx:DataGrid dataProvider="{queryBizDelegate.queryResults}"/>
```

第 39 章

データの格納

39

このトピックでは、アプリケーションにデータを格納するためのデータモデル機能について説明します。

目次

データモデルについて

" データモデル " とは、アプリケーション固有のデータを格納するためのプロパティを持つ ActionScript オブジェクトです。Adobe Flex アプリケーションとサーバーの間の通信が必要になるのは、Flex アプリケーションが入手できないデータを取得するときと、Flex アプリケーションからの新しいデータでサーバーサイドのデータソースを更新するときだけです。データモデルを使用して、Flex アプリケーションからサーバーへ送信する前のデータを格納したり、サーバーから送られてきたデータをアプリケーションで使用するまで格納したりします。

データモデルはデータ検証に使用できます。また、データモデルにクライアントサイドのビジネスロジックを含めることができます。データモデルは、MXML または ActionScript で定義します。MVC (Model-View-Controller : モデルビューコントローラ) デザインパターンにおいて、データモデルはモデル層を表します。

アプリケーションの計画を立てる際に、アプリケーションで格納する必要のあるデータの種類や、データの操作方法を決定します。これにより、必要なデータモデルの種類が決まります。たとえば、従業員のデータを格納するアプリケーションを作成するとします。単純な従業員モデルには、名前、部署、電子メールアドレスなどのプロパティを含めます。

データモデルの定義

データモデルの定義は、MXML タグ、ActionScript 関数、または ActionScript クラスで行うことができます。一般に、簡単なデータ構造には MXML ベースのモデルを使用し、複雑なデータ構造やクライアントサイドのビジネスロジックには ActionScript を使用します。

メモ

<mx:Model> および <mx:XML> タグは、Flex コンパイラタグであり、ActionScript クラスに直接的には対応しません。これらを含めたコンパイラタグの詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。『Adobe Flex 2 リファレンスガイド』で、メインページの「付録」リンクをクリックしてください。

<mx:Model> タグまたは <mx:XML> タグは、Flex アプリケーションのファイル、または MXML コンポーネントのファイルに配置することができます。このタグには、id の値を設定する必要があります。また、MXML コンポーネントのルートタグとすることはできません。

<mx:Model> タグ

MXML ベースのモデルの最も一般的なタイプは <mx:Model> タグで、mx.utils.ObjectProxy 型の ActionScript オブジェクトにコンパイルされます。データが型情報を含まない階層内のデータの場合、これにはオブジェクトのツリーが含まれます。オブジェクトツリーのリーフはスカラー値です。<mx:Model> タグで定義されたモデルに型情報やビジネスロジックが含まれない場合は、ごく単純な事例でのみ使用してください。型付けのあるプロパティが必要な場合やビジネスロジックを追加する場合は、ActionScript クラスでモデルを定義します。

従業員モデルが <mx:Model> タグで宣言されている例を次に示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Models\ModelTag.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Model id="employeemodel">
        <employee>
            <name>
                <first/>
                <last/>
            </name>
            <department/>
            <email/>
        </employee>
    </mx:Model>
</mx:Application>
```

<mx:Model> の子タグに値が指定されていない場合は、null と判断されます。代わりに空のストリングを指定する場合は、タグの値としてバインディング式を使用します。次に例を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Models\ModelTagEmptyString.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
```

```
    <mx:Model id="employeemodel">
        <employee>
            <name>
                <!--Fill the first property with empty string.-->
                <first>{""}</first>
                <!--Fill the last property with empty string.-->
                <last>{""}</last>
            </name>
            <!--department is null-->
            <department/>
            <!--email is null-->
            <email/>
        </employee>
    </mx:Model>
</mx:Application>
```

ActionScript クラスを使用すると、<mx:Model> タグのこの制限を回避できます。

<mx:XML> タグ

<mx:XML> タグでは、リテラルな XML データを表現します。format プロパティを e4x に設定することにより、XML オブジェクトが作成され、ECMAScript for XML (E4X) 仕様 (ECMA-357 Edition 2) で定義されている強力な XML 処理規格が実装されます。後方互換性のため、format プロパティが e4x に明示的に設定されない場合は、flash.xml.XMLNode オブジェクトタイプが作成されます。

メモ

現時点では、<mx:XML> データモデル内のノードをバインディングソースとして使用することはできません。

スクリプトベースのモデル

MXML ベースのモデルを使用する代わりに、<mx:Script> タグの変数としてモデルを定義できます。次の例では、ActionScript スクリプトブロック内で定義される簡単なモデルを示しています。このモデルを <mx:Model> タグで宣言する方が簡単です。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Models\ScriptModel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Script>
        <![CDATA[
            [Bindable]
            public var myEmployee:Object={
                name:{
                    first:null,
                    last:null
                    },
                department:null,
                email:null
```

```
            };
        ]]>
    </mx:Script>
</mx:Application>
```

MXML ベースのモデルを使用せずに、スクリプトベースのモデルを使用するメリットはありません。MXML ベースのモデルのように、スクリプトベースのモデルではプロパティの型を指定できません。プロパティの型を指定するには、クラスベースのモデルを使用します。

クラスベースのモデル

ActionScript クラスをモデルとして使用する方法は、型付けされたプロパティを持つ複雑なデータ構造を格納する場合や、アプリケーションデータを使用してクライアントサイドのビジネスロジックを実行する場合に適しています。クラスベースのモデルがサーバーサイドのデータサービスに渡されるとき、そのモデルの型情報はサーバーに保持されます。

次の例では、ActionScript クラスでモデルを定義しています。このモデルは、電子商取引アプリケーションでショッピングカートのアイテムを格納するのに使用されます。また、アイテムを追加 / 削除するメソッド、アイテム数を取得するメソッド、および合計金額を取得するメソッドの形式で、ビジネスロジックも提供しています。ActionScript コンポーネントの詳細については、『Flex 2 コンポーネントの作成と拡張』を参照してください。

```
package
{

[Bindable]
public class ShoppingCart {
  public var items:Array = [];

  public var total:Number = 0;

  public var shippingCost:Number = 0;

  public function ShoppingCart() {
  }

  public function addItem(item:Object, qty:int = 1,
    index:int = 0):void {
    items.splice(index, 0, { id: item.id,
    name: item.name,
    description: item.description,
    image: item.image,
    price: item.price,
    qty: qty });
    total += parseFloat(item.price) * qty;
   }

  public function removeItemAt(index:Number):void {
```

```
      total -= parseFloat(items[index].price) * items[index].qty;
      items.splice(index, 1);
      if (getItemCount() == 0)
      shippingCost = 0;
     }

   public function getItemCount():int {
      return items.length;
   }

   public function getTotal():Number {
      return total;
   }
}
}
```

メモ

クラスベースモデルでは、認識されるすべての型のプロパティを使用できます。たとえば、Employee クラスを作成し、Employee データを読み込む別のクラスで Employee 型のプロパティを定義できます。

次の例に示すように、クラスベースのモデルを MXML ファイルの ActionScript コンポーネントタグとして宣言します。

```
<local:ShoppingCart id="cart" xmlns:local="*"/>
```

XML 名前空間の値 * で示されているように、このコンポーネントは MXML ファイルと同じディレクトリにあります。コンポーネントの場所を指定する方法の詳細については、『Flex 2 コンポーネントの作成と拡張』を参照してください。

<mx:Model> タグまたは <mx:XML> タグでの外部ソースの指定

source プロパティを使用して、<mx:Model> または <mx:XML> タグの外部ソースを指定できます。ユーザーインターフェイスを定義する MXML とモデルのコンテンツを分離することによって、アプリケーションの保守性と再利用性が向上します。Adobe では、この方法で、静的 XML コンテンツを Flex アプリケーションに追加することを推奨します。

外部ソースファイルには、<mx:Model> タグや <mx:XML> タグの本体で定義されるモデルと同じように、静的データおよびデータバインディング式を含めることができます。source プロパティで参照されるファイルは、クライアントコンピュータではなくサーバーに存在します。コンパイラは source 値を読み取り、ソースをアプリケーションにコンパイルします。source 値は実行時には読み取られません。実行時に XML データを取得するには、<mx:HTTPService> タグを使用します。詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

<mx:Model> タグと <mx:XML> タグを外部ソースに使用すると、データモデル構造およびデータバインディング式を簡単に再利用できます。これらのタグを使用して、モデルエレメントのデータをユーザーインターフェイスコントロールにバインディングすることにより、静的データをユーザーインターフェイスコントロールにあらかじめ入力しておくこともできます。

source プロパティには、現在の Web アプリケーションディレクトリを基準とする相対ファイル名を指定でき、接頭辞として HTTP:// を含む URL も指定できます。次の例では、myEmployee1 データモデルの内容は、Web アプリケーションのローカルディレクトリに存在する content.xml という XML ファイルになります。myEmployee2 データモデルの内容は、XML を返す架空の HTTP URL です。

```
<mx:Model source="employees.xml" id="employee1"/>

<mx:Model source="http://www.somesitel.com/employees.xml" id="employee2"/>
```

ソースファイルは、単一のルートノードを持つ有効な XML ドキュメントであることが必要です。次の例は、<mx:Model source="employees.xml" id="Model1"/> タグのソースとして使用できる XML ファイルを示します。<

```
<?xml version="1.0"?>
<employees>
  <employee>
       <name>John Doe</name>
       <phone>555-777-66555</phone>
       <email>jdoe@fictitious.com</email>
       <active>true</active>
    </employee>
    <employee>
       <name>Jane Doe</name>
       <phone>555-777-66555</phone>
       <email>jndoe@fictitious.com</email>
       <active>true</active>
    </employee>
</employees>
```

データモデルでの検証の使用

データモデルに格納されたデータを検証するには、検証を使用します。次の例では、<mx:EmailValidator>、<mx:PhoneNumberValidator>、<mx:ZipCodeValidator>、<mx:SocialSecurityValidator> の各タグによって、登録データモデルの email、phone、zip、ssn の各フィールドを検証する検証を宣言しています。データモデルフィールドにバインドされる TextInput コントロールに、ユーザーが不適切なデータを入力すると、検証によってエラーメッセージが生成されます。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Models\ModelTagEmptyString.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Model id="reg">
        <registration>
            <name>{username.text}</name>
            <email>{email.text}</email>
            <phone>{phone.text}</phone>
            <zip>{zip.text}</zip>
            <ssn>{ssn.text}</ssn>
        </registration>
    </mx:Model>

    <mx:Validator required="true" source="{reg}" property="name"
        trigger="{submit}" triggerEvent="click" listener="{username}"/>
    <mx:EmailValidator source="{reg}" property="email"
        trigger="{submit}" triggerEvent="click" listener="{email}"/>
    <mx:PhoneNumberValidator source="{reg}" property="phone"
        trigger="{submit}" triggerEvent="click" listener="{phone}"/>
    <mx:ZipCodeValidator source="{reg}"
       property="zip" trigger="{submit}" triggerEvent="click" listener="{zip}"/>
    <mx:SocialSecurityValidator source="{reg}" property="ssn"
        trigger="{submit}" triggerEvent="click" listener="{ssn}"/>


    <!-- Form contains user input controls. -->
    <mx:Form>
        <mx:FormItem label="Name" required="true">
            <mx:TextInput id="username" width="200"/>
        </mx:FormItem>
        <mx:FormItem label="Email" required="true">
            <mx:TextInput id="email" width="200"/>
        </mx:FormItem>
        <mx:FormItem label="Phone" required="true">
            <mx:TextInput id="phone" width="200"/>
        </mx:FormItem>
        <mx:FormItem label="Zip" required="true">
            <mx:TextInput id="zip" width="60"/>
        </mx:FormItem>
```

```
        <mx:FormItem label="Social Security" required="true">
            <mx:TextInput id="ssn" width="200"/>
        </mx:FormItem>
        <mx:FormItem>
        <!-- User clicks Button to trigger validation. -->
            <mx:Button id="submit" label="Validate"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

この例では、ユーザーインターフェイスとアプリケーション固有のデータを明確に分離しています。登録データモデルから RPC サービス要求にデータをバインディングすることにより、簡単に拡張して 3 層アーキテクチャを作成することができます。また、1287 ページ、第 45 章の「RPC コンポーネントの使用」で説明しているように、それ自体がデータモデルである RPC サービス要求に、ユーザー入力データを直接バインドすることもできます。

検証の詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

値オブジェクトとしてのデータモデルの使用

データモデルを値オブジェクトとして使用できます。これは、オブジェクトのメソッド呼び出しで取得されるデータのセットに対する中央リポジトリとして機能します。この方法により、データの管理とアプリケーション内でのデータの操作が簡単になります。

この例では、Web サービス処理の結果が tentModel データモデルに格納されます。TentDetail コンポーネントは、データを tentModel データモデルから取得して現在選択されているテントの詳細を表示する、カスタム MXML コンポーネントです。

```
...
<!-- データモデルには、選択されたテントからのデータが格納されます。 -->
<mx:Model id="tentModel">
  <tent>
    <name>{selectedTent.name}</name>
    <sku>{selectedTent.sku}</sku>
    <capacity>{selectedTent.capacity}</capacity>
    <season>{selectedTent.seasonStr}</season>
    <type>{selectedTent.typeStr}</type>
    <floorarea>{selectedTent.floorArea}</floorarea>
    <waterproof>{getWaterProof(selectedTent.waterProof)}</waterproof>
    <weight>{getWeight(selectedTent)}</weight>
    <price>{selectedTent.price}</price>
  </tent>
</mx:Model>
...
  <TentDetail id="detail" tent="{tentModel}"/>
...
```

次の例は、TentDetail コンポーネントの MXML ソースコードを示しています。tentModel データモデルを含む tent プロパティへの参照と、対応する tentModel プロパティがボールド体で強調表示されています。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" title="Tent Details">

  <mx:Script>
    <![CDATA[
      [Bindable]
      public var tent:Object;
    ]]>
  </mx:Script>

  <mx:Style>
    .title{fontFamily:Arial;fontWeight:bold;color:#3D3D3D;fontSize:16pt;}
    .flabelColor
      {fontFamily:Arial;fontWeight:bold;color:#3D3D3D;fontSize:11pt}
    .productSpec{fontFamily:Arial;color:#5B5B5B;fontSize:10pt}
  </mx:Style>

  <mx:VBox paddingLeft="10" paddingTop="10" paddingRight="10">

    <mx:Form verticalGap="0" paddingLeft="10" paddingTop="10"
      paddingRight="10" paddingBottom="0">

      <mx:VBox width="209" height="213">
        <mx:Image width="207" height="211"
          source="./images/{tent.sku}_detail.jpg"/>
      </mx:VBox>

      <mx:FormHeading label="{tent.name}" paddingTop="1"
        styleName="title"/>

      <mx:HRule width="209"/>

      <mx:FormItem label="Capacity" styleName="flabelColor">
        <mx:Label text="{tent.capacity} person"
          styleName="productSpec"/>
      </mx:FormItem>
      <mx:FormItem label="Season"
        styleName="flabelColor">
        <mx:Label text="{tent.season}"
          styleName="productSpec"/>
      </mx:FormItem>
      <mx:FormItem label="Type" styleName="flabelColor">
        <mx:Label text="{tent.type}"
          styleName="productSpec"/>
      </mx:FormItem>
      <mx:FormItem label="Floor Area" styleName="flabelColor">
        <mx:Label text="{tent.floorarea}
          square feet" styleName="productSpec"/>
```

```
        </mx:FormItem>
        <mx:FormItem label="Weather" styleName="flabelColor">
          <mx:Label text="{tent.waterproof}"
            styleName="productSpec"/>
        </mx:FormItem>
        <mx:FormItem label="Weight" styleName="flabelColor">
          <mx:Label text="{tent.weight}"
            styleName="productSpec"/>
        </mx:FormItem>
      </mx:Form>
    </mx:VBox>
</mx:Panel>
```

XML データモデルへのデータバインディング

Flex では、<mx:XML> タグを ActionScript の xml.XMLNode オブジェクト (XML オブジェクト) のリテラル XML データにコンパイルします。これは、ActionScript オブジェクトのツリーを含む Action オブジェクトにコンパイルされる <mx:Model> タグとは異なります。データを <mx:XML> データモデルにバインドする場合は、他のデータモデルの場合と同様に、中括弧シンタックスを使用できます。ただし、データモデル内のノードをバインディング元として使用することはできません。

このタイプのバインディングに <mx:Binding> タグを使用する場合、<mx:Binding> タグの destination プロパティとして適切な ActionScript XML コマンドを記述する必要があるので、この方法は推奨しません。<mx:XML> タグの詳細については、1156 ページの「データモデルの定義」を参照してください。

次の例では、バインディング先を中括弧で囲んだ <mx:XML> データモデルを示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Models\XMLBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:XML id="myEmployee" format="e4x">
        <employee>
            <name>
                <first>{firstName.text}</first>
                <last>{lastName.text}</last>
                </name>
            <department>{department.text}</department>
            <email>{email.text}</email>
        </employee>
    </mx:XML>
    <mx:TextInput id="firstName"/>
    <mx:TextInput id="lastName"/>
    <mx:TextInput id="department"/>
    <mx:TextInput id="email"/>
</mx:Application>
```

メモ

1 つの XML を別の XML にバインドすることはできません。

第 40 章

データ検証

40

このトピックでは、アプリケーション内でデータを検証できるようにする Adobe Flex 2 のデータ検証メカニズムについて説明します。Flex には、日付、番号、通貨の値などユーザーが入力する一般的なデータタイプに対応した、事前定義済みの検証があります。

通常、Flex バリデータはデータモデルに対して使用します。データモデルの詳細については、1155 ページ、第 39 章の「データの格納」を参照してください。

目次

データ検証

ユーザーがユーザーインターフェイスに入力するデータは、アプリケーションにとって適切な内容の場合もあれば、そうでないこともあります。Flex では、オブジェクトのフィールド値が特定の条件に適合することを保証するために " バリデータ " を使用します。たとえば、ユーザーが有効な電話番号値を入力したことを確認したり、ストリング値が設定された最小値以上であることや、郵便番号フィールドに正しい桁数の数字が含まれていることを確認したりするために、バリデータを使用できます。

典型的なクライアントサーバー環境では、データがクライアントからサーバーへ送信された後に、サーバー上でデータ検証が実行されます。Flex バリデータを使用するメリットの 1 つは、バリデータをクライアント上で実行し、サーバーに送信される前に入力データを検証できる点です。Flex バリデータを使用することで、データをサーバーに送信し、後でサーバーからエラーメッセージを受信する必要がなくなるため、アプリケーション全体の応答性が向上します。

メモ

Flex バリデータを使用すると、サーバー上でデータ検証を実行する必要がなくなるわけではありませんが、一部のデータ検証をクライアント上で実行することで、パフォーマンスを向上させるメカニズムを提供します。

Flex には、郵便番号、電話番号、クレジットカード番号などの共通するタイプのユーザー入力データ用に一連のバリデータが用意されています。Flex には次のバリデータがあります。

- CreditCardValidator クラスの使用
- CurrencyValidator クラスの使用
- DateValidator クラスの使用
- EmailValidator クラスの使用
- NumberValidator クラスの使用
- PhoneNumberValidator クラスの使用
- RegExpValidator クラスの使用
- SocialSecurityValidator クラスの使用
- StringValidator クラスの使用
- ZipCodeValidator クラスの使用

バリデータについて

バリデータは MXML または ActionScript を使用して定義します。MXML 内でバリデータを宣言するには、<mx:Validator> タグか、適切なバリデータタイプのタグを使用します。たとえば、標準バリデータ PhoneNumberValidator を宣言する場合は、<mx:PhoneNumberValidator> タグを使用します。次に例を示します。

```
<?xml version="1.0"?>
<!-- validators\PNValidator.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define the PhoneNumberValidator. -->
    <mx:PhoneNumberValidator id="pnV"
        source="{phoneInput}" property="text"/>

    <!-- Define the TextInput control for entering the phone number. -->
    <mx:TextInput id="phoneInput"/>
    <mx:TextInput id="zipCodeInput"/>
</mx:Application>
```

前の例では、電話番号用の TextInput コントロールに値を入力します。郵便番号用の TextInput コントロールを選択し、直前の TextInput コントロールからフォーカスを移動すると、バリデータが実行されます。

バリデータの source プロパティを使ってオブジェクトを指定し、property プロパティを使って検証するオブジェクトのフィールドを指定します。source および property プロパティの詳細については、1168 ページの「source プロパティと property プロパティについて」を参照してください。

Validator タグは、MXML ファイルのルートタグ直下の子としてのみ指定できます。前の例では、バリデータによって、ユーザーが TextInput コントロールに有効な電話番号を入力したことを確認します。有効な電話番号には、少なくとも 10 桁の番号と、追加の書式文字が含まれます。詳細については、1201 ページの「PhoneNumberValidator クラスの使用」を参照してください。

次の例に示すように、バリデータは MXML ファイル内のスクリプトブロックまたは ActionScript ファイル内のいずれかで、ActionScript を使用して宣言します。

```
<?xml version="1.0"?>
<!-- validators\PNValidatorAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // Import PhoneNumberValidator.
            import mx.validators.PhoneNumberValidator;

            // Create the validator.
            private var v:PhoneNumberValidator = new PhoneNumberValidator();

            private function createValidator():void {
```

```
                // Configure the validator.
                v.source = phoneInput;
                v.property = "text";
            }
        ]]>
    </mx:Script>

    <!-- Define the TextInput control for entering the phone number. -->
    <mx:TextInput id="phoneInput" creationComplete="createValidator();"/>
    <mx:TextInput id="zipCodeInput"/>
</mx:Application>
```

source プロパティと property プロパティについて

バリデータは、次の 2 つのプロパティを使用して検証するアイテムを指定します。

source　検証するプロパティを含むオブジェクトを指定します。このプロパティには、コンポーネントまたはデータモデルのインスタンスを設定します。source プロパティに値を指定するには、MXML 内でデータバインディングシンタックスを使用します。ネストされたプロパティを指定するために、ドット区切りのストリングがサポートされています。

property　検証する値を含む source のプロパティ名を指定するストリングです。

これらのプロパティは、次のいずれかの方法で設定できます。

- バリデータタグを使用するときは MXML 内で設定する。
- ActionScript 内でプロパティに値を割り当てることで設定する。
- プログラムを使用してバリデータを呼び出すために `Validator.validate()` メソッドを呼び出して設定する。詳細については、1175 ページの「プログラムによる検証のトリガ」を参照してください。

通常は、次の例に示すように、source プロパティの値として Flex ユーザーインターフェイスコントロールを指定し、property プロパティの値として検証するコントロールのプロパティを指定します。

```
<?xml version="1.0"?>
<!-- validators\ZCValidator.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ZipCodeValidator id="zipV"
        source="{myZip}"
        property="text"/>

    <mx:TextInput id="phoneInput"/>
    <mx:TextInput id="myZip"/>
</mx:Application>
```

この例では、Flex の ZipCodeValidator を使用して、TextInput コントロールに入力されたデータを検証します。TextInput コントロールは、text プロパティに入力データを格納します。

検証のトリガについて

検証は、イベントに応答して自動的にトリガされるようにするか、バリデータの `Validator.validate()` メソッドを明示的に呼び出してプログラムからトリガします。

イベントを使用する場合は、ユーザーアクションに応答して自動的にバリデータを実行するように設定できます。たとえば、Button コントロールの `click` イベントを使用してフォームのフィールド上で検証をトリガできます。また、TextInput コントロールの `valueCommit` イベントを使用してユーザーが情報をコントロールに入力した後に検証をトリガすることもできます。詳細については、1170 ページの「イベントを使用した検証のトリガ」を参照してください。

プログラムから検証をトリガすることもできます。たとえば、1 つの検証を実行するために、複数の関連する入力フィールドを調べなければならない場合があります。また、ユーザーの入力に基づいて条件付きで検証を実行しなければならない場合もあります。たとえば、米ドルやユーロなど、支払いに使用する通貨をユーザーが選択できるようにすることができます。そのためには、特定の通貨に設定されたバリデータを呼び出す必要があります。この場合、入力値に適したバリデータをトリガする `validate()` メソッドを明示的に呼び出すように指定できます。詳細については、1175 ページの「プログラムによる検証のトリガ」を参照してください。

必須フィールドの検証について

Flex バリデータは、ユーザーがユーザーインターフェイスコントロールに不正な値を入力したときにそのことを判別できます。すべてのバリデータが `required` プロパティをサポートしています。このプロパティが `true` の場合、ユーザーインターフェイスコントロールで値の欠落や空の値があると、検証エラーが発生します。デフォルト値は `true` です。したがって、バリデータに関連付けられたコントロールにユーザーが 1 つでも必須データを入力できなかった場合は、デフォルトで検証エラーが発生します。このチェックを無効にするには、`required` プロパティを `false` に設定します。詳細については、1183 ページの「必須フィールドの検証」を参照してください。

検証エラーについて

デフォルトでは、検証エラーが発生すると、障害に関連するコンポーネントの周りに赤いボックスが描画されます。マウスポインタをそのコンポーネントの上に重ねると、エラーに関連付けられたエラーメッセージが表示されます。コンポーネントの外観とエラーに関連付けられるエラーメッセージは、カスタマイズ可能です。検証エラーの詳細については、1188 ページの「検証エラーの操作」を参照してください。

検証イベントについて

検証はイベント駆動型です。イベントを使用して、検証のトリガ、イベントに応答したプログラマティックなバリデータの作成と設定、バリデータにより送出されたイベントの受け取りを実行できます。

たとえば、検証操作が完了すると、バリデータは検証結果に応じて `valid` イベントまたは `invalid` イベントを送出します。これらのイベントを受け取り、その後、アプリケーションに必要な追加の処理を実行できます。

別の方法として、Flex コンポーネントが検証結果に応じて `valid` イベントまたは `invalid` イベントを送出することもあります。この場合は、バリデータによって送出されたイベントではなく、検証対象のコンポーネントから送出されたイベントを受け取ることになります。

検証イベントを受け取る必要はありません。デフォルトでは、検証が失敗した場合、データバインディングのソースに関連付けられたコントロールの周りに赤いボックスが描画されます。検証に合格すると、以前失敗したときのインジケータはクリアされます。詳細については、1192 ページの「検証イベントの操作」を参照してください。

カスタム検証について

Flex は、多数の事前定義済みのバリデータを提供していますが、独自の検証ロジックを実装する必要が生じる場合もあります。ActionScript の mx.validators.Validator クラスを継承することで、カスタム検証ロジックをカプセル化したサブクラスを作成できます。カスタムバリデータの作成の詳細については、『Flex 2 コンポーネントの作成と拡張』の第 14 章の「カスタムバリデータの作成」を参照してください。

バリデータの使用

このセクションでは、Flex バリデータを使用するための基本プロセスについて説明します。特定のバリデータクラスの詳細については、1195 ページの「標準バリデータの使用」を参照してください。

イベントを使用した検証のトリガ

バリデータをイベントに関連付けることで、バリデータを自動的にトリガできます。次の例では、ユーザーが TextInput コントロールに郵便番号を入力し、Button コントロールをクリックすると、検証がトリガされます。

```
<?xml version="1.0"?>
<!-- validators\ZCValidatorTriggerEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ZipCodeValidator id="zipV"
        source="{myZip}"
```

```
        property="text"
        trigger="{mySubmit}"
        triggerEvent="click"/>

    <mx:TextInput id="myZip"/>
    <mx:Button id="mySubmit" label="Submit"/>
</mx:Application>
```

この例では、ZipCodeValidator クラスの trigger プロパティと triggerEvent プロパティを使用して、イベントをバリデータに関連付けます。これらのプロパティには、次の値が設定されています。

trigger　バリデータをトリガするイベントを生成するコンポーネントを指定します。省略した場合、Flex のデフォルトでは source プロパティの値が使用されます。

triggerEvent　検証をトリガするイベントを指定します。省略した場合、Flex では valueCommit イベントが使用されます。コントロールの値が変化するたびに、Flex によって valueCommit イベントが送出されます。通常これは、ユーザーがコンポーネントからフォーカスを移動したときか、プロパティ値がプログラムによって変更されたときです。バリデータにすべてのイベントを無視させる場合は、triggerEvent を空のストリング ("") に設定します。

デフォルトのイベントを使用した検証のトリガ

次の例に示すように、前セクションの例を書き換えて、trigger プロパティと triggerEvent プロパティのデフォルト値を使用するように変更できます。

```
<?xml version="1.0"?>
<!-- validators\ZCValidatorDefEvent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ZipCodeValidator id="zipV"
        source="{myZip}"
        property="text"/>

    <mx:TextInput id="myZip"/>
    <mx:Button id="mySubmit" label="Submit"/>
</mx:Application>
```

trigger プロパティと triggerEvent プロパティを省略すると、TextInput コントロールから valueCommit イベントが送出されたときに、Flex によってバリデータがトリガされます。ユーザー操作またはプログラムよって値が変更されると、Flex のコントロールによって valueCommit イベントが送出されます。

データバインディングの検証のトリガ

データバインディングは、実行時に、オブジェクトのプロパティの値を自動的に別のオブジェクトのプロパティにコピーするシンタックスを提供します。データバインディングを使用すると、通常はソースの変更に応答して、ソース値が宛先にコピーされます。データバインディングのソースと宛先は、通常は Flex のコンポーネントまたはデータモデルになります。

次の例では、TextInput コントロール内のデータが自動的にデータモデルにコピーされるように、TextInput コントロールに入力されたデータをデータモデルにバインドします。

```
<?xml version="1.0"?>
<!-- validators\ValWithDataBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define a data model for storing the phone number. -->
    <mx:Model id="userInfo">
        <phoneInfo>
            <phoneNum>{phoneInput.text}</phoneNum>
        </phoneInfo>
    </mx:Model>

    <!-- Define the TextInput control for entering the phone number. -->
    <mx:TextInput id="phoneInput"/>
    <mx:TextInput id="zipCodeInput"/>
</mx:Application>
```

次の例に示すように、バリデータとデータバインディングを一緒に使用して、データバインディングのソースまたは宛先を検証できます。

```
<?xml version="1.0"?>
<!-- validators\ValTriggerWithDataBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define a data model for storing the phone number. -->
    <mx:Model id="userInfo">
        <phoneInfo>
            <phoneNum>{phoneInput.text}</phoneNum>
        </phoneInfo>
    </mx:Model>

    <!-- Define the PhoneNumberValidator. -->
    <mx:PhoneNumberValidator id="pnV"
        source="{phoneInput}"
        property="text"/>

    <!-- Define the TextInput control for entering the phone number. -->
    <mx:TextInput id="phoneInput"/>
    <mx:TextInput id="zipCodeInput"/>
</mx:Application>
```

この例では、PhoneNumberValidator を使用して、TextInput コントロールに入力されたデータを検証します。ここでは、次のことが実行されます。

- バリデータをデータバインディングのソースに割り当てます。
- TextInput コントロールでデフォルトイベント `valueCommit` を使用してバリデータをトリガします。つまり、ユーザーが郵便番号用の TextInput コントロールを選択して、直前の TextInput コントロールからフォーカスを移動したときに、バリデータが実行されます。
- ソースが変更されるたびに、データバインディングの宛先が Flex によって更新されます。つまり、`userInfo.phoneNum` フィールドは TextInput コントロールが変更されるたびに更新されますが、バリデータは、ユーザーが TextInput コントロールからフォーカスを移動し `valueCommit` イベントがトリガされたときにだけ実行されます。バリデータの `triggerEvent` プロパティを使用して、検証をトリガする異なるイベントを指定できます。

MVC (Model-View-Controller : モデルビューコントローラ) デザインパターンでは、アプリケーションのモデル部をビュー部およびコントローラ部から分離します。前の例では、データモデルがモデルを表し、バリデータがビューを表します。

TextInput コントロールは、そのデータがデータモデルにバインドされているかどうか、または何らかのバインディングがあるかどうかを一切認識しません。バリデータも TextInput コントロールに割り当てられているため、アプリケーションのモデル部とビュー部を分離したまま、その他の部分には影響を与えずに変更できます。

ただし、次の例に示すように、Flex にはバリデータのデータモデルへの割り当てを妨げるものが何もありません。

```
<?xml version="1.0"?>
<!-- validators\ValTriggerWithDataBindingOnModel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define a data model for storing the phone number. -->
    <mx:Model id="userInfo">
        <phoneInfo>
            <phoneNum>{phoneInput.text}</phoneNum>
        </phoneInfo>
    </mx:Model>

    <!-- Define the PhoneNumberValidator. -->
    <mx:PhoneNumberValidator id="pnV"
        source="{userInfo}"
        property="phoneNum"
        trigger="{phoneInput}"
        listener="{phoneInput}"/>

    <!-- Define the TextInput control for entering the phone number. -->
    <mx:TextInput id="phoneInput"/>
    <mx:TextInput id="zipCodeInput"/>
</mx:Application>
```

この例では、TextInput コントロールの valueCommit イベントを使用してデータバリデータをトリガしますが、バリデータは TextInput コントロールのプロパティではなく、データモデルのフィールドに割り当てます。

また、この例では、バリデータの listener プロパティも使用します。このプロパティは、検証のソースではなく指定されたオブジェクトに関して検証エラー情報を表示するように、バリデータを設定します。この例では、検証のソースがモデルであるため、モデルにデータを提供した TextInput コントロールのビジュアル情報が表示されます。詳細については、1191 ページの「検証のリスナーの指定」を参照してください。

モデルにネスト構造のエレメントがある場合、次の例に示すように、source プロパティにはドット区切りのストリングを使用して、検証するモデルエレメントを指定します。

```
<?xml version="1.0"?>
<!-- validators\ValTriggerWithDataBindingComplexModel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define a data model for storing the phone number. -->
    <mx:Model id="userInfo">
        <user>
            <phoneInfo>
                <homePhoneNum>{homePhoneInput.text}</homePhoneNum>
                <cellPhoneNum>{cellPhoneInput.text}</cellPhoneNum>
            </phoneInfo>
        </user>
    </mx:Model>

    <!-- Define the PhoneNumberValidator. -->
    <mx:PhoneNumberValidator id="hPNV"
        source="{userInfo.phoneInfo}"
        property="homePhoneNum"
        trigger="{homePhoneInput}"
        listener="{homePhoneInput}"/>

    <!-- Define the PhoneNumberValidator. -->
    <mx:PhoneNumberValidator id="cPNV"
        source="{userInfo.phoneInfo}"
        property="cellPhoneNum"
        trigger="{cellPhoneInput}"
        listener="{cellPhoneInput}"/>

    <!-- Define the TextInput controls for entering the phone number. -->
    <mx:Label text="Home Phone:"/>
    <mx:TextInput id="homePhoneInput"/>

    <mx:Label text="Cell Phone:"/>
    <mx:TextInput id="cellPhoneInput"/>
</mx:Application>
```

プログラムによる検証のトリガ

Validator クラスと Validator のすべてのサブクラスには、validate() メソッドが含まれています。イベントを使用して自動的にバリデータをトリガするのではなく、このメソッドを呼び出して直接バリデータを呼び出すことができます。

validate() メソッドのシグネチャは次のとおりです。

```
validate(value:Object = null,  supressEvents:Boolean =
  false):ValidationResultEvent
```

パラメータは次の値になります。

value　value が null の場合は、source プロパティと property プロパティを使用して検証するデータを指定します。value が null 以外の場合は、this キーワードを基準とするオブジェクトのフィールド、つまりドキュメントのスコープ内のオブジェクトを指定します。

また、value パラメータを指定する場合は、Validator.listener プロパティも設定する必要があります。検証が実行されると、listener プロパティで指定されたオブジェクトに対して、Flex によって視覚的な変更が適用されます。デフォルトでは、listener プロパティが source プロパティの値に設定されます。ただし、value パラメータを渡す場合には source プロパティを指定しないため、それを明示的に設定する必要があります。詳細については、1191 ページの「検証のリスナーの指定」を参照してください。

supressEvents　false の場合、完了時に valid イベントまたは invalid イベントのいずれかが送出されます。true の場合は、イベントは送出されません。

このメソッドは、検証結果を含むイベントオブジェクトを返します。このオブジェクトは、ValidationResultEvent クラスのインスタンスです。返された結果の使用の詳細については、1176 ページの「validate() メソッドの戻り値の処理」を参照してください。

次の例では、ユーザーが Button コントロールをクリックしたときに、プログラムからバリデータを作成して呼び出します。

```
<?xml version="1.0"?>
<!-- validators\ValTriggerProg.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Import ZipCodeValidator.
            import mx.validators.ZipCodeValidator;

            private var v:ZipCodeValidator = new ZipCodeValidator();

            private function performValidation():void {
                v.domain = "US or Canada";
                // Set the listener property to the component
                // used to display validation errors.
```

```
                v.listener=myZip;
                v.validate(myZip.text);
            }
        ]]>
    </mx:Script>

    <mx:TextInput id="myZip"/>
    <mx:Button label="Submit" click="performValidation();"/>
</mx:Application>
```

引き続きイベントを使用して、バリデータを作成して呼び出す performValidation() 関数をトリガしていますが、イベントそのものによってバリデータが自動的に呼び出されるわけではありません。

イベントを使用して検証を直接トリガした場合と同様に、バリデータのエラーは関連付けられたコンポーネント上に表示されます。

validate() メソッドの戻り値の処理

validate() メソッドの戻り値を調べ、検証が成功したか失敗したかに応じてアクションを実行することができます。validate() メソッドは、ValidationResultEvent クラスによって定義されたタイプのイベントオブジェクトを返します。

ValidationResultEvent クラスは、type プロパティを含むいくつかのプロパティを定義します。イベントオブジェクトの type プロパティには、検証結果に基づいて、ValidationResultEvent.VALID または ValidationResultEvent.INVALID のいずれかが含まれます。次の例に示すように、このプロパティは検証ロジックの一部として使用できます。

```
<?xml version="1.0"?>
<!-- validators\ValTriggerProgProcessReturnResult.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Import the ValidationResultEvent class.
            import mx.events.ValidationResultEvent;
            import mx.validators.ZipCodeValidator;

            public var v:ZipCodeValidator = new ZipCodeValidator();

            // Define variable for storing the validation event object.
            public var vResult:ValidationResultEvent;

            public function performValidation():void {
                v.domain = "US or Canada";
                v.listener=myZip;
                vResult = v.validate(myZip.text);

                if (vResult.type==ValidationResultEvent.VALID) {
                    // Validation succeeded.
                    myTA.text='OK';
```

```
                }
                else {
                    // Validation failed.
                    myTA.text='Fail';
                }
            }
        ]]>
    </mx:Script>

    <mx:TextInput id="myZip"/>
    <mx:Button label="Submit" click="performValidation();"/>

    <mx:TextArea id="myTA"/>
</mx:Application>
```

ValidationResultEvent クラスには、検証イベントを処理するときに使用できる追加のプロパティがあります。詳細については、1192 ページの「検証イベントの操作」を参照してください。

DateValidator および CreditCardValidator のトリガ

DateValidator および CreditCardValidator は、単一のバリデータで複数のフィールドを検証できます。CreditCardValidator は、クレジットカード番号を含む 1 番目のフィールドと、クレジットカードのタイプを含む 2 番目のフィールドを確認します。DateValidator は、日付を含む単一のフィールドを確認したり、日付を構成する複数のフィールドを確認したりできます。

独立して設定された複数のプロパティを含むオブジェクトを検証する場合、通常は、検証に必要なすべての情報が単一のフィールドに含まれているとはかぎらないため、イベントを使用してバリデータを自動的にトリガできません。

複雑なオブジェクトを検証するには、ユーザーによる何らかの種類の対話操作に基づいてバリデータの `validate()` メソッドを呼び出す方法があります。たとえば、次の例に示すように、Button コントロールの `click` イベントに応答して、日付を構成する複数のフィールドを検証することができます。

```
<?xml version="1.0"?>
<!-- validators\DateAndCC.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define the data model. -->
    <mx:Model id="date">
        <dateInfo>
            <month>{monthInput.text}</month>
            <day>{dayInput.text}</day>
            <year>{yearInput.text}</year>
        </dateInfo>
    </mx:Model>

    <!-- Define the validators. -->
    <mx:DateValidator id="dayV"
        triggerEvent=""
        daySource="{dayInput}"
        dayProperty="text"
```

```
        monthSource="{monthInput}"
        monthProperty="text"
        yearSource="{yearInput}"
        yearProperty="text"/>

    <!-- Define the form to populate the model. -->
    <mx:Form>
        <mx:TextInput id="monthInput"/>
        <mx:TextInput id="dayInput"/>
        <mx:TextInput id="yearInput"/>
    </mx:Form>

    <!-- Define the button to trigger validation. -->
    <mx:Button label="Submit"
        click="dayV.validate();"/>
</mx:Application>
```

この例では、バリデータは 3 つの入力フィールドをすべて調べます。いずれかのフィールドが無効な場合、検証は失敗します。バリデータは、失敗した無効なフィールドだけを強調表示します。バリデータが検証エラーを通知する方法の詳細については、1188 ページの「検証エラーの操作」を参照してください。DateValidator および CreditCardValidator の詳細については、1195 ページの「標準バリデータの使用」を参照してください。

1 つの関数内での複数のバリデータの呼び出し

プログラムを使用して、1 つの関数から複数のバリデータを呼び出すことができます。この例では、ZipCodeValidator バリデータと PhoneNumberValidator バリデータを使用して、データモデルへの郵便番号と電話番号の入力を検証します。

```
<?xml version="1.0"?>
<!-- validators\ValidatorCustomFuncStaticVals.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Import event class.
            import mx.events.ValidationResultEvent;

            // Define variable for storing the validation event object.
            private var vResult:ValidationResultEvent;

            private function validateZipPhone():void {
                // Validate the ZIP code.
                vResult = zipV.validate();
                // If the ZIP code is invalid,
                // do not move on to the next field.
                if (vResult.type==ValidationResultEvent.INVALID)
                    return;
```

```
                // Validate the phone number.
                vResult = pnV.validate();
                // If the phone number is invalid,
                // do not move on to the validation.
                if (vResult.type==ValidationResultEvent.INVALID)
                    return;
            }
        ]]>
    </mx:Script>

    <mx:Model id="person">
        <userInfo>
            <zipCode>{zipCodeInput.text}</zipCode>
            <phoneNumber>{phoneNumberInput.text}</phoneNumber>
        </userInfo>
    </mx:Model>

    <!-- Define the validators. -->
    <mx:ZipCodeValidator id="zipV"
        source="{zipCodeInput}"
        property="text"/>
    <mx:PhoneNumberValidator id="pnV"
        source ="{phoneNumberInput}"
        property="text"/>

    <mx:Form>
        <!-- Collect input data -->
        <mx:TextInput id="zipCodeInput"/>
        <mx:TextInput id="phoneNumberInput"/>
    </mx:Form>

    <mx:Button label="Validate"
        click="validateZipPhone();"/>
</mx:Application>
```

この例では、事前定義済みの ZipCodeValidator と PhoneNumberValidator を使用して、ユーザーが入力したユーザー情報を検証します。その後、ユーザーがフォームを送信するために [Submit] ボタンをクリックした時点で、郵便番号が指定された電話番号の市外局番に実際に適合しているかどうかを検証します。

静的メソッドの `Validator.validateAll()` を使用して、Array 内のすべてのバリデータを呼び出すこともできます。このメソッドは、失敗したバリデータごとに 1 つの ValidationResultEvent オブジェクトを格納した Array を返します。すべてのバリデータが成功した場合は空の Array を返します。次の例では、このメソッドを使用し、Button コントロールの `click` イベントに応答して 2 つのバリデータを呼び出します。

```
<?xml version="1.0"?>
<!-- validators\ValidatorMultipleValids.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initValidatorArray();">
```

```
    <mx:Script>
        <![CDATA[
            import mx.validators.Validator;

            // Define the validator Array.
            private var myValidators:Array;

            private function initValidatorArray():void {
                myValidators=[zipV, pnV];
            }
        ]]>
    </mx:Script>

    <mx:Model id="person">
        <userInfo>
            <zipCode>{zipCodeInput.text}</zipCode>
            <phoneNumber>{phoneNumberInput.text}</phoneNumber>
        </userInfo>
    </mx:Model>

    <!-- Define the validators. -->
    <mx:ZipCodeValidator id="zipV"
        source="{zipCodeInput}"
        property="text"/>
    <mx:PhoneNumberValidator id="pnV"
        source ="{phoneNumberInput}"
        property="text"/>

    <mx:Form>
        <!-- Collect input data -->
        <mx:TextInput id="zipCodeInput"/>
        <mx:TextInput id="phoneNumberInput"/>
    </mx:Form>

    <mx:Button label="Validate"
        click="Validator.validateAll(myValidators);"/>
</mx:Application>
```

再利用可能なバリデータの作成

複数のフィールドを検証するために使用できるように、再利用可能なバリデータを定義できます。再利用可能にするには、次の例に示すように、プログラムを使用して source プロパティと property プロパティを設定し検証するフィールドを指定するか、その情報を validate() メソッドに渡します。

```
<?xml version="1.0"?>
<!-- validators\ReusableVals.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import flash.events.Event;

            private function performValidation(eventObj:Event):void {
                zipV.listener=eventObj.currentTarget;
                zipV.validate(eventObj.currentTarget.text);
            }
        ]]>
    </mx:Script>

    <mx:ZipCodeValidator id="zipV"
        triggerEvent=""/>

    <mx:TextInput id="shippingZip"
        focusOut="performValidation(event);"/>
    <mx:TextInput id="billingZip"
        focusOut="performValidation(event);"/>
</mx:Application>
```

この例では、1ユーザーごとに、請求先と配送先の 2 つの住所領域があります。どちらの住所にも郵便番号フィールドがあるため、1つの ZipCodeValidator を両方のフィールドで再利用できます。focusOut イベントのイベントリスナーは、検証するフィールドを validate() メソッドに渡します。

また、次の例に示すように、performValidation() 関数を記述することもできます。

```
<?xml version="1.0"?>
<!-- validators\ReusableValsSpecifySource.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import flash.events.Event;

            private function performValidation(eventObj:Event):void {
                zipV.source = eventObj.currentTarget;
                zipV.property = "text";
                zipV.validate();
            }
```

```
        ]]>
    </mx:Script>

    <mx:ZipCodeValidator id="zipV"
        triggerEvent=""/>

    <mx:TextInput id="shippingZip"
        focusOut="performValidation(event);"/>
    <mx:TextInput id="billingZip"
        focusOut="performValidation(event);"/>
</mx:Application>
```

バリデータ実行の条件設定

プログラムを使用してバリデータを呼び出すことで、アプリケーション内で条件ロジックを使用して、複数のバリデータの中からどれを呼び出すかの決定、バリデータプロパティの設定、検証の一部としてのその他の事前処理または事後処理などを実行できます。

次の例では、Button コントロールを使用してバリデータを呼び出しますが、最初にアプリケーションがユーザー入力に基づいて、実行するバリデータを決定します。

```
<?xml version="1.0"?>
<!-- validators\ConditionalVal.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.events.ValidationResultEvent;

            private var vEvent:ValidationResultEvent;

            private function validateData():void {
                if (country.selectedValue == "Canada") {
                    vEvent = zipCN.validate(zipInput.text);
                }
                else            {
                    vEvent = zipUS.validate(zipInput.text);
                }
            }
        ]]>
    </mx:Script>

    <mx:ZipCodeValidator id="zipUS"
        domain="US Only"
        listener="{zipInput}"/>
    <mx:ZipCodeValidator id="zipCN"
        domain="US or Canada"
        listener="{zipInput}"/>
```

```
    <mx:RadioButtonGroup id="country"/>
    <mx:RadioButton groupName="country" label="US"/>
    <mx:RadioButton groupName="country" label="Canada"/>

    <mx:TextInput id="zipInput"/>

    <mx:Button label="Submit" click="validateData();"/>
</mx:Application>
```

この例では、ZipCodeValidator を使用して、TextInput コントロールに入力された郵便番号を検証します。ただし、検証を実行する前に、validateData() 関数によって、最初に郵便番号が米国またはカナダのどちらに対応しているかを判別する必要があります。この例では、ユーザーが郵便番号入力の一部として国を指定できるように、アプリケーションで RadioButton コントロールを使用します。

必須フィールドの検証

Flex のすべてのバリデータの基本クラスである Validator クラスには、required プロパティが含まれています。これが true に設定され、フィールドが空の場合、検証は失敗します。このプロパティを使用し、ユーザーが必須入力フィールドにデータを入力しなかった場合に検証が失敗するようにバリデータを設定できます。

通常は、validate() メソッドを呼び出して、必須フィールドのバリデータを呼び出します。イベントによって必ず検証がトリガされるとは保証できないため、この操作が必要となります。多くの場合、空の入力フィールドは、ユーザーが一度も入力コントロールにフォーカスを合わせなかったことを意味しています。

次の例では、必須入力フィールドの検証を実行します。

```
<?xml version="1.0"?>
<!-- validators\RequiredVal.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:StringValidator id="reqV"
        source="{inputA}"
        property="text"
        required="true"/>

    <mx:TextInput id="inputA"/>

    <mx:Button label="Submit"
        click="reqV.validate();"/>
</mx:Application>
```

この例では、次のいずれかが発生したときに StringValidator を実行します。

- TextInput コントロールによって valueCommit イベントが送出されたとき。ただし、そのイベントを送出するには、ユーザーが TextInput コントロールにフォーカスを合わせ、その後フォーカスを移動させる必要があります。ユーザーが一度も TextInput コントロールにフォーカスを合わせなかった場合、バリデータはトリガされず、Flex はコントロールが空であることを認識しません。そのために、validate() メソッドを呼び出して、バリデータが未入力のデータをチェックするように保証する必要があります。
- ユーザーが Button コントロールをクリックしたとき。TextInput コントロールにデータが入力されていない場合、バリデータで検証エラーが発生します。また、無効なストリング値が入力された場合も検証エラーが発生します。

バリデータの有効化と無効化

Validator.enabled プロパティを使用して、バリデータを有効または無効にすることができます。enabled プロパティの値が true の場合、バリデータは有効になります。値が false の場合は、バリデータは無効になります。バリデータが無効にされると、バリデータからイベントが送出されず、validate() メソッドは null を返します。

たとえば、次のコードのように、データバインディングを使用して enabled プロパティを設定することもできます。

```
<?xml version="1.0"?>
<!-- validators\EnableVal.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ZipCodeValidator id="zcVal"
        source="{inputA}"
        property="text"
        required="true"
        enabled="{enableV.selected}"/>

    <mx:TextInput id="inputA"/>
    <mx:TextInput/>
    <mx:CheckBox id="enableV"
        label="Validate input?"/>
</mx:Application>
```

この例では、ユーザーが CheckBox コントロールを選択した場合にのみバリデータを有効にします。

データバインディングを使用したバリデータの設定

アプリケーション要件に合致するようにバリデータを設定します。たとえば、StringValidator を使用して、有効なストリング長の最小値と最大値を指定できます。ストリングが有効と見なされるためには、その長さが最小文字数以上、最大文字数以下である必要があります。

多くの場合、バリデータプロパティは静的に設定されます。つまり、それらのプロパティがアプリケーションの実行時に変更されることはありません。たとえば、次の StringValidator は、入力ストリングの長さを 1 文字以上 10 文字以下として定義します。

```
<mx:StringValidator required="true" minlength="1" maxLength="10"/>
```

ユーザー入力によってバリデータのプロパティを定義することもできます。次の例では、NumberValidator の最小値と最大値をユーザーが設定できるようにします。

```
<?xml version="1.0"?>
<!-- validators\ConfigWithBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:NumberValidator
        source="{inputA}"
        property="text"
        minValue="{Number(inputMin.text)}"
        maxValue="{Number(inputMax.text)}"/>

    <mx:TextInput id="inputA"/>
    <mx:TextInput id="inputMin" text="1"/>
    <mx:TextInput id="inputMax" text="10"/>
</mx:Application>
```

この例では、データバインディングを使用してバリデータのプロパティを設定します。

検証の一般的なガイドライン

フォーム上で検証を実行するときには、いくつかのガイドラインに従う必要があります。通常、フォームはデータモデルに関連付けます。そのようにすることで、入力ユーザーインターフェイスコントロールをデータモデルのフィールドにバインディングする一部として検証をトリガできます。また、次のアクションを実行することもできます。

- 可能な場合は、バリデータをフォームの個々のユーザーインターフェイスすべてに割り当てます。実行する操作が、ユーザーが値を入力したことを確認するだけの場合も、バリデータを使用できます。
- 必要に応じて、バリデータを複数のフィールドに割り当てます。たとえば、CreditCardValidator または DateValidator を複数のフィールドに対して使用します。
- 必須フィールドがある場合は、バリデータで `validate()` メソッドを明示的に呼び出します。詳細については、1183 ページの「必須フィールドの検証」を参照してください。

■ サーバーにデータを送信する前にバリデータを呼び出すために、[Submit] ボタンを定義します。通常は、Button コントロールの click イベントを使用してプログラムからバリデータを呼び出し、すべての検証が成功したらデータを送信します。

次の例では、これらのガイドラインに従い、複数の TextInput コントロールで構成されたフォームを検証します。

```
<?xml version="1.0"?>
<!-- validators\FullApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

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

            private var vResult:ValidationResultEvent;

            // Function to validate data and submit it to the server.
            private function validateAndSubmit():void           {
                // Validate the required fields.
                vResult = fNameV.validate();
                if (vResult.type==ValidationResultEvent.INVALID)
                    return;

                vResult = lNameV.validate();
                if (vResult.type==ValidationResultEvent.INVALID)
                    return;

                // Since the date requires 3 fields, perform the validation
                // when the Submit button is clicked.
                vResult = dayV.validate();
                if (vResult.type==ValidationResultEvent.INVALID)
                    return;

                // Invoke any other validators or validation logic to make
                // an additional check before submitting the data.

                // Submit data to server.
            }
        ]]>
    </mx:Script>

    <!-- Define the data model. -->
    <mx:Model id="formInfo">
        <formData>
            <date>
                <month>{monthInput.text}</month>
                <day>{dayInput.text}</day>
                <year>{yearInput.text}</year>
            </date>
```

```
        <name>
            <firstName>{fNameInput.text}</firstName>
            <lastName>{lNameInput.text}</lastName>
        </name>
        <phoneNum>{phoneInput.text}</phoneNum>
    </formData>
</mx:Model>

<!-- Define the validators. -->
<mx:StringValidator id="fNameV"
    required="true"
    source="{fNameInput}"
    property="text"/>
<mx:StringValidator id="lNameV"
    required="true"
    source="{lNameInput}"
    property="text"/>
<mx:PhoneNumberValidator id="pnV"
    source="{phoneInput}"
    property="text"/>

<!-- Invoke the DataValidator programmatically. -->
<mx:DateValidator id="dayV"
    triggerEvent=""
    daySource="{dayInput}" dayProperty="text"
    monthSource="{monthInput}" monthProperty="text"
    yearSource="{yearInput}" yearProperty="text"/>

<!-- Define the form to populate the model. -->
<mx:Form>
    <mx:FormItem label="Month">
        <mx:TextInput id="monthInput"/>
    </mx:FormItem>
    <mx:FormItem label="Day">
        <mx:TextInput id="dayInput"/>
    </mx:FormItem>
    <mx:FormItem label="Year">
        <mx:TextInput id="yearInput"/>
    </mx:FormItem>
    <mx:FormItem label="First name">
        <mx:TextInput id="fNameInput"/>
    </mx:FormItem>
    <mx:FormItem label="Last name">
        <mx:TextInput id="lNameInput"/>
    </mx:FormItem>
    <mx:FormItem label="Phone">
        <mx:TextInput id="phoneInput"/>
    </mx:FormItem>
</mx:Form>
```

```
    <!-- Define the button to trigger validation. -->
    <mx:Button label="Submit"
        click="validateAndSubmit();"/>
</mx:Application>
```

この例では、次の処理が実行されます。

- TextInput コントロールから valueCommit イベントが送出されるたびに、関連付けられたバリデータが実行されます。
- Button コントロールの click イベントによって validateAndSubmit() が呼び出され、最終的な検証を実行してからデータがサーバーに送信されます。
- validateAndSubmit() 関数は、すべての必須フィールドについてバリデータを呼び出します。
- 日付には 3 つの異なる入力フィールドが必要であるため、validateAndSubmit() 関数は DateValidator を呼び出します。
- 最初の検証エラーが検出されると、validateAndSubmit() 関数から値は返されますが、データは送信されません。
- すべての検証が成功した場合にだけ、validateAndSubmit() 関数によってデータがサーバーに送信されます。

検証エラーの操作

Flex のユーザーインターフェイスコンポーネントを含む、UIComponent 基本クラスのサブクラスは、通常、境界線の色を変更してエラーメッセージを表示することで、検証の失敗を処理します。検証が成功した場合は、コンポーネントは既存の検証エラーメッセージを非表示にし、境界線を除去します。

このセクションでは、エラーメッセージのコンテンツの設定方法と、検証エラーの表示特性について説明します。

エラーメッセージの設定

Flex のバリデータでは、そのすべてでデフォルトのエラーメッセージが定義されています。ほとんどの場合、これらのメッセージは任意でオーバーライドできます。

すべてのバリデータのデフォルトエラーメッセージは、アプリケーションのローカライズの一部として簡単に変更できるように、リソースバインディングを使用して定義されています。バリデータクラスから作成されたすべてのバリデータオブジェクトのエラーメッセージのデフォルト値は、そのクラスに関連付けられたリソースバンドルを編集することでオーバーライドできます。

特定のバリデータオブジェクトのエラーメッセージを編集するには、バリデータのプロパティにストリング値を書き込みます。たとえば、PhoneNumberValidator では、入力された電話番号の桁数が違うことを示すデフォルトのエラーメッセージを定義します。特定の PhoneNumberValidator オブジェクトのデフォルトのエラーメッセージをオーバーライドするには、次の例に示すように、新しいメッセージストリングを wrongLengthError プロパティに割り当てます。

```
<?xml version="1.0"?>
<!-- validators\PNValidatorErrMessage.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Define the PhoneNumberValidator. -->
    <mx:PhoneNumberValidator id="pnV"
        source="{phoneInput}" property="text"
        wrongLengthError="Please enter a 10-digit number."/>

    <!-- Define the TextInput control for entering the phone number. -->
    <mx:TextInput id="phoneInput"/>
    <mx:TextInput id="zipCodeInput"/>
</mx:Application>
```

検証エラーメッセージの色の変更

デフォルトでは、マウスポインタをユーザーインターフェイスコントロール上に重ねたときに表示される検証エラーメッセージの背景色は赤です。次の例に示すように、ErrorTip スタイルを使用して色を変更できます。

```
<?xml version="1.0"?>
<!-- validators\PNValidatorErrMessageStyle.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Use blue for the error message. -->
    <mx:Style>
        .errorTip { borderColor: #0000FF}
    </mx:Style>

    <!-- Define the PhoneNumberValidator. -->
    <mx:PhoneNumberValidator id="pnV"
        source="{phoneInput}" property="text"
        wrongLengthError="Please enter a 10-digit number."/>

    <!-- Define the TextInput control for entering the phone number. -->
    <mx:TextInput id="phoneInput"/>
    <mx:TextInput id="zipCodeInput"/>
</mx:Application>
```

この例では、エラーメッセージは薄い緑色で表示されます。

errorString を使用した検証エラーの表示

UIComponent クラスで定義されている errorString プロパティを使用すると、実際にバリデータクラスを使用しなくてもコンポーネントの検証エラーを表示できます。UIComponent.errorString プロパティにストリング値を書き込むと、検証エラーを示す赤い境界線がコンポーネントの周りに描画されます。バリデータによって検証エラーが検出された場合と同様に、マウスポインタをコンポーネント上に移動すると、そのストリングが検証エラーメッセージとしてツールヒントに表示されます。

検証エラーをクリアするには、UIComponent.errorString プロパティに空のストリング ("") を書き込みます。

メモ

UIComponent.errorString プロパティに値を書き込んでも、valid イベントまたは invalid イベントはトリガされません。単に、境界線の色が変更され、検証エラーメッセージが表示されるだけです。

カスタムの ToolTip コントロールの作成に関する詳細については、857 ページ、第 23 章の「ツールヒントの使用」を参照してください。

検証エラーのクリア

errorString プロパティは、検証のソースとなるフィールドをリセットし、そのフィールドのリセット時に検証エラーが発生しないようにする場合に役立ちます。

たとえば、ユーザー入力を収集するフォームを提供するものとします。フォーム内で、ユーザーがフォームをリセットするためのボタンなどのメカニズムも提供できます。ただし、バリデータに関連付けられたフォームフィールドをクリアすると、検証エラーがトリガされます。次の例では、TextInput コントロールの text プロパティのリセットの一部として errorString プロパティを使用し、フォームのリセット時に検証エラーが発生しないようにします。

```
<?xml version="1.0"?>
<!-- validators\ResetVal.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.events.ValidationResultEvent;
            private var vResult:ValidationResultEvent;

            // Function to validate data and submit it to the server.
            private function validateAndSubmit():void {
                // Validate the required fields.
                vResult = zipV.validate();
                if (vResult.type==ValidationResultEvent.INVALID)
                    return;

                // Submit data to server.
```

```
            }

            <!-- Clear the input controls and the errorString property
                when resetting the form. -->
            private function resetForm():void {
                zipInput.text = '';
                zipInput.errorString = '';
            }
        ]]>
    </mx:Script>

    <mx:ZipCodeValidator id="zipV"
        source="{zipInput}"
        property="text"/>

    <mx:Form>
        <mx:FormItem label="Enter ZIP code">
            <mx:TextInput id="zipInput"/>
        </mx:FormItem>
        <mx:FormItem label="Enter Country">
            <mx:TextInput id="cntryInput"/>
        </mx:FormItem>
    </mx:Form>

    <!-- Trigger submit. -->
    <mx:Button label="Submit" click="validateAndSubmit();"/>

    <!-- Trigger reset. -->
    <mx:Button label="Reset" click="resetForm();"/>
</mx:Application>
```

この例では、フォームアイテムをクリアする関数が、各アイテムに関連付けらている errorString プロパティもクリアし、検証エラーをすべてクリアします。

検証のリスナーの指定

すべてのバリデータで listener プロパティがサポートされています。検証が実行されると、listener プロパティで指定されたオブジェクトに対して、Flex によって視覚的な変更が適用されます。

デフォルトでは、listener プロパティが source プロパティの値に設定されます。つまり、検証イベントに反映させるビジュアル的な変更はすべて、検証対象のコンポーネント上で実行されます。ただし、次の例に示すように、コンポーネントを検証して、その検証結果を別のコンポーネントに適用する場合があります。

```
<?xml version="1.0"?>
<!-- validators\SetListener.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ZipCodeValidator id="zipV"
```

```
        source="{zipCodeInput}"
        property="text"
        listener="{errorMsg}"/>

    <mx:TextInput id="zipCodeInput"/>
    <mx:TextArea id="errorMsg"/>
</mx:Application>
```

検証イベントの操作

Flex で検証イベントを受け取る方法は、2 通りあります。

- 検証対象のコンポーネントによって送出された検証イベントの受け取り

  Flex コンポーネントは、検証結果に応じて、valid イベントまたは invalid イベントを送出します。これによって、検証対象のコンポーネントから送出されたイベントを受け取り、その検証結果に基づいてコンポーネントに対して追加の処理を実行できます。

  イベントリスナーに渡されるイベントオブジェクトは、Event タイプです。例を含む詳細については、1192 ページの「コンポーネントの検証イベントの明示的な処理」を参照してください。

- バリデータによって送出された検証イベントの受け取り

  バリデータはすべて、検証結果に応じて、valid イベントまたは invalid イベントを送出します。これらのイベントを受け取り、その後、バリデータの必要に応じて追加の処理を実行できます。

  イベントリスナーに渡されるイベントオブジェクトは、ValidationResultEvent タイプです。例を含む詳細については、1193 ページの「バリデータの検証イベントの明示的な処理」を参照してください。

検証イベントを受け取る必要はありません。これらのイベントが発行されると、デフォルトでは、Flex は対象のコンポーネントの境界線色を適切な色に変更し、invalid イベントに対してエラーメッセージを表示するか、valid イベントに対して前のエラーメッセージを非表示にします。

コンポーネントの検証イベントの明示的な処理

検証が失敗したか成功したかによって、コンポーネントに追加の処理を実行することができます。その場合、valid イベントと invalid イベントを任意で処理することができます。次の例では、検証が失敗した場合に追加処理を実行するために、invalid イベントのイベントリスナーを定義します。

```
<?xml version="1.0"?>
<!-- validators\ValCustomEventListener -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Import event class.
```

```
            import flash.events.Event;

            // Define vars for storing text colors.
            private var errorTextColor:Object = "red";
            private var currentTextColor:Object;

            // Initialization event handler for getting default text color.
            private function myCreationComplete(eventObj:Event):void {
                currentTextColor = getStyle('color');
            }

            // For an invalid event, change the text color.
            private function handleInvalidVal(eventObject:Event):void {
                setStyle('color', errorTextColor);
            }

            // For a valid event, restore the text color.
            private function handleValidVal(eventObject:Event):void {
                setStyle('color', currentTextColor);
            }
        ]]>
    </mx:Script>

    <mx:PhoneNumberValidator source="{phoneInput}" property="text"/>

    <mx:TextInput id="phoneInput"
        initialize="myCreationComplete(event);"
        invalid="handleInvalidVal(event);"
        valid="handleValidVal(event);"/>
    <mx:TextInput id="zipInput"/>
</mx:Application>
```

バリデータの検証イベントの明示的な処理

バリデータによって送出された valid イベントおよび invalid イベントを明示的に処理するには、次の例に示すように、イベントリスナーを定義します。

```
<?xml version="1.0"?>
<!-- validators\ValEventListener.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            // Import event class
            import mx.events.ValidationResultEvent;

            private function handleValid(event:ValidationResultEvent):void {
                if(event.type==ValidationResultEvent.VALID)
```

```
                submitButton.enabled = true;
            else
                submitButton.enabled = false;
        }

        // Submit form is everything is valid.
        private function submitForm():void {
            // Handle submit.
        }

    ]]>
</mx:Script>

<mx:ZipCodeValidator
    source="{inputZip}" property="text"
    valid="handleValid(event);"
    invalid="handleValid(event);"/>

<mx:TextInput id="inputZip"/>
<mx:TextInput id="inputPn"/>

<mx:Button id="submitButton"
    label="Submit"
    enabled="false"
    click="submitForm();"/>
</mx:Application>
```

この例では、TextInput フィールドに有効な郵便番号が入力されるまで、Button コントロールは無効です。イベントオブジェクトの type プロパティは、検証結果に基づいて、ValidationResultEvent.VALID または ValidationResultEvent.INVALID のいずれかになります。

イベントリスナー内では、次のプロパティを含む、ValidationResultEvent クラスのすべてのプロパティを使用できます。

field　検証に失敗しイベントをトリガしたフィールドの名前を示すストリング。

message　検証によって作成されたすべてのバリデータエラーメッセージを含むストリング。

results　バリデータが調べたフィールドごとに 1 つずつの ValidationResult オブジェクトを含む配列。検証が成功した場合、ValidationResultEvent.results Array プロパティは空になります。検証に失敗した場合、ValidationResultEvent.results Array プロパティには、バリデータがチェックしたフィールド (検証に失敗したフィールドと合格したフィールドの両方) ごとに 1 つずつの ValidationResult オブジェクトが含まれます。ValidationResult.isError プロパティを調べれば、フィールドが検証に合格したか失敗したかを判別できます。

標準バリデータの使用

Flex には Validator サブクラスが含まれてます。それらは、一般的なデータの種類 (クレジットカードの番号、日付、電子メールアドレス、数値、電話番号、社会保障番号、ZIP コードなど) の検証に使用できるバリデータです。このセクションでは、次のバリデータについて説明します。

- CreditCardValidator クラスの使用
- CurrencyValidator クラスの使用
- DateValidator クラスの使用
- EmailValidator クラスの使用
- NumberValidator クラスの使用
- PhoneNumberValidator クラスの使用
- RegExpValidator クラスの使用
- SocialSecurityValidator クラスの使用
- StringValidator クラスの使用
- ZipCodeValidator クラスの使用

CreditCardValidator クラスの使用

CreditCardValidator クラスは、クレジットカード番号が正しい長さで、かつ正しい接頭辞で始まっていることを検証し、指定されたカードタイプ用の Luhn mod10 アルゴリズムを渡します。検証では、クレジットカードが実際に有効なクレジットカードアカウントであるかどうかはチェックされません。

通常は、`cardNumberSource` プロパティと `cardNumberProperty` プロパティを使用してクレジットカード番号の位置を指定し、`cardTypeSource` プロパティと `cardTypeProperty` プロパティを使用して検証するクレジットカードの種類の位置を指定します。

次のように、CreditCardValidator クラスは、指定されたカードの種類に対してクレジットカードの番号が適切な長さかどうかを検証します。

- Visa : 13 桁または 16 桁
- MasterCard : 16 桁
- Discover : 16 桁
- American Express : 15 桁
- DinersClub : 14 桁、MasterCard としても機能する場合は 16 桁

cardTypeProperty プロパティに定数を割り当てることで、検証するクレジットカード番号の種類を指定できます。MXML では、次の定数値が有効です。

- "American Express"
- "Diners Club"
- "Discover"
- "MasterCard"
- "Visa"

ActionScript では、cardTypeProperty プロパティを設定するために次の定数を使用できます。

- CreditCardValidatorCardType.AMERICAN_EXPRESS
- CreditCardValidatorCardType.DINERS_CLUB
- CreditCardValidatorCardType.DISCOVER
- CreditCardValidatorCardType.MASTERCARD
- CreditCardValidatorCardType.VISA

次の例では、ユーザーが指定したカードの種類に基づいてクレジットカード番号を検証します。検証エラーは Application オブジェクトに通知され、Alert ウィンドウが開きます。

```
<?xml version="1.0"?>
<!-- validators\CCExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:CreditCardValidator id="ccV"
        cardTypeSource="{cardTypeCombo.selectedItem}"
        cardTypeProperty="data"
        cardNumberSource="{cardNumberInput}"
        cardNumberProperty="text"/>

    <mx:Form id="creditCardForm">
        <mx:FormItem label="Card Type">
            <mx:ComboBox id="cardTypeCombo">
                <mx:dataProvider>
                    <mx:Object label="American Express"
                        data="American Express"/>
                    <mx:Object label="Diners Club"
                        data="Diners Club"/>
                    <mx:Object label="Discover"
                        data="Discover"/>
                    <mx:Object label="MasterCard"
                        data="MasterCard"/>
                    <mx:Object label="Visa"
                        data="Visa"/>
                </mx:dataProvider>
            </mx:ComboBox>
        </mx:FormItem>
        <mx:FormItem label="Credit Card Number">
```

```
            <mx:TextInput id="cardNumberInput"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button label="Check Credit" click="ccV.validate();"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

CurrencyValidator クラスの使用

CurrencyValidator クラスは、ストリングが通貨の表記として有効かどうかを一連のパラメータに基づいて検証します。CurrencyValidator クラスは、通貨の値の形式、負の値を許可するかどうか、値の表示する桁数を指定できるプロパティを定義します。

次の例では、CurrencyValidator クラスを使用して、米ドルとユーロで入力された通貨の値を検証します。

```
<?xml version="1.0"?>
<!-- validators\CurrencyExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Example for US currency. -->
    <mx:CurrencyValidator id="usV"
        source="{priceUS}" property="text"
        alignSymbol="left"
        trigger="{valButton}"
        triggerEvent="click"/>

    <mx:Label text="Enter a US-formatted price:"/>
    <mx:TextInput id="priceUS"/>

    <!-- Example for European currency. -->
    <mx:CurrencyValidator id="eurV"
        source="{priceEU}" property="text"
        alignSymbol="right"
        decimalSeparator="," thousandsSeparator="."
        trigger="{valButton}"
        triggerEvent="click"/>

    <mx:Label text="Enter a European-formatted price:"/>
    <mx:TextInput id="priceEU"/>

    <mx:Button id="valButton" label="Validate Currencies"/>
</mx:Application>
```

DateValidator クラスの使用

DateValidator クラスは、ストリングまたはオブジェクトが日付として有効であり、指定した形式に適合するかどうかを検証します。月、日、および年については、1 桁または 2 桁の数字で入力できます。デフォルトでは、バリデータは次の情報が提供されることを確認します。

- 月は 1 ～ 12 までです。
- 日は、1 ～ 31 までです。
- 年は、数値です。

検証する単一のストリングを指定した場合、ストリングには `allowedFormatChars` プロパティが指定する桁数と書式文字 (スラッシュ (/)、円記号 (\)、ハイフン (-)、ピリオド (.) を含む) を含めることができます。デフォルトでは、ストリングの日付の入力形式は “mm/dd/yyyy” です。ここで、“mm” は月、“dd” は日、“yyyy” は年です。`inputFormat` プロパティを使用すると、異なる形式を指定できます。

また、単一のオブジェクトによって表された日付、または異なるオブジェクトの複数のフィールドによって表された日付を検証するように指定することもできます。たとえば、日付の日、月、年の部分を表す 3 つのフィールド、または 3 つの個別のフィールドで日付を入力させる 3 つの TextInput コントロールを含むデータモデルを使用できます。日、月、年のエレメントを除外した日付形式を指定する場合でも、3 つのフィールドをすべてバリデータに指定する必要があります。

次の表は、DateValidator に日付を指定する方法を示しています。

検証のソース	必須プロパティ	デフォルトリスナー
日付を含む単一のストリング	`source` プロパティと `property` プロパティを使用して、ストリングを指定します。	Flex は、`property` プロパティによって指定されたフィールドにエラーメッセージを関連付けます。
日、月、および年を含むオブジェクトまたは複数のフィールド	`daySource`、`dayProperty`、`monthSource`、`monthProperty`、`yearSource`、`yearProperty` のプロパティをすべて使用して、日、月、年の入力値を指定します。	Flex は、検証エラーの原因となったフィールドに応じて、`daySource`、`monthSource`、または `yearSource` プロパティによって指定されたフィールドにエラーメッセージを関連付けます。

次の例では、フォームに入力された日付を検証します。

```
<?xml version="1.0"?>
<!-- validators\DateExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:DateValidator id="dateV"
        daySource="{dayInput}" dayProperty="text"
        monthSource="{monthInput}" monthProperty="text"
        yearSource="{yearInput}" yearProperty="text"/>
```

```
    <mx:Form >
        <mx:FormItem label="Month">
            <mx:TextInput id="monthInput"/>
        </mx:FormItem>
        <mx:FormItem label="Day">
            <mx:TextInput id="dayInput"/>
        </mx:FormItem>
        <mx:FormItem label="Year">
            <mx:TextInput id="yearInput"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button label="Check Date" click="dateV.validate();"/>
        </mx:FormItem>
    </mx:Form>

    <!-- Alternate method for a single field containing the date. -->
    <mx:Model id="alternateDate">
        <dateInfo>
            <date>{dateInput.text}</date>
        </dateInfo>
    </mx:Model>

    <mx:DateValidator id="stringDateV"
        source="{dateInput}" property="text"
        inputFormat="dd/mm/yyyy"
        allowedFormatChars="*#~/"/>

    <mx:Form>
        <mx:FormItem label="Date of Birth (dd/mm/yyyy)">
            <mx:TextInput id="dateInput"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button label="Check Date" click="stringDateV.validate();"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

EmailValidator クラスの使用

EmailValidator クラスは、ストリングに "@" 記号が含まれていて、ドメインにピリオド文字 "." が含まれているかどうかを検証します。IP ドメイン名も、角括弧で囲めば使用できます (例 : myname@[206.132.22.1])。0 ～ 255 の範囲の個別の IP 番号を使用できます。この検証では、ドメインとユーザー名が実在するかどうかはチェックされません。

次の例では、電子メールアドレスが正しい形式であるかどうかを検証します。

```
<?xml version="1.0"?>
<!-- validators\EmailExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form id="contactForm">
        <mx:FormItem id="homePhoneItem" label="Home Phone">
            <mx:TextInput id="homePhoneInput"/>
        </mx:FormItem>
        <mx:FormItem id="cellPhoneItem" label="Cell Phone">
            <mx:TextInput id="cellPhoneInput"/>
        </mx:FormItem>
        <mx:FormItem id="emailItem" label="Email">
            <mx:TextInput id="emailInput"/>
        </mx:FormItem>
    </mx:Form>

    <mx:PhoneNumberValidator id="pnVHome"
        source="{homePhoneInput}" property="text"/>
    <mx:PhoneNumberValidator id="pnVCell"
        source="{cellPhoneInput}" property="text"/>
    <mx:EmailValidator id="emV"
        source="{emailInput}" property="text"/>
</mx:Application>
```

NumberValidator クラスの使用

NumberValidator クラスは、ストリングが有効な数値を表しているかどうかを検証します。このバリデータは、入力値が、指定された範囲内であり (minValue プロパティと maxValue プロパティによって指定される)、整数で (domain プロパティによって指定される)、負の数でなく (allowNegative プロパティによって指定される)、指定された表示桁数を超えていないことを確認できます。NumberValidator では、12,345.67 のように形式が適用された数値を正確に検証します。thousandsSeparator プロパティと decimalSeparator プロパティは、国際化に対応してカスタマイズできます。

次の例では、NumberValidator クラスを使用して、整数が 1 ～ 10 の範囲内であることを確認します。

```
<?xml version="1.0"?>
<!-- validators\NumberExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:Form >
        <mx:FormItem
            label="Number of Widgets (max 10 per customer)">
            <mx:TextInput id="quantityInput"/>
        </mx:FormItem>
        <mx:FormItem >
            <mx:Button label="Submit"/>
        </mx:FormItem>
    </mx:Form>

    <mx:NumberValidator id="numV"
        source="{quantityInput}" property="text"
        minValue="1" maxValue="10" domain="int"/>
</mx:Application>
```

PhoneNumberValidator クラスの使用

PhoneNumberValidator クラスでは、ストリングが電話番号として有効かどうかを検証します。有効な電話番号には、少なくとも 10 桁の番号と、追加の書式文字が含まれます。この検証では、電話番号が実在するかどうかはチェックされません。

次の例では、2 つの PhoneNumberValidator タグを使用して、自宅の電話番号と携帯電話番号が正しく入力されたことを確認します。

```
<?xml version="1.0"?>
<!-- validators\PhoneExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form id="contactForm">
        <mx:FormItem id="homePhoneItem" label="Home Phone">
            <mx:TextInput id="homePhoneInput"/>
        </mx:FormItem>
        <mx:FormItem id="cellPhoneItem" label="Cell Phone">
            <mx:TextInput id="cellPhoneInput"/>
        </mx:FormItem>
        <mx:FormItem id="emailItem" label="Email">
            <mx:TextInput id="emailInput"/>
        </mx:FormItem>
    </mx:Form>

    <mx:PhoneNumberValidator id="pnVHome"
        source="{homePhoneInput}" property="text"/>
    <mx:PhoneNumberValidator id="pnVCell"
        source="{cellPhoneInput}" property="text"/>
    <mx:EmailValidator id="emV"
        source="{emailInput}" property="text"/>
</mx:Application>
```

RegExpValidator クラスの使用

RegExpValidator クラスでは、正規表現を使用してフィールドを検証できます。expression プロパティを使用して正規表現をバリデータに渡し、さらに flags プロパティを使用して正規表現のパターンマッチングを制御する追加フラグを渡します。

バリデータが検証対象のフィールドで正規表現に一致するものを検出できた場合、検証は成功です。バリデータが、一致を検出できなかった場合は、検証エラーが発生します。

このセクションでは、RegExpValidator と共に正規表現を使用する方法について説明します。正規表現の記述の詳細については、『ActionScript 3.0 のプログラミング』を参照してください。

RegExpValidator クラスは、valid イベントと invalid イベントを送出します。invalid イベントの場合、イベントオブジェクトは ValidationResultEvent クラスのインスタンスで、ValidationResult オブジェクトの Array を含んでいます。

一方、valid イベントの場合、ValidationResultEvent オブジェクトには、RegExpValidationResult オブジェクトの配列が含まれます。RegExpValidationResult クラスは ValidationResult クラスの子クラスで、次のような正規表現で使用される追加のプロパティが含まれます。

matchedIndex　一致させる入力ストリングの開始インデックスを含む整数。

matchedString　正規表現に一致させる入力ストリングのサブストリングを含むストリング。

matchedSubStrings　一致させる括弧付きのサブストリングがある場合に、そのサブストリングを含むストリングの配列。サブストリングに一致するものが見つからない場合、この Array の長さは 0 です。matchedSubStrings[0] を使用すると、最初にサブストリングが一致したものにアクセスできます。

次の例では、正規表現 ABC\d を使用します。バリデータは、文字 A、B、C が連続して続き、その後数字が続くパターンをマッチングします。

```
<?xml version="1.0"?>
<!-- validators\RegExpExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
          import mx.events.ValidationResultEvent;
          import mx.validators.*;

          private function handleResult(event:ValidationResultEvent):void {
              if (event.type == "valid")
              {
                  // For valid events, the results Array contains
                  // RegExpValidationResult objects.
                  var xResult:RegExpValidationResult;
                  myTA.text="";
                  for (var i:uint = 0; i < event.results.length; i++)
                  {
```

```
                    xResult = event.results[i];
                    myTA.text=myTA.text + xResult.matchedIndex + " " +
                        xResult.matchedString + "\n";
                }
            }
            else
            {
                // Not necessary, but if you needed to access it,
                // the results array contains ValidationResult objects.
                var result:ValidationResult;
                myTA.text="";
            }
        }
        ]]>
    </mx:Script>

    <mx:RegExpValidator id="regExpV"
        source="{exp}" property="text"
        flags="g"
        expression="{source.text}"
        valid="handleResult(event);"
        invalid="handleResult(event);"/>

    <mx:Form>
        <mx:FormItem label="Search string">
            <mx:TextInput id="exp"/>
        </mx:FormItem>
        <mx:FormItem label="Regular expression">
            <mx:TextInput id="source" text="ABC\d"/>
        </mx:FormItem>
        <mx:FormItem label="Results">
            <mx:TextArea id="myTA"/>
        </mx:FormItem>
    </mx:Form>
</mx:Application>
```

この例では、source という名前の TextInput コントロールで正規表現を指定し、それをバリデータの expression プロパティにバインドします。正規表現を変更するには、TextInput コントロールに新しい正規表現を入力します。flags プロパティの g の値は、入力フィールドで複数の一致を検索するように指定しています。

valid イベントのイベントハンドラは、入力ストリングのインデックスと、正規表現に一致するものすべてに共通するサブストリングを TextArea コントロールに書き込みます。invalid イベントハンドラは、TextArea コントロールをクリアします。

SocialSecurityValidator クラスの使用

SocialSecurityValidator クラスは、ストリングが米国の社会保障番号として有効かどうかを検証します。実在の社会保障番号であるかどうかはチェックしません。

次の例では、社会保障番号を検証します。

```
<?xml version="1.0"?>
<!-- validators\SSExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form id="identityForm">
        <mx:FormItem id="ssnItem" label="Social Security Number">
            <mx:TextInput id="ssnField"/>
        </mx:FormItem>
        <mx:FormItem id="licenseItem" label="Driver's License Number">
            <mx:TextInput id="licenseInput"/> <!-- Not validated -->
        </mx:FormItem>
    </mx:Form>

    <mx:SocialSecurityValidator id="ssV"
        source="{ssnField}" property="text"/>
</mx:Application>
```

StringValidator クラスの使用

StringValidator クラスは、ストリングの長さが指定の範囲内かどうかを検証します。次の例では、ストリングの長さが 6 ～ 12 文字の範囲であることを確認します。

```
<?xml version="1.0"?>
<!-- validators\StringExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form id="membershipForm">
        <mx:FormItem id="fullNameItem" label="Full Name">
        <!-- Not validated -->
            <mx:TextInput id="fullNameInput"/>
        </mx:FormItem>
        <mx:FormItem id="userNameItem" label="Username">
            <mx:TextInput id="userNameInput"/>
        </mx:FormItem>
    </mx:Form>

    <mx:StringValidator source="{userNameInput}" property="text"
        minLength="6" maxLength="12"/>
</mx:Application>
```

ZipCodeValidator クラスの使用

ZipCodeValidator クラスは、ストリングが 5 桁の ZIP コード、5 桁 + 4 桁の米国 ZIP コードまたはカナダ郵便番号として適切な長さかどうかを検証します。

次の例では、米国 ZIP コードまたはカナダ郵便番号を検証します。

```
<?xml version="1.0"?>
<!-- validators\ZCExample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Form id="addressForm">
        <mx:FormItem id="zipCodeItem" label="Zip Code">
            <mx:TextInput id="zipInput"/>
        </mx:FormItem>
        <mx:FormItem id="submitArea">
            <mx:Button label="Submit"/>
        </mx:FormItem>
    </mx:Form>

    <mx:ZipCodeValidator id="zipV"
        source="{zipInput}" property="text"
        domain="US or Canada"/>
</mx:Application>
```

第 41 章

41 データのフォーマット

このトピックでは、データフォーマッタの使用方法について説明します。データフォーマッタとは、生のデータをカスタマイズしたストリングにフォーマットする、ユーザー設定オブジェクトです。多くの場合、フォーマッタはデータバインディングと共に使用され、コンポーネントにバインドされた生データの有意な表示を作成します。これによりデータのフォーマット作業が自動化され、アプリケーションのフィールドの形式を簡単に変更できるようになるため、時間の節約になります。

目次

フォーマッタの使用

Adobe Flex フォーマッタは、データをストリングにフォーマットするために使用するコンポーネントです。フォーマッタは、生データからフォーマット済みストリングへの不可逆変換を実行します。通常、フォーマッタはデータがテキストフィールドに表示される直前に呼び出します。Flex には、通貨、日付、数値、電話番号、郵便番号をフォーマットできる標準フォーマッタが用意されています。

Flex フォーマッタは、すべて mx.formatters.Formatter クラスのサブクラスです。Formatter クラスでは、値を受け取ってストリング値を返す `format()` メソッドが宣言されています。

ほとんどのフォーマッタでは、エラーが発生すると空のストリングが返され、エラーを記述したストリングがフォーマッタの `error` プロパティに書き込まれます。`error` プロパティは、Formatter クラスから継承されています。

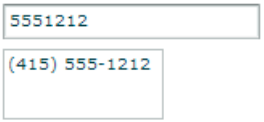
次の手順では、フォーマッタを使用する際の一般的なプロセスについて説明します。

1. MXML コード内でフォーマッタを宣言し、適切なフォーマットプロパティを指定します。
2. データをバインディングするために、中括弧 ({ }) `format()` メソッドを呼び出し、フォーマットする値をパラメータとして `format()` メソッドに指定します。

次の例では、ユーザーがアプリケーションで TextInput コントロールを使用して入力した電話番号をフォーマットします。

```
<?xml version="1.0"?>
<!-- formatters\Formatter2TextFields.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Declare a PhoneFormatter and define formatting parameters.-->
    <mx:PhoneFormatter id="phoneDisplay"
        areaCode="415"
        formatString="###-####" />

    <!-- Declare the input control.-->
    <mx:Label text="Enter 7 digit phone number (########):"/>
    <mx:TextInput id="myTI"/>

    <!-- Declare the control to display the formatted data.-->
    <mx:TextArea text="{phoneDisplay.format(myTI.text)}"/>
</mx:Application>
```

この例では、Flex PhoneFormatter クラスを使用して、フォーマットされた電話番号を TextArea コントロールに表示します。次の例は、このアプリケーションの出力を示しています。



データバインディング式の中括弧 ({ }) 内で format() メソッドを使用する必要はありません。アプリケーション内のどこからでもこのメソッドを呼び出すことができます。通常はイベントへの応答で呼び出します。次の例では、MM/DD/YYYY の日付形式で DateFormatter を宣言します。その後、Button コントロールの click イベントへの応答で、フォーマットされた日付が TextInput コントロールの text プロパティに書き込まれます。

```
<?xml version="1.0"?>
<!-- formatters\FormatterDateField.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Declare a DateFormatter and define formatting parameters.-->
    <mx:DateFormatter id="dateFormatter"
        formatString="month: MM, day: DD, year: YYYY"/>

    <mx:Label text="Enter date (mm/dd/yyyy):"/>
    <mx:TextInput id="dob" text=""/>

    <mx:Label text="Formatted date: "/>
    <mx:TextInput id="formattedDate"
        text=""
        editable="false"/>

    <!-- Format and update the date.-->
    <mx:Button label="Format Input"
        click="formattedDate.text=dateFormatter.format(dob.text);"/>
</mx:Application>
```

エラーハンドラ関数の記述

エラーを検出したときのフォーマッタのプロトコルでは、空のストリングが返され、エラー状況を記述するストリングがフォーマッタの error プロパティに書き込まれます。format() メソッドの戻り値が空のストリングかどうかをチェックできます。空のストリングが見つかった場合は、error プロパティにアクセスしてエラーの原因を確認します。

代わりに、フォーマットのエラーでエラーメッセージを返すエラーハンドラ関数を記述できます。次の例に、単純なエラーハンドラ関数を示します。

```
<?xml version="1.0"?>
<!-- formatters\FormatterSimpleErrorForDevApps.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            private function formatWithError(value:Object):String {
                var formatted:String = myFormatter.format(value);
                if (formatted == "") {
                    if (myFormatter.error != null ) {
```

```
                        if (myFormatter.error == "Invalid value") {
                            formatted = ": The value is not valid.";
                        }
                        else {
                            formatted = ": The formatString is not valid.";
                        }
                    }
                }
                return formatted;
            }
        ]]>
    </mx:Script>

    <!-- Declare a formatter and specify formatting properties.-->
    <mx:DateFormatter id="myFormatter"
        formatString="MXXXMXMXMXMXM"/>

    <!-- Trigger the formatter while populating a string with data.-->
    <mx:TextInput id="myTI"
        width="75%"
        text="Your order shipped on {formatWithError('May 23, 2005')}"/>
</mx:Application>
```

この例では、不正なフォーマットストリングで DateFormatter を定義します。次に、Date フォーマッタの format() メソッドを直接呼び出す代わりに、formatWithError() メソッドを使用して TextInput コントロールでフォーマッタを呼び出します。この例では、入力ストリングまたはフォーマットストリングのいずれかが不正である場合、formatWithError() メソッドは、フォーマットされたストリング値ではなくエラーメッセージを返します。

標準フォーマッタの使用

このセクションでは、Flex に組み込まれている標準フォーマッタについて説明します。

CurrencyFormatter クラスの使用

CurrencyFormatter クラスは、NumberFormatter クラスと同じ機能に加えて、通貨記号の機能も備えています。このクラスには追加のプロパティ、currencySymbol および alignSymbol があります。NumberFormatter クラスの詳細については、1216 ページの「NumberFormatter クラスの使用」を参照してください。

CurrencyFormatter クラスによって、小数点のフォーマット、千の桁区切りのフォーマット、負符号のフォーマットなど、数値データの基本フォーマットオプションが設定されます。format() メソッドは、数値またはストリング形式の数値を受け取り、結果のストリングをフォーマットします。

ストリング形式の数値が format() メソッドに渡されると、ストリングが左から右に解析され、最初に認識された連続する数字の抽出が試みられます。このメソッドでは、後に続く数字と共に桁区切り記号と小数点が解析されます。パーサーは、thousandsSeparatorFrom プロパティに別の文字が設定されていない限り、桁区切り記号のカンマ (,) を検索します。パーサーは、decimalSeparator プロパティに別の文字が設定されていない限り、小数点のピリオド (.) を検索します。

メモ

数値がストリング値として format() メソッドに渡されると、連続する数字の直前にダッシュ (-) がある場合、それが負符号と認識されます。ダッシュの直後にスペースがある場合は、負符号とは見なされません。

例 : CurrencyFormatter クラスの使用

次の例では、MXML ファイルで CurrencyFormatter クラスを使用します。

```
<?xml version="1.0"?>
<!-- formatters\MainCurrencyFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // Define variable to hold the price.
            [Bindable]
            private var todaysPrice:Number=4025;
        ]]>
    </mx:Script>

    <!-- Declare a CurrencyFormatter and define parameters.-->
    <mx:CurrencyFormatter id="Price" precision="2"
        rounding="none"
        decimalSeparatorTo="."
        thousandsSeparatorTo=","
        useThousandsSeparator="true"
        useNegativeSign="true"
        currencySymbol="$"
        alignSymbol="left"/>

    <!-- Trigger the formatter while populating a string with data.-->
    <mx:TextInput text="Today's price is {Price.format(todaysPrice)}."/>
</mx:Application>
```

実行時に、次のテキストが表示されます。

Today's price is $4,025.00.

エラー処理 : CurrencyFormatter クラス

エラーが発生した場合、空のストリングが返され、エラーの説明が `error` プロパティに保存されます。エラーは、送信された値に関する問題、またはユーザー設定を含むフォーマットストリングに関する問題を示します。次の表に説明を示します。

error プロパティの値	エラーが発生する状況
`Invalid value`	無効な数値が `format()` メソッドに渡された場合。値は、Number 型または String 型の有効な数値である必要があります。
`Invalid format`	1 つ以上のパラメータに、使用できない設定が含まれる場合。

DateFormatter クラスの使用

DateFormatter クラスを使用すると、さまざまな表示方法を使用して日時のデータを表示できます。`format()` メソッドは Date オブジェクトを受け取り、それをユーザー定義のパターンに基づいてストリングに変換します。また `format()` メソッドでは、ストリング形式の日付を受け取り、フォーマットする前に有効な Date オブジェクトへの解析を試みます。

DateFormatter クラスには、ストリングとしてフォーマットされた日付を受け取る `parseDateString()` メソッドがあります。`parseDateString()` メソッドは、ストリング内の数字と文字の部分を調べて、Date オブジェクトを作成します。このパーサーでは、省略しない月名と省略した (3 文字の) 月名、時刻、午前と午後、さまざまな日付の表現方法を解釈することができます。`parseDateString()` メソッドでストリングを Date オブジェクトに解析できない場合、`null` が返されます。

次に、解析できるストリングの例をいくつか示します。

```
"12/31/98" or "12-31-98" or "1998-12-31" or "12/31/1998"
"Friday, December 26, 2005 8:35 am"
"Jan. 23, 1989 11:32:25"
```

DataFormatter クラスはストリングを左から右に解析します。日付の値は必須であり、時刻の前に現れる必要があります。時刻の値は必須ではありません。時刻が定義されていない日付を持つ Date オブジェクトについては、0:0:0 がデフォルトの時刻になります。タイムゾーンオフセットは解析されません。

パターンストリングの使用

DateFormatter クラスには、パターン文字のストリングを指定します。これを解析することで、適切なフォーマットが決定されます。フォーマットのオプションと、返されるストリングのフォーマットを制御するには、パターン文字のストリングの構成方法を理解する必要があります。

パターンストリングは YYYY/MM のように特定の大文字を使用して構成します。DateFormatter パターンストリングには、パターン文字に加えてそれ以外のテキストを含めることができます。パターン文字が1つ含まれていれば、有効なパターンストリングになります。

パターンストリングを作成するときは、通常、パターン文字を繰り返します。反復される文字の数によって、表示方法が決まります。数値の場合、パターン文字の数が最小限の桁数になります。これより少ない桁の数は、この桁になるように 0 が追加されます。たとえば、4 文字のパターンストリング "YYYYY" は、4 桁の年に対応しています。

月名など、テキスト説明の対応するマッピングがある場合、パターン文字の数が 4 つ以上であれば、完全な形式が使用されます。3 つ以下であれば、短い省略形 (利用できる場合) が使用されます。たとえば、月名のパターンストリングに "MMMM" を指定した場合、フォーマッタには "Dec" のような省略形ではなく、"December" のような完全な月名が必要です。

時間の値の場合は、1 つのパターン文字が 1 桁または 2 桁に解釈されます。2 つのパターン文字は 2 桁として解釈されます。

次の表で、使用できる各パターン文字について説明します。

パターン文字	説明
Y	年です。パターン文字の数が 2 の場合、年は 2 桁に切り詰められます。それ以外の場合、4 桁で表示されます。次の例の 3 番目に示すように、指定された桁数になるように 0 が追加されます。 例 : YY = 05 YYYY = 2005 YYYYY = 02005
M	月名です。形式は、次の条件により決まります。 • パターン文字の数が 1 つの場合、形式は 1 桁または 2 桁の数値として解釈されます。 • パターン文字の数が 2 つの場合、形式は 2 桁の数値として解釈されます。 • パターン文字の数が 3 つの場合、形式は省略したテキストとして解釈されます。 • パターン文字の数が 4 つの場合、形式は省略されないテキストとして解釈されます。 例 : M = 7 MM= 07 MMM=Jul MMMM= July

パターン文字	説明
D	日付です。1文字の日付パターンストリングも有効ですが、通常は 2 文字のパターンストリングを使用します。 例 : D=4 DD=04 DD=10
E	曜日です。形式は、次の条件により決まります。 • パターン文字の数が 1 つの場合、形式は 1 桁または 2 桁の数値として解釈されます。 • パターン文字の数が 2 つの場合、形式は 2 桁の数値として解釈されます。 • パターン文字の数が 3 つの場合、形式は省略したテキストとして解釈されます。 • パターン文字の数が 4 つの場合、形式は省略されないテキストとして解釈されます。 例 : E = 1 EE = 01 EEE = Mon EEEE = Monday
A	午前 / 午後を表します。
J	時刻 (0 ～ 23) です。
H	時刻 (1 ～ 24) です。
K	午前 / 午後で区切った時刻 (0 ～ 11) です。
L	午前 / 午後で区切った時刻 (1 ～ 12) です。
N	分数です。 例 : N = 3 NN = 03
S	秒数です。 例 : SS = 30
その他のテキスト	その他のテキストをパターンストリングに追加して、ストリングをさらに設定することができます。句読点や数字、任意の小文字を使用できます。大文字はパターン文字として解釈される場合があるため、大文字は使用しないでください。 例 : EEEE, MMM.D, YYYY at H:NN A = Tuesday, Sept. 8, 2003 at 1:26 PM

次の表に、パターンストリングの例と、その結果の例を示します。

パターン	結果
YYYY.MM.DD at HH:NN:SS	2005.07.04 at 12:08:56
EEE, MMM D, 'YY	Wed, Jul 4, '05
H:NN A	12:08 PM
HH o'clock A	12 o'clock PM
K:NN A	0:08 PM
YYYYY.MMMM.DD. JJ:NN A	02005.July.04. 12:08 PM
EEE, D MMM YYYY HH:NN:SS	Wed, 4 Jul 2005 12:08:56

例 : DateFormatter クラスの使用

次の例では、MXML ファイルで DateFormatter クラスを使用して、Date オブジェクトを String としてフォーマットします。

```
<?xml version="1.0"?>
<!-- formatters\MainDateFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // Define variable to hold the date.
            [Bindable]
            private var today:Date = new Date();
        ]]>
    </mx:Script>

    <!-- Declare a DateFormatter and define parameters.-->
    <mx:DateFormatter id="DateDisplay"
        formatString="MMMM D, YYYY"/>

    <!-- Display the date in a TextArea control.-->
    <mx:TextArea id="myTA" text="{DateDisplay.format(today)}"/>
</mx:Application>
```

実行時には、TextArea コントロールにより現在の日付が表示されます。

エラー処理 : DateFormatter クラス

エラーが発生した場合、空のストリングが返され、エラーの説明が error プロパティに保存されます。エラーは、送信された値に関する問題、またはユーザー設定を含むフォーマットストリングに関する問題を示します。次の表に説明を示します。

error プロパティの値	エラーが発生する状況
Invalid value	Date オブジェクトではない値が format() メソッドに渡された場合 (空の引数は許可されます)。
Invalid format	• formatString プロパティが空 ("") に設定されている場合。 • formatString プロパティに、パターン文字が含まれていない場合。

NumberFormatter クラスの使用

NumberFormatter クラスには、小数点、桁区切り記号、負の記号を使用したフォーマットなど、数値データの基本フォーマットオプションがあります。format() メソッドは、数値またはストリング形式の数値を受け取り、結果のストリングをフォーマットします。

ストリング形式の数値が format() メソッドに渡されると、ストリングが左から右に解析され、最初に認識された連続する数字の抽出が試みられます。このメソッドでは、後に続く数字と共に桁区切り記号と小数点が解析されます。パーサーは、thousandsSeparatorFrom プロパティに別の文字が設定されていない限り、桁区切り記号のカンマ (,) を検索します。パーサーは、decimalSeparator プロパティに別の文字が設定されていない限り、小数点のピリオド (.) を検索します。

メモ

format() メソッドでは、連続する数字の直前にダッシュ (-) がある場合、それが負数として認識されます。ダッシュの直後にスペースがある場合は、負数とは見なされません。

NumberFormatter クラスの rounding および precision プロパティは、数値の小数点のフォーマットにも影響します。rounding と precision の両方のプロパティを使用した場合、まず rounding が適用され、次に precision に指定された値を使用して小数点以下の桁数が設定されます。そのため、たとえば 303.99 = 304.00 のように、数値を丸めながら、小数点以下の桁を残すことができます。

例 : NumberFormatter クラスの使用

次の例では、MXML ファイルで NumberFormatter クラスを使用します。

```
<?xml version="1.0"?>
<!-- formatters\MainNumberFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // Define variable to hold the number.
            [Bindable]
            private var bigNumber:Number = 6000000000.65;
        ]]>
    </mx:Script>

    <!-- Declare and define parameters for the NumberFormatter.-->
    <mx:NumberFormatter id="PrepForDisplay"
        precision="0"
        rounding="up"
        decimalSeparatorTo="."
        thousandsSeparatorTo=","
        useThousandsSeparator="true"
        useNegativeSign="true"/>

    <!-- Trigger the formatter while populating a string with data.-->
    <mx:TextInput text="{PrepForDisplay.format(bigNumber)}"/>
</mx:Application>
```

実行時に、次のテキストが表示されます。

6,000,000,001

エラー処理 : NumberFormatter クラス

エラーが発生した場合、空のストリングが返され、エラーの説明が `error` プロパティに保存されます。エラーは、送信された値に関する問題、またはユーザー設定を含むフォーマットストリングに関する問題を表します。次の表で詳しく説明します。

error プロパティの値	エラーが発生する状況
`Invalid value`	無効な数値が `format()` メソッドに渡された場合。値は、Number 型または String 型の有効な数値である必要があります。
`Invalid format`	1 つ以上のパラメータに、使用できない設定が含まれる場合。

PhoneFormatter クラスの使用

PhoneFormatter クラスを使用すると、エリアコードと加入者コードのフォーマットを調整して、電話番号のフォーマットを設定できます。国コードと国際フォーマットの設定を調整することもできます。PhoneFormatter.format() メソッドに渡す値は、Number オブジェクトまたは数字のみを含むストリングであることが必要です。

PhoneFormatter の formatString プロパティは、フォーマット済みストリングをフォーマットパターンの定義として受け取ります。次に示す表に、formatString の一般的な設定値を示します。PhoneFormatter の format() メソッドは、連続する数字を受け取ります。この数字は、formatString の値のプレースホルダー (#) 記号に対応します。formatString プロパティのプレースホルダー記号の数と、format() メソッドの数字の数が一致する必要があります。

formatString の値	入力	出力
###-####	1234567	(xxx) 456-7890
(###) ### ####	1234567890	(123) 456-7890
###-###-####	11234567890	123-456-7890
#(###) ### ####	11234567890	1(123) 456 7890
#-###-###-####	11234567890	1-123-456-7890
+###-###-###-####	1231234567890	+123-123-456-7890

前に示した表では、ダッシュ (-) が区切りエレメントとして使用されています。このダッシュは、ピリオド (.) やスペースに置き換えることができます。必要に応じて validPatternChars プロパティを使用することで、使用可能なデフォルトの文字セットを変更できます。numberSymbol プロパティを使用して、数値のプレースホルダーを表すデフォルトの文字を変更できます。たとえば、# を $ に変更できます。

メモ

7 桁の米国形式の電話番号については、ショートカットが用意されています。areaCode プロパティに値が含まれ、7 桁の形式のストリングを使用した場合、返されるストリングに自動的に市外局番が追加されます。エリアコードのデフォルトの形式は、(###) です。この形式は areaCodeFormat プロパティを使用して変更できます。エリアコードは、3 桁のプレースホルダーを使用する限り任意に設定できます。

例 : PhoneFormatter クラスの使用

次の例では、MXML ファイルで PhoneFormatter クラスを使用します。

```
<?xml version="1.0"?>
<!-- formatters\MainPhoneFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // Define variable to hold the phone number.
            [Bindable]
            private var newNumber:Number = 1234567;
        ]]>
    </mx:Script>

    <!-- Declare a PhoneFormatter and define formatting parameters.-->
    <mx:PhoneFormatter id="PhoneDisplay"
        areaCode="415"
        formatString="###-####"/>

    <!-- Trigger the formatter while populating a string with data-->
    <mx:TextInput id="myTI"
        initialize="myTI.text=PhoneDisplay.format(newNumber);"/>
</mx:Application>
```

実行時に、次のテキストが表示されます。

(415) 123-4567

エラー処理 : PhoneFormatter クラス

エラーが発生した場合、空のストリングが返され、エラーの説明が `error` プロパティに保存されます。エラーは、送信された値に関する問題、またはユーザー設定を含むフォーマットストリングに関する問題を示します。次の表に説明を示します。

error プロパティの値	エラーが発生する状況
`Invalid value`	• 無効な数値が `format()` メソッドに渡された場合。値は、Number 型または String 型の有効な数値である必要があります。 • 値の桁数が、format ストリングで指定されている桁数と異なります。
`Invalid format`	• `formatString` 内の文字が、`validPatternChars` プロパティで指定されている文字と一致しない場合。 • `areaCodeFormat` プロパティが指定されているのに対して、数値プレースホルダーの数が 3 つではない場合。

ZipCodeFormatter クラスの使用

ZipCodeFormatter クラスを使用すると、米国の 5 桁または 9 桁の郵便番号、またはカナダの 6 文字の郵便番号にフォーマットを設定できます。ZipCodeFormatter クラスの `formatString` プロパティは、フォーマット済みストリングをフォーマットパターンの定義として受け取ります。`formatString` プロパティはオプションです。省略した場合、デフォルト値 *#####* が使用されます。

フォーマットする値の桁数と `formatString` プロパティの値の桁数は、米国の郵便番号では 5 桁または 9 桁、カナダの郵便番号では 6 文字にする必要があります。

次の表に、一般的な `formatString` の値と、入力値、出力値を示します。

formatString の値	入力	出力	フォーマット
#####	94117, 941171234	94117, 94117	5 桁の米国郵便番号
#####-####	941171234, 94117	94117-1234, 94117-0000	9 桁の米国郵便番号
### ###	A1B2C3	A1B 2C3	6 文字のカナダ郵便番号

米国の郵便番号の場合、9 桁のフォーマットが要求されたときに 5 桁の値が入力されると、9 桁のフォーマットに合わせるために値に -0000 が付加されます。逆に、5 桁のフォーマットが要求されたときに 9 桁の値が入力されると、値が 5 桁に切り詰められます。

カナダの郵便番号の場合、`formatString` にも入力値にも 6 桁の値だけが許可されます。

メモ

米国の郵便番号では、数値のみが有効です。カナダの郵便番号では、英数字を使用できます。アルファベット文字は大文字である必要があります。

例 : ZipCodeFormatter クラスの使用

次の例では、MXML ファイルで ZipCodeFormatter クラスを使用します。

```
<?xml version="1.0"?>
<!-- formatters\MainZipFormatter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            // Define variable to hold the ZIP code.
            [Bindable]
            private var storedZipCode:Number=123456789;
        ]]>
    </mx:Script>

    <!-- Declare a ZipCodeFormatter and define parameters.-->
    <mx:ZipCodeFormatter id="ZipCodeDisplay"
        formatString="#####-####"/>

    <!-- Trigger the formatter while populating a string with data.-->
```

```
    <mx:TextInput text="{ZipCodeDisplay.format(storedZipCode)}"/>
</mx:Application>
```

実行時に、次のテキストが表示されます。

12345-6789

エラー処理 : ZipCodeFormatter クラス

エラーが発生した場合、空のストリングが返され、エラーの説明が `error` プロパティに保存されます。エラーは、送信された値に関する問題、またはユーザー設定を含むフォーマットストリングに関する問題を表します。次の表で詳しく説明します。

error プロパティの値	エラーが発生する状況
`Invalid value`	• 無効な数値が `format()` メソッドに渡された場合。値は、Number 型または String 型の有効な数値である必要があります。ただし、カナダの郵便番号の場合は英数字の値を使用できます。 • 数値の桁数が、`formatString` プロパティで指定されている桁数と一致しない場合。
`Invalid format`	• `formatString` 内の文字が、`validFormatChars` プロパティで指定されている文字と一致しない場合。 • 数値プレースホルダーの数が、9、5、または 6 ではない場合。

第 6 部

データアクセスと相互接続性

6

第VI部では、外部データを操作するための Adobe Flex 2 機能について説明します。

次のトピックが含まれます。

第 42 章

サーバーサイドデータへのアクセス 42

このトピックでは、外部データの取り扱いを可能にする Adobe Flex 2 の機能について説明します。また、それらの機能が Flex 2 SDK で利用できるのか Adobe Flex データサービスで利用できるのかについて具体的に示します。Flex データサービスは標準の J2EE プラットフォーム上およびサーブレットコンテナで動作します。

目次

Flex のデータアクセスについて

Adobe Flex 2 は、多様なデータを扱うアプリケーションを構築するための新しいメッセージングインフラストラクチャを備えています。このインフラストラクチャは、アプリケーションへのデータの移動とアプリケーションからのデータの移動を行う Flex の 3 つの機能が基になっています。これらは、RPC サービス、Data Management Service、および Message Service です。

Flex のデータアクセス機能について次の表で説明します。

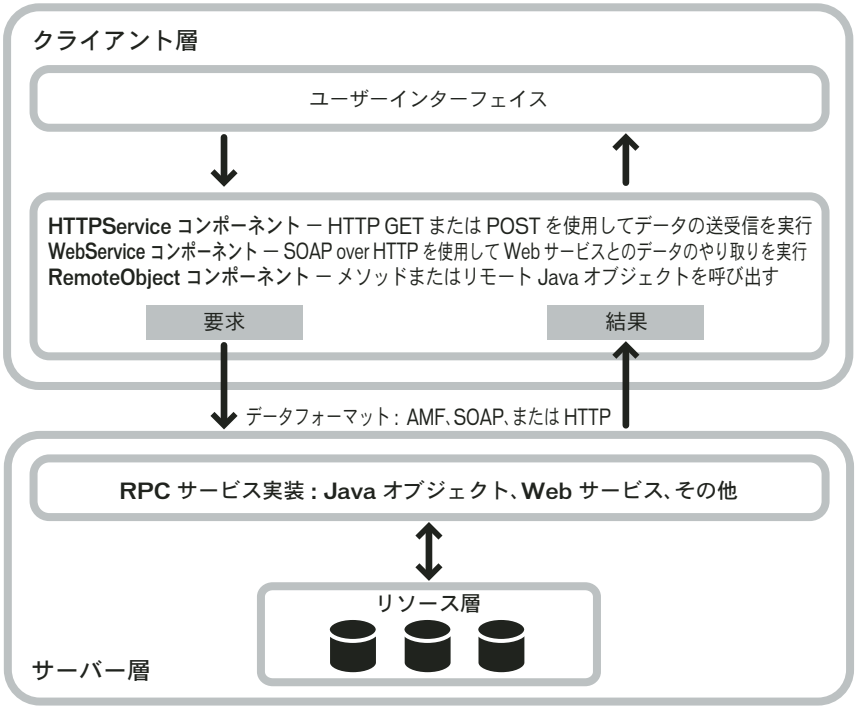
機能	説明	利用対象
RPC サービス	外部データにアクセスするための呼び出しおよび応答モデルを提供します。この機能を使用することで、リモートサービスへの非同期要求を行うアプリケーションを作成できます。リモートサービスでは要求を処理した後、データを Flex アプリケーションに返します。 Flex 2 SDK を使用する場合のみ、サービスを直接呼び出すことができますが、Flex データサービスを必要とするサーバーサイド機能を使用することはできません。 詳細については、1227 ページの「RPC サービスについて」を参照してください。	Flex 2 SDK Flex データサービス
Data Management Service	アプリケーション層間のデータの同期、リアルタイムでのデータの更新、データの複製、状況に応じて接続するアプリケーションサービス、および、アダプタを介したデータソースとの統合を実現します。この機能を使用することで、分散データを操作するアプリケーションを作成でき、大量のデータのコレクションやネストされたデータの関係 (1 対 1 や 1 対多の関係など) を管理できます。 詳細については、1229 ページの「Data Management Service について」を参照してください。	Flex データサービス
Message Service	メッセージングサービスをコラボレーションアプリケーションおよびリアルタイムアプリケーションに提供します。この機能を使用することで、Flex アプリケーションや JMS (Java Message Service) アプリケーションなどの他のアプリケーションとの間でメッセージを送受信できるアプリケーションを作成できます。 詳細については、1229 ページの「メッセージングについて」を参照してください。	Flex データサービス

RPC サービスについて

RPC サービス機能は、外部データへのアクセス手段として呼び出しおよび応答モデルが適切な選択肢となるアプリケーションを作成するためのものです。この機能を使用することで、リモートサービスへの非同期要求を行うことができます。リモートサービスでは要求を処理した後、データを直接 Flex アプリケーションに返します。

クライアントサイドの RPC コンポーネントは MXML または ActionScript で作成できます。このコンポーネントは、リモートサービスを呼び出した後、サービスから返された応答データを ActionScript オブジェクトに格納し、そこからデータを簡単に取得できるようにします。RPC サービスは HTTP URL として実装できます。HTTP URL では、HTTP の POST または GET、SOAP 準拠の Web サービス、または Java Web アプリケーション内では Java オブジェクトが使用されます。クライアントサイドの RPC コンポーネントは、HTTPService、WebService、RemoteObject の各コンポーネントです。

次の図は、RPC コンポーネントと RPC サービスとのやり取りを簡単に示したものです。



RPC コンポーネントと Flex 2 SDK のみの使用

Flex データサービスを使用せずに Flex 2 SDK を使用する場合は、サーバーサイドプロキシサービスを介さずに直接 HTTP サービスや Web サービスを呼び出す非エンタープライズアプリケーションを作成できます。Flex データサービスを使用せずに RemoteObject コンポーネントを使用することはできません。

Flash Player のデフォルトでは、アプリケーションのロードに使用されたものと完全には一致しないホストへのアクセスがブロックされます。このため、アプリケーションをホストするサーバー上で RPC サービスを実行するか、または RPC サービスをホストするリモートサーバーで "crossdomain.xml" ファイルを定義する必要があります。

"crossdomain.xml" ファイルは、サーバーのデータとドキュメントをどのドメインの SWF ファイルが利用できるかを示す XML ファイルで、特定のドメインのみを指定することも、すべてのドメインを指定することもできます。"crossdomain.xml" ファイルは、Flex アプリケーションが接続しようとするサーバーの Web ルートに置く必要があります。

詳細については、1279 ページ、第 44 章の「RPC コンポーネントについて」を参照してください。

RPC コンポーネントと Flex データサービス の使用

Flex データサービス を RPC コンポーネントと共に使用すると、さまざまなドメインからのサービストラフィックのプロキシ、クライアント認証、許可された RPC サービス URL のホワイトリスト、サーバーサイドロギング、ローカリゼーションサポート、RPC サービスの集中管理などの、各種エンタープライズ機能を提供できます。また、Flex データサービス には、リモートの Java オブジェクトにアクセスするために RemoteObject コンポーネントを SOAP 対応の Web サービスとして設定しないで使用するオプションもあります。

Flex データサービスを使用する場合は、サービスに直接接続する代わりに、RPC コンポーネントが宛先に接続します。この場合の " 宛先 " は、サーバーサイドの XML ベース設定ファイルを通じて管理することのできるサービスエンドポイントです。

詳細については、1279 ページ、第 44 章の「RPC コンポーネントについて」を参照してください。

Data Management Service について

Data Management Service は、分散データを操作するアプリケーションの作成を可能にする Flex データサービス の機能です。この機能を使用してアプリケーションを構築すると、データの同期、データの複製、および状況に応じて接続するアプリケーションサービスを実現できます。また、大量のデータのコレクションやネストされたデータの関係 (1 対 1 や 1 対多の関係など) を管理したり、データアダプタを使用してデータリソースを統合したりできます。

クライアントサイドの DataService コンポーネントは MXML または ActionScript で作成できます。このコンポーネントは、サーバーサイドの Data Management Service 宛先のメソッドを呼び出し、クライアントサイドのデータコレクションにリモートデータソースのデータを設定する、クライアント側のデータとサーバー側のデータとの同期をとる、などのアクティビティを実行します。

メッセージングについて

メッセージングは、コラボレーションアプリケーションやリアルタイムのメッセージングアプリケーションの作成を可能にする Flex データサービスの機能です。この機能は、メッセージングクライアントとの間でテキストやオブジェクトの受け渡しを行うクライアントサイド API と、メッセージ宛先のサーバーサイド設定を提供します。

メッセージングコンポーネントは MXML または ActionScript で作成できます。このコンポーネントは、サーバーサイドのメッセージング宛先に接続し、それらの宛先にメッセージを送信して、他のメッセージングクライアントからメッセージを受信します。メッセージを送信するコンポーネントのことをプロデューサと呼び、メッセージを受信するコンポーネントのことをコンシューマと呼びます。メッセージはトランスポート " チャンネル " を経由して送信されます。これらのチャンネルでは、RTMP (Realtime Message Protocol) や AMF (Action Message Format) などの特定のトランスポートプロトコルが使用されます。

メッセージングクライアントとなることができるのは、Flex アプリケーションや、JMS (Java Message Service) アプリケーションなどの他のタイプのアプリケーションです。JMS は、アプリケーションでのメッセージの作成、送受信、および読み取りを可能にする Java API です。JMS アプリケーションでは、Flex アプリケーションと同じメッセージ宛先にパブリッシュおよびサブスクライブできます。ただし、Flex のメッセージングを使用するだけでも、広範囲なメッセージングアプリケーションを作成できます。

メッセージ宛先は、サーバーサイドの XML ベース設定ファイルを通じて管理します。宛先に対してクライアント認証を設定することができます。また、メッセージングトラフィックに対してサーバーサイドロギングを設定することもできます。

Flex Builder での Flex データサービス の使用

Flex Builder では、Flex データサービス を操作する方法が 2 つあります。Flex Builder でアプリケーションをコンパイルする方法と、コンパイルしていないアプリケーション (MXML ファイル) をサーバー上の Flex データサービス Web アプリケーションに保存し、ページが表示されるときにアプリケーションをコンパイルする方法です。Flex Builder で Flex データサービス プロジェクトをコンパイルすると、そのたびにコンパイル済みのアプリケーション (SWF ファイル) がサーバー上の Flex データサービス Web アプリケーションに自動的に保存されます。

Flex Builder で Flex データサービス プロジェクトを設定するための 2 つのオプションは、Flex Builder の [ファイル] メニューから [新規]-[Flex プロジェクト] を選択すると表示される [Flex プロジェクトの作成] ダイアログボックスにあります。

Flex Builder で Flex データサービス プロジェクトを作成すると、プロジェクトと同じ名前の新規ディレクトリが作成されるか、同じ名前の既存のディレクトリが使用されます。このディレクトリは、指定したルートフォルダのサブディレクトリになります。同じ名前のデフォルトのアプリケーションファイル (MXML ファイル) も作成されます。Flex Builder のナビゲータツリーでファイルを選択して右クリックし、[デフォルトのアプリケーションに設定] をクリックすると、デフォルトのアプリケーションファイルとして別の MXML を設定することもできます。

ローカルにコンパイルするアプリケーションの場合は、MXML ファイルがローカルの Flex Builder プロジェクトディレクトリに保存され、SWF ファイルと HTML ラッパーファイルがサーバーに保存されます。完成したアプリケーションを MXML ファイルではなく、SWF ファイルおよび HTML ラッパーファイルとしてデプロイする場合は、ローカルでコンパイルする方が便利です。

サーバーでコンパイルするアプリケーションの場合は、MXML ファイルがサーバーに保存されます。アプリケーションのコンパイル時、HTML ラッパーファイルは生成および保存されません。

Flex データサービス 用に Flex Builder プロジェクトを使用するときも、Flex Builder を使用しないときと同様にサーバーサイドの Java クラスなどのリソースを管理します。Flex Builder [プロジェクトプロパティ] ダイアログの [ソースパス] パネルで、ActionScript クラスなど、クライアントサイドの追加のソースファイルの場所を指定できます。デフォルトでは、Flex Builder にはサーバーでコンパイルするアプリケーション用に WEB-INF\flex\user_classes ディレクトリが含まれています。

メモ

Flex Builder で Flex データサービス プロジェクトにどのオプションを選択するかにかかわらず、有効な Flex データサービス ルートフォルダとルート URL を指定する必要があります。これらの値では、Flex データサービス Web アプリケーションのルートをマップしてください。Flex Builder で Flex アプリケーションをコンパイルする前に、対応する Flex データサービス Web アプリケーションを起動する必要があります。

Flex Builder プロジェクトの詳細については、『Flex Builder 2 ユーザーガイド』の第 3 章の「プロジェクトの操作」を参照してください。

第 43 章

データサービスの設定

43

Adobe Flex データサービスのサービスでは、サービス固有の機能に加えて、一連の共通の機能が使用されています。このトピックでは、すべてのサービスタイプに共通する以下の機能について説明します。

- サービス設定ファイル
- メッセージチャネル
- データの直列化
- 認証および承認
- サーバーサイドのログ
- セッションデータ
- ソフトウェアクラスタ化
- 実行時のサービスの監視と管理
- カスタムのエラー処理
- クラスのロード

目次

サービス設定ファイルについて

Remoting Service、Proxy Service、Message Service、および Data Management Service は、データサービス設定ファイル services-config.xml の <services> セクションで設定します。このファイルのデフォルトの場所は、Flex データサービス Web アプリケーションの WEB-INF/flex ディレクトリです。この場所の設定は、WEB-INF/web.xml ファイルの MessageBrokerServlet の設定で行います。

services-config.xml ファイルでは、サービス定義を格納したファイルを参照によってインクルードできるので便利です。通常、Adobe では、Remoting Service、Proxy Service、Message Service、および Data Management Service が、参照によってインクルードされます。

次の表は、Adobe における設定ファイルの典型的な設定方法を示したものです。これらのファイルにコメントが記載されたものが、Flex データサービスのインストールディレクトリの resources/config サブディレクトリで参照できます。これらのファイルは独自に設定を行う際のテンプレートとして使用できます。

ファイル名	説明
services-config.xml	Flex データサービスの最上位の設定ファイル。通常、このファイルには、セキュリティ制限の定義と、各サービスで使用できるログ設定が含まれています。また、Adobe で通常行っているように、サービス定義をインラインで含めることや、サービス定義を参照によってインクルードすることが可能です。Adobe では通常、各サービスを remoting-config.xml、proxy-config.xml、messaging-config.xml、data-management-config.xml の各ファイルで定義しています。
remoting-config.xml	Remoting Service の設定ファイル。リモートオブジェクトを扱うための Remoting Service の宛先を定義します。 Remoting Service の設定の詳細については、1329 ページ、第 46 章の「RPC サービスの設定」を参照してください。
proxy-config.xml	Proxy Service の設定ファイル。Web サービスと HTTP サービス (REST サービス) を扱うための Proxy Service の宛先を定義します。 Proxy Service の設定の詳細については、1329 ページ、第 46 章の「RPC サービスの設定」を参照してください。

ファイル名	説明
messaging-config.xml	Message Service の設定ファイル。パブリッシュ / サブスクライブメッセージングを実行するための Message Service の宛先を定義します。 Messaging Service の設定の詳細については、1357 ページ、第 49 章の「Message Service の設定」を参照してください。
data-management-config.xml	Data Management Service の設定ファイル。Data Management Service の宛先を扱うための Data Management Service の宛先を定義します。 Data Management Service の設定の詳細については、1389 ページ、第 52 章の「Data Management Service の設定」を参照してください。

ファイルを参照によってインクルードする場合は、参照先のファイルの内容が、サービスの適切な XML 構造に従っている必要があります。file-path の値は services-config.xml ファイルの場所を基準とする相対位置になります。次の例は、参照によってインクルードされたサービス定義を示しています。

```
<services>
  <!-- Remoting Service -->
  <service-include file-path="remoting-config.xml"/>

  <!-- Proxy Service -->
  <service-include file-path="proxy-config.xml"/>

  <!-- Message Service -->
  <service-include file-path="messaging-config.xml"/>

  <!-- Data Management Service -->
  <service-include file-path="data-management-config.xml"/>
</services>
```

データサービス設定ファイルのシンタックス

次の表では、services-config.xml ファイルの XML エレメントについて説明します。ルートエレメントは services-config エレメントです。

XML エレメント	説明
services	個々のデータサービスの定義、またはサービスの定義を含む他の XML ファイルへの参照を格納します。Adobe では、各種の標準サービスについて別々の設定ファイルがインクルードされます。これらのサービスには、Proxy Service、Remoting Service、Message Service、および Data Management Service があります。 services エレメントは、ルートエレメント services-config の子として、設定の最上位で宣言されています。 特定のサービスタイプの設定に関する詳細については、次のトピックを参照してください。 • 1329 ページ、第 46 章の「RPC サービスの設定」 • 1357 ページ、第 49 章の「Message Service の設定」 • 1389 ページ、第 52 章の「Data Management Service の設定」
services/service-include	サービス定義の設定エレメントを格納した XML ファイルへのパスを指定します。 **属性：** file-path　サービス定義を格納した XML ファイルへのパス。
services/service	Proxy Service、Remoting Service、Message Service、Data Management Service、カスタムサービスなどのデータサービスの定義を格納します。 (オプション) service-include エレメントを使用すると、サービス定義のあるファイルを、services-config.xml ファイル内のインラインではなく参照によってインクルードできます。
services/service/properties	サービスレベルのプロパティのエレメントを格納します。
services/service/adapters	宛先で参照され、特定の種類の機能を提供する、サービスアダプタの定義を格納します。

XML エレメント	説明
services/service/adapters/adapter-definition	サービスアダプタの定義を格納します。サービスタイプごとに、そのサービスタイプに関連した一連のアダプタを独自に持ちます。たとえば、Data Management Service の宛先では、Java アダプタや ActionScript オブジェクトアダプタを使用できます。 **属性：** • id　アダプタの識別子。宛先定義の中でアダプタを参照するのに使用します。 • class　アダプタ機能を提供する Java クラスの完全修飾名。 • default　このアダプタがサービスの宛先のデフォルトアダプタかどうかを示すブール値。デフォルトアダプタは、宛先定義の中でアダプタを明示的に参照しない場合に使用されます。
services/service/default-channels	サービスのデフォルトチャネルへの参照を格納します。デフォルトチャネルは、宛先定義の中でチャネルが明示的に参照されていない場合に使用されます。チャネルは、ファイルにインクルードされた順に試行されます。あるチャネルが使用できない場合は、次のチャネルが試行されます。
services/service/default-channels/channel	チャネル定義の id への参照を格納します。 **属性：** ref　チャネル定義の id 値。
services/service/destination	宛先定義を格納します。
services/service/destination/adapter	サービスアダプタへの参照を格納します。このエレメントを省略した場合、宛先ではデフォルトアダプタが使用されます。
services/service/destination/properties	宛先のプロパティを格納します。 使用できるプロパティはサービスタイプによって異なります。サービスタイプは、指定されたサービスクラスによって決まります。
services/service/destination/channels	サービスでデータ伝達に使用できるチャネルへの参照を格納します。チャネルは、ファイルにインクルードされた順に試行されます。あるチャネルが使用できない場合は、次のチャネルが試行されます。
services/service/destination/channels/channel	チャネルの id 値への参照を格納します。 チャネルは、設定の最上位にある channels エレメントにおいて、ルートエレメント services-config の子として定義されます。

XML エレメント	説明
services/service/destination/security	セキュリティ制限の定義への参照と、ログインコマンドの定義への参照を格納します。これらは認証および承認に使用されます。 このエレメントには、最上位のセキュリティエレメントでグローバルに定義されるセキュリティ制限への参照ではなく、完全なセキュリティ制限の定義を格納することもできます。 詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。
services/service/destination/security/security-constraint	セキュリティ制限の定義の id 値への参照、またはセキュリティ制限の定義を格納します。 **属性：** ref　サービス設定の最上位にある security エレメントで定義されている security-constraint エレメントの id 値。 id　実際のセキュリティ制限をこのエレメントで定義するときの、セキュリティ制限の識別子。
services/service/destination/security/security-constraint/auth-method	セキュリティ制限で使用される認証方法を指定します。セキュリティ制限では、基本認証かカスタム認証のいずれかを使用できます。 有効な値は basic および custom です。
services/service/destination/security/security-constraint/roles	アプリケーションサーバーのユーザーストアで定義されているロールの名前を指定します。
services/service/destination/security/login-command	カスタム認証の実行に使用されるログインコマンド定義の id 値への参照を格納します。 **属性：** ref　ログインコマンド定義の id 値。 詳細については、1263 ページの「カスタム認証」を参照してください。
security	認証および承認のためのセキュリティ制限定義とログインコマンド定義を格納します。 詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。
security/security-constraint	セキュリティ制限を定義します。
security/security-constraint/auth-method	セキュリティ制限で使用される認証方法を指定します。セキュリティ制限では、基本認証かカスタム認証を使用できます。
security/security-constraint/roles	アプリケーションサーバーのユーザーストアで定義されているロールの名前を指定します。

XML エレメント	説明
security/login-command	カスタム認証に使用されるログインコマンドを定義します。 **属性：** • class　ログインコマンドクラスの完全修飾クラス名。 • server　カスタム認証が実行されるアプリケーションサーバー。 詳細については、1263 ページの「カスタム認証」を参照してください。
channels	サーバーとクライアント間のデータ伝達に使用されるメッセージチャネルの定義を格納します。詳細については、1240 ページの「メッセージチャネルの設定」を参照してください。
channels/channel-definition	データ伝達に使用できるメッセージチャネルを定義します。 **属性：** • id　チャネル定義の識別子。 • class　チャネルクラスの完全修飾クラス名。
channels/channel-definition/endpoint	チャネル定義のエンドポイント URI とエンドポイントクラスを指定します。 **属性：** • uri　エンドポイント URI。 • class　エンドポイントクラスの完全修飾クラス名。
channels/channel-definition/properties	チャネル定義のプロパティを格納します。使用できるプロパティは、指定されたチャネルのタイプによって異なります。 詳細については、1240 ページの「メッセージチャネルの設定」を参照してください。
clusters	クラスタ定義を格納します。クラスタ定義は、複数のホスト間でのソフトウェアクラスタ化を設定します。 詳細については、1268 ページの「ソフトウェアクラスタ化の使用」を参照してください。
clusters/cluster	クラスタを定義します。 **属性：** id　クラスタ定義の識別子。 properties　JGroups プロパティファイルの名前。

XML エレメント	説明
logging	サーバーサイドのログ機能の設定を格納します。詳細については、1264 ページの「サーバーサイドのサービスログ機能の設定」を参照してください。
logging/target	ログターゲットクラスとログレベルを指定します。 **属性：** • class　完全修飾されたログターゲットクラス名。 • level　ログレベル。詳細については、1264 ページの「サーバーサイドのサービスログ機能の設定」を参照してください。
logging/target/properties	以下の各行に示すログのプロパティを格納します。これらのプロパティはすべて省略可能です。
logging/target/properties/prefix	ログメッセージに含める接頭辞を指定します。
logging/target/properties/includeDate	ログメッセージに日付を含めるかどうかを示すブール値。
logging/target/properties/includeTime	ログメッセージに時刻を含めるかどうかを示すブール値。
logging/target/properties/includeLevel	ログメッセージにログレベルを含めるかどうかを示すブール値。
logging/target/properties/includeCategory	ログメッセージにログカテゴリを含めるかどうかを示すブール値。ログカテゴリは、以下で説明する filters/pattern エレメントの値に対応しています。サーバーでは、ワイルドカードパターンではなく特定のログカテゴリに対してログが記録されます。
logging/target/filters	ログメッセージ作成時に比較対象となるパターンを格納します。指定されたパターンに一致するメッセージだけがログに記録されます。
logging/target/filters/pattern	ログメッセージ作成時に比較対象となるパターン。指定されたパターンに一致するメッセージだけがログに記録されます。有効なパターン値の一覧については、1264 ページの「サーバーサイドのサービスログ機能の設定」を参照してください。
system	前のカテゴリに属さない、システム全体に影響する設定。ロケール情報のほか、再展開や監視ファイルの設定も格納します。

XML エレメント	説明
system/locale	(オプション) ロケールストリング。たとえば、"en"、"de"、"fr"、"es" などが有効なロケールストリングです。
system/default-locale	デフォルトのロケールストリング。default-locale エレメントを指定しない場合は、英語のエラーメッセージの基本セットが使用されます。
system/redeploy	設定ファイルが更新されるときの Web アプリケーションの再展開のサポート。この機能は J2EE アプリケーションサーバーでの Web アプリケーションの再展開で使用できます。touch-file の値は、Web での再展開を強制するためにアプリケーションサーバーで使用されるファイルです。touch-file で指定する値の詳細については、アプリケーションサーバーのマニュアルを参照してください。
system/redeploy/enabled	再展開が有効になっているかどうかを示すブール値。
system/redeploy/watch-interval	設定ファイルが変更されているかどうかを調べるまでの待機時間 (秒数)。
system/redeploy/watch-file	変更について監視されるデータサービス設定ファイル。watch-file の値は、{context.root} で始めるか、絶対パスにする必要があります。次の例では {context.root} を使用しています。 {context.root}/WEB-INF/flex/data-management-config.xml
system/redeploy/touch-file	Web アプリケーションの再展開を強制するためにアプリケーションサーバーで使用されるファイル。touch-file の値は、{context.root} で始めるか、絶対パスにする必要があります。touch-file で指定する値の詳細については、アプリケーションサーバーのマニュアルを参照してください。JRun では、touch-file の値は次の例のような web.xml ファイルです。 {context.root}/WEB-INF/web.xml

メッセージチャネルの設定

Flex データサービスでは、Flex メッセージングシステムの一部であるメッセージチャネルを経由して、サービスの宛先との間でメッセージが相互に伝達されます。任意のタイプのサービスと任意のタイプのチャネルを、ペアにすることができます。

Flex データサービスでは AMF、HTTP、RTMP の各チャネルと、セキュアな AMF、HTTP、RTMP の各チャネルがサポートされています。データのセキュリティを必要とする場合は、いずれかのセキュアなチャネルを使用してください。セキュアな AMF チャネルとセキュアな HTTP チャネルでは、HTTPS (HTTP over Secure Socket Layer) を使用してデータが伝達されます。セキュアな RTMP チャネルでは、TLS (RTMP over Transport Layer Security) を使用してデータが伝達されます。

チャネル定義はデータサービス設定ファイルで作成します。各チャネル定義のエンドポイント URI は一意である必要があります。

AMF エンドポイントを繰り返しポーリングするように AMF ポーリングチャネルを設定することで、クライアントプルのメッセージコンシューマを作成できます。ポーリングの発生間隔はチャネル上で設定できます。

次の例は AMF チャネル定義を示しています。チャネル定義の前には、その定義を参照する宛先が付きます。

```
...
  <destination id="MyTopic">
...
     <channels>
       <channel ref="samples-amf"/>
     </channels>
...
  </destination>
...


  <channel-definition id="samples-amf"
    type="mx.messaging.channels.AMFChannel">
    <endpoint uri="/myapp/messagebroker/amf" port="8100"
      type="flex.messaging.endpoints.AmfEndpoint"/>
  </channel-definition>
</channels>
...
```

ChannelSet オブジェクトを作成して実行時にクライアントでチャネルを割り当てることもできます。ChannelSet オブジェクトには Channel オブジェクトを格納します。次の例に示すように、ChannelSet をサービスコンポーネントに割り当てます。

```
...
// ChannelSet を作成します。
var cs:ChannelSet = new ChannelSet();
// Channel を作成します。
```

```
var customChannel:Channel = new AMFChannel("my-polling-amf", endpointUrl);
// Channel を ChannelSet に追加します。
cs.addChannel(customChannel);
// ChannelSet を RemoteObject インスタンスに割り当てます。
myRemoteObject.channelSet = cs;
...
```

チャネルを実行時に設定するときは、endpointURL を動的に制御できます。ChannelSet には Channel オブジェクトをいくつでも追加できます。ChannelSet オブジェクトではセット全体が検索され、接続可能なチャネルが見つけられます。宛先が認識する ID を指定することをお勧めしますが、ID の確認が不要な場合は null を渡すことができます。

ChannelSet で提供される検索動作に加えて、Channel クラスでは `failoverURIs` プロパティが定義されています。このプロパティを使用すると、ActionScript でチャネルを直接設定して、チャネルから宛先への接続を試みたときにこのエンドポイント URI の配列全体にわたってフェイルオーバーを発生させるようにすることができます。

接続プロセスでは、最初のチャネルが検索され、そのチャネルへの接続が試みられます。この接続に失敗した場合、チャネルでフェイルオーバー URI が定義されていれば、それぞれの URI が試みられ、すべて失敗した場合はそのチャネルが断念され、次に使用可能なチャネルが ChannelSet で検索されます。セット内のどのチャネルへの接続もできなかった場合、保留されている未送信のメッセージがあればクライアントで失敗が生成されます。

Flex データサービスでは次のメッセージチャネルがサポートされています。

メッセージチャネル	説明

RTMP	RTMP エンドポイントへの接続に使用するチャネル。リアルタイムのメッセージングとサーバープッシュのブロードキャストをサポートします。
	次の例は RTMP チャネル定義を示しています。
	`<channel-definition id="my-rtmp"`
	`  class="mx.messaging.channels.RTMPChannel">`
	`  <endpoint uri="rtmp://{server.name}:2035"`
	`  class="flex.messaging.endpoints.RTMPEndpoint"/>`
	`  <properties>`
	`  <idle-timeout-minutes>20</idle-timeout-minutes>`
	`  </properties>`
	`</channel-definition>`
	RTMP エンドポイントの設定では server.port トークンまたは context.root トークンは使用できません。server.name トークンは使用できますが、クラスタ化の使用時は使用できません。
	ファイアウォールを設置している場合は、RTMP トラフィックを許可するために、RTMP エンドポイントに割り当てたポートを開く必要があります。HTTP トンネリングは RTMP トラフィックには使用できません。
	SSL (Secure Socket Layer) を使用する安全なデータ送信を提供する場合は、標準の RTMP チャネルではなくセキュアな RTMP チャネルを使用してください。
	RTMP チャネル定義の `properties` セクションにあるオプションの `max-worker-threads` エレメントを使用すると、固定 RTMP エンドポイントワーカープールサイズを設定できます。デフォルトの動作は、必要に応じて上限なしで大きくなるキャッシュスレッドプールを使用するためのものです。次の例は、`max-worker-threads` エレメントを示しています。
	`<channel-definition id="rtmp-ac"`
	`  class="mx.messaging.channels.RTMPChannel">`
	`  <endpoint uri="rtmp://10.132.64.63:2266/eqa/rtmpac"`
	`  class="flex.messaging.endpoints.RTMPEndpoint"/>`
	`  <properties>`
	`  <max-worker-threads>10</max-worker-threads>`
	`...`

AMF	AMF エンコードされたメッセージを AMF エンドポイントに送信するために使用するチャネル。HTTP 経由で送信された AMF 要求および応答をサポートします。
	このチャネルをパブリッシュ / サブスクライブメッセージングに使用する場合は、通常はチャネル定義で `polling` プロパティを `true` に設定します。また、ポーリング間隔をチャネル定義で設定することもできます。
	次の例は AMF チャネル定義を示しています。
	`<channel-definition id="samples-amf"`
	`  type="mx.messaging.channels.AMFChannel">`
	`  <endpoint uri="http://{server.name}:8100/myapp/messagebroker/`
	`  amf" type="flex.messaging.endpoints.AmfEndpoint"/>`
	`</channel-definition>`

メッセージチャネル	説明

AMF ポーリング	AMF エンコードされたメッセージを AMF エンドポイントに送信するために使用するチャネル。HTTP 経由で送信された AMF 要求および応答をサポートします。
	このチャネルをパブリッシュ / サブスクライブメッセージングに使用する場合は、チャネル定義で `polling` プロパティを `true` に設定します。また、ポーリング間隔をチャネル定義で設定することもできます。
	次の例は、ポーリングを有効にした AMF ポーリングチャネル定義を示しています。
	`<channel-definition id="samples-polling-amf"`
	`type="mx.messaging.channels.AMFChannel">`
	`<endpoint uri="http://{server.name}:8100/dev/messagebroker/`
	`amfpolling" type="flex.messaging.endpoints.AmfEndpoint"/>`
	`<properties>`
	`<polling-enabled>true</polling-enabled>`
	`<polling-interval-seconds>8</polling-interval-seconds>`
	`</properties>`
	`</channel-definition>`

メッセージチャネル	説明

セキュアな RTMP	TLS (Transport Layer Security) 経由の RTMP エンドポイントへの接続に使用するチャネル。このチャネルは、リアルタイムのメッセージングとサーバープッシュのブロードキャストをサポートします。このチャネルではデジタル証明書が必要です。また、キーストアファイル名とパスワードを指定するための子エレメントが含まれます。次の例はセキュアな RTMP チャネル定義を示しています。キーストアファイルは必ずしも示された場所に存在するとは限りません。.
	`<channel-definition id="my-rtmps"`
	`class="mx.messaging.channels.SecureRTMPChannel">`
	`<endpoint uri="rtmps://{server.name}:2099"`
	`class="flex.messaging.endpoints.SecureRTMPEndpoint"/>`
	`<properties>`
	`<idle-timeout-minutes>30</idle-timeout-minutes>`
	`<keystore-file>C:¥somedir¥flex¥dataservices¥apps/j2ee¥dev¥WEB-INF/flex/keys/server.keystore</keystore-file>`
	`<keystore-password>password</keystore-password>`
	`</properties>`
	`</channel-definition>`
	Java keytool ユーティリティを使用して、公開キーと秘密キーのペア、および自己署名証明書を作成できます。自己署名証明書のユーザー名は、`localhost`、または RTMPS エンドポイントが使用できる IP アドレスに、設定する必要があります。キーおよび証明書の作成と管理の詳細については、Java keytool のマニュアル (http://java.sun.com) を参照してください。オプションで、リムーバブルメディアなどの別のファイルにキーストアのパスワードを保存することができます。指定が必要なのは、1 つの keystore-password または keystore-password-file だけです。次の例に、`keystore-password-file` エレメントを示します。
	`<keystore-password-file>a:\password</keystore-password-file>`
	また、SSL 使用時のデフォルトの JVM アルゴリズムの代わりに、代替 JVM アルゴリズムを指定することもできます。デフォルトの JVM アルゴリズムは、セキュリティプロパティ ssl.KeyManagerFactory.algorithm で制御できます。通常は、デフォルトのアルゴリズムを使用し、このプロパティを省略してください。使用できる値は次のとおりです。
	• `<algorithm>Default</algorithm>` デフォルトの JVM アルゴリズムを明示的に使用します。
	• `<algorithm>SunX509</algorithm>` Sun のアルゴリズムを使用します。Sun セキュリティプロバイダが必要です。
	• `<algorithm>IbmX509</algorithm>` IBM のアルゴリズムを使用します。IBM セキュリティプロバイダが必要です。
	RTMP エンドポイントの設定では server.port トークンまたは context.root トークンは使用できません。server.name トークンは使用できますが、クラスタ化の使用時は使用できません。
	ファイアウォールを設置している場合は、RTMP トラフィックを許可するために、RTMP エンドポイントに割り当てたポートを開く必要があります。HTTP トンネリングは RTMP トラフィックには使用できません。

メッセージチャネル	説明

セキュアな AMF	AMF チャネルと同様のチャネルですが、HTTP ではなく HTTPS を使用します。AMF チャネルとは別のエンドポイントとクラスを使用します。
	次の例はセキュアな AMF チャネル定義を示しています。
	`<channel-definition id="my-secure-amf"`
	`  class="mx.messaging.channels.SecureAMFChannel">`
	`  <endpoint uri="https://{server.name}:9100/dev/messagebroker/`
	`  amfsecure"`
	`  class="flex.messaging.endpoints.SecureAMFEndpoint"/>`
	`</channel-definition>`

HTTP	HTTP エンドポイントへの接続に使用するチャネル。HTTP 要求および応答をサポートします。このチャネルはテキストベースの (XML) メッセージ形式を使用します。
	次の例は HTTP チャネル定義を示しています。
	`<channel-definition id="my-http"`
	`  class="mx.messaging.channels.HTTPChannel">`
	`  <endpoint uri="http://{server.name}:8100/dev/messagebroker/http"`
	`  class="flex.messaging.endpoints.HTTPEndpoint"/>`
	`</channel-definition>`
	メモ ：正確な、参照による直列化が必要な場合は、HTTPChannel による IExternalizable インターフェイス (カスタム直列化で使用します) を実装する型は使用しないでください。使用した場合、循環オブジェクト間の参照が失われ、エンドポイントで参照のクローンが作成されたような状態になります。IExternalizable インターフェイスの詳細については、1255 ページの「カスタム直列化の使用」を参照してください。
	HTTP ベースのチャネルエンドポイントを使用するときは、チャネル定義の `properties` セクションにある `redirectURL` プロパティで URL を指定することができます。これは従来の機能です。要求の MIME タイプが正しくなかった場合に、これを使用して要求を別の URL にリダイレクトできます。

セキュアな HTTP	HTTP チャネルと同様のチャネルですが、HTTP ではなく HTTPS を使用します。HTTP チャネルとは別のチャネルクラスを使用します。
	次の例はセキュアな HTTP チャネル定義を示しています。
	`<channel-definition id="my-secure-http"`
	`  class="mx.messaging.channels.SecureHTTPChannel">`
	`  <endpoint uri=`
	`  "https://{server.name}:9100/dev/messagebroker/`
	`  httpsecure"`
	`  class="flex.messaging.endpoints.SecureHTTPEndpoint"/>`
	`</channel-definition>`

チャネルエンドポイントの設定

特定のファイアウォール、ルータ、または Web サーバーの IP アドレスをリストするホワイトリストとブラックリストを使用して、HTTP ベースおよび RTMP ベースのチャネルエンドポイントを保護することができます。ホワイトリストには、エンドポイントへのアクセスを許可されているクライアント IP アドレスが含まれます。ブラックリストには、エンドポイントへのアクセスを制限されているクライアント IP アドレスが含まれます。

クライアント IP アドレスがホワイトリストとブラックリストの両方のメンバーである場合は、ブラックリストがホワイトリストより優先されます。

whitelist および blacklist エレメントには、0 ～ N 個の ip-address エレメントや ip-address-pattern エレメントを含めることができます。ip-address エレメントは簡単な IP アドレスのマッチングをサポートするため、アスタリスク文字 (*) を使用してアドレス内の個々のバイトをワイルドカードにすることができます。

ip-address-pattern エレメントは、IP アドレスの正規表現のパターンマッチングをサポートします。これによって、範囲に基づく強力な IP アドレスのフィルタリングが可能になります。

次の例に、ホワイトリストとブラックリストを示します。

```
<whitelist>
  <ip-address-pattern>237.*</ip-address-pattern>
  <ip-address>10.132.64.63</ip-address>
</whitelist>

<blacklist>
  <ip-address>10.60.147.*</ip-address>
  <ip-address-pattern>10\\.132\\.17\\.5[0-9]{1,2}</ip-address-pattern>
</blacklist>
```

デフォルトでは、RTMP エンドポイントを開始する RTMP サーバーは、指定されたポート上のすべてのローカルなネットワークインターフェイスカード (NIC) にバインドされます。ポートはエンドポイントの URL に基づきます。RTMP チャネルの properties セクションにある <bind-address>xxx.xxx.xxx.xxx</bind-address> エレメントを指定して、起動時に RTMP サーバーを特定のローカル NIC にバインドすることができます。こういう状況は、内部向けの NIC と一般公開された NIC が装備されたマシンで起こります。RTMP サーバーを内部 NIC だけにバインドして、パブリックアクセスから保護することもできます。

デフォルトでは、RTMP サーバーの受け入れソケットは、プラットフォームのデフォルト設定を使用して保留中の接続を受け入れるキューのサイズを制御します。RTMP チャネルの properties セクションにある accept-backlog エレメントを使用して、このキューのサイズを制御できます。

データの直列化

このセクションでは、クライアント上の ActionScript オブジェクトとサーバー上の Java オブジェクトとの間の直列化について説明します。また、クライアント上の ActionScript オブジェクトと SOAP および XML スキーマ型との間の直列化についても説明します。

ActionScript から Java へのデータ変換

メソッドパラメータによって Flex アプリケーションから Java オブジェクトに送信されるデータは、自動的に ActionScript のデータ型から Java のデータ型に変換されます。Flex が Java オブジェクトの適切なメソッドを探す際には、さらに穏やかな変換方法が使用されます。

クライアント側のデータ型が Boolean や String などの単純な型ならば、多くの場合はリモート API に完全に一致しますが、完全に一致しない場合にも、Java オブジェクトの適切なメソッドを見つけるために単純な変換処理が試行されます。

ActionScript 配列のエントリに対するインデックス指定の方法には 2 種類あります。厳密な配列は、すべてのインデックスが Number 型の配列です。結合配列は、String 型のインデックスが少なくとも 1 つ含まれる配列です。Java オブジェクトのメソッドの呼び出しに使用されるパラメータのデータ型は、この配列の種類によって変わるため、いずれの種類の配列をサーバーに送信するのかについて認識しておく必要があります。

単純なデータ型の変換

次の表に、単純なデータ型に対してサポートされる ActionScript (AMF 3) から Java への変換を示します。

ActionScript 型 (AMF 3)	Java への非直列化	サポートされている Java 型バインディング
Array (密)	java.util.List	java.util.Collection、`Object`[ ] (ネイティブ配列) 型がインターフェイスの場合は、次のインターフェイス実装にマップされます。 • List は ArrayList にマップされます。 • SortedSet は TreeSet にマップされます。 • Set は HashSet にマップされます。 • Collection は Arraylist にマップされます。 Collection のカスタム実装の新しいインスタンスはその型にバインドされます。
Array (疎)	java.util.Map	java.util.Collection、ネイティブ配列

ActionScript 型 (AMF 3)	Java への非直列化	サポートされている Java 型バインディング
Boolean "true" または "false" のストリング	java.lang.Boolean	Boolean、boolean、String
flash.utils.ByteArray	byte []	
flash.utils.IExternalizable	java.io.Externalizable	
Date	java.util.Date (世界標準時 (UTC) に対応するようにフォーマットされます)	java.util.Date、java.util.Calendar、java.sql.Timestamp、java.sql.Time、java.sql.Date
int/uint	java.lang.Integer	java.lang.Byte、java.lang.Double、java.lang.Float、java.lang.Long、java.lang.Short、java.math.BigDecimal、String、および、byte、double、float、long、short の各プリミティブ型
ヌル (null)	ヌル (null)	プリミティブ
数値	java.lang.Double	java.lang.Byte、java.lang.Double、java.lang.Float、java.lang.Long、java.lang.Short、java.math.BigDecimal、String、0 (null が送られた場合)、および、byte、double、float、long、short の各プリミティブ型
Object (汎用)	java.util.Map	Map インターフェイスが指定された場合、java.util.Map では新しい java.util.HashMap が、java.util.SortedMap では新しい java.util.TreeMap が、Flex データサービスによって作成されます。
String	java.lang.String	java.lang.String、java.lang.Boolean、java.lang.Number
型指定された Object	型指定された Object リモートのクラス名を指定する [RemoteClass] メタデータを使用した場合。Bean 型には、引数を持たないパブリックコンストラクタが必要です。	型指定された Object
未定義 (undefined)	ヌル (null)	オブジェクトの場合は null、プリミティブの場合はデフォルト値

ActionScript 型 (AMF 3)	Java への非直列化	サポートされている Java 型バインディング
XML	org.w3c.dom.Document	org.w3c.dom.Document
XMLDocument (古い XML 型)	org.w3c.dom.Document	org.w3c.dom.Document XMLDocument 型に対する古い XML のサポートは、services-config.xml ファイルで定義されている任意のチャネルで有効にできます。この設定は、データをサーバーからクライアントに送り戻す場合にのみ重要です。この設定によって、org.w3c.dom.Document インスタンスが ActionScript に送られる方法が制御されます。詳細については、1253 ページの「チャネルでの古い AMF 直列化の実現」を参照してください。

プリミティブ値は Java では null に設定できません。クライアントから Java オブジェクトに Boolean 型や Number 型の値を渡す場合、null 値はプリミティブ型のデフォルト値に変換されます。たとえば、double、float、long、int、short、byte の場合は 0、char の場合は \u0000、Boolean の場合は false になります。デフォルト値を取得するのはプリミティブの Java 型のみです。

Flex データサービスでは、java.lang.Throwable オブジェクトは他の型指定されたオブジェクトと同様に扱われます。これらは、getter および setter を探す規則に従って処理され、型指定されたオブジェクトがクライアントに返されます。これらの規則は通常の Bean 規則と同様ですが、読み取り専用プロパティに対しては getter を探し、読み書き可能プロパティに対しては getter と setter を探す点が異なります。このため、Java 例外からすべての情報を取得できます。Throwable オブジェクトに対して古い動作を必要とする場合は、チャネルで `legacy-throwable` プロパティを `true` に設定できます。詳細については、1253 ページの「チャネルでの古い AMF 直列化の実現」を参照してください。

厳密な配列は、java.util.Collection API または java.lang.reflect.Array API の実装を受け取るメソッドにパラメータとして渡すことができます。

Java コレクションは Object 型を何種類でも含むことができますが、Java 配列の場合はすべてのエントリが同じ型である必要があります (たとえば、java.lang.Object[ ]、int[ ] など)。

また、ActionScript の厳密な配列は、共通の Collection API インターフェイスを実装した適切な型に変換されます。たとえば、ActionScript の厳密な配列が Java オブジェクトメソッド `public void addProducts(java.util.Set products)` に送られた場合は、java.util.HashSet インスタンスに変換された後でパラメータとして渡されます。これは、HashSet は java.util.Set インターフェイスの適切な実装の 1 つであるためです。同様に、java.util.TreeSet のインスタンスは java.util.SortedSet インターフェイスによって型指定されたパラメータに渡されます。

Flex では、java.util.ArrayList のインスタンスは、java.util.List インターフェイスなど、java.util.Collection を拡張する任意のインターフェイスによって型指定されたパラメータに渡された後、mx.collections.ArrayCollection インスタンスとしてクライアントに送り返されます。通常の ActionScript 配列をクライアントに送り返す必要がある場合は、チャネル定義のプロパティの `<serialization>` セクションで、`legacy-collection` エレメントを `true` に設定する必要があります。詳細については、1253 ページの「チャネルでの古い AMF 直列化の実現」を参照してください。

Java から ActionScript へのデータ変換

Java メソッドから返されたオブジェクトは、Java から ActionScript に変換されます。Flex はオブジェクト内のオブジェクトも処理します。Flex は、次の表に示す Java データ型を暗黙的に処理します。

Java 型	ActionScript 型 (AMF 3)
java.lang.String	String
java.lang.Boolean、boolean	Boolean
java.lang.Integer	int i < 0xF0000000 || i > 0x0FFFFFFF の場合、値は Number 型に格上げされます。
java.lang.Short	int i < 0xF0000000 || i > 0x0FFFFFFF の場合、値は Number 型に格上げされます。
java.lang.Byte	int i < 0xF0000000 || i > 0x0FFFFFFF の場合、値は Number 型に格上げされます。
java.lang.Byte[]	flash.utils.ByteArray
java.lang.Double	Number
java.lang.Long	Number
java.lang.Float	Number
java.lang.Character	String
java.lang.Character[]	String
java.util.Calendar	Date 日付は世界標準時 (UTC) タイムゾーンで送られます。クライアントとサーバーでタイムゾーンに合わせて時間を適切に調整する必要があります。
java.util.Date	Date データは UTC タイムゾーンで送られます。クライアントとサーバーでタイムゾーンに合わせて時間を適切に調整する必要があります。

Java 型	ActionScript 型 (AMF 3)
java.lang.Object (上記以外の型)	型指定された Object オブジェクトは Java Bean のイントロスペクション規則を使用して直列化されます。静的フィールド、一時フィールド、および非パブリックフィールドは、除外されます。
java.util.Collection	mx.collection.ArrayCollection
java.lang.Object[]	Array
java.util.Map	オブジェクト (型指定なし) Flex 1.5, では、java.util.Map は結合配列または ECMA 配列として送られていました。この方法は推奨されなくなりました。結合配列に対して古い Map のサポートを有効にすることもできますが、この方法はお勧めしません。詳細については、1253 ページの「チャネルでの古い AMF 直列化の実現」を参照してください。
java.util.Dictionary	オブジェクト (型指定なし)
org.w3c.dom.Document	XML オブジェクト XMLDocument 型に対する古い XML のサポートは、services-config.xml ファイルで定義されている任意のチャネルで有効にできます。詳細については、1253 ページの「チャネルでの古い AMF 直列化の実現」を参照してください。
ヌル (null)	ヌル (null)
java.lang.Object を拡張するその他のクラス	オブジェクト (型指定) オブジェクトは Java Bean のイントロスペクション規則を使用して直列化されます。静的フィールド、一時フィールド、および非パブリックフィールドは、除外されます。

Flex で暗黙的に処理されない Java オブジェクトの場合は、パブリックの getter および setter メソッドペア内の値と、パブリック変数内の値が、オブジェクトのプロパティとしてクライアントに渡されます。プライベートプロパティ、定数、静的プロパティ、読み取り専用プロパティなどは、直列化されません。

Flex は、標準の Java クラスである java.beans.Introspector を使用して Java Bean クラスのプロパティ記述子を取得します。このクラスには、Flex が対応する ActionScript オブジェクトのプロパティ名を判定するための規則が含まれます。

ActionScript クラスでは、[RemoteClass(alias=" ")] メタデータタグを使用して、Java オブジェクトに直接マップされる ActionScript オブジェクトを作成できます。MXML ファイルでは、データの変換先の ActionScript クラスを使用するか参照する必要があります。これには、次の例に示すように結果オブジェクトをキャストすることをお勧めします。

```
var result:MyClass = MyClass(event.result);
```

クラス自体では、依存関係もリンクされるようにするために、厳密に型指定された参照を使用してください。

次の例は、[RemoteClass(alias=" ")] メタデータタグを使用した ActionScript クラスのソースコードを示しています。

```
package samples.contact {
  [Bindable]
  [RemoteClass(alias="samples.contact.Contact")]
  public class Contact {
    public var contactId:int;

    public var firstName:String;

    public var lastName:String;

    public var address:String;

    public var city:String;

    public var state:String;

    public var zip:String;
  }
}
```

サーバー上の Java オブジェクトにマップせず、サーバーからオブジェクト型を送り返す場合は、エイリアスを指定しない [RemoteClass] メタデータタグを使用できます。ActionScript オブジェクトは、サーバーに送られるときは Map オブジェクトに直列化されますが、サーバーからクライアントに返されるときは元の ActionScript 型になります。

チャネルでの古い AMF 直列化の実現

以前のバージョンの Flex で使用されていた古い AMF 型の直列化をサポートする場合は、チャネル定義内の古い直列化プロパティを services-config.xml ファイルで設定できます。宛先に対して古い直列化を使用するには、古いチャネルをその宛先に割り当てます。

次の例に、古い直列化を使用したチャネル定義を示します。

```
<channel-definition id="my-legacy-amf" class="mx.messaging.channels.AMFChannel">
  <endpoint uri="http://{server.name}:8100/dev/messagebroker/amflegacy"
    class="flex.messaging.endpoints.AMFEndpoint"/>
    <properties>
      <serialization>
        <!-- ignore-property-errors はデフォルトで true です。このプロパティで、
        受信クライアントオブジェクトがサーバーオブジェクトで設定できない
        プロパティを持つときに、エンドポイントがエラーを
        スローするかどうかが決まります。 -->
        <ignore-property-errors>true</ignore-property-errors>
        <!-- log-property-errors はデフォルトで false です。true に設定すると、
        予期しないプロパティエラーがログに記録されます。 -->
        <log-property-errors>false</log-property-errors>
        <legacy-collection>true</legacy-collection>
        <legacy-map>true</legacy-map>
        <legacy-xml>true</legacy-xml>
        <legacy-throwable>true</legacy-throwable>
      </serialization>
  </properties>
</channel-definition>
```

Web サービスにおける ActionScript からスキーマ型および SOAP 型への変換

Flex では、次の表にある XML スキーマ型を使用する SOAP 要求を送信する前に、情報を ActionScript の対応する型で適切に表現する必要があります。複合型と SOAP-Encoded Array 型もサポートしており、他の複合型、配列型、またはビルトイン XML スキーマ型で構成されているものでもかまいません。

Web サービスにおける XML スキーマ型と ActionScript 型の型マッピングを、次の表に示します。

XML スキーマ型	ActionScript 型
ComplexType	匿名オブジェクト、または必要なプロパティが定義された、その他の型指定されたオブジェクト
sequence	Array
boolean	Boolean
string	String

XML スキーマ型	ActionScript 型
normalizedString	String
すべての数値型	Number、int、uint
date	Date
dateTime	Date
time	Date
Base64Binary	flash.utils.ByteArray
hexBinary	flash.utils.ByteArray
anyType (WSDL)	ActionScript 型を次のスキーマ型に変換します。 • uint を unsignedInt に変換します。 • int を int に変換します。 • Number を double に変換します。 • Boolean を boolean に変換します。 • String を string に変換します。 • XMLDocument を schema に変換します。 • Date を dateTime に変換します。 • Array を array に変換します。使用できる型情報がない場合は、配列内の最初のアイテムが型の決定に使用されます。

Web サービスにおける SOAP 型と ActionScript 型の型マッピングを、次の表に示します。

SOAP 型	ActionScript 型
base64	flash.utils.ByteArray
数値型	Number
boolean	Boolean
Array	Array
string	String

RPC エンコードされた Web サービスにおける、Apache Axis に固有の型と ActionScript 型の型マッピングを、次の表に示します。

Apache Axis 型	ActionScript 型
Map	Object
RowSet	RowSet の受信のみ可能です。RowSet の送信はできません。
Document	Document の受信のみ可能です。Document の送信はできません。
Element	flash.xml.XMLNode

カスタム直列化の使用

クライアントの ActionScript とサーバーの Java との間でデータの直列化および非直列化を行う標準のメカニズムではニーズが満たされない、という場合は、ActionScript ベースの flash.utils.IExternalizable インターフェイスをクライアントで実装し、対応する Java-based java.io.Externalizable インターフェイスをサーバーで実装することで、独自の直列化スキームを記述できます。

通常、カスタム直列化は、クライアントサイドかサーバーサイドのいずれかのオブジェクト表現について、そのプロパティをネットワーク層を経由して受け渡す際に、すべてのプロパティを渡すことを避けるために行います。カスタム直列化を実装する際は、クライアントのみまたはサーバーのみに存在する特定のプロパティが渡されないように、クラスのコードを記述できます。標準の直列化スキームを使用した場合は、すべてのパブリックプロパティがクライアントとサーバー間で受け渡されます。

クライアントサイドでは、flash.utils.IExternalizable インターフェイスを実装するクラスの識別子が直列化ストリームに書き込まれます。クラスでは、自身のインスタンスの状態を直列化し、再構築します。クラスでは IExternalizable インターフェイスの `writeExternal()` メソッドと `readExternal()` メソッドを実装して、直列化ストリームの内容と形式を制御しますが、オブジェクトとそのスーパータイプのクラス名や型は制御しません。これらのメソッドは、AMF 直列化の動作よりも優先されます。また、これらのメソッドは、その状態を保存するためにスーパータイプと対称になっている必要があります。

サーバーサイドでは、java.io.Externalizable インターフェイスを実装する Java クラスによって、flash.utils.IExternalizable インターフェイスを実装する ActionScript クラスと同様の機能が実行されます。

メモ	正確な、参照による直列化が必要な場合は、HTTPChannel による IExternalizable インターフェイスを実装する型は使用しないでください。使用した場合、循環オブジェクト間の参照が失われ、エンドポイントで参照のクローンが作成されたような状態になります。

次の例は、サーバーで Java ベースの Product クラスにマップされる、クライアント (ActionScript) 側の Product クラスのソースコードを示しています。クライアントの Product で IExternalizable インターフェイスを実装し、サーバーの Product で Externalizable インターフェイスを実装します。

```
// Product.as
package samples.externalizable {

import flash.utils.IExternalizable;
import flash.utils.IDataInput;
import flash.utils.IDataOutput;

[RemoteClass(alias="samples.externalizable.Product")]
public class Product implements IExternalizable {
  public function Product(name:String=null) {
    this.name = name;
  }
```

```
    public var id:int;
    public var name:String;
    public var properties:Object;
    public var price:Number;

    public function readExternal(input:IDataInput):void {
      name = input.readUTF();
      properties = input.readObject();
      price = input.readFloat();
    }

    public function writeExternal(output:IDataOutput):void {
      output.writeUTF(name);
      output.writeObject(properties);
      output.writeFloat(price);
    }
}
}
```

クライアントの Product では 2 種類の直列化を使用します。つまり、java.io.Externalizable インターフェイスと互換性のある標準の直列化と、AMF 3 の直列化を使用します。次の例は、クライアントの Product の `writeExternal()` メソッドを示しています。このメソッドでは両方の種類の直列化を使用しています。

```
public function writeExternal(output:IDataOutput):void {
  output.writeUTF(name);
  output.writeObject(properties);
  output.writeFloat(price);
}
```

次の例に示すように、サーバーの Product の `writeExternal()` メソッドはクライアント側のこのメソッドとほとんど同じです。

```
public void writeExternal(ObjectOutput out) throws IOException {
  out.writeUTF(name);
  out.writeObject(properties);
  out.writeFloat(price);

}
```

クライアントの Product の `writeExternal()` メソッド、`flash.utils.IDataOutput.writeUTF()` メソッド呼び出し、および `flash.utils.IDataOutput.writeFloat()` メソッド呼び出しは、プリミティブ型を操作する Java の `java.io.DataInput.readUTF()` メソッドと `java.io.DataInput.readFloat()` メソッドの仕様を満たす標準の直列化メソッドの例です。これらのメソッドは、String 型である `name` プロパティと、Float 型である `price` プロパティを、サーバーの Product に送信します。

クライアントの Product の writeExternal() メソッドは、AMF 3 の直列化を示す唯一の例で、flash.utils.IDataOutput.writeObject() メソッドを呼び出します。このメソッドは、サーバーの Product の readExternal() メソッドの中で java.io.ObjectInput.readObject() メソッド呼び出しにマップされます。flash.utils.IDataOutput.writeObject() メソッドは、Object 型である properties プロパティをサーバーの Product に送信します。このことが可能になる理由は、writeObject() メソッドから送られたデータを AMF 3 としてフォーマットされたデータとして受け取る java.io.ObjectInput インターフェイスが、AMFChannel エンドポイントで実装されているからです。

一方、サーバーの Product の readExternal() メソッドの中で readObject() メソッドが呼び出されるときは、AMF 3 の非直列化が使用されます。このため、ActionScript 側の properties 値は Map 型であると見なされます。

次の例は、サーバーの Product の readExternal() メソッドを示しています。

```
public void readExternal(ObjectInput in) throws IOException,
  ClassNotFoundException {
  // サーバーのプロパティをクライアントの表現から読み取ります。
  name = in.readUTF();
  properties = (Map)in.readObject();
  price = in.readFloat();
  setInventoryId(lookupInventoryId(name, price));
  }
```

クライアントの Product の writeExternal() メソッドでは、直列化の中で id プロパティをサーバーに送信しません。これは、サーバー側の Product オブジェクトではこのプロパティが役に立たないためです。同様に、サーバーの Product の writeExternal() メソッドでも、サーバー固有のプロパティである inventoryId プロパティをクライアントに送信しません。

いずれの方向の直列化でも、Product のプロパティの名前は送信していません。クラスの状態が固定であり管理可能であるため、プロパティは名前が付けられずに、適切に定義された順序で送信され、readExternal() ではそれらが適切な順序で読み取られます。

サーバーの Product クラスの詳細なソースを次の例に示します。

```
// Product.java
package samples.externalizable;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.Map;

/**
* この Externalizable クラスでは、この型のインスタンスを
* 送受信するクライアントが、直列化に必要なデータ形式に
* 従っている必要があります。
*/
```

```
public class Product implements Externalizable {
   private String inventoryId;
   public String name;
   public Map properties;
   public float price;

   public Product()
   {
   }

     /**
     * サードパーティのインベントリの追跡に使用するローカル識別子。このプロパティは
     * サーバー固有であるためクライアントには送信されません。
     * 識別子は 'X' で始める必要があります。
     */
     public String getInventoryId() {
        return inventoryId;
     }

     public void setInventoryId(String inventoryId) {
        if (inventoryId != null && inventoryId.startsWith("X"))
        {
           this.inventoryId = inventoryId;
        }
        else
        {
           throw new IllegalArgumentException("3rd party product
           inventory identities must start with 'X'");
        }
     }

     /**
      * ThirdPartyProxy のインスタンスのクライアント状態を非直列化します。
      * これには、UTF フォーマットされた String を name として読み取り、
      * UTF フォーマットされた String を description として読み取り、
      * 浮動小数点整数 ( 単精度 ) を price として読み取ります。
      */
     public void readExternal(ObjectInput in) throws IOException,
        ClassNotFoundException {
        // サーバーのプロパティをクライアントの表現から読み取ります。
        name = in.readUTF();
        properties = (Map)in.readObject();
        price = in.readFloat();
        setInventoryId(lookupInventoryId(name, price));
     }

     /**
      * ThirdPartyProxy のインスタンスのサーバー状態を直列化します。
      * これには、UTF フォーマットされた String を name として送信し、
      * UTF フォーマットされた String を description として送信し、
```

```
     * 浮動小数点整数（単精度）を price として送信します。インベントリの識別子は
     * 外部クライアントには送信されません。
     */
    public void writeExternal(ObjectOutput out) throws IOException {
      // クライアントのプロパティをサーバーの表現から書き出します。
      out.writeUTF(name);
      out.writeObject(properties);
      out.writeFloat(price);
    }

    private static String lookupInventoryId(String name, float price) {
      String inventoryId = "X" + name + Math.rint(price);
      return inventoryId;
    }
}
```

宛先のセキュリティ保護

宛先がパブリックではない場合、Flex サービス設定ファイルにある宛先定義でセキュリティ制限を適用することで、アクセスを、特権を持つユーザーグループに制限することができます。セキュリティ制限を適用することにより、宛先にアクセスする前に、カスタム認証または基本認証を使用して確実にユーザーを認証できるようになります。Flex データサービスのセキュリティ制限では、デフォルトでカスタム認証が使用されます。セキュリティ制限では、宛先にアクセスする前にユーザーがユーザーストアに対して承認されることを必要とすることもできます。ユーザーストアとは、ユーザーのセキュリティ属性を格納したリポジトリのことです。

認証とは、ユーザーがシステムに対して本人であることを証明するためのプロセスのことです。承認とは、ユーザーがシステムの中でどのような種類のアクティビティを実行できるのかを決めるためのプロセスのことです。ユーザーは、認証された後、承認によって特定のリソースにアクセスできるようになります。

セキュリティ制限を 1 つの宛先のみで使用する場合は、宛先に対するセキュリティ制限を宛先定義内でインラインで宣言できます。次の例は、宛先定義内で宣言されたセキュリティ制限を示しています。

```
<service>
...
  <destination id="roDest">
...
    <security>
      <security-constraint>
        <auth-method>Custom</auth-method>
        <roles>
          <role>roDestUser</role>
        </roles>
      </security-constraint>
    </security>
  </destination>
```

```
...
</service>
```

セキュリティ制限をグローバルに宣言することもできます。複数の宛先で同じセキュリティ設定を使用する場合は、Flex サービス設定ファイルの security セクションで 1 つのセキュリティ制限を定義し、各宛先からそのセキュリティ制限を参照してください。次の例は、2 つの宛先定義内で参照されているセキュリティ制限を示しています。

```
<service>
  <destination id="SecurePojo1">
...
     <security>
       <security-constraint ref="trusted"/>
     </security>
  </destination>
  <destination id="SecurePojo2">
...
     <security-constraint ref="trusted"/>
  </destination>
...
</service>
...
<security>
  <security-constraint id="trusted">
    <auth-method>Custom</auth-method>
    <roles>
      <role>trustedUsers</role>
    </roles>
  </security-constraint>
...
</security>
```

クライアントサイドコンポーネントからの資格情報の受け渡し

セキュリティ制限を使用する宛先にユーザー資格情報を送信するには、ユーザー名とパスワードの値を、アプリケーションで使用している RemoteObject、HTTPService、WebService、DataService、Publisher、または Consumer コンポーネントの setCredentials() メソッドのパラメータとして指定します。資格情報を削除するには、コンポーネントの logout() メソッドを呼び出します。

メモ

サービスコンポーネントの useProxy プロパティが false に設定されている場合は、setCredentials() または setRemoteCredentials() メソッドを呼び出しても効果はありません。

次の例は、ユーザー名とパスワードの値を HTTPService コンポーネントから宛先に送信するための ActionScript コードを示しています。

```
import mx.rpc.http.HTTPService;

var employeeHTTP:HTTPService = new HTTPService();
employeeHTTP.destination = "SecureDest";
employeeHTTP.setCredentials("myUserName", "myPassword");
empoyeeHTTP.send({param1: 'foo'});
```

setRemoteCredentials() メソッドを使用すると、資格情報を必要とするリモートの HTTP サービスや Web サービスに資格情報を渡すことができます。たとえば、HTTPService コンポーネントを使用している場合に、資格情報を安全な JSP ページに渡すことができます。資格情報はメッセージヘッダの中で宛先に送信されます。

CFC (ColdFusion Component) など、ユーザー名とパスワードによる認証を必要としている外部サービスによって管理されている Remoting Service の宛先に対しても、setRemoteCredentials() メソッドを呼び出すことができます。

リモートの資格情報の受け渡しは、Flex サービス設定ファイルで定義されたセキュリティ制限を満たすためにユーザー名とパスワードの資格情報を受け渡すこととは異なります。しかし、これら 2 種類の資格情報は組み合わせて使用できます。

次の例は、リモートユーザー名とリモートパスワードの値を宛先に送信するための ActionScript コードを示しています。これらの資格情報は、宛先設定によって、それらの資格情報を必要としている実際の JSP ページに渡されます。

```
var employeeHTTP:mx.rpc.http.HTTPService = new HTTPService();
employeeHTTP.destination = "secureJSP";
employeeHTTP.setRemoteCredentials("myRemoteUserName", "myRemotePassword");
employeeHTTP.send({param1: 'foo'});
```

クライアントでリモート資格情報を実行時に設定する代わりに、サーバーサイドの宛先定義の <properties> セクションにある remote-username エレメントおよび remote-password エレメントで、リモート資格情報を設定することもできます。次の例は、これらのプロパティを指定した宛先を示しています。

```
  <destination id="samplesProxy">
    <channels>
      <channel ref="samples-amf"/>
    </channels>

    <properties>
      <url>
        http://someserver/SecureService.jsp
      </url>
      <remote-username>johndoe</remote-username>
      <remote-password>opensaysme</remote-password>
    </properties>
  </destination>
...
```

基本認証

基本認証では、Web アプリケーションコンテナの標準の J2EE 基本認証が利用されます。この認証形式を使用する場合は、URL などのリソースを Web アプリケーションの web.xml ファイルでセキュリティ保護します。基本認証を使用して宛先へのアクセスをセキュリティ保護する場合、通常は web.xml ファイルの中で、宛先で使用されるチャネルのエンドポイントをセキュリティ保護します。そして、セキュリティ保護されたリソースにアクセスするように宛先を設定し、ユーザー名 (原則) とパスワード (資格情報) に対してチャレンジが行われるようにします。Web ブラウザでは、Flex とは独立してチャレンジが実行されます。Web アプリケーションコンテナではユーザーの資格情報が認証されます。

次の例は、セキュリティ保護されたチャネルエンドポイントに対する web.xml ファイルでの設定を示しています。

```
...
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Channel</web-resource-name>

    <url-pattern>/messagebroker/amf</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>

  <auth-constraint>
    <role-name>sampleusers</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
</login-config>

<security-role>
  <role-name>sampleusers</role-name>
</security-role>
...
```

ログインに成功したユーザーは、ブラウザを閉じるまでログインしたままの状態になります。基本認証によるセキュリティ制限を宛先に適用している場合は、現在認証されている原則が存在するかどうかが Flex によって調べられ、それからメッセージが宛先に転送されます。現在認証されている原則が存在する場合でも、原則に対してカスタム承認を実行してください。カスタム承認を有効にするには、宛先のセキュリティ制限定義でロールを指定します。web.xml ファイルで参照されるロール、および Flex データサービス設定ファイルで定義されたセキュリティ制限の中で参照されるロールは、すべてアプリケーションサーバーのユーザーストア内に定義します。

メモ

ユーザーおよびロールの定義方法は、使用しているアプリケーションサーバーの種類によって異なります。たとえば、Adobe JRun のユーザーおよびロールは、デフォルトで "servers/<サーバー名>/SERVER-INF/jrun-users.xml" ファイル内に定義します。

次の例は、承認のためのロールを指定するセキュリティ制限定義を示しています。

```
<security-constraint id="privileged-users">
  <auth-method>Basic</auth-method>
  <roles>
    <role>privilegedusers</role>
    <role>admins</role>
  </roles>
</security-constraint>
```

カスタム認証

基本認証の代わりにカスタム認証を使用することにより、アプリケーションの外観に合わせたカスタムログインフォームを MXML で作成できます。

Flex のカスタム認証ではカスタムログインアダプタが使用されます。これはログインコマンドとも呼ばれ、ユーザーの資格情報を確認し、原則をアプリケーションサーバーにログインさせます。ログインコマンドでは `flex.messaging.security.LoginCommand` API を実装する必要があります。Flex サービス設定ファイルの `<security>` セクションでは、複数のログインコマンドを登録できます。`login-command` エレメントの `server` 属性は、`servletConfig.getServletContext().getServerInfo()` メソッドから返された値に対して部分一致を実行する場合に使用します。`server` の値は、この値の最初のサブストリング、またはストリング全体と、大文字と小文字を区別せずに一致する必要があります。

Flex データサービスには、Adobe JRun、Apache Tomcat、Oracle Application Server、BEA WebLogic、および IBM WebSphere に対応したログインコマンドが実装されています。Tomcat または JBoss のいずれかの場合は、TomcatLoginCommand クラスを使用してください。ログインコマンドは一度に 1 つだけ有効にし、他はすべてコメントアウトしてください。

次の例は、JRun 用に設定されたログインコマンドを示しています。

```
<security>
...
  <login-command class="flex.messaging.security.JRunLoginCommand"
    server="JRun"/>
<!--
  <login-command class="flex.messaging.security.TomcatLoginCommand"
    server="Tomcat"/>
  <login-command class="flex.messaging.security.OracleLoginCommand"
    server="Oracle"/>

  <login-command class="flex.messaging.security.WeblogicLoginCommand"
    server="Weblogic"/>
```

```
   <login-command class="flex.messaging.security.WebSphereLoginCommand"
     server="WebSphere"/>
        -->
...
</security>
```

ロールなしのログインコマンドはカスタム認証の場合のみ使用できます。カスタム承認も使用する必要がある場合は、指定したロール参照を、アプリケーションサーバーのユーザーストア内で定義されたロールにリンクする必要があります。

サーバーサイドのサービスログ機能の設定

Flex データサービスの要求および応答に対して、サーバーサイドのログ機能を実行できます。サーバーサイドのログ機能は Flex サービス設定ファイルのログのセクションで設定します。デフォルトでは、出力は System.out に送信されます。クライアントサイドのログ機能の詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の 247 ページの「ログ機能」を参照してください。

ログレベルを、次のいずれかの使用可能なレベルに設定します。

- all
- debug
- info
- warn
- error
- none

フィルタパターンエレメントでは、ログに記録する情報のカテゴリを指定できます。target エレメントの class 属性では、flex.messaging.log.ConsoleTarget を指定してメッセージを標準出力に記録できます。または、flex.messaging.log.ServletLogTarget を指定して、アプリケーションサーバーのデフォルトのサーブレット用ログメカニズムを使用するようにメッセージを記録できます。

次の例は、Debug ログレベルを使用したログ設定を示しています。

```
<logging>

<!-- flex.messaging.log.ServletLogTarget も使用できます。 -->
  <target class="flex.messaging.log.ConsoleTarget" level="Debug">
    <properties>
      <prefix>[Flex]</prefix>
      <includeDate>false</includeDate>
      <includeTime>false</includeTime>
      <includeLevel>false</includeLevel>
      <includeCategory>false</includeCategory>
    </properties>
    <filters>
      <pattern>Endpoint</pattern>
      <!--<pattern>Service.*</pattern>-->
```

```
        <!--<pattern>Message.*</pattern>-->
      </filters>
    </target>
</logging>
```

運用環境では、警告とエラーの両方が表示されるように、ログレベルを warn に設定することをお勧めします。無視しても問題のない警告が頻繁に発生する場合は、ログレベルを error に変更してエラーだけが表示されるようにできます。

次の表は、ログレベルとその説明をまとめたものです。

ログレベル	説明
all	すべてのメッセージをログに記録することを指定します。
debug	DEBUG メッセージは、Flex の内部的なアクティビティを示します。 Debug、Info、Warn、Error のログメッセージをログファイルに含める場合は、debug ログレベルを選択します。
info	INFO メッセージは、開発者や管理者向けの一般的な情報を示します。 info、Warn、Error のログメッセージをログファイルに含める場合は、Info ログレベルを選択します。
warn	Warn メッセージは、強制終了を伴わない問題がアプリケーションに発生したことを示します。 warn および error のログメッセージをログファイルに含める場合は、warn ログレベルを選択します。
error	Error メッセージは、重要なサービスが利用できないこと、あるいは、アプリケーションの使用を制限するような状況が発生したことを示します。 Error のログメッセージをログファイルに含める場合は、Error ログレベルを選択します。
none	メッセージを一切ログに記録しないことを指定します。

次の一致パターンを pattern エレメントの値として使用できます。

- Configuration
- DataService.General
- DataService.Hibernate
- DataService.Transaction
- Endpoint.*
- Endpoint.AMF
- Endpoint.HTTP
- Endpoint.RTMP
- Endpoint.Deserialization
- Endpoint.General

- Message.*
- Message.Command.*
- Message.Command.operation-name。operation-name は次のいずれかです。

  subscribe、unsubscribe、poll、poll_interval、client_sync、server_ping、client_ping、cluster_request、login、logout。
- Message.General
- Message.Data.*
- Message.Data.operation-name。operation-name は次のいずれかです。

  create、fill get、update、delete、batched、multi_batch、transacted、page、count、get_or_create、create_and_sequence、get_sequence_id、association_add、association_remove、fillids、refresh_fill、update_collection。
- Message.RPC
- MessageSelector
- Resource
- Service.*
- Service.Cluster
- Service.HTTP
- Service.Message
- Service.Message.JMS (永続的な JMS サブスクライバを正常にサブスクライブ解除できない場合に警告をログに記録します)。
- Service.Remoting
- Security

メモ

ユーザーおよびロールの定義方法は、使用しているアプリケーションサーバーの種類によって異なります。たとえば、Adobe JRun のユーザーおよびロールは、デフォルトで "servers/< サーバー名 >/SERVER-INF/jrun-users.xml" ファイル内に定義します。

セッションデータの操作

Flex データサービスでは、セッションデータを操作するための次のクラスが提供されています。これらのクラスは、公開されている Flex データサービス Javadoc のマニュアルに含まれています。

- flex.messaging.FlexContext
- flex.messaging.FlexSession
- flex.messaging.FlexSessionListener
- flex.messaging.FlexSessionAttributeListener
- flex.messaging.FlexSessionBindingEvent
- flex.messaging.FlexSessionBindingListener

FlexContext クラスは、セッションにアクセスする場合や、HTTP サーブレット要求および応答などのセッションの HTTP 部分にアクセスする場合に便利です。比較的大規模な Web アプリケーションでは、JSP や Struts アクションなどの他のクラスによって情報が格納される場合がありますが、そのようなコンテキストで Flex アプリケーションを使用する場合に、このクラスを使用して HTTP データにアクセスできます。

FlexSession クラスは、ID へのアクセスを提供するほか、setAttribute および getAttribute の機能も提供します。このクラスは、クライアントに戻す必要のないデータをサーバーに格納する場合に便利です。ただし、FlexSession はクラスタには対応していません。クライアントからクラスタ内の別のサーバーに接続した場合、クライアントでは新しい FlexSession を受け取ります。クラスタ化に対応する目的で FlexSession の属性に格納される情報は何もありません。FlexSession を使用する際は次の点に注意してください。

- RTMP チャネルを使用している場合は、ブラウザの更新後に FlexSession が残りません。しかし、AMF チャネルまたは HTTP チャネルを使用している場合は残ります。
- RTMP チャネルを使用している場合は、FlexSession の属性は他のチャネルと共有されません。しかし、AMF チャネルまたは HTTP チャネルを使用している場合は共有されます。
- RTMP チャネルを認証付きで使用している場合は、ブラウザの更新後に再びログインする必要があります。しかし、AMF チャネルまたは HTTP チャネルを使用している場合は再びログインすることは不要です。
- RTMP チャネルを認証付きで使用している場合は、AMF チャネルまたは HTTP チャネルへの切り替え後に再びログインする必要があります。

FlexSessionListener クラスは、どのユーザーが接続しているのかを監視する場合に便利です。リスナーを追加するには、静的メソッドを使用して、確立中の新しい接続を追跡します。すると、追加されたセッションへの参照が返されます。その後、各セッションが破棄されたときに、セッションからそのことをそれらの同じリスナーに知らせることができます。これを使用して、閉じた接続を監視でき、リソースをクリーンアップすることもできます。

ソフトウェアクラスタ化の使用

Flex データサービス のソフトウェアクラスタ化機能は、ステートフルサービスを使用しているとき、および RTMP チャネルなどの非 HTTP チャネルを使用しているときの、フェイルオーバーを処理します。Flex クライアントがサブネット内のホスト上で実行されている特定のサービスにサブスクライブしていて、そのホストが障害を起こした場合に、クライアントで引き続きメッセージを同じサブネット内の別のホストに送ることができます。この機能は、Flex サービス設定ファイルで定義されているすべてのサービスタイプで使用できます。Message Service と Data Management Service では、アプリケーションのメッセージング状態について、フェイルオーバーと複製の両方がサポートされています。Remoting Service と Proxy Service では、フェイルオーバーのみサポートされています。

Flex データサービスにおける最も一般的なクラスタ化形式では、ソフトウェアクラスタ化が伴いません。代わりに、1 組のロードバランサが、通常はハードウェアの形式で、HTTP サーバーの前面に設置されます。これらのロードバランサによって、個々の HTTP サーバーに対して要求が指示され、クライアント要求が同じクライアントからその HTTP サーバーに固定されます。この形式のクラスタ化は特別な機能を実装せずに Flex で可能になります。また、ソフトウェアクラスタ化機能と連動することもできます。

メッセージの処理

クライアントのフェイルオーバーの提供に加えて、クラスタノードではすべてのノードのメッセージを処理しなければならない場合があります。通常、この処理は、共通のクラスタ状態を調整するためのバックエンドリソースが使用できない場合に実行されます。共有バックエンドが使用できない場合、クラスタノードはメッセージの再処理とブロードキャストを他のノードに指示します。共有バックエンドが使用できる場合、ノードは、接続先のクライアントにメッセージをブロードキャストするよう他のノードに指示しますが、メッセージの再処理を他のノードに指示することはしません。

クラスタの定義と参照

ソフトウェアクラスタは、Flex サービス設定ファイルの `clusters` セクションで定義します。クラスタは複数定義できます。各クラスタ定義では、`id` と、JGroups プロパティファイルへのパスを指定する必要があります。JGroups は、ソフトウェアクラスタ化を提供するために Flex データサービスで使用されているオープンソースソフトウェアです。JGroups の詳細については、www.jgroups.org/javagroupsnew/docs/index.html を参照してください。

jgroups.jar ファイルと jgroups-*.xml プロパティファイルは、インストールされている Flex データサービスの "resources/clustering" フォルダにあります。クラスタを定義する前に、jgroups.jar ファイルを WEB-INF/lib ディレクトリにコピーし、jgroups-*.xml ファイルを WEB-INF/flex ディレクトリにコピーする必要があります。

名前付きのクラスタは、クラスタに含める必要のある宛先の <network> セクションで参照します。クラスタ化された宛先が指し示すチャネルでは、そのエンドポイントとして指定された URI の中にトークンを含めることはできません。

次の例は、services-config.xml ファイルでのクラスタの設定を示しています。

```
<?xml version="1.0"?>
<services-config>
...
  <clusters>
        <cluster id="MyCluster" properties="jgroups-udp.xml"/>
        <cluster id="AnotherCluster" properties="jgroups-tcp.xml"/>
    </clusters>
...
</services-config>
```

次の例は、サービスの宛先でのクラスタへの参照を示しています。shared back-end プロパティを true に設定しているため、クラスタノードでは、接続先のクライアントにメッセージをブロードキャストするよう他のノードに指示しますが、メッセージの再処理を他のノードに指示することはしません。shared-backend 属性は必須ではなく、Data Management Service の宛先に対してのみ有効です。

```
<destination id="MyDestination">
...
  <properties>
    <network>
      <cluster ref="default-cluster" shared-backend="true"/>
    </network>
  </properties>
...
</destination>
```

サービスの管理

Flex データサービスでは、Flex サービス設定ファイルで設定されたサービスを実行時に監視および管理するために、Java Management Bean (MBean) が使用されます。実行時の監視および管理コンソールは、ランタイム MBean へのアクセスを提供する Flex クライアントアプリケーションの一例です。このアプリケーションは、MBean を呼び出す Java クラスである Remoting Service 宛先を呼び出します。コンソールは flex-admin Web アプリケーションにあります。コンソールを実行するには、Web アプリケーションを実行しているときに http://‹ サーバー : ポート ›/flex-admin を開きます。ここで、‹ サーバー : ポート › には、使用しているサーバー名とポート名を指定します。

メモ

実行時の監視および管理コンソールでは、承認の確認を行わない管理機能が公開されています。flex-admin Web アプリケーションは、Flex Web アプリケーションを展開しているアプリケーションサーバーと同じアプリケーションサーバーに展開してください。また、flex-admin Web アプリケーションをロックダウンする場合は J2EE セキュリティその他の手段を使用してアクセスを保護してください。

MBean の作成と登録

ランタイム MBean によって公開されている API は、Flex データサービス設定ファイルに影響を与えることや、それらのファイルとやり取りすることはありません。これらの API は、古い接続の終了やメッセージスロットルの監視などの操作には使用できますが、新しいサービスの登録や既存のサーバーコンポーネントの設定変更などの操作には使用できません。

ランタイム MBean は、対応する管理対象リソースによってインスタンス化され、ローカルの MBean サーバーに登録されます。たとえば、MessageBroker がインスタンス化されると、対応する MessageBrokerControlMBean が作成され、MBean サーバーに登録されます。ランタイム MBean の属性は基になるリソースによって設定され、それらの属性は MBean API によって読み取り専用として公開されます。場合によっては、MBean で、属性値について基になるリソースをポーリングすることもできます。

MBean の命名規則

ランタイム MBean のモデルは MessageBrokerControlMBean で始まります。他の MBean を参照する MBean の属性を使用することで、モデルをひととおり移動することができます。たとえば、EndpointControlMBean にアクセスするには、MessageBrokerControlMBean から移動を始め、`Endpoints` 属性を取得します。この属性には、管理ブローカーに登録されているすべてのエンドポイントの ObjectName が格納されています。任意の管理クライアントでこの属性を使用することで、ルートの MessageBrokerControlMBean に対応する単一の ObjectName を生成でき、そこからシステム内の他の MBean に移動して管理できます。他の MBean の ObjectName を取得する必要はありません。

ランタイム MBean のモデルは階層的に構成されていますが、システムに登録されている Mean はそれぞれ MBean ObjectName を持っており、必要に応じて取得またはクエリできます。ランタイム MBean は、登録、ルックアップ、およびクエリが簡素化されるように、ObjectName の命名規則に従っています。MBean インスタンスの ObjectName は次のように定義されています。

```
{domain}:{key}={value}[,{keyN}={valueN}]*
```

MBean インスタンスを一意に識別するキーと値のペアをいくつでも追加できます。

すべてのランタイム MBean は flex.run-time ドメインに属します。アプリケーション名が使用できる場合は、`flex.runtime.`‹アプリケーション名› という形式でもこのドメインに含まれます。

ランタイム MBean の ObjectName にはそれぞれ次のキーが含まれています。

キー	説明
type	MBean によって管理されているリソースの型名。MessageBrokerControlMBean は flex.messaging.MessageBroker を管理しているので、その型は MessageBroker になります。
id	MBean によって管理されているリソースの `id` 値。`name` または `id` がリソースで使用できない場合は、次の手順に従って `id` が作成されます。 `id = {type} + N` ここで、N は、この型のインスタンスに対する増分値です。

ObjectName には追加のオプションキーを含めることもできます。

ランタイム MBean の詳細については、公開されている Flex データサービス Javadoc マニュアルを参照してください。Javadoc には flex.management.jmx.MBeanServerGateway クラスに関する説明もあります。実行時の監視および管理コンソールは、このクラスを Remoting Service 宛先として使用します。

カスタム ServiceAdapter クラス用のカスタム MBean の作成

カスタム ServiceAdapter 用のメトリクスや管理 API を公開するためにカスタム MBean を記述する際、対応する MBean を構成できるように、ServiceAdapter クラスでは次のメソッドが定義されています。

```
public void setupAdapterControl(Destination destination);
```

カスタム ServiceAdapter コントロールの MBean では、flex.management.runtime.messaging.services.ServiceAdapterControl を拡張する必要があります。MBean ではカスタム MBean インターフェイスを実装する必要があり、そのインターフェイスではさらに flex.management.runtime.messaging.services.ServiceAdapterControlMBean を拡張する必要があります。これにより、コアの Flex データサービス MBean の階層にカスタム MBean を追加できるようになります。

次のコード例は、setupAdapterControl() メソッドをカスタム ServiceAdapter に実装する方法を示しています。ここで、controller は CustomAdapterControl (カスタム MBean の実装クラス) 型のインスタンス変数です。

```
public void setupAdapterControl(Destination destination) {
   controller = new CustomAdapterControl(getId(), this,
   destination.getControl());
   controller.register();
   setControl(controller);
}
```

カスタムアダプタでは、controller (MBean インスタンス) のメソッドを呼び出すかプロパティを更新することによって、対応する MBean に格納されているメトリクスを更新できます。カスタム MBean には、そのコンストラクタの中でカスタムアダプタへの参照が渡されます。protected ServiceAdapter serviceAdapter インスタンス変数を使用することによって、カスタム MBean の中でこの参照にアクセスできます。こうすることで、MBean から対応するサービスアダプタをクエリしてその状態を調べることや、アダプタのメソッドを呼び出すことが可能になります。

カスタムのエラー処理の使用

メッセージ例外の一部として追加の情報をクライアントに返す場合は、flex.messaging.MessageException クラスの extendedData プロパティを使用できます。このプロパティは HashMap であり、失敗が発生した場合に追加のデータをクライアントに返す手段として柔軟に使用できます。インストールされている Flex データサービスの docs ディレクトリにある Javadoc マニュアルでは、flex.messaging.MessageException クラスに関する説明があります。

メモ

Flex データサービスの直列化機能は、Bean における任意の Throwable 型の直列化機能を備えています。これにより、クライアントに送信する必要のあるプロパティについて、getter で独自の Throwable 例外をスローするオプションが提供されます。

次の例は、データを例外に追加する Java のテストクラスを示しています。

```
package errorhandling;

import java.util.HashMap;
import java.util.Map;
import flex.messaging.MessageException;

public class TestException {

public String generateMessageExceptionWithExtendedData(String extraData)
{
   MessageException me = new MessageException("Testing extendedData.");
   Map extendedData = new HashMap();
      // このクラスを呼び出すメソッドは、このマップ内の "extraData" スロットを// 受け取ります。
      extendedData.put("extraData", extraData);
      me.setExtendedData(extendedData);
```

```
     me.setCode("999");
     me.setDetails("Some custom details.");
     throw me;
     }

}
```

次の例は、例外を追加のデータと共に生成する ActionScript メソッドを示しています。

```
<?xml version="1.0"?>
<mx:Application
     xmlns:mx="http://www.adobe.com/2006/mxml"
     creationComplete="run()">

  <mx:RemoteObject
    destination="myDest"
    id="myException"
    fault="onServiceFault(event)"
/>
  <mx:Script>
  <![CDATA[
    import mx.rpc.events.*;
    import mx.messaging.messages.*;"

    public var data : String = "Extra data.";
    public var actualData : String;

    public function onServiceFault(event:FaultEvent):void {
      actualData = ErrorMessage(event.message).extendedData.extraData;
    }

    public function run_exception():void {
      var call : Object =
        myException.generateMessageExceptionWithExtendedData(data);
    }

    public function run():void {
      run_exception();
    }
  ]]>
  </mx:Script>
</mx:Application>
```

データサービスのクラスのロードについて

通常、Flex データサービスでは、そのクラスがロードされる際、アプリケーションサーバーによって提供されているデフォルトのクラスローダーを使用して、標準の J2EE Web アプリケーションと同じ方法でロードされます。ただし、唯一の例外として Web 層コンパイラの場合があります。Web 層コンパイラはブートストラップクラスローダーを使用して、web.xml 内で `flex.class.path` という名前を持つ context-param の値から、JAR ファイルをロードします。デフォルトでは、Flex Web 層コンパイラの jar ファイルを WEB-INF/flex/jars ディレクトリで検索するように設定されています。この処理は、Web 層のクラスを Web アプリケーションのクラスローダー内の潜在的な競合から避けるための試みです。ただし、Flex データサービスの実際のサービスはこの classloader を使用することはありません。J2EE Web アプリケーションではよくあることですが、独自のバージョンの JAR ファイルを使用する場合に、アプリケーションサーバーのソースパスに既に存在するファイルと競合する、という問題に直面することがあります。問題のトラブルシューティングを行う際は、必ず最初に、コードが Flex データサービスのないアプリケーションで動作することを確認してください。

Web 層コンパイラを使用する予定がない場合は、Web 層コンパイラなしでデータサービスを使用できます。詳細については、1275 ページの「Web 層コンパイラなしでのデータサービスの使用」を参照してください。

Web 層コンパイラのクラスのノード

web.xml ファイルでは、FlexMxmlServlet のサーブレットクラスは flex.bootstrap.BootstrapServlet クラスであり、init-param では、MXML のコンパイルを処理するためにロードされるサーブレットとして flex2.server.j2ee.MxmlServlet が設定されています。コンパイラは WEB-INF/flex/jars にある mxmlc.jar および asc.jar ファイル内に存在します。WEB-INF/flex/jars からは、Web アプリケーションのクラスローダーではなく Flex Web 層のクラスローダーにアクセスできます。

次の例は、web.xml ファイルでの FlexMxmlServlet の定義を示しています。

```
<servlet>
  <servlet-name>FlexMxmlServlet</servlet-name>
  <display-name>MXML Processor</display-name>
  <description>Servlet wrapper for the Mxml Compiler</description>
  <servlet-class>flex.bootstrap.BootstrapServlet</servlet-class>
  <init-param>
    <param-name>servlet.class</param-name>
    <param-value>flex2.server.j2ee.MxmlServlet</param-value>
  </init-param>
  <init-param>
    <param-name>webtier.configuration.file</param-name>
    <param-value>/WEB-INF/flex/flex-webtier-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

BootstrapServlet では独自のクラスパスを使用しています。このクラスパスは web.xml ファイルで flex.class.path という context-param で指定されています。次の例では、Web 層コンパイラをロードする BootstrapServlet において、flex/hotfixes および the flex/jars という 2 つのディレクトリでクラスを検索することを指定しています。

```
<context-param>
  <param-name>flex.class.path</param-name>
  <param-value>/WEB-INF/flex/hotfixes,/WEB-INF/flex/jars</param-value>
</context-param>
```

データサービスのクラスのロード

MessageBrokerServlet は Flex データサービスメッセージブローカーを作成します。このサーブレットは、RTMP サーバーを含むすべてのサービスとエンドポイントを開始します。このサーブレットは標準のクラスローダーによってロードされます。標準のクラスローダーは、アプリケーションサーバーがその Web アプリケーション用のクラスをロードするために使用します。アプリケーションサーバーは、特殊なブートストラップクラスローダーを使用する BootstrapServlet は使用しません。標準の J2EE Web アプリケーションの場合と同様に、Flex データサービスは Web アプリケーションクラスローダーを使用して WEB-INF/lib ディレクトリからクラスをロードします。MXML コンパイラまたは Web 層キャッシュへのアクセス時にはクラスローダーのコンテキスト切り替えが発生することがありますが、Flex データサービスのサービスへのアクセス時にはコンテキスト切り替えは発生しません。

次の例は、web.xml ファイルでの MessageBrokerServlet の定義を示しています。

```
<servlet>
  <servlet-name>MessageBrokerServlet</servlet-name>
  <display-name>MessageBrokerServlet</display-name>
  <servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>
  <init-param>
    <param-name>services.configuration.file</param-name>
    <param-value>/WEB-INF/flex/services-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Web 層コンパイラなしでのデータサービスの使用

データサービスは Web 層コンパイラがなくても使用できます。Web 層コンパイラを使用しない場合、WEB-INF/flex ディレクトリで必要になるファイルは、license.properties ファイルと、services-config.xml、proxy-config.xml、messaging-config.xml、および data-management-config.xml ファイルになります。

MXML コンパイラを使用しない場合は、flex-bootstrap.jar ファイルを WEB-INF/lib ディレクトリから削除できます。また、FlexMxmlServlet のサーブレット定義も web.xml ファイルから削除してください。

ファクトリメカニズムの使用

Flex の Remoting Service 宛先と Data Management Service 宛先ではいずれも、Flex クライアントとの統合のために記述した Java クラスが使用されます。デフォルトでは、Flex データサービスによってこれらのインスタンスが作成されます。これらがアプリケーションのスコープを持つコンポーネントの場合は、宛先の名前を属性名として、ServletContext 属性空間に格納されます。セッションのスコープを持つコンポーネントを使用する場合には、それらのコンポーネントは同様に宛先名を属性として、FlexSession に格納されます。宛先で `attribute-id` エレメントを指定した場合は、どの属性にコンポーネントが格納されるのかを制御できます。そのため、複数の宛先で同じインスタンスを共有できます。

次の例は、`attribute-id` エレメントのある宛先定義を示しています。

```
<destination id="WeatherService">
  <properties>
    <source>weather.WeatherService</source>
    <scope>application</scope>
    <attribute-id>MyWeatherService</attribute-id>
  </properties>
</destination>
```

この例では、Flex によってクラス weather.WeatherService のインスタンスが作成され、MyWeatherService という名前を持つ ServletContext の一連の属性に格納されます。同じ `attribute-id` 値と同じ Java クラスを持つ別の宛先を定義する場合は、同じコンポーネントインスタンスが使用されます。

Flex データサービスが備えているファクトリメカニズムを使用すると、独自のコンポーネント作成および保守システムを Flex にプラグインでき、コンポーネントを独自の名前空間に格納する EJB や Spring などのシステムと統合できるようになります。これを実現するには、flex.messaging.FlexFactory インターフェイスを実装するクラスを提供します。このクラスを使用して、特定の宛先用に設定されたコンポーネントに対応する FactoryInstance を作成します。そして、この FactoryInstance を使用して、与えられた要求に対して使用される特定のコンポーネントを参照します。実装された FlexFactory からは、Flex データサービス設定ファイルの設定属性にアクセスでき、FlexSession オブジェクトや ServletContext オブジェクトにもアクセスできます。詳細については、公開されている Flex データサービス Javadoc のマニュアルで、FlexFactory クラスに関する説明を参照してください。

FlexFactory クラスを実装した後、宛先でそのクラスを使用できるように設定できます。これには、次の例のように、services-config.xml ファイルの `factories` エレメントに `factory` エレメントを配置します。Flex 対応 Web アプリケーションごとに 1 つの FlexFactory インスタンスが作成されます。このインスタンスには独自の設定プロパティを持たせることができますが、この例ではファクトリを設定するのに必須のプロパティはありません。

```
<factories>
  <factory id="spring" class="flex.samples.factories.SpringFactory" />
</factories>
```

指定した factory エレメントごとに 1 つの FlexFactory インスタンスが作成されます。宛先定義の properties セクションにある factory エレメントを使用して ID でファクトリを指定する宛先ごとに、この 1 つのグローバルな FlexFactory の実装が使用されます。たとえば、remoting-config.xml ファイルに次のような宛先を持たせることができます。

```
<destination id="WeatherService">
  <properties>
    <factory>spring</factory>
    <source>weatherBean</source>
  </properties>
</destination>
```

起動時に factory エレメントが検出されると、FlexFactory.createFactoryInstance() メソッドが呼び出されます。このメソッドは、source 値や設定で予測されるその他の属性を取得します。ConfigMap パラメータからアクセスする属性は予測済みとしてマークされ、設定エラーは発生しないので、この方法でデフォルトの Flex 設定を拡張できます。この宛先用のコンポーネントのインスタンスが必要な場合は、FactoryInstance.lookup() メソッドが呼び出され、この宛先用の個々のインスタンスが取得されます。

オプションで、ファクトリインスタンスで追加の属性を受け取ることもできます。これには 2 つの方法があります。最初にファクトリを定義する場合は、追加の属性をファクトリ定義の一部として提供できます。そのファクトリのインスタンスを定義する際、ファクトリインスタンスの作成に使用される宛先定義に、独自の属性を追加することもできます。

次の例では、ファクトリ定義に追加された属性をボールド体のテキストで示しています。

```
<factories>
  <factory id="myFactoryId" class="myPackage.MyFlexFactory">
    <properties>
      <myfactoryattributename>
        myfactoryattributevalue
      </myfactoryattributename>
    </properties>
  </factory>
</factories>
```

この種の設定の使用例としては、Spring Framework Java アプリケーションフレームワークの統合があります。統合によって、Spring ファクトリのすべてのコンポーネントを参照するのに使用される Spring コンテキストを初期化するためのデフォルトのパスが、その Spring ファクトリに提供されます。FlexFactory を実装するクラスでは、たとえば次の例のように、myfactoryattributename の値を configMap パラメータから取得して FlexFactory インターフェイスの initialize() メソッドに渡すための呼び出しを含めます。

```
public void initialize(String id, ConfigMap configMap){
  System.out.println("**** MyFactory initialized with: " +
  configMap.getPropertyAsString("myfactoryattributename", "not set"));
}
```

initialize() メソッドはストリング値を取得します。メソッドの第 1 パラメータは属性の名前で、第 2 パラメータはデフォルト値が設定されていない場合に使用するデフォルト値です。設定マップ内のプロパティを取得するための各種の呼び出しに関する詳細については、公開されている Flex データサービス Javadoc のマニュアルで、flex.messaging.config.ConfigMap クラスの説明を参照してください。

次の例のように、各ファクトリインスタンスでは、そのファクトリインスタンスが定義されるときに使用される設定属性を追加できます。

```
<destination id="myDestination">
  <properties>
    <source>mypackage.MyRemoteClass</source>
    <factory>myFactoryId</factory>
    <myfactoryinstanceattribute>
      myfoobar2value
    </myfactoryinstanceattribute>
  </properties>
</destination>
```

次の例のように、createFactoryInstance() メソッドでは、FlexFactory の実装の一部として、ファクトリのそのインスタンスの属性にアクセスします。

```
public FactoryInstance createFactoryInstance(String id, ConfigMap properties) {
  System.out.println("**** MyFactoryInstance instance initialized with
  myfactoryinstanceattribute=" +
  properties.getPropertyAsString("myfactoryinstanceattribute", "not set"));
….
}
```

第 44 章

RPC コンポーネントについて

44

このトピックでは、Adobe Flex RPC (リモートプロシージャコール) コンポーネントについて紹介します。Flex RPC コンポーネントはサービス指向アーキテクチャ (SOA) をベースとしています。RPC コンポーネントによって、サーバーサイドの RPC サービスとやり取りし、データを自分のアプリケーションに提供できるようになります。

データには、HTTP の GET または POST (HTTP サービス)、SOAP (Web サービス)、あるいは Java オブジェクト (リモートオブジェクトサービス) を通じてアクセスできます。一般に、HTTP サービスは REST スタイルの Web サービスとも呼ばれます。REST は Representational State Transfer の略で、分散ハイパーメディアシステムのアーキテクチャスタイルです。REST の詳細については、www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm を参照してください。

典型的な Flex アプリケーションでは、RPC コンポーネントから RPC サービスにデータを入力として送信します。RPC サービスが実行されると、要求元の RPC コンポーネントに結果データが返されます。

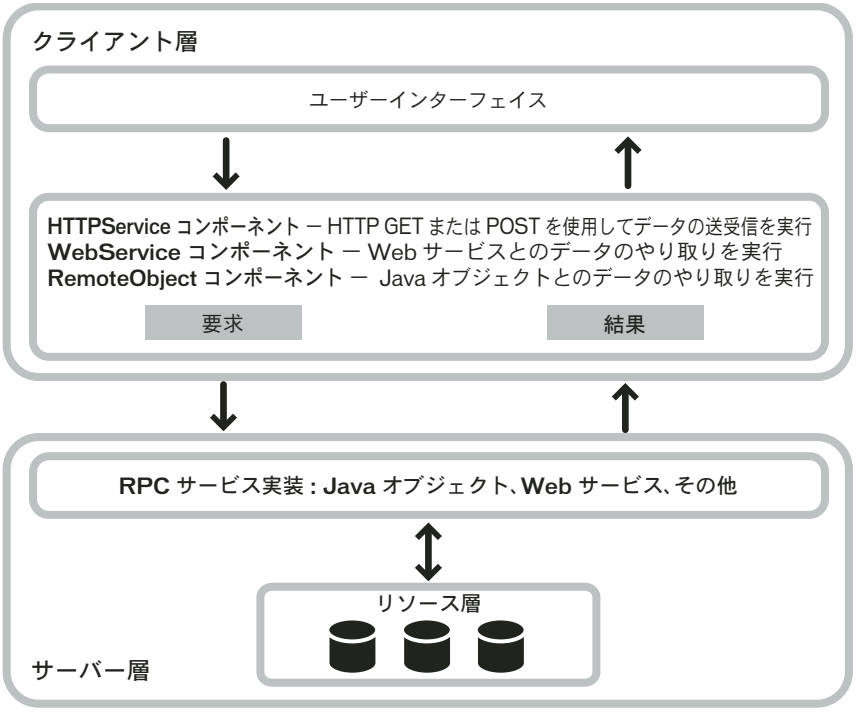
目次

RPC コンポーネントについて

Flex RPC コンポーネントは MXML または ActionScript で使用でき、リモートオブジェクトサービス、Web サービス、および HTTP サービスの、3 種類の RPC サービスを操作できます。このセクションでは、各種の RPC コンポーネントと、それらをどのような場合に使用するのかについて簡単に説明します。

Flex フレームワークを Adobe Flex データサービスなしで使用する場合は、RPC サービスに直接接続します。Flex データサービスを使用する場合は、宛先にも接続できます。宛先は、対応するサーバーサイドの設定を持ち、サーバープロキシ経由でのアクセスを提供する、RPC サービスです。

次の図は、RPC コンポーネントと RPC サービスとのやり取りを簡単に示したものです。



次に、RPC サービスへのアクセスを必要とするアプリケーションを作成する際に検討する必要がある重要事項をいくつか示します。

1. 使用するサービスのタイプはどれが最適なのか。詳細については、1281 ページの「RemoteObject コンポーネント」、1282 ページの「WebService コンポーネント」、および 1282 ページの「HTTPService コンポーネント」を参照してください。
2. サービスに直接接続する必要があるか、それとも宛先を経由する必要があるか。詳細については、1288 ページの「RPC コンポーネントの宣言」を参照してください。
3. サービスにデータを渡す最適な方法は何か。詳細については、1293 ページの「サービスの呼び出し」を参照してください。
4. サービスから返されたデータ結果をどのように処理するか。詳細については、1306 ページの「サービス結果の処理」を参照してください。
5. RPC コンポーネントを使用するコードをどのようにデバッグできるか。詳細については、1264 ページの「サーバーサイドのサービスログ機能の設定」を参照してください。
6. どのようなセキュリティ対策を実装できるか、または実装する必要があるか。詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。

RemoteObject コンポーネント

RemoteObject コンポーネントを使用すると、Java オブジェクトなどのサーバーサイドオブジェクトを Web サービスとして手動で設定しなくても、それらのオブジェクトのメソッドにアクセスできます。WebService コンポーネントや HTTPService コンポーネントと異なり、RemoteObject コンポーネントは、Flex データサービスとの組み合わせ、または Macromedia ColdFusion MX 7.0.2 および Remoting Update との組み合わせでのみ、使用できます。アクセスするオブジェクトは、Flex データサービスの一部である Flex サービス設定ファイルで Remoting Service 宛先として設定します。RemoteObject コンポーネントは MXML または ActionScript で使用できます。

オブジェクトがまだ Web サービスとしてパブリッシュされていない場合や、環境で Web サービスを使用していない場合、または Web サービスではなく Java オブジェクトを使用したい場合などに、WebService コンポーネントの代わりに RemoteObject コンポーネントを使用できます。RemoteObject コンポーネントを使用して、Flex データサービスまたは ColdFusion MX 7.0.2 Web アプリケーションのソースパスにあるローカルの Java オブジェクトに接続できます。

RemoteObject タグを使用する場合は、データがアプリケーションとサーバーサイドオブジェクト間でバイナリの AMF (Action Message Format) 形式で受け渡されます。

RemoteObject コンポーネントの使用の詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

WebService コンポーネント

WebService コンポーネントを使用すると Web サービスにアクセスできます。Web サービスは、一般にオペレーションと呼ばれるメソッドを備えたソフトウェアモジュールです。Web サービスインターフェイスは XML を使用して定義されます。Web サービスは、さまざまなプラットフォームで実行されるソフトウェアモジュールが互いに通信を行うための規格に準拠しています。Web サービスの詳細については、World Wide Web Consortium の Web サイト (www.w3.org/2002/ws/) で、Web サービスのセクションを参照してください。

Flex アプリケーションは、URL として提供される WSDL (Web Services Description Language) ドキュメントでインターフェイスが定義された Web サービスとやり取りすることができます。WSDL は、Web サービスで解釈可能なメッセージ、そのメッセージに対する Web サービスの応答形式、Web サービスがサポートするプロトコル、およびメッセージの送信先を記述するために使用される標準形式です。

Flex では WSDL 1.1 がサポートされています。WSDL 1.1 の詳細については、www.w3.org/TR/wsdl を参照してください。また、Flex では RPC エンコードされた Web サービスとドキュメント / リテラルの Web サービスの両方がサポートされています。

Flex アプリケーションでは、SOAP メッセージの形式を持ち、HTTP で伝達される、Web サービスの要求および結果がサポートされています。SOAP では XML ベースの形式が定義されており、これを使用することで、構造化され型指定された情報を Flex アプリケーションなどの Web サービスクライアントと Web サービスの間で交換できます。

Web サービスが標準として確立されている環境の場合は、WebService コンポーネントを使用して SOAP 準拠の Web サービスに接続できます。また、エンタープライズ環境の中に置かれているものの、必ずしも Flex Web アプリケーションのソースパスでアクセスできないオブジェクトを使用する場合にも、WebService コンポーネントが役立ちます。

WebService コンポーネントの使用の詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

HTTPService コンポーネント

HTTPService コンポーネントを使用すると、HTTP の GET、POST、HEAD、OPTIONS、PUT、TRACE、または DELETE 要求を送信でき、HTTP 応答から返されたデータを Flex アプリケーションに取り込むことができます。Flex ではマルチパートフォームの POST はサポートされていません。

HTTP 要求を受け入れて応答を送信する任意の HTTP URI を、HTTP サービスにすることができます。一般に、この種のサービスは REST スタイルの Web サービスとも呼ばれます。REST は Representational State Transfer の略で、分散ハイパーメディアシステムのアーキテクチャスタイルです。REST の詳細については、www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm を参照してください。

HTTPService コンポーネントは、SOAP Web サービスやリモートオブジェクトサービスと同じ機能を提供できない場合に役立つオプションです。たとえば、Web サービスや Remoting Service 宛先として利用できない、JSP (JavaServer Page)、サーブレット、および ASP ページと、HTTPService コンポーネントを使用してやり取りすることができます。

HTTPService コンポーネントを使用して CGI のようなやり取りを実行し、HTTP の GET、POST、HEAD、OPTIONS、PUT、TRACE、または DELETE を使用して、指定した URI に要求を送信できます。HTTPService オブジェクトの `send()` メソッドを呼び出すと、指定した URI に対して HTTP 要求が実行され、HTTP 応答が返されます。指定した URI に引数を渡すこともできます。

例 : RPC コンポーネント

次の例は RemoteObject コンポーネントの MXML コードです。このコードは、Remoting Service 宛先に接続し、Button コントロールの `click` イベントの中で要求をデータソースに送信して、結果データを TextArea コントロールの `text` プロパティに表示します。TextArea コントロールの中括弧 ({ }) 内の値がバインディング式で、この式によってサービスの結果データが TextArea コントロールの `text` プロパティにコピーされます。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCIntroExample1.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Connect to a service destination.-->
    <mx:RemoteObject id="remoteService" destination="myRO"/>

    <!-- Provide input data for calling the service. -->
    <mx:TextInput id="inputText"/>

    <!-- Call the web service, use the text in a TextInput control as input
data.-->
    <mx:Button click="remoteService.getData(inputText.text)"/>

    <!-- Display results data in the user interface. -->
    <mx:TextArea text="{remoteService.getData.lastResult.prop1}"/>
</mx:Application>
```

詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

Flex RPC サービス機能と他のテクノロジーとの比較

Flex でのデータソースおよびデータの操作方法は、JSP、ASP、ColdFusion など他の Web アプリケーション環境の場合とは異なります。また、Flex アプリケーションでのデータアクセスは、Flash Professional で作成されたアプリケーションでのデータアクセスとも大きく異なります。このセクションでは、これらの違いについて説明します。

クライアントサイドの処理とサーバーサイドの処理

JSP とサーブレット、ASP、または CFML を使用して作成された HTML テンプレートとは異なり、Flex アプリケーションのファイルはバイナリの SWF ファイルにコンパイルされてから、クライアントに送信されます。Flex アプリケーションから外部サービスに要求を行う場合は、SWF ファイルは再コンパイルされず、ページの更新も不要です。

次の例は、Web サービスを直接呼び出す MXML コードです。この例では Proxy Service を経由していません。`WebService.useProxy` プロパティのデフォルト値は `false` であるため、タグ内で宣言されていません。ユーザーが Button コントロールをクリックすると、クライアントサイドのコードによって Web サービスが呼び出され、ページを更新することなく、結果データがバイナリの SWF ファイルに返されます。返された結果データは、アプリケーションでダイナミックコンテンツとして使用できます。次の例は、WebService コンポーネントの `wsdl` プロパティですが、サービスの呼び出し方法は、サービスに直接接続する場合でも宛先を経由する場合でも同じです。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCIntroExample2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <!-- Declare a WebService component (the specified WSDL URL is not
functional). -->
    <mx:WebService id="WeatherService"
        wsdl="/ws/WeatherService?wsdl"/>

    <mx:Button label="Get Weather"
        click="WeatherService.GetWeather(input.text);"/>

    <mx:TextInput id="input"/>
</mx:Application>
```

次の例は、JSP カスタムタグを使用して Web サービスを呼び出す JSP コードです。ユーザーがこの JSP を要求すると、Web サービス要求はクライアント上ではなくサーバー上で行われ、結果を使用して HTML ページが生成されます。アプリケーションサーバーは HTML ページ全体を再生成してから、ユーザーの Web ブラウザに送信します。

```
<%@ taglib prefix="web" uri="webservicetag" %>

<% String str1="BRL";
```

```
String str2="USD";%>

<!-- Web サービスを呼び出します。-->
<web:invoke
   url="http://www.itfinity.net:8008/soap/exrates/default.asp"
   namespace="http://www.itfinity.net/soap/exrates/exrates.xsd"
   operation="GetRate"
   resulttype="double"
   result="myresult">
   <web:param name="fromCurr" value="<%=str1%>"/>
   <web:param name="ToCurr" value="<%=str2%>"/>
</web:invoke>

<!-- Web サービスの結果を表示します。-->
<%= pageContext.getAttribute("myresult") %>
```

データソースへのアクセス

Flex が他の Web アプリケーションテクノロジーと異なるもう 1 つの特徴は、Flex ではデータソースと直接通信しないということです。Flex では、サービスコンポーネントを使用してサーバーサイドのサービスに接続し、データソースを操作します。

次の例では、ColdFusion ページからデータソースに直接アクセスします。

```
...
<CFQUERY DATASOURCE="Dsn"
   NAME="myQuery">
   SELECT * FROM table
</CFQUERY>
...
```

Flex で同様の機能を実現するには、HTTPService、WebService、または RemoteObject コンポーネントを使用してサーバーサイドオブジェクトを呼び出し、データソースから結果を返します。

Flash Professional のデータ管理

Flash Professional と Flex 2 では、データ管理アーキテクチャが異なります。これらのアーキテクチャは、それぞれのオーサリング環境とユーザーコミュニティの要件を満たすように開発されています。Flash Professional では、Flash Professional のオーサリング環境で使用するための一連のデータコンポーネントが提供されています。コンポーネントの一部の機能は、Flex の機能と共通していますが、ベースとなっているアーキテクチャは異なります。

Flash Professional は、Flash Professional のコンポーネントと連動する独自のデータバインディング機能も備えていますが、これは Flex のデータバインディングとは完全に異なる機能です。

第 45 章

RPC コンポーネントの使用

45

Adobe Flex 2 ではサービス指向アークテクチャがサポートされています。このアークテクチャでは、Flex アプリケーションは複数の種類のサービスを呼び出し、それらのサービスから応答を受信することによって、リモートのデータソースとやり取りします。典型的な Flex アプリケーションでは、クライアントサイドの RPC コンポーネントからリモートサービスに非同期要求を送信します。すると、結果データがクライアントサイドコンポーネントに返されます。

このトピックでは、Flex RPC コンポーネントの使用方法について説明します。RPC コンポーネントの概要については、1279 ページ、第 44 章の「RPC コンポーネントについて」を参照してください。

目次

RPC コンポーネントの宣言

RPC サービスに接続する RPC コンポーネントは MXML または ActionScript で宣言できます。Adobe Flex データサービスを使用しない場合は、WebService または HTTPService コンポーネントを使用して RPC サービスに直接接続できます。

メモ	RemoteObject コンポーネントを使用するには、Flex データサービス、または Macromedia ColdFusion MX 7.0.2 と Remoting Update の組み合わせが必要です。

Flex データサービスを使用する場合は、RPC サービスに直接接続するか、あるいは、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしているファイルで定義された宛先に接続するかを、選択できます。宛先定義の名前には、サーバープロキシ経由での RPC サービスへのアクセスを提供するサービス設定が付けられます。宛先は、呼び出す必要のある、実際のサービスまたはオブジェクトです。RemoteObject コンポーネントの宛先は Java オブジェクトになります。HTTPService コンポーネントの宛先は、JSP ページ、または HTTP 経由でアクセスされる別のリソースになります。WebService コンポーネントの宛先は SOAP 準拠の Web サービスになります。

宛先定義により、RPC サービスの集中管理が可能になります。また、宛先定義により、基本認証またはカスタム認証を使用して宛先へのアクセスをセキュリティ保護できるようになります。宛先とのデータの受け渡しを行うチャネルとして、セキュアなチャネルを含む複数の異なるトランスポートチャネルから選択できます。さらに、サーバーサイドのログ機能を使用して RPC サービストラフィックのログを記録することもできます。

オプションで、HTTPService コンポーネントまたは WebService コンポーネントを使用してデフォルト宛先に接続することも可能です。

このセクションでは、MXML でのサービス接続の宣言と宛先の設定に関する基本的な事項について説明します。MXML および ActionScript でのサービスの呼び出しに関する詳細については、1293 ページの「サービスの呼び出し」を参照してください。宛先の設定の詳細については、1329 ページ、第 46 章の「RPC サービスの設定」を参照してください。

サーバーサイド設定のないサービスの使用

Flex データサービス宛先を設定せずに、RPC コンポーネントを使用して HTTP サービスおよび Web サービスに接続できます。これには、コンポーネントの `destination` プロパティを設定するのではなく、HTTPService コンポーネントの `url` プロパティまたは WebService コンポーネントの `wsdl` プロパティを設定します。

RPC コンポーネントの `useProxy` プロパティをデフォルト値の `false` に設定した場合、コンポーネントは `url` または `wsdl` プロパティの値に基づいて RPC サービスと直接通信します。Flex データサービスを使用しない場合は、`useProxy` プロパティの値を `false` にする必要があります。この方法でサービスに接続する場合は、次のうち少なくとも 1 つが成り立つ必要があります。

- RPC サービスが Flex アプリケーションと同じドメイン内にあること。
- アプリケーションのドメインからのアクセスを許可する "crossdomain.xml" (クロスドメインポリシー) ファイルが、RPC サービスのホストとなっている Web サーバーにインストールされていること。クロスドメインポリシーの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 4 章の「Flex セキュリティの適用」を参照してください。

HTTPService または WebService コンポーネントの useProxy プロパティを true に設定し、url または wsdl プロパティを設定して、destination プロパティを設定しない場合、コンポーネントではデフォルト宛先が使用されます。デフォルト宛先は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしている proxy-config.xml などのファイルの <proxy-service> セクションで設定します。この場合は Flex データサービスが必要になり、設定ファイルの defaultHTTP または defaultHTTPS 宛先に、それぞれコンポーネントの url または wsdl プロパティの値に一致する URL パターンを含める必要があります。defaultHTTPS 宛先は url または wsdl プロパティで HTTPS URL を指定した場合に使用されます。

次の例は、RPC サービスを直接参照する HTTPService コンポーネントと WebService コンポーネントを宣言するための MXML タグです。id プロパティは、サービスを呼び出してサービス結果を処理するために必要になります。これらの例では useProxy プロパティをタグで設定していません。コンポーネントでは、useProxy のデフォルト値である false が使用され、サービスに直接接続します。この設定は Flex データサービスを使用しない場合に必要になります。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCNoServer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:HTTPService
        id="yahoo_web_search"
        url="http://api.search.yahoo.com/WebSearchService/V1/webSearch"/>

    <mx:WebService
        id="macr_news"
        wsdl="http://ws.invesbot.com/companysearch.asmx?wsdl"/>
</mx:Application>
```

次の例は、対応するサーバーサイド設定のない HTTPService コンポーネントと WebService コンポーネントを宣言するための ActionScript コードです。これらの例では useProxy プロパティを明示的に設定していません。コンポーネントでは、useProxy のデフォルト値である false が使用され、サービスに直接接続します。この設定は Flex データサービスを使用しない場合に必要になります。コンポーネントを宣言するためのコードは ActionScript のメソッドに配置します。

メモ

HTTP サービスや Web サービスから正しい HTTP ステータスコードが確実に返されるようにするためには、Flex データサービスの一部である Flex プロキシを使用する必要があります。Web ブラウザでは、200 以外のステータスコードがサービスから返された場合、Adobe Flash Player は応答のボディを読み取ることができません。ステータスコードが 500 で、ボディに失敗が含まれている場合、その失敗を取得する方法はありません。この場合、プロキシは失敗のステータスコードを強制的に 200 にすることによって問題を回避します。Player は応答のボディと失敗をそのまま一緒に渡します。

サーバーサイド宛先のある RPC コンポーネントの使用

RPC コンポーネントの destination プロパティでは、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしているファイルで設定された、Flex データサービス 宛先を参照します。宛先では、RPC サービスのクラスまたは URL、使用するトランスポートチャネル、RPC サービスへのアクセスに使用するアダプタ、およびセキュリティ設定を、それぞれ指定します。宛先の設定の詳細については、1329 ページ、第 46 章の「RPC サービスの設定」を参照してください。

宛先への接続を MXML タグで宣言するには、3 つの RPC サービスタグのいずれかで、id プロパティと destination プロパティを設定します。id プロパティは、サービスを呼び出してサービス結果を処理するために必要になります。

次の例は、MXML タグにおける RemoteObject、HTTPService、WebService の各コンポーネントの簡単な宣言です。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCServerMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <!-- Declare a RemoteObject component. -->
    <mx:RemoteObject
        id="employeeRO"
        destination="SalaryManager"
        useProxy="true"/>

    <!-- Declare an HTTPService component. -->
    <mx:HTTPService
        id="employeeHTTP"
        destination="SalaryManager"
        useProxy="true"/>

    <!-- Declare a WebService component. -->
    <mx:WebService
        id="employeeWS"
        destination="SalaryManager"
        useProxy="true"/>
</mx:Application>
```

次の例は、ActionScript における RemoteObject、HTTPService、WebService の各コンポーネントの簡単な宣言です。これらの例のコードは ActionScript のメソッドの中に配置します。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCNoServerAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            // Import required packages.
            import mx.rpc.remoting.RemoteObject;
            import mx.rpc.http.HTTPService;
            import mx.rpc.soap.WebService;
```

```
            private var employeeRO:RemoteObject;
            private var employeeHTTP:HTTPService;
            private var employeeWS:WebService;

            public function declareServices():void{
                // Create a RemoteObject component.
                employeeRO = new RemoteObject();
                employeeRO.destination = "SalaryManager";
                // Create an HTTPService component.
                employeeHTTP = new HTTPService();
                employeeHTTP.destination = "SalaryManager";
                // Create a WebService component.
                employeeWS = new WebService();
                employeeWS.destination = "SalaryManager";
            }
        ]]>
    </mx:Script>

</mx:Application>
```

宛先の設定

宛先は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしているファイルのサービス定義で設定します。宛先では、RPC サービスのクラスまたは URL、使用するトランスポートチャネル、サービスへのアクセスに使用するアダプタ、および宛先をセキュリティ保護するための設定を、それぞれ指定します。

RemoteObject 宛先は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしている remoting-config.xml などのファイルにある、Remoting Service 定義で設定します。HTTP 宛先および Web サービス宛先は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしている proxy-config.xml などのファイルにある、Proxy Service 定義で設定します。

Proxy Service とそのアダプタは、アプリケーションから別のドメイン上にある HTTP サービスや Web サービスにアクセスできるようにする機能を提供します。また、Proxy Service によって、特定の URL および URL パターンへのアクセスが制限され、セキュリティが提供されます。Flex データサービスがない場合や、Proxy Service によって提供される機能を必要としない場合は、Proxy Service を迂回できます。プロキシを迂回するには、HTTPService コンポーネントまたは WebService コンポーネントの `useProxy` プロパティをデフォルト値の `false` に設定します。Proxy Service の詳細については、1329 ページ、第 46 章の「RPC サービスの設定」を参照してください。

基本認証では、Web アプリケーションコンテナの標準の J2EE 基本認証が利用されます。基本認証の要素は、Flex Web アプリケーションの "services-config.xml" ファイルおよび web.xml ファイルで設定します。カスタム認証についてはすべて、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしているファイルで設定します。サービス宛先のセキュリティ保護の詳細については、1329 ページ、第 46 章の「RPC サービスの設定」を参照してください。

次の例は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしている remoting-config.xml などのファイルにおける、Remoting Service に対するサーバーサイドの基本設定です。RemoteObject コンポーネントは Remoting Service 宛先に接続します。HTTP サービスおよび Web サービスの設定もこの例とほぼ同様です。

```
<service id="remoting-service"
  class="flex.messaging.services.RemotingService"
  messageTypes="flex.messaging.messages.RemotingMessage">

  <adapters>
    <adapter-definition id="java-object"
      class="flex.messaging.services.remoting.adapters.JavaAdapter"
      default="true"/>
  </adapters>

  <default-channels>
    <channel ref="samples-amf"/>
    </default-channels>

    <destination id="restaurant">
      <properties>
        <source>samples.restaurant.RestaurantService</source>
        <scope>application</scope>
      </properties>
    </destination>

</service>
```

この Remoting Service 宛先では、AMF (Action Message Format) メッセージチャネルをデータ伝達に使用しています。あるいは、サポートされている他のいずれかのメッセージチャネルを使用することもできます。メッセージチャネルは、"services-config.xml" ファイルで `services-config` エレメントの下にある `<channels>` セクションで定義します。メッセージチャネルの詳細については、1240 ページ、第 43 章の「メッセージチャネルの設定」を参照してください。

Remoting Service 宛先では、リモートオブジェクト宛先が通常行う場合と同様に、Java オブジェクトアダプタを使用して Java オブジェクト宛先に接続します。宛先の設定の詳細については、1329 ページ、第 46 章の「RPC サービスの設定」を参照してください。

サービスの呼び出し

RPC コンポーネントの宣言とサービスの呼び出しは MXML または ActionScript で実行できます。入力データのソースに関係なく、サービスの呼び出しは非同期に実行されます。また、RPC 呼び出しの完了後に呼び出される、イベントリスナーと呼ばれる ActionScript メソッドが必要になります。

イベントは、通常はユーザーイベントとシステムイベントの 2 種類に分類されます。ユーザーイベントは、ユーザーによるアプリケーションとのやり取りの結果として発生します。たとえば、ユーザーが Button コントロールをクリックするとユーザーイベントが発生します。システムイベントは、システムのコードが実行された結果として発生します。イベントの詳細については、81 ページ、第 5 章の「イベントの使用」を参照してください。

Flex では 2 つの方法でサービスを呼び出すことができます。1 つは明示的にパラメータを渡す方法で、もう 1 つはパラメータのバインディングです。明示的にパラメータを渡す方法は、RPC コンポーネントを MXML または ActionScript で宣言する場合に使用できます。パラメータのバインディングは、RPC コンポーネントを MXML で宣言する場合にのみ使用できます。

RPC コンポーネントの `requestTimeout` プロパティを使用すると、要求がタイムアウトになるまでに要求を保留し続けることを許可する秒数を定義できます。`requestTimeout` プロパティは、RemoteObject、HTTPService、WebService の各コンポーネントで使用できます。

明示的にパラメータを渡す方法の使用

明示的にパラメータを渡す場合は、ActionScript 関数へのパラメータの形式でサービスに入力を提供します。この方法によるサービスの呼び出しは、Java におけるメソッドの呼び出し方法とよく似ています。明示的にパラメータを渡す場合、Flex データバリデータを同時に自動的に使用することはできません。

RemoteObject コンポーネントおよび WebService コンポーネントで明示的にパラメータを渡す方法

RemoteObject コンポーネントと WebService コンポーネントでは、明示的にパラメータを渡す方法がたいへんよく似ています。次の例の MXML コードは、RemoteObject コンポーネントを宣言し、Button コントロールの click イベントリスナーの中で、明示的にパラメータを渡す方法でサービスを呼び出します。データは ComboBox コントロールを通じてサービスに提供されます。簡単なイベントリスナーで、サービスレベルの result イベントと fault イベントを処理します。この例では `destination` プロパティを示していますが、サービスを呼び出す方法は、サービスに直接接続する場合でも宛先を経由する場合でも同じです。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCParamPassing.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalGap="10">
```

```
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            [Bindable]
            public var empList:Object;
        ]]>
    </mx:Script>

    <mx:RemoteObject
        id="employeeRO"
        destination="SalaryManager"
        result="empList=event.result"
        fault="Alert.show(event.fault.faultString, 'Error');"/>

    <mx:ComboBox id="dept" width="150">
        <mx:dataProvider>
            <mx:ArrayCollection>
                <mx:source>
                    <mx:Object label="Engineering" data="ENG"/>
                    <mx:Object label="Product Management" data="PM"/>
                    <mx:Object label="Marketing" data="MKT"/>
                </mx:source>
            </mx:ArrayCollection>
        </mx:dataProvider>
    </mx:ComboBox>

    <mx:Button label="Get Employee List"
click="employeeRO.getList(dept.selectedItem.data);"/>
</mx:Application>
```

HTTPService タグで明示的にパラメータを渡す方法

HTTPService コンポーネントで明示的にパラメータを渡す方法は、RemoteObject および WebService コンポーネントの場合とは異なります。必ず、HTTPService コンポーネントの send() メソッドを使用してサービスを呼び出します。これは RemoteObject および WebService コンポーネントとは異なります。これらのコンポーネントでは、RPC サービスのクライアントサイドのメソッドまたは処理であるメソッドを呼び出します。

明示的にパラメータを渡す方法を使用する場合、名前と値のペアを格納したオブジェクトを send() メソッドのパラメータとして指定できます。send() メソッドのパラメータは、単純な基本型である必要があります。複雑にネストされたオブジェクトは、名前と値のペアに変換する一般的な方法がないため、使用できません。

send() メソッドへのパラメータを指定しない場合、HTTPService コンポーネントでは、<mx:request> タグで指定した任意のクエリパラメータが使用されます。

次の例は、send() メソッドにパラメータを指定して HTTP サービスを呼び出す 2 つの方法を示しています。2 つ目の例では、cancel() メソッドを呼び出すことで HTTP サービス呼び出しをキャンセルする方法も示しています。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCSend.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[
            public function callService():void {
                // Cancel all previous pending calls.
                myService.cancel();

                var params:Object = new Object();
                params.param1 = 'val1';
                myService.send(params);
            }
        ]]>
    </mx:Script>

    <mx:HTTPService
        id="myService"
        destination="Dest"
        useProxy="true"/>
<!-- HTTP service call with a send() method that takes a variable as its
parameter. The value of the variable is an Object. -->
    <mx:Button click="myService.send({param1: 'val1'});"/>

<!-- HTTP service call with an object as a send() method parameter that provides
query parameters. -->
    <mx:Button click="callService();"/>
</mx:Application>
```

パラメータのバインディングの使用

パラメータのバインディングを使用すると、ユーザーインターフェイスのコントロールやモデルから要求パラメータにデータをコピーすることができます。パラメータのバインディングは、MXML で宣言する RPC コンポーネントの場合にのみ使用できます。要求をサービスに送信する前に、パラメータ値にバリデータを適用できます。データバインディングとデータモデルの詳細については、1133 ページ、第 38 章の「データバインディング」および 1155 ページ、第 39 章の「データの格納」を参照してください。データ検証の詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

パラメータのバインディングを使用する際、RemoteObject メソッドパラメータタグは、<mx:method> タグの下の <mx:arguments> タグ内でネストして宣言します。HTTPService パラメータタグは、<mx:request> タグ内でネストして宣言します。WebService 処理パラメータタグは、<mx:operation> タグの下の <mx:request> タグ内でネストして宣言します。要求を送信するには、send() メソッドを呼び出します。

RemoteObject コンポーネントでのパラメータのバインディング

RemoteObject コンポーネントでパラメータのバインディングを使用する場合は、必ず RemoteObject コンポーネントの <mx:method> タグでメソッドを宣言します。

<mx:method> タグには、メソッドのパラメータに使用する子タグを含む <mx:arguments> タグを含めることができます。<mx:method> タグの name プロパティは、サービスのいずれかのメソッド名に合わせる必要があります。引数タグの順序はサービスのメソッドパラメータの順序に合わせる必要があります。引数タグの名前を、対応するメソッドパラメータの実際の名前にできる限り合わせることもできますが、必ずしもそのようにする必要はありません。

メモ

<mx:arguments> タグの内側に同じ名前の引数タグがある場合、リモートメソッドで Array を唯一の入力ソースとして想定していなければ、サービス呼び出しは失敗します。これについては、アプリケーションのコンパイル時に警告されません。

データは、RemoteObject コンポーネントのメソッドパラメータにバインドできます。パラメータのタグ名を参照して、データのバインディングおよび検証を実行します。

次の例は、TextInput コントロールの text プロパティに 2 つのパラメータをバインドしたメソッドです。PhoneNumberValidator バリデータを、最初の引数タグの名前である arg1 に割り当てています。この例では RemoteObject コンポーネントの destination プロパティを示していますが、サービスを呼び出す方法は、サービスに直接接続する場合でも宛先を経由する場合でも同じです。

```
<?xml version="1.0"?>
<!-- fds\rpc\ROParamBind1.mxml. Compiles -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:RemoteObject
        id="ro"
        destination="roDest" >

        <mx:method name="setData">
            <mx:arguments>
                <arg1>{text1.text}</arg1>
                <arg2>{text2.text}</arg2>
            </mx:arguments>
        </mx:method>
    </mx:RemoteObject>
    <mx:TextInput id="text1"/>
    <mx:TextInput id="text2"/>

    <mx:PhoneNumberValidator source="{ro.setData.arguments}" property="arg1"/>
</mx:Application>
```

引数タグの値は、MXML タグで指定した順序でメソッドに渡されます。

次の例では、RemoteObject コンポーネントの <mx:method> タグでパラメータのバインディングを使用し、ユーザーが Button コントロールをクリックしたときに、選択された ComboBox アイテムのデータを employeeRO.getList 処理にバインドしています。パラメータのバインディングを行う場合は、パラメータなしの send() メソッドを使用してサービスを呼び出します。

```
<?xml version="1.0"?>
<!-- fds\rpc\ROParamBind2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
                import mx.utils.ArrayUtil;
        ]]>
    </mx:Script>
    <mx:RemoteObject
        id="employeeRO"
        destination="roDest"
        showBusyCursor="true"
        fault="Alert.show(event.fault.faultString, 'Error');">
        <mx:method name="getList">
            <mx:arguments>
                <deptId>{dept.selectedItem.data}</deptId>
            </mx:arguments>
        </mx:method>
    </mx:RemoteObject>
    <mx:ArrayCollection id="employeeAC"
        source="{ArrayUtil.toArray(employeeRO.getList.lastResult)}"/>

    <mx:HBox>
        <mx:Label text="Select a department:"/>
        <mx:ComboBox id="dept" width="150">

            <mx:dataProvider>
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
            <mx:Button label="Get Employee List"
                click="employeeRO.getList.send()"/>
    </mx:HBox>
    <mx:DataGrid dataProvider="{employeeAC}" width="100%">
        <mx:columns>
            <mx:DataGridColumn dataField="name" headerText="Name"/>
```

```
            <mx:DataGridColumn dataField="phone" headerText="Phone"/>
            <mx:DataGridColumn dataField="email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

サービス呼び出しの結果に、配列が含まれるのか、単独のオブジェクトが含まれるのかがわからない場合には、この例のように、mx.utils.ArrayUtil クラスの toArray() メソッドを使用して、結果を配列に変換することができます。toArray() メソッドに単独のオブジェクトを渡すと、そのオブジェクトを唯一の配列エレメントとする配列が返されます。このメソッドに配列を渡すと、同じ配列が返されます。ArrayCollection オブジェクトの操作の詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

HTTPService コンポーネントでのパラメータのバインディング

HTTP サービスがクエリパラメータを受け取る場合、それらのパラメータを <mx:request> タグの子タグとして宣言できます。タグの名前は、サービスで想定されているクエリパラメータの名前に合わせる必要があります。

次の例では、HTTPService コンポーネントの <mx:request> タグでパラメータのバインディングを使用し、ユーザーが Button コントロールをクリックしたときに、選択された ComboBox アイテムのデータを employeeSrv 要求にバインドしています。パラメータのバインディングを行う場合は、パラメータなしの send() メソッドを使用してサービスを呼び出します。この例では HTTPService コンポーネントの url プロパティを示していますが、サービスを呼び出す方法は、サービスに直接接続する場合でも宛先を経由する場合でも同じです。

```
<?xml version="1.0"?>
<!-- fds\rpc\HttpServiceParamBind.mxml. Compiles -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="20">
    <mx:Script>
        <![CDATA[
            import mx.utils.ArrayUtil;
        ]]>
    </mx:Script>

    <mx:HTTPService
        id="employeeSrv"
        url="employees.jsp">
        <mx:request>
            <deptId>{dept.selectedItem.data}</deptId>
        </mx:request>
    </mx:HTTPService>
    <mx:ArrayCollection
        id="employeeAC"
        source=
            "{ArrayUtil.toArray(employeeSrv.lastResult.employees.employee)}"/>
    <mx:HBox>
        <mx:Label text="Select a department:"/>
```

```
        <mx:ComboBox id="dept" width="150">
            <mx:dataProvider>
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
        <mx:Button label="Get Employee List" click="employeeSrv.send();"/>
    </mx:HBox>
    <mx:DataGrid dataProvider="{employeeAC}"
        width="100%">
        <mx:columns>
            <mx:DataGridColumn dataField="name" headerText="Name"/>
            <mx:DataGridColumn dataField="phone" headerText="Phone"/>
            <mx:DataGridColumn dataField="email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

サービス呼び出しの結果に、配列が含まれるのか、単独のオブジェクトが含まれるのかがわからない場合には、前の例のように、mx.utils.ArrayUtil クラスの `toArray()` メソッドを使用して、結果を配列に変換することができます。`toArray()` メソッドに単独のオブジェクトを渡すと、そのオブジェクトを唯一の配列エレメントとする配列が返されます。このメソッドに配列を渡すと、同じ配列が返されます。ArrayCollection オブジェクトの操作の詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

WebService コンポーネントでのパラメータのバインディング

WebService コンポーネントでパラメータのバインディングを使用する場合は、必ず WebService コンポーネントの `<mx:operation>` タグで処理を宣言します。`<mx:operation>` タグには、処理で想定される XML ノードを含む `<mx:request>` タグを含めることができます。`<mx:operation>` タグの `name` プロパティは、Web サービスのいずれかの処理名に合わせる必要があります。

データは、Web サービス処理のパラメータにバインドできます。パラメータのタグ名を参照して、データのバインディングおよび検証を実行します。

次の例では、WebService コンポーネントの `<mx:operation>` タグでパラメータのバインディングを使用し、ユーザーが Button コントロールをクリックしたときに、選択された ComboBox アイテムのデータを `employeeWS.getList` 処理にバインドしています。`<deptId>` タグは、`getList` 処理の `deptId` パラメータに直接対応しています。パラメータのバインディングを行う場合は、パラメータなしの `send()` メソッドを使用してサービスを呼び出します。この例では WebService コンポーネントの `destination` プロパティを示していますが、サービスを呼び出す方法は、サービスに直接接続する場合でも宛先を経由する場合でも同じです。

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceParamBind.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.utils.ArrayUtil;
            import mx.controls.Alert;
        ]]>
    </mx:Script>

    <mx:WebService
        id="employeeWS"
        destination="wsDest"
        showBusyCursor="true"
        fault="Alert.show(event.fault.faultString)">
        <mx:operation name="getList">
            <mx:request>
                <deptId>{dept.selectedItem.data}</deptId>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:ArrayCollection
        id="employeeAC"
        source="{ArrayUtil.toArray(employeeWS.getList.lastResult)}"/>
    <mx:HBox>
        <mx:Label text="Select a department:"/>
        <mx:ComboBox id="dept" width="150">
            <mx:dataProvider>
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
            <mx:Button label="Get Employee List"
                click="employeeWS.getList.send()"/>
    </mx:HBox>
    <mx:DataGrid dataProvider="{employeeAC}" width="100%">
        <mx:columns>
                <mx:DataGridColumn dataField="name" headerText="Name"/>
                <mx:DataGridColumn dataField="phone" headerText="Phone"/>
                <mx:DataGridColumn dataField=" to email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

SOAP 要求のボディ全体を XML 内で手動で指定し、すべての正しい名前空間情報を <mx:request> タグで定義することもできます。これには、次の例のように、<mx:request> タグの format 属性の値を xml に設定する必要があります。

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceSOAPRequest.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:WebService id="ws" wsdl="http://api.google.com/GoogleSearch.wsdl"
        useProxy="true">
        <mx:operation name="doGoogleSearch" resultFormat="xml">
            <mx:request format="xml">
                <ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                    <key xsi:type="xsd:string">XYZ123</key>
                    <q xsi:type="xsd:string">Balloons</q>
                    <start xsi:type="xsd:int">0</start>
                    <maxResults xsi:type="xsd:int">10</maxResults>
                    <filter xsi:type="xsd:boolean">true</filter>
                    <restrict xsi:type="xsd:string"/>
                    <safeSearch xsi:type="xsd:boolean">false</safeSearch>
                    <lr xsi:type="xsd:string" />
                    <ie xsi:type="xsd:string">latin1</ie>
                    <oe xsi:type="xsd:string">latin1</oe>
                </ns1:doGoogleSearch>
            </mx:request>
        </mx:operation>
    </mx:WebService>
</mx:Application>
```

RemoteObject のメソッドまたは WebService の処理におけるプロパティの設定

次のプロパティを、RemoteObject コンポーネントの <mx:method> タグ、または WebService コンポーネントの <mx:operation> タグで、設定できます。必須のプロパティは name プロパティのみです。これらのプロパティについて次の表で説明します。

プロパティ	説明
concurrency	同じメソッドに対して複数の呼び出しが行われる場合の処理方法を示す値です。デフォルトでは、実行中の処理またはメソッドに対して新しい要求が行われても、既存の要求はキャンセルされません。 使用できる値は次のとおりです。 • multiple　既存の要求はキャンセルされません。返されるデータは開発者が責任を持って、イベントストリームを慎重に管理することで整合性を確保する必要があります。これがデフォルト値です。 • single　メソッドに対して要求を行えるのは一度に 1 回のみになります。複数の要求を行うと、失敗が生成されます。 • last　要求を行うと、既存の要求がすべてキャンセルされます。 **メモ :** ここでの要求とは、HTTP 要求のことではありません。これはクライアントアクション要求、つまり pendingCall オブジェクトです。HTTP 要求は、サーバーに送られてサーバーサイドで処理されます。しかし、要求がキャンセルされると結果は無視されます。single または last の値を使用すると要求がキャンセルされます。last 要求は、必ずしもサーバーが HTTP 経由で受信する最後の要求になるとは限りません。
fault	エラーが発生したときに実行される ActionScript コード。fault がイベントパラメータとして渡されます。
name	(必須) 呼び出す処理またはメソッドの名前。
result	lastResult オブジェクトが使用できる場合に実行される ActionScript コード。result オブジェクトがイベントパラメータとして渡されます。

ActionScript でのサービスの呼び出し

ActionScript でサービスを呼び出すときは、常に明示的にパラメータを渡す方法を使用します。このセクションでは、RemoteObject、HTTPService、WebService の各コンポーネントの ActionScript 例を示します。

ActionScript での RemoteObject コンポーネントの呼び出し

次の ActionScript の例は、1293 ページの「RemoteObject コンポーネントおよび WebService コンポーネントで明示的にパラメータを渡す方法」で示した MXML の例と等価なものです。useRemoteObject() メソッド呼び出しでは、サービスを宣言し、宛先を設定し、result および fault イベントリスナーを設定して、サービスの getList() メソッドを呼び出しています。

```
<?xml version="1.0"?>
<!-- fds\rpc\ROInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.remoting.RemoteObject;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;

            [Bindable]
            public var empList:Object;
            public var employeeRO:RemoteObject;

            public function useRemoteObject(intArg:int, strArg:String):void {
                employeeRO = new RemoteObject();
                employeeRO.destination = "SalaryManager";
                employeeRO.getList.addEventListener("result",
getListResultHandler);
                employeeRO.addEventListener("fault", faultHandler);
                employeeRO.getList(deptComboBox.selectedItem.data);
            }

            public function getListResultHandler(event:ResultEvent):void {
                 // Do something
                empList=event.result;
            }

            public function faultHandler (event:FaultEvent):void {
             // Deal with event.fault.faultString, etc.
                Alert.show(event.fault.faultString, 'Error');
            }
        ]]>
    </mx:Script>
    <mx:ComboBox id="deptComboBox"/>
</mx:Application>
```

ActionScript での Web サービスの呼び出し

次の例は、ActionScript スクリプトブロック内の Web サービス呼び出しを示しています。useWebService() メソッド呼び出しでは、サービスを宣言し、宛先を設定し、WSDL ドキュメントを取得して、サービスの echoArgs() メソッドを呼び出しています。ActionScript で WebService コンポーネントを宣言する場合は WebService.loadWSDL() メソッドを呼び出す必要があります。

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import mx.rpc.soap.WebService;
        import mx.rpc.events.ResultEvent;
        import mx.rpc.events.FaultEvent;
        private var ws:WebService;
        public function useWebService(intArg:int, strArg:String):void {
            ws = new mx.rpc.soap.WebService();
            ws.destination = "echoArgService";
            ws.echoArgs.addEventListener("result", echoResultHandler);
            ws.addEventListener("fault", faultHandler);
            ws.loadWSDL();
            ws.echoArgs(intArg, strArg);
        }

        public function echoResultHandler(event:ResultEvent):void {
            var retStr:String = event.result.echoStr;
            var retInt:int = event.result.echoInt;
         //Do something.
        }

        public function faultHandler(event:FaultEvent):void {
      //deal with event.fault.faultString, etc
        }
        ]]>
    </mx:Script>
</mx:Application>
```

ActionScript での HTTP サービスの呼び出し

次の例は、ActionScript スクリプトブロック内の HTTP サービス呼び出しを示しています。useHTTPService() メソッド呼び出しでは、サービスを宣言し、宛先を設定し、result および fault イベントリスナーを設定して、サービスの send() メソッドを呼び出しています。

```
<?xml version="1.0"?>
<!-- fds\rpc\HttpServiceInAS.mxml. Compiles -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.http.HTTPService;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;

            private var service:HTTPService

            public function useHttpService(parameters:Object):void {
                service = new HTTPService();
                service.destination = "sampleDestination";
                service.method = "POST";
                service.addEventListener("result", httpResult);
                service.addEventListener("fault", httpFault);
                service.send(parameters);
            }

            public function httpResult(event:ResultEvent):void {
                var result:Object = event.result;
            //Do something with the result.
            }

            public function httpFault(event:FaultEvent):void {
                var faultstring:String = event.fault.faultString;
                Alert.show(faultstring);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

サービス結果の処理

RPC コンポーネントでサービスを呼び出すと、サービスから返されたデータが lastResult オブジェクトに格納されます。デフォルトでは、HTTPService コンポーネントおよび WebService コンポーネント処理の `resultFormat` プロパティ値は `object` であり、返されたデータは ActionScript オブジェクトの単純なツリーとして表現されます。Flex では、Web サービスや HTTP サービスから返された XML データが、String、Number、Boolean、Date などの基本型を表すように適切に解釈されます。厳密に型指定されたオブジェクトを扱う場合は、Flex により作成されたオブジェクトツリーを使用してそれらのオブジェクトを設定する必要があります。

WebService コンポーネントと HTTPService コンポーネントはいずれも、複合型である匿名のオブジェクトおよび配列を返します。`makeObjectsBindable` がデフォルトの `true` の場合、オブジェクトは mx.utils.ObjectProxy インスタンスにラップされ、配列は mx.collections.ArrayCollection インスタンスにラップされます。

メモ

Macromedia ColdFusion では大文字と小文字が区別されないため、データはすべて内部で大文字に変換されます。ColdFusion の Web サービスを利用する場合は、この点に注意してください。

結果を E4X 結果形式の XML として処理する方法

HTTPService コンポーネントおよび WebService 処理の `resultFormat` プロパティ値を `e4x` に設定すると、XML 型の lastResult オブジェクトを作成できます。lastResult オブジェクトには E4X (ECMAScript for XML) 式を使用してアクセスできます。`resultFormat` として `e4x` を使用する方法は XML を操作する場合に適していますが、`resultFormat` プロパティを `xml` に設定することによって、XML 操作用の古いオブジェクトである flash.xml.XMLNode 型の lastResult オブジェクトを作成することもできます。また、HTTPService コンポーネントの `resultFormat` プロパティを `flashvars` または `text` に設定することで、前者は名前と値のペアを含む ActionScript オブジェクトとして、後者は純粋なテキストだけを含む ActionScript オブジェクトとして、それぞれ結果を作成できます。詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

メモ

サービス結果で E4X のシンタックスを使用する場合は、HTTPService または WebService コンポーネントの `resultFormat` プロパティを `e4x` に設定する必要があります。デフォルト値は `object` です。

WebService 処理の `resultFormat` プロパティを `e4x` に設定した場合、Web サービスから返される SOAP エンベロープのボディに含まれる名前空間情報を処理しなければならないことがあります。次の例は、名前空間情報を含む SOAP ボディの一部です。このデータは株式相場を取得する Web サービスから返されたものです。名前空間情報はボールド体のテキストで示しています。

```
...
<soap:Body>
<GetQuoteResponse
xmlns="http://ws.invesbot.com/">
<GetQuoteResult><StockQuote xmlns="">
<Symbol>ADBE</Symbol>
<Company>ADOBE SYSTEMS INC</Company>
<Price>&lt;big&gt;&lt;b&gt;35.90&lt;/b&gt;&lt;/big&gt;</Price>
...
</soap:Body>
...
```

この soap:Body には名前空間情報が含まれているため、WebService 処理の resultFormat プロパティを e4x に設定した場合は、http://ws.invesbot.com/ 名前空間に対応する名前空間オブジェクトを作成する必要があります。次の例は、この処理を実行するアプリケーションです。

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceE4XResult1.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
    pageTitle="Test" >
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            private namespace invesbot = "http://ws.invesbot.com/";
            use namespace invesbot;
        ]]>
    </mx:Script>
    <mx:WebService
        id="WS"
        destination="stockservice" useProxy="true"
        fault="Alert.show(event.fault.faultString), 'Error'">
        <mx:operation name="GetQuote" resultFormat="e4x">
            <mx:request>
                <symbol>ADBE</symbol>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:HBox>
        <mx:Button label="Get Quote" click="WS.GetQuote.send()"/>
        <mx:Text
            text="{WS.GetQuote.lastResult.GetQuoteResult.StockQuote.Price}"
        />
    </mx:HBox>
</mx:Application>
```

次の例のように、名前空間用の変数を作成し、サービス結果へのバインディングの中で変数にアクセスすることもできます。

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceE4XResult2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
    pageTitle="Test" >
    <mx:Script>
        <![CDATA[
        import mx.controls.Alert;
        public var invesbot:Namespace =
            new Namespace("http://ws.invesbot.com/");
        ]]>
    </mx:Script>
    <mx:WebService
        id="WS"
        destination="stockservice" useProxy="true"
        fault="Alert.show(event.fault.faultString), 'Error'">
        <mx:operation name="GetQuote" resultFormat="e4x">
            <mx:request>
                <symbol>ADBE</symbol>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:HBox>
        <mx:Button label="Get Quote" click="WS.GetQuote.send()"/>
        <mx:Text

text="{WS.GetQuote.lastResult.invesbot::GetQuoteResult.StockQuote.Price}"
        />
    </mx:HBox>
</mx:Application>
```

E4X のシンタックスは、lastResult オブジェクトの中で返される XML のエレメントや属性にアクセスする場合に使用します。使用するシンタックスは、XML で宣言された名前空間があるかどうかによって異なります。

名前空間なし

次の例は、エレメントや属性で名前空間が指定されていない場合にエレメント値や属性値を取得する方法を示しています。

```
var attributes:XMLList = XML(event.result).Description.value;
```

前のコードは、次の XML ドキュメントの xxx を返します。

```
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Description>
    <value>xxx</value>
  </Description>
</RDF>
```

任意の名前空間

次の例は、エレメントや属性で任意の名前空間が指定されている場合にエレメント値や属性値を取得する方法を示しています。

```
var attributes:XMLList = XML(event.result).*::Description.*::value;
```

前のコードは、次のいずれかの XML ドキュメントの xxx を返します。

XML ドキュメント 1:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description>
    <rdf:value>xxx</rdf:xxx>
  </rdf:Description>
</rdf:RDF>
```

XML ドキュメント 2:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cm="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <cm:Description>
    <rdf:value>xxx</rdf:xxx>
  </cm:Description>
</rdf:RDF>
```

特定の名前空間

次の例は、宣言された rdf 名前空間がエレメントや属性で指定されている場合にエレメント値や属性値を取得する方法を示しています。

```
var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
var attributes:XMLList = XML(event.result).rdf::Description.rdf::value;
```

前のコードは、次の XML ドキュメントの xxx を返します。

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description>
    <rdf:value>xxx</rdf:xxx>
  </rdf:Description>
</rdf:RDF>
```

次の例は、宣言された rdf 名前空間がエレメントや属性で指定されている場合にエレメント値や属性値を取得する別の方法を示しています。

```
namespace rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
use namespace rdf;
var attributes:XMLList = XML(event.result).rdf::Description.rdf::value;
```

前のコードは、次の XML ドキュメントの xxx を返します。

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description>
    <rdf:value>xxx</rdf:xxx>
  </rdf:Description>
</rdf:RDF>
```

他のオブジェクトへのサービス結果のバインド

RPC コンポーネントの lastResult オブジェクトのプロパティを、ユーザーインターフェイスコンポーネントやデータモデルなどの他のオブジェクトのプロパティにバインドできます。lastResult は、直前の呼び出しから得られた結果オブジェクトです。サービス要求が実行されるたびに、lastResult が更新され、さらに関連するバインディングもすべて更新されます。

次に示す例では、lastResult オブジェクトの 2 つのプロパティである CityShortName および CurrentTemp が、2 つの TextArea コントロールの text プロパティにバインドされます。ユーザーが MyService.GetWeather( ) 処理に対して要求を行い、処理の要求パラメータとして郵便番号を指定すると、CityShortName プロパティおよび CurrentTemp プロパティが返されます。

```
<mx:TextArea text="{MyService.GetWeather.lastResult.CityShortName}"/>
<mx:TextArea text="{MyService.GetWeather.lastResult.CurrentTemp}"/>
```

lastResult オブジェクトを、ArrayCollection オブジェクトの source プロパティにバインドできます。HTTPService または WebService コンポーネントを使用する場合、HTTPService コンポーネントまたは WebService 処理の resultFormat プロパティを e4x に設定すると、lastResult を XMLListCollection オブジェクトにバインドできます。RemoteObject のメソッドには resultFormat プロパティはありません。そして、ArrayCollection オブジェクトまたは XMLListCollection オブジェクトを、List、ComboBox、DataGrid コントロールなどのユーザーインターフェイスコンポーネントの複雑なプロパティにバインドできます。

データを操作するには ArrayCollection または XMLListCollection の API を使用します。XMLListCollection オブジェクトを使用している場合は、E4X (ECMAScript for XML) 式を使用してデータを操作できます。

ArrayCollection オブジェクトへの結果のバインド

次の例では、サービスの lastResult オブジェクト employeeWS.getList.lastResult を ArrayCollection オブジェクトの source プロパティにバインドしています。ArrayCollection オブジェクトを DataGrid コントロールの dataProvider プロパティにバインドし、従業員の名前、電話番号、電子メールアドレスを表示します。

```
<?xml version="1.0"?>
<!-- fds\rpc\BindingResultArrayCollection.mxml. Warnings on mx:Object  -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.utils.ArrayUtil;
        ]]>
    </mx:Script>
    <mx:WebService id="employeeWS" destination="employeeWS"
        showBusyCursor="true"
        fault="Alert.show(event.fault.faultString, 'Error');">
        <mx:operation name="getList">
```

```
            <mx:request>
                <deptId>{dept.selectedItem.data}</deptId>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:ArrayCollection id="ac"
        source="{ArrayUtil.toArray(employeeWS.getList.lastResult)}"/>
    <mx:HBox>
        <mx:Label text="Select a department:"/>
        <mx:ComboBox id="dept" width="150">
            <mx:dataProvider>
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
        <mx:Button label="Get Employee List" click="employeeWS.getList.send()"/>
    </mx:HBox>
    <mx:DataGrid dataProvider="{ac}" width="100%">
        <mx:columns>
                <mx:DataGridColumn dataField="name" headerText="Name"/>
                <mx:DataGridColumn dataField="phone" headerText="Phone"/>
                <mx:DataGridColumn dataField="email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

サービス呼び出しの結果に、配列が含まれるのか、単独のオブジェクトが含まれるのかがわからない場合には、前の例のように、mx.utils.ArrayUtil クラスの toArray() メソッドを使用して、結果を配列に変換することができます。toArray() メソッドに単独のオブジェクトを渡すと、そのオブジェクトを唯一の配列エレメントとする配列が返されます。このメソッドに配列を渡すと、同じ配列が返されます。

ArrayCollection オブジェクトの操作の詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

XMLListCollection オブジェクトへの結果のバインド

次の例では、サービスの lastResult オブジェクト employeeWS.getList.lastResult を XMLListCollection オブジェクトの source プロパティにバインドしています。XMLListCollection オブジェクトを DataGrid コントロールの dataProvider プロパティにバインドし、従業員の名前、電話番号、電子メールアドレスを表示します。

メモ

サービス結果を XMLListCollection にバインドするには、HTTPService または WebService コンポーネントの resultFormat プロパティを e4x に設定する必要があります。このプロパティのデフォルト値は object です。

```
<?xml version="1.0"?>
<!-- fds\rpc\BindResultXMLListCollection.mxml. Warnings on mx:Object -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" verticalGap="10">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
        ]]>
    </mx:Script>
    <mx:WebService id="employeeWS"
        destination="employeeWS"
        showBusyCursor="true"
        fault="Alert.show(event.fault.faultString, 'Error');">
        <mx:operation name="getList" resultFormat="e4x">
            <mx:request>
                <deptId>{dept.selectedItem.data}</deptId>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:HBox>
        <mx:Label text="Select a department:"/>
        <mx:ComboBox id="dept" width="150">
            <mx:dataProvider>
                <mx:ArrayCollection>
                    <mx:source>
                        <mx:Object label="Engineering" data="ENG"/>
                        <mx:Object label="Product Management" data="PM"/>
                        <mx:Object label="Marketing" data="MKT"/>
                    </mx:source>
                </mx:ArrayCollection>
            </mx:dataProvider>
        </mx:ComboBox>
        <mx:Button label="Get Employee List"
            click="employeeWS.getList.send()"/>
    </mx:HBox>
    <mx:XMLListCollection id="xc"
        source="{employeeWS.getList.lastResult}"/>
    <mx:DataGrid dataProvider="{xc}" width="100%">
        <mx:columns>
```

```
                <mx:DataGridColumn dataField="name" headerText="Name"/>
                <mx:DataGridColumn dataField="phone" headerText="Phone"/>
                <mx:DataGridColumn dataField="email" headerText="Email"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

XMLListCollection オブジェクトの操作の詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

result イベントと fault イベントの処理

サービス呼び出しが完了すると、RemoteObject のメソッド、WebService の処理、または HTTPService のコンポーネントから、result イベントまたは fault イベントが送出されます。result イベントは、使用できる結果が返されたことを示します。fault イベントは、エラーが発生したことを示します。result イベントは、lastResult にバインドされているプロパティを更新するトリガとして機能します。result イベントおよび fault イベントを明示的に処理するには、イベントリスナーを RemoteObject のメソッドまたは WebService の処理に追加します。HTTPService コンポーネントの場合は、result および fault イベントリスナーをコンポーネント自身で指定します。これは、HTTPService コンポーネントには複数の処理やメソッドがないためです。

RemoteObject のメソッドや WebService の処理で result または fault イベントのイベントリスナーを指定しない場合、イベントはコンポーネントレベルで渡されます。この場合、コンポーネントレベルの result および fault イベントリスナーを指定できます。

次の MXML の例では、WebService 処理の `result` および `fault` イベントでイベントリスナーを指定しています。また、WebService コンポーネントの `fault` イベントでもイベントリスナーを指定しています。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCResultFaultMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;
            import mx.controls.Alert;
            public function showErrorDialog(event:FaultEvent):void {
                // Handle operation fault.
                Alert.show(event.fault.faultString, "Error");
            }
            public function defaultFault(event:FaultEvent):void {
                // Handle service fault.
                Alert.show(event.fault.faultString, "Error");
            }
            public function log(event:ResultEvent):void {
                // Handle result.
```

```
            }
        ]]>
    </mx:Script>
    <mx:WebService id="WeatherService" destination="wsDest"
        fault="defaultFault(event)">
        <mx:operation name="GetWeather"
            fault="showErrorDialog(event)"
            result="log(event)">
            <mx:request>
                <ZipCode>{myZip.text}</ZipCode>
            </mx:request>
        </mx:operation>
    </mx:WebService>
    <mx:TextInput id="myZip"/>
</mx:Application>
```

次の ActionScript の例では、result イベントリスナーを WebService 処理に追加し、fault イベントリスナーを WebService コンポーネントに追加しています。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCResultFaultAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.rpc.soap.WebService;
            import mx.rpc.events.ResultEvent;
            import mx.rpc.events.FaultEvent;
            private var ws:WebService;

            public function useWebService(intArg:int, strArg:String):void {
                ws = new WebService();
                ws.destination = "wsDest";
                ws.echoArgs.addEventListener("result", echoResultHandler);
                ws.addEventListener("fault", faultHandler);
                ws.loadWSDL();
                ws.echoArgs(intArg, strArg);
            }

            public function echoResultHandler(event:ResultEvent):void {
                var retStr:String = event.result.echoStr;
                var retInt:int = event.result.echoInt;
                //do something
        }

            public function faultHandler(event:FaultEvent):void {
            //deal with event.fault.faultString, etc.
            }
        ]]>
    </mx:Script>
</mx:Application>
```

mx.rpc.events.InvokeEvent イベントを使用して、RPC コンポーネント要求がいつ呼び出されたのかを示すこともできます。このイベントは、処理をキューに格納して後で呼び出す場合に便利です。

サービスと、バインディング、検証、およびイベントリスナーの使用

データをサービスに渡す前に検証し、サービスから result または fault が返されたときにイベントを送出することができます。次の例は、サービス要求データを検証し、イベントリスナーを result イベントおよび fault イベントに割り当てるアプリケーションです。

この 2 層アプリケーションでは、次の処理を実行します。

1. Web サービスを宣言します。
2. ユーザーインターフェイスデータを Web サービス要求にバインドします。
3. 郵便番号を検証します。
4. Web サービス結果のデータをユーザーインターフェイスコントロールにバインドします。
5. WebService 処理の result および fault イベントリスナーを指定します。

ユーザーインターフェイスと Web サービスとの間にデータモデルレイヤーを追加した、多層アプリケーションも作成できます。データモデルとデータバインディングの詳細については、1127 ページ、第 37 章の「データの表現と操作」を参照してください。データ検証の詳細については、1165 ページ、第 40 章の「データ検証」を参照してください。

```
<?xml version="1.0"?>
<!-- fds\rpc\RPCBindValidateEvents.mxml. Compiles w destination error -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="600" height="400">

    <!-- WebService component handles web service requests and results. -->
    <mx:WebService id="WeatherService" destination="wsDest">
        <mx:operation name="GetWeather"
            fault="showErrorDialog(event.fault.faultString)" result="log();">

    <!-- The mx:request data model stores web service request data. -->
            <mx:request>
                <ZipCode>{myZipField.text}</ZipCode>
            </mx:request>
        </mx:operation>
    </mx:WebService>

    <!-- Validator validates ZIP code using the standard Zip Code validator. -->
    <mx:ZipCodeValidator
        source="{WeatherService.GetWeather.request}" property="ZipCode"
        trigger="{mybutton}" triggerEvent="click"/>
    <mx:VBox>
    <mx:TextInput id="myZipField" text="enter zip" width="80"/>

    <!-- Button triggers service request. -->
    <mx:Button id="mybutton" label="Get Weather"
```

```
        click="WeatherService.GetWeather.send();"/>

    <!-- TextArea control displays the results that the service returns. -->
        <mx:TextArea id="temp" text="The current temperature in
        {WeatherService.GetWeather.lastResult.CityShortName} is
        {WeatherService.GetWeather.lastResult.CurrentTemp}."
        height="30" width="200"/>
    </mx:VBox>
    <mx:Script>
        <![CDATA[
            public function log():void {
                // function implementation
            }

            public function showErrorDialog(error:String):void {
                // function implementation
            }
        ]]>
    </mx:Script>
</mx:Application>
```

サービスに対する非同期呼び出しの処理

ActionScript コードは非同期で実行されるため、サービスに対する同時呼び出しを許可する場合は、サービスを呼び出したコンテキストに従って結果を適切に処理するようコードを作成する必要があります。デフォルトでは、既に実行中の Web サービス処理に対して要求が行われても、既存の要求がキャンセルされることはありません。複数の場所からサービスを呼び出すことのできる Flex アプリケーションの場合、コンテキストが異なるとサービスの応答が変わる場合があります。

Flex アプリケーションを設計する際は、アプリケーションに異なるデータソースが必要かどうか、およびアプリケーションで何種類のサービスが必要かを検討してください。この検討の結果が、アプリケーションのデータレイヤーに指定する抽象レベルを決定する際に役立ちます。

ごく単純なアプリケーションでは、ユーザーインターフェイスコンポーネントがサービスを直接呼び出すことがあります。やや規模の大きいアプリケーションでは、ビジネスオブジェクトがサービスを呼び出すことがあります。さらに規模の大きいアプリケーションでは、サービスを呼び出すサービスブローカーオブジェクトとビジネスモデルとの間でデータをやり取りすることがあります。

アプリケーションにおけるオブジェクトに対する非同期サービス呼び出しの結果を理解するには、ActionScript のスコープ設定について理解する必要があります。

ACT (Asynchronous Completion Token) デザインパターンの使用

Flex は、コードが非同期的に実行されるサービス指向フレームワークであるため、ACT (Asynchronous Completion Token) デザインパターンに適しています。このデザインパターンを使用することで、クライアントが実行した非同期処理の完了に応答して、クライアント内部の処理を効率的に進めることができます。詳細については、www.cs.wustl.edu/~schmidt/PDF/ACT.pdf を参照してください。

ACT デザインパターンを使用する場合、アプリケーション固有のアクションと状態に、非同期処理の完了を表す応答を関連付けます。各非同期処理について、処理の結果を処理するために必要なアクションと状態を指定する ACT を作成します。結果が返されると、その ACT を使用して、どの非同期処理による結果なのかを判別できます。クライアントでは ACT を使用して結果の処理に必要な状態を識別します。

特定の非同期処理に対応する ACT は、その処理が呼び出される前に作成されます。処理の実行中、クライアントは実行を継続します。サービスが応答を送信すると、クライアントはその応答に関連付けられている ACT を使用して適切なアクションを実行します。

Flex のリモートオブジェクトサービス、Web サービス、HTTP サービスを呼び出すと、Flex はサービス呼び出しのインスタンスを返します。`concurrency` のデフォルト値である `multiple` を使用する場合、データサービスの `send()` メソッドにより返される call オブジェクトを使用して、同じサービスに対して同時に行われた呼び出しによる各結果を処理することができます。この call オブジェクトが返されたときに call オブジェクトに情報を追加してから、result イベントリスナーでこの call オブジェクトを event.call として返すことができます。これで、各データサービス呼び出しの call オブジェクトを ACT として使用する、ACT デザインパターンが実装されます。コードにおける ACT デザインパターンの使用方法は、要件に応じて異なります。たとえば、個々の呼び出し、独自の機能のセットを実行する複雑なオブジェクト、または中央のリスナーが呼び出す関数に、単純な識別子を割り当てることができます。

次に、ACT デザインパターンの単純な実装の例を示します。この例では HTTP サービスを使用し、call オブジェクトに単純な変数を割り当てています。

```
...
<mx:HTTPService id="MyService" destination="httpDest"
  result="resultHandler(event)"/>
...
<mx:Script>
  <![CDATA[
    ...
    public function storeCall():void {
      // 返されるサービス呼び出しのインスタンスを
      // 格納する、call という名前の変数を作成します。
      var call:Object = MyService.send();
```

```
        // 返される call オブジェクトに変数を追加します。
        // この変数には任意の名前を付けることができます。
        call.marker = "option1";
        ...
      }

      // result イベントリスナーで、call.marker の値に基づいて
      // 条件式を実行します。
      private function resultHandler(event:ResultEvent):void {
        var call:object = event.token
        if (call.marker == "option1") {
        // option 1 を実行します。
        }
        else
        ...
      }
  ]]>
</mx:Script>
...
```

別のサービス呼び出しが完了してからのサービス呼び出しの実行

データサービスを使用する際の一般的な要件の１つに、あるサービス呼び出しを実行している場合、その呼び出しが終了してからでないと、他のサービス呼び出しを実行できないという依存性が挙げられます。コードでは、最初の呼び出しの結果が得られるまで、２番目の呼び出しは行わないでください。次の例に示すように、最初のサービス呼び出しの result イベントリスナーで、２番目の呼び出しを行う必要があります。

```
...
<mx:WebService id="ws" destination="wsDest"...>
  <mx:operation name="getCurrentSales" result="resultHandler(event.result)"/>
  <mx:operation name="setForecastWithSalesInput"/>
</mx:WebService>
<mx:Script>
  <![CDATA[
    // getCurrentSales 処理の結果を使用して
    // getForecastWithSalesInput 処理を呼び出します。
      public function resultHandler(currentsales:String):void {
      ws.setForecastWithSalesInput(currentsales);
      // データバインディングを使用する方法もあります。
      }
  ]]>
</mx:Script>
...
```

RemoteObject コンポーネントに固有の機能の使用

RemoteObject コンポーネントを使用して Remoting Service 宛先のメソッドを呼び出し、Flex データサービスが実行されているアプリケーションサーバーに存在し、Web アプリケーションのソースパスにある Java オブジェクトを指定することができます。

ソースパスにある Java オブジェクトへのアクセス

RemoteObject コンポーネントを使用すると、Flex データサービス Web アプリケーションのソースパスにあるステートレスオブジェクトおよびステートフルオブジェクトにアクセスできます。Web アプリケーションの WEB-INF/classes ディレクトリにスタンドアローンクラスファイルを配置すると、それらのクラスファイルがソースパスに追加されます。Web アプリケーションの WEB-INF/lib ディレクトリに、JAR (Java アーカイブ) ファイルに収められたクラスを配置すると、それらのクラスがソースパスに追加されます。"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしている remoting-config.xml などのファイルにある、Remoting Service 宛先の `source` プロパティに、完全修飾クラス名を指定する必要があります。また、クラスには引数なしのコンストラクタも設定する必要があります。Remoting Service 宛先の設定の詳細については、第 46 章の「RPC サービスの設定」を参照してください。

ステートレスオブジェクト (要求のスコープ) にアクセスするように Remoting Service 宛先を設定した場合、Flex では、同じオブジェクトのメソッドが呼び出されるのではなく、メソッド呼び出しごとに新しいオブジェクトが作成されます。オブジェクトのスコープは、要求のスコープ (デフォルト値)、アプリケーションのスコープ、またはセッションのスコープに設定できます。アプリケーションのスコープ内のオブジェクトは、そのオブジェクトのある Web アプリケーションで利用できます。セッションのスコープ内のオブジェクトは、クライアントセッション全体で利用できます。

ステートフルオブジェクトにアクセスするようにリモートオブジェクト宛先を設定した場合、Flex では、オブジェクトがサーバー上に 1 回だけ作成され、メソッド呼び出し間で状態が維持されます。アプリケーションのスコープまたはセッションのスコープでオブジェクトを格納した場合にメモリの問題が生じる場合は、要求のスコープを使用してください。

JNDI にある EJB その他のオブジェクトへのアクセス

JNDI (Java Naming and Directory Interface) 内のオブジェクトを参照してそのメソッドを呼び出すサービスファサードクラスとなっている、宛先のメソッドを呼び出すことにより、JNDI 内に格納されている EJB (Enterprise JavaBean) その他のオブジェクトにアクセスすることができます。

ステートレスオブジェクトまたはステートフルオブジェクトを使用し、JNDI を使用する EJB その他のオブジェクトのメソッドを呼び出すことができます。EJB の場合、JNDI から EJB オブジェクトを返し、EJB のメソッドを呼び出す、サービスファサードクラスを呼び出すことができます。

Java クラスでは標準の Java コーディングパターンを使用し、初期コンテキストを作成して JNDI 参照を実行します。EJB でも標準のコーディングパターンを使用し、EJB ホームオブジェクトの `create()` メソッドと結果の EJB のビジネスメソッドを呼び出すメソッドを、クラスに含めます。

次の例では、ファサードクラスの宛先で `getHelloData()` という名前のメソッドを使用しています。

```
<mx:RemoteObject id="Hello" destination="roDest">
   <mx:method name="getHelloData"/>
</mx:RemoteObject>
```

Java 側では、`getHelloData()` メソッドを使用して、EJB のビジネスメソッドを呼び出すために必要なものをすべて簡単にカプセル化することができます。次の例に示す Java メソッドでは、次の処理を実行します。

- EJB を呼び出すための新しい初期コンテキストを作成します。
- JNDI 参照を実行して EJB ホームオブジェクトを取得します。
- EJB ホームオブジェクトの `create()` メソッドを呼び出します。
- EJB の `sayHello()` メソッドを呼び出します。

```
...
public void getHelloData() {
  try{
     InitialContext ctx = new InitialContext();

     Object obj = ctx.lookup("/Hello");

     HelloHome ejbHome = (HelloHome)

     PortableRemoteObject.narrow(obj, HelloHome.class);

     HelloObject ejbObject = ejbHome.create();

     String message = ejbObject.sayHello();
     }
  catch (Exception e);
  }
...
```

予約されているメソッド名

次のメソッド名は Flex Remoting ライブラリで使用されています。メソッドを作成する際にこれらの名前を使用しないでください。

```
addHeader()
addProperty()
deleteHeader()
hasOwnProperty()
isPropertyEnumerable()
isPrototypeOf()
registerClass()
toLocaleString()
toString()
unwatch()
valueOf()
watch()
```

また、メソッド名の先頭にはアンダースコア文字 (_) を使用しないでください。

RemoteObject および WebService 処理 (メソッド) には、通常はサービス変数名を付けるだけでアクセスできます。ただし、処理名と、サービスで定義されているメソッド名が同じ場合は、ActionScript で RemoteObject または WebService コンポーネントの次のメソッドを使用して、対象の名前を持つ処理を返すことができます。

```
public function getOperation(name:String):Operation
```

WebService コンポーネントに固有の機能の使用

Flex アプリケーションは、URL として提供される WSDL 1.1 (Web Services Description Language 1.1) ドキュメントでインターフェイスが定義された Web サービスとやり取りすることができます。WSDL は、Web サービスで解釈可能なメッセージ、そのメッセージに対する Web サービスの応答形式、Web サービスがサポートするプロトコル、およびメッセージの送信先を記述するために使用される標準形式です。Flex では RPC エンコードされた Web サービスとドキュメントリテラル Web サービスの両方がサポートされています。Web サービスでは、RPC スタイルまたはドキュメントスタイルのいずれかの SOAP バインディングが使用されます。これらは最も一般的な 2 種類の Web サービスで、RPC エンコードされた SOAP バインディングまたはドキュメントリテラル SOAP バインディングが使用されます。エンコードおよびリテラルという言葉は、サービスで使用される WSDL から SOAP へのマッピングの種類を示しています。

メモ	Flex では、choice、union、default、list、group の各 XML スキーマ型はサポートされていません。また、duration、gMonth、gYear、gYearMonth、gDay、gMonthDay、Name、Qname、NCName、anyURI、language の各データ型もサポートされていません。(Flex では anyURL がサポートされていますが、ストリングと同様に扱われます。

Flex アプリケーションでは、SOAP (Simple Object Access Protocol) メッセージとしてフォーマットされた Web サービス要求および結果がサポートされています。SOAP では XML ベースの形式が定義されており、これを使用することで、構造化され型指定された情報を Flex アプリケーションなどの Web サービスクライアントと Web サービスの間で交換できます。

Adobe Flash Player はセキュリティ Sandbox 内で動作するため、どの Flex アプリケーションやその他の Flash アプリケーションから HTTP 経由でアクセスできるかについて制限があります。Flash アプリケーションでは、その Flash アプリケーションが提供されるドメインと同じドメインに置かれているリソースに対する、同じプロトコルによる HTTP アクセスのみが許可されます。Web サービスは一般にリモートからアクセスされるので、問題が発生します。Flex データサービスで使用できる Flex プロキシは、リモートの Web サービスに対する要求を受け取り、その要求をリダイレクトした後、応答をクライアントに返します。

Flex データサービスを使用しない場合は、Flex アプリケーションと同じドメインにある Web サービスにアクセスできます。または、アプリケーションのドメインからのアクセスを許可する crossdomain.xml (クロスドメインポリシー) ファイルを、RPC サービスのホストとなっている Web サーバーにインストールする必要があります。"crossdomain.xml" ファイルの詳細については、『Flex 2 アプリケーションの構築および展開ガイド』の第 4 章の「Flex セキュリティの適用」を参照してください。

WSDL ドキュメントの読み取り

WSDL ドキュメントは、Web ブラウザ、簡易テキストエディタ、XML エディタ、あるいは、WSDL ドキュメントを読みやすい形式で表示するユーティリティを備えた Adobe Dreamweaver などの開発環境で、表示できます。

次の表に、WSDL ドキュメントに含まれるタグを示します。

タグ	内容
<binding>	Flex アプリケーションなどのクライアントが Web サービスとの通信に使用するプロトコルを指定します。SOAP、HTTP GET、HTTP POST、および MIME に対応したバインディングがあります。Flex でサポートされているバインディングは SOAP のみです。
<fault>	メッセージの処理に関する問題の結果として返されるエラー値を指定します。
<input>	Flex アプリケーションなどのクライアントが Web サービスに送信するメッセージを指定します。
<message>	Web サービスの処理によって転送されるデータを定義します。
<operation>	<input> タグ、<output> タグ、<fault> タグの組み合わせを定義します。
<output>	Web サービスが、Flex アプリケーションなどの Web サービスクライアントに送信するメッセージを指定します。

タグ	内容
<port>	Web サービスのエンドポイントを指定します。エンドポイントは、バインディングとネットワークアドレスとの間の関連付けを指定します。
<portType>	Web サービスによって提供される 1 つまたは複数の処理を定義します。
<service>	<port> タグの集まりを定義します。各サービスは 1 つの <portType> タグに割り当てられ、その <portType> タグの処理にアクセスするさまざまな方法を指定します。
<types>	Web サービスのメッセージで使用されるデータ型を定義します。

RPC 指向処理とドキュメント指向処理

WSDL ファイルでは、RPC (リモートプロシージャコール) 指向の処理またはドキュメント指向 (ドキュメントまたはリテラル) の処理を指定できます。Flex では両方の処理スタイルがサポートされています。

RPC 指向の処理を呼び出す場合は、Flex アプリケーションから、処理とそのパラメータを指定する SOAP メッセージを送信します。ドキュメント指向処理を呼び出す場合は、Flex アプリケーションから、XML ドキュメントを収めた SOAP メッセージを送信します。

WSDL ドキュメントの各 <port> タグには、次の例に示すように、特定の <soap:binding> タグの名前を指定する binding プロパティが設定されています。

```
<binding name="InstantMessageAlertSoap" type="s0:InstantMessageAlertSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document"/>
```

関連付けられた <soap:binding> タグの style プロパティによって、処理スタイルが決まります。この例では、スタイルは document になります。

サービス内の処理では、同じスタイルを指定することや、サービスに関連付けられたポートで指定されているスタイルを変更することができます。次に例を示します。

```
<operation name="SendMSN">
  <soap:operation soapAction="http://www.bindingpoint.com/ws/imalert/
    SendMSN"style="document"/>
```

ステートフル Web サービス

Flex では、Cookie を使用してセッション情報を格納する Web サービスのエンドポイントの状態を、Java サーバーセッションを使って保持します。この機能は、Flex アプリケーションと Web サービスの間に介在するかたちで機能します。エンドポイントから Flex アプリケーションに渡されるものすべてに対して、エンドポイントの ID が付加されます。エンドポイントからセッション情報が送信されると、Flex アプリケーションでそのセッション情報が受信されます。この機能には設定は必要ありません。Proxy Service を使用する際に RTMP チャネルを使用する宛先では、この機能はサポートされません。

SOAP ヘッダーの操作

SOAP ヘッダーは、SOAP エンベロープ内のオプションのタグです。通常、SOAP エンベロープには認証情報など、アプリケーション固有の情報が含まれます。このセクションでは、SOAP ヘッダーを Web サービス要求に追加する方法、SOAP ヘッダーをクリアする方法、および Web サービス結果に含まれている SOAP ヘッダーを取得する方法について説明します。

Web サービス要求への SOAP ヘッダーの追加

Web サービスによっては、処理を呼び出す際に SOAP ヘッダーを一緒に渡さなければならない場合があります。

SOAP ヘッダーはすべての Web サービス処理または個々の処理に追加できます。これは、WebService または Operation オブジェクトの `addHeader()` メソッドまたは `addSimpleHeader()` メソッドをイベントリスナー関数で呼び出します。

`addHeader()` メソッドを使用する場合は、まず SOAPHeader オブジェクトと QName オブジェクトを別々に作成しておく必要があります。`addHeader()` メソッドには次のシグネチャがあります。

```
addHeader(header:mx.rpc.soap.SOAPHeader):void
```

SOAPHeader オブジェクトを作成するには、次のコンストラクタを使用します。

```
SOAPHeader(qname:QName, content:Object)
```

`SOAPHeader()` メソッドの第1パラメータで QName オブジェクトを作成するには、次のコンストラクタを使用します。

```
QName(uri:String, localName:String)
```

`SOAPHeader()` コンストラクタの `content` パラメータは、次の形式に基づく一連の名前と値のペアです。

```
{name1:value1, name2:value2}
```

`addSimpleHeader()` メソッドは、SOAP ヘッダーの名前と値の単一ペアに対するショートカットです。`addSimpleHeader()` メソッドを使用する場合には、メソッドのパラメータで SOAPHeader オブジェクトおよび QName オブジェクトを作成します。`addSimpleHeader()` メソッドには次のシグネチャがあります。

```
addSimpleHeader(qnameLocal:String, qnameNamespace:String, headerName:String,
  headerValue:Object):void
```

`addSimpleHeader()` メソッドは次のパラメータを受け取ります。

- `qnameLocal` は、ヘッダーの QName のローカル名です。
- `qnameNamespace` は、ヘッダーの QName の名前空間です。
- `headerName` は、ヘッダーの名前です。

- headerValue は、ヘッダーの値です。単純な値の場合は String を、基本的な XML エンコードを使用する場合は Object を、ヘッダー XML を自分で指定する場合は XML を、それぞれ指定します。

次のコード例は、addHeader() メソッドと addSimpleHeader() メソッドを使用して SOAP ヘッダーを追加する方法を示しています。これらのメソッドは headers という名前のイベントリスナー関数で呼び出され、イベントリスナーは <mx:WebService> タグの load プロパティで割り当てられます。

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceAddHeader.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="600">
    <mx:WebService id="ws" destination="wsDest" load="headers();"/>
    <mx:Script>
        <![CDATA[
            import mx.rpc.soap.SOAPHeader;
            private var header1:SOAPHeader;
            private var header2:SOAPHeader;

            public function headers():void {

            // Create QName and SOAPHeader objects.
            var q1:QName=new QName("http://soapinterop.org/xsd", "Header1");
            header1=new SOAPHeader(q1, {string:"bologna",int:"123"});
            header2=new SOAPHeader(q1, {string:"salami",int:"321"});


            // Add the header1 SOAP Header to all web service requests.
            ws.addHeader(header1);

            // Add the header2 SOAP Header to the getSomething operation.
            ws.getSomething.addHeader(header2);

            // Within the addSimpleHeader method,
            // which adds a SOAP header to web
            //service requests, create SOAPHeader and QName objects.
            ws.addSimpleHeader
                ("header3", "http://soapinterop.org/xsd", "foo","bar");
            }
        ]]>
    </mx:Script>
</mx:Application>
```

SOAP ヘッダーのクリア

オブジェクトに追加した SOAP ヘッダーを削除するには、次の WebService オブジェクトの例のように、WebService または Operation オブジェクトの clearHeaders() メソッドを使用します。clearHeaders() は、ヘッダーを追加したレベル (WebService または Operation) で呼び出す必要があります。

```
<?xml version="1.0"?>
<!-- fds\rpc\WebServiceClearHeader.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="600" >

    <!-- The value of the destination property is for demonstration only and is
not a real destination. -->

    <mx:WebService id="ws" destination="wsDest" load="headers();"/>

    <mx:Script>
        <![CDATA[
            import mx.rpc.*;
            import mx.rpc.soap.SOAPHeader;

            private function headers():void {
                // Create QName and SOAPHeader objects.
                var q1:QName=new QName("Header1", "http://soapinterop.org/xsd");
                var header1:SOAPHeader=new SOAPHeader(q1,
{string:"bologna",int:"123"});
                var header2:SOAPHeader=new SOAPHeader(q1,
{string:"salami",int:"321"});
                    // Add the header1 SOAP Header to all web service request.
                ws.addHeader(header1);
                // Add the header2 SOAP Header to the getSomething operation.
                ws.getSomething.addHeader(header2);

               // Within the addSimpleHeader method, which adds a SOAP header to
all
                // web service requests, create SOAPHeader and QName objects.
              ws.addSimpleHeader("header3","http://soapinterop.org/xsd", "foo",
"bar");
            }

            // Clear SOAP headers added at the WebService and Operation levels.
            private function clear():void {
                ws.clearHeaders();
                ws.getSomething.clearHeaders();
            }
        ]]>
    </mx:Script>

    <mx:HBox>
            <mx:Button label="Clear headers and run again" click="clear();"/>
    </mx:HBox>

</mx:Application>
```

異なる URL への Web サービスのリダイレクト

Web サービスによっては、WSDL の処理後に別のエンドポイント URL に変更し、Web サービスへの初期呼び出しを行わなければならない場合があります。たとえば、セキュリティ証明書を渡すことを要求する Web サービスを使用するとします。Web サービスを呼び出してログイン証明書を送ると、Web サービスは証明書を受け取り、サービスのビジネス処理を使用するために必要な実際のエンドポイント URL を返します。これらのビジネス処理を呼び出す前に、WebService コンポーネントの `endpointURI` プロパティを変更する必要があります。

次の result イベントリスナーの例では、Web サービスから返されたエンドポイント URL を変数に格納した後、その変数を関数に渡して、以降の要求に使用するエンドポイント URL を変更しています。

```
...
public function onLoginResult(event:ResultEvent):void {

// ログインの結果から、新しいサービスのエンドポイントを抽出します。
    var newServiceURL = event.result.serverUrl;

// すべてのサービスの処理を、ログインの結果で受け取った URL にリダイレクトします。
  serviceName.endpointURI=newServiceURL;

}
...
```

セキュリティ証明書を渡す必要のある Web サービスの場合、以降の要求を行う際に SOAP ヘッダーに付加する必要がある識別子がサービスから返されることもあります。詳細については、1324 ページの「SOAP ヘッダーの操作」を参照してください。

第 46 章

RPC サービスの設定

46

Adobe Flex クライアントアプリケーションから RPC (リモートプロシージャコール) サービスに接続するには、MXML または ActionScript で RPC コンポーネントを宣言します。RPC コンポーネントでは、HTTP サービスまたは Web サービスの実際の URL または WSDL URL、あるいは Flex データサービス 宛先定義の名前を指定できます。宛先は、呼び出す必要のある、実際のサーバーサイドのサービスまたはオブジェクトです。宛先は、Flex サービス設定ファイル、またはサービス設定ファイルを参照によってインクルードしているファイルで、設定します。このトピックでは、宛先の設定方法とセキュリティ保護の方法について説明します。

RPC コンポーネントでの宛先の指定に関する詳細については、1287 ページ、第 45 章の「RPC コンポーネントの使用」を参照してください。

目次

宛先の設定について

RPC サービスの宛先は、<mx:RemoteObject>、<mx:WebService>、<mx:HTTPService> タグ、または対応する ActionScript API を使用して接続する、接続先のオブジェクトまたはサービスです。宛先の設定は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしているファイルで、最も頻繁に実行する作業です。リモートオブジェクトの宛先は "Remoting Service" 定義で定義します。慣例として、これらは "remoting-config.xml" ファイルにあります。Web サービス宛先および HTTP サービス宛先は Proxy Service 定義で定義します。慣例として、これらは "proxy-config.xml" ファイルにあります。

宛先を設定するときはメッセージチャネルを指定します。メッセージチャネルは、対応するサーバーサイドのオブジェクトまたはサービスに接続するために使用できます。また、アダプタも参照できます。アダプタは、オブジェクトやサービスと直接やり取りするサーバーサイドコードです。デフォルトアダプタを設定すると、宛先ごとにアダプタを明示的に参照せずに済ませることができます。

次の例は、基本的な Remoting Service 定義です。サービスには、セキュリティ制限を参照している宛先が含まれています。例ではこのセキュリティ制限も示しています。宛先では、サービスレベルで定義された java-object というデフォルトアダプタが使用されます。

```
<service id="remoting-service"
  class="flex.messaging.services.RemotingService"
  messageTypes="flex.messaging.messages.RemotingMessage">

  <adapters>
    <adapter-definition id="java-object"
      class="flex.messaging.services.remoting.adapters.JavaAdapter"
      default="true"/>
  </adapters>

  <default-channels>
    <channel ref="samples-amf"/>
  </default-channels>

  <destination id="SampleEmployeeRO">
    <properties>
      <source>samples.explorer.EmployeeManager</source>
      <scope>application</scope>
    </properties>
    <security>
      <security-constraint ref="privileged-users"/>
    </security>
  </destination>
</service>

...
<security>
  <security-constraint id="privileged-users">
    <auth-method>Custom</auth-method>
    <roles>
      <role>privilegedusers</role>
      <role>admins</role>
    </roles>
  </security-constraint>
...
</security>
```

メッセージチャネル

宛先ではメッセージチャネルを参照します。メッセージチャネルは同じ設定ファイル内の任意の場所で定義します。特定の宛先でチャネルを明示的に参照するのではなく、デフォルトチャネルを設定ファイル内でサービスレベルで参照することもできます。

メッセージチャネルの詳細については、1240 ページの「メッセージチャネルの設定」を参照してください。

宛先アダプタ

サービス定義には、サービス要求に対して機能するアダプタの定義が含まれます。たとえば、HTTP サービスは HTTP 要求メッセージを Proxy Service に送信します。Proxy Service ではその HTTP プロキシアダプタを使用して、対象の URL に対して HTTP 要求を行います。アダプタ定義では、flex.messaging.services.ServiceAdapter の実装であるアダプタクラスを参照する必要があります。

デフォルトアダプタを定義すると、宛先定義ごとにアダプタを明示的に参照せずに済ませることができます。アダプタをデフォルトアダプタにするには、アダプタのデフォルトプロパティの値を `true` に設定します。

次の例は、Remoting Service 宛先で使用する Java オブジェクトアダプタです。このアダプタはデフォルトアダプタとして設定されます。

```
...
<adapters>
  <adapter-definition id="java-object"
    class="flex.messaging.services.remoting.adapters.JavaAdapter"
    default="true"/>
</adapters>
...
```

セキュリティ

セキュリティ制限は、ユーザーが宛先にアクセスできるようにする前にユーザーの認証および承認を行う場合に使用します。認証では、基本認証を使用するかカスタム認証を使用するかを指定できます。承認では、承認に必要なロールを指定できます。

セキュリティ制限は宛先定義内でインラインで宣言できます。または、セキュリティ制限をグローバルに宣言し、宛先定義内でその `id` によって参照できます。

セキュリティの詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。

HTTP サービスのデフォルト宛先

HTTP サービスのデフォルト宛先は、Flex サービス設定ファイル、またはサービス設定ファイルを参照によってインクルードしているファイルで、設定できます。この宛先を使用すると、複数の HTTP サービスの `url` プロパティ値をクライアントサイドのサービスタグまたは ActionScript コードで使用でき、同時に、Proxy Service を経由してクロスドメインのサポートおよびセキュリティを実現できます。

デフォルト宛先の `id` 値は常に `defaultHTTP` になります。通常は `dynamic-url` パラメータを使用し、HTTP プロキシアダプタに対して 1 つまたは複数の URL ワイルドカードパターンを指定します。

次の例は、`dynamic-url` 値を指定したデフォルト宛先定義です。

```
...
```

```
<destination id="defaultHTTP">
   <channels>
     <channel ref="my-amf"/>
     </channels>
   <properties>
        <dynamic-url>http://mysite.com/myservices/*</dynamic-url>
   </properties>
   <security>
...
   </security>
</destination>
...
```

HTTP サービスアダプタの詳細については、1334 ページの「HTTP サービスのプロパティ」を参照してください。

宛先プロパティの設定

Remoting Service 宛先および Proxy Service 宛先と通信するには、特定の種類の宛先プロパティを宛先定義の properties セクションで設定します。

Remote オブジェクトのプロパティ

Remoting Service 宛先定義では、source エレメントを使用して、宛先で使用される Java オブジェクトを指定します。また、scope エレメントを使用して、要求のスコープ内で利用できるようにするか (ステートレス)、アプリケーションのスコープ内で利用できるようにするか、またはセッションのスコープ内で利用できるようにするかを指定します。これらのプロパティについて次の表で説明します。

エレメント	説明
source	Java オブジェクト (リモートオブジェクト) の完全修飾クラス名。
scope	オブジェクトを要求のスコープ内で利用できるようにするか、アプリケーションのスコープ内で利用できるようにするか、またはセッションのスコープ内で利用できるようにするかを指定します。要求のスコープ内のオブジェクトはステートレスになります。アプリケーションのスコープ内のオブジェクトは、そのオブジェクトのある Web アプリケーションで利用できます。セッションのスコープ内のオブジェクトは、クライアントセッション全体で利用できます。 有効な値は、request、application、および session です。デフォルト値は request です。

次の例は、source プロパティと scope プロパティのある Remoting Service 宛先定義です。

```
...
  <destination id="SampleEmployeeRO">
    <properties>
      <source>samples.explorer.EmployeeManager</source>
      <scope>application</scope>
    </properties>
    <adapter ref="java-object" />
  </destination>
...
```

Web サービスのプロパティ

wsdl エレメントと soap エレメントを使用して、Web サービスの URL を設定します。これらのエレメントは、宛先に対してどの URL を許可するのかを定義します。これらのエレメントについて次の表で説明します。

エレメント	内容
wsdl	(オプション) デフォルトの WSDL URL。
soap	SOAP エンドポイントの URL パターン。通常は、WSDL ドキュメント内の操作ごとに定義されます。 複数の soap エントリを使用すると複数の SOAP エンドポイントパターンを指定できます。これらの値と、クライアントサイドのサービスタグまたは ActionScript コードで指定した endpointURI プロパティの値とが比較されます。

次の例は、Web サービスの宛先定義です。

```
...
  <destination id="ContactManagerWS">
    <adapter ref="soap-proxy" />
    <properties>
      <wsdl>{context.root}/services/ContactManagerWS?wsdl</wsdl>
      <soap>{context.root}/services/ContactManagerWS</soap>
    </properties>
  </destination>
...
```

HTTP サービスのプロパティ

url エレメントと dynamic-url エレメントを使用して、HTTP サービスの URL を設定します。これらのエレメントは、宛先に対してどの URL を許可するのかを定義します。これらのエレメントについて次の表で説明します。

エレメント	内容
url	(オプション) デフォルトの URL。
dynamic-url	(オプション) HTTP サービスの URL パターン。複数の dynamic-url エントリを使用すると複数の URL パターンを指定できます。これらの値と、クライアントサイドのサービスタグまたは ActionScript コードで指定した url プロパティの値とが比較されます。

次の例は、HTTP サービスの宛先定義です。

```
...
  <destination id="samplesProxy">
    <channels>
      <channel ref="samples-amf"/>
    </channels>

    <properties>
      <dynamic-url>{context.root}/photoviewer/*</dynamic-url>
      <dynamic-url>{context.root}/blogreader/*</dynamic-url>
      <dynamic-url>{context.root}/services/*</dynamic-url>
    </properties>
  </destination>
...
```

Proxy Service の設定

Web サービス宛先と HTTP サービス宛先の両方を定義する Proxy Service には、Apache 接続マネージャ、SSL 自己署名証明書、および外部プロキシを設定するための子エレメントを持つ properties エレメントがあります。これらの XML エレメントについて次の表で説明します。

エレメント	内容
connection-manager	max-connections エレメントと default-max-connections-per-host エレメントを格納します。max-connections エレメントは、プロキシがサポートする同時接続の最大総数を制御します。この値が 0 よりも大きい場合、Flex データサービスでは、基になる Apache HttpClient プロキシに対してマルチスレッド接続マネージャが使用されます。 default-max-connections-per-host エレメントは、ハードウェアクラスタ化を使用する環境において各ホストで許容されるデフォルトの接続数を設定します。
content-chunked	まとまったコンテンツを使用しているかどうかを示します。デフォルト値は false です。Flash Player では、まとまったコンテンツはサポートされません。
allow-lax-ssl	true に設定した場合、SSL の使用時に自己署名証明書が使用できるようになります。運用環境ではこの値を true に設定しないでください。
external-proxy	外部プロキシの場所を指定します。また、インターネットにアクセスする前に Proxy Service から外部プロキシに接続しなければならない場合に必要となるユーザー名とパスワードも指定します。

次の例は、max-connections と external-proxy の設定です。

```
...
<service id="proxy-service" class="flex.messaging.services.HTTPProxyService"
  messageTypes="flex.messaging.messages.HTTPMessage,flex.messaging.
  messages.SOAPMessage">
...

  <properties>
    <connection-manager>
      <max-total-connections>100</max-total-connections>
      <default-max-connections-per-host>2
      </default-max-connections-per-host>
       </connection-manager>

<!-- 自己署名証明書を使用できるようにします。運用環境では使用しないでください。 -->
     <allow-lax-ssl>true</allow-lax-ssl>
```

```
     <external-proxy>
       <server>10.10.10.10</server>
       <port>3128</port>
       <nt-domain>mycompany</nt-domain>
       <username>flex</username>
       <password>flex</password>
       </external-proxy>
    </properties>
...
</service>
```

第 47 章

Flex メッセージングについて

47

Adobe Flex データサービスには、サーバーサイドの Message Service と組み合わせて使用する、クライアントサイドのメッセージング API が組み込まれています。このトピックでは、メッセージングと Adobe Flex メッセージング機能の概要について紹介します。

目次

メッセージングについて

Flex メッセージング機能は、確立されたメッセージング標準と用語をベースとしています。Flex メッセージングでは、Flex メッセージングアプリケーションを作成するためのクライアントサイド API および対応するサーバーサイド Message Service (Flex Message Service) が提供されています。Flex メッセージングでは JMS (Java Message Service) メッセージングへの参加も可能です。詳細については、1357 ページ、第 49 章の「Message Service の設定」を参照してください。

ColdFusion Event Gateway Adapter を使用して CFC (ColdFusion Component) にメッセージを送信することもできます。詳細については、ColdFusion Event Gateway Adapter のマニュアルを参照してください。

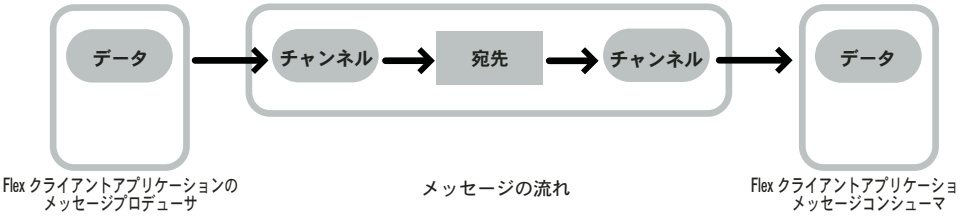
メッセージングシステムにより、異なるアプリケーションどうしがメッセージサービスを介してメッセージと呼ばれるデータパケットを互いに渡し合い、ピアとして非同期に通信できます。メッセージは通常、ヘッダーとボディで構成されます。ヘッダーには識別子とルーティング情報が含まれます。ボディにはアプリケーションデータが含まれます。

メッセージを送信するアプリケーションはプロデューサと呼ばれます。メッセージを受信するアプリケーションはコンシューマと呼ばれます。ほとんどのメッセージングシステムでは、プロデューサとコンシューマが互いについて何も知る必要はありません。プロデューサはメッセージを特定のメッセージ宛先に送信し、メッセージサービスはメッセージを適切なコンシューマに転送します。

メッセージチャネルは、プロデューサとコンシューマをメッセージ宛先に接続します。特定のチャネルを経由してメッセージを送信する場合、アプリケーションはメッセージチャネルに関連付けられたメッセージエンドポイントに接続します。メッセージエンドポイントは、データをメッセージにエンコードし、メッセージをコンシューマが使用できる形式にデコードする役割を持つコードです。一部のメッセージングシステムでは、エンドポイントはデコードされたメッセージをメッセージブローカーに渡すことができ、メッセージブローカーはそれらを適切な宛先に転送します。

メッセージアダプタは、Flex Message Service とその他のメッセージングシステム間のパイプ役として機能するコードです。たとえば、JMS (Java Message Service) アダプタは、Flex アプリケーションが JMS トピックおよびキューにサブスクライブするのを可能にするメッセージアダプタです。このアダプタを使用して、純粋な Java JMS メッセージアプリケーションは Flex アプリケーションと同じ宛先を共有できます。Java アプリケーションはメッセージを Flex にパブリッシュでき、Java コードは Flex アプリケーションが送信するメッセージに応答できます。

次の図は、メッセージプロデューサからメッセージコンシューマへのデータの流れを示しています。データは、メッセージとしてエンコードされ、チャネルを経由して宛先に送信されます。次に、メッセージはチャネルを経由してメッセージコンシューマに送信され、コンシューマが使用できるデータにデコードされます。



Flex では 2 種類のメッセージングがサポートされています。1つはパブリッシュ/ サブスクライブメッセージングで、トピックベースメッセージングとも呼ばれます。もう 1 つはポイントツーポイントメッセージングで、キューベースメッセージングとも呼ばれます。これらは、エンタープライズメッセージングシステムで使用されている最も一般的な 2 種類のメッセージングです。パブリッシュ / サブスクライブメッセージングは、プロデューサとコンシューマ間で 1 対多の関係を必要とするアプリケーションで使用します。ポイントツーポイントメッセージングは、プロデューサとコンシューマ間で 1 対 1 の関係を必要とするアプリケーションで使用します。

Flex メッセージングでは、トピックまたはキューはメッセージサービス宛先によって表現されます。パブリッシュ / サブスクライブメッセージングでは、各メッセージが複数のコンシューマを持つことができます。この種のメッセージングは、複数のコンシューマに同じメッセージを受信させるときに使用します。パブリッシュ/ サブスクライブメッセージングを使用するアプリケーションの例は、オークションサイト、株価情報サービス、および 1 つのメッセージを多数のサブスクライバに送信する必要があるその他のアプリケーションです。キューベースメッセージングでは、メッセージがそれぞれ 1 つのコンシューマに配信されます。

プロデューサは、メッセージサーバー上の特定のトピックにメッセージをパブリッシュし、コンシューマはメッセージを受信するためにそれらのトピックにサブスクライブします。コンシューマは、トピックにサブスクライブした後で初めて、そのトピックにパブリッシュされたメッセージを受信できます。

次の図は、簡単なパブリッシュ / サブスクライブメッセージの流れを示しています。

アプリケーションでのメッセージングの使用

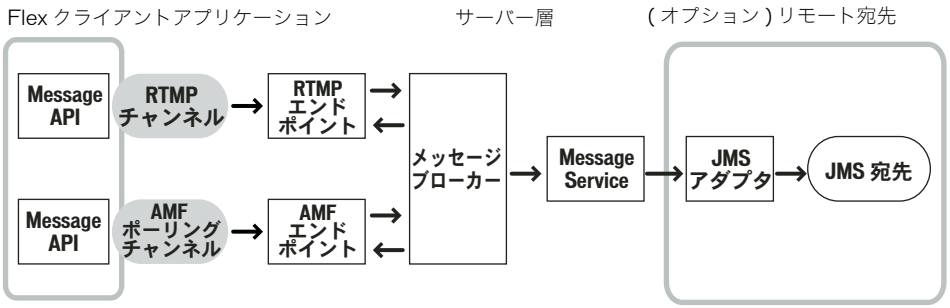
Flex では、メッセージングを Flex アプリケーションに追加できるようにする MXML および ActionScript API が提供されています。プロデューサ、コンシューマ、または両方の役割をするアプリケーションを作成できます。Flex アプリケーションは、メッセージを Flex サーバーで宣言されたチャネルを経由して、同じく Flex サーバーで宣言された宛先に送信します。詳細については、1240 ページの「メッセージチャネルの設定」を参照してください。

メッセージサービスのチャネルと宛先は Flex サービス設定ファイルで設定します。メッセージ宛先はリモートリソースです。メッセージ宛先のセキュリティポリシーと、メッセージ宛先を必要とするメッセージアダプタを設定できます。メッセージングの設定の詳細については、1357 ページ、第 49 章の「Message Service の設定」を参照してください。

Flex メッセージングアーキテクチャについて

Flex メッセージングにより、Flex アプリケーションはメッセージ宛先に接続し、メッセージを宛先に送信し、他のメッセージングクライアントからメッセージを受信できます。これらのメッセージングクライアントは、Flex アプリケーションや、JMS (Java Message Service) クライアントなどの他のタイプのクライアントの場合があります。JMS は、アプリケーションでのメッセージの作成、送受信、および読み取りを可能にする Java API です。JMS クライアントは、Flex アプリケーションと同じメッセージ宛先にパブリッシュおよびサブスクライブできます。つまり、Flex アプリケーションは Java クライアントアプリケーションとメッセージを交換できます。ただし、Flex のメッセージングを使用するだけでも、広範囲なメッセージングアプリケーションを作成できます。

次の図に、メッセージングアーキテクチャの概略を示します。



Flex Message Service

Flex Message Service には、チャネル、宛先、プロデューサ、コンシューマの 4 つのコンポーネントがあります。Flex クライアントアプリケーションで ActionScript または MXML のメッセージング API を使用してメッセージを宛先にパブリッシュすると、クライアントアプリケーションからサーバーサイドの Message Service にメッセージが送信されます。

Flex Message Service により、Adobe Flash Player でサポートされている AMF (Action Message Format)、RTMP (Realtime Message Protocol)、XMLSocket などのトランスポートプロトコルの上層における抽象化が実現されます。Message Service は、1 つまたは複数のチャネルを経由してメッセージを伝達するように設定します。チャネルはそれぞれ特定のトランスポートプロトコルに対応しています。特定の宛先でどのチャネルを使用するかは、Flex サービス設定ファイルの中で指定します。

メッセージチャネル

Flex アプリケーションからは、複数の異なるメッセージチャネルを経由して Flex Message Service にアクセスできます。各チャネルは、特定のネットワークプロトコルに対応しており、対応するサーバーサイドエンドポイントを持ちます。また、各チャネルに対応するクライアントサイドコードが存在します。メッセージがエンドポイントに到達すると、エンドポイントでメッセージがデコードされてメッセージブローカーに渡されます。メッセージブローカーでは、メッセージをどこに送信する必要があるかが決定されます。パブリッシュ / サブスクライブメッセージングの場合、メッセージブローカーはメッセージを Message Service に送信します。また、メッセージブローカーはメッセージを Flex RPC (リモートプロシージャコール) サービスおよび Data Management Service の機能に転送します。Flex クライアントは、使用可能なチャネルが見つかるか、リストにそれ以上試行するチャネルがなくなるまで、指定された順序でチャネルを試行します。

メッセージチャネルの詳細については、1359 ページの「メッセージチャネル」を参照してください。

JMS メッセージアダプタ

JMS は、アプリケーションでのメッセージの作成、送受信、および読み取りを可能にする Java API です。Flex メッセージングでは、JMS メッセージアダプタを使用して外部のメッセージングシステムとやり取りします。

Flex アプリケーションでこのアダプタを使用することにより、JMS トピックおよびキューにサブスクライブできます。また、このアダプタによって、Flex アプリケーションが既存の MOM (Messaging Oriented Middleware) システムに参加できるようになります。

第 48 章

Flex メッセージングの使用

48

Adobe Flex データサービスには、サーバーサイドの Flex Message Service と組み合わせて使用する、クライアントサイドのメッセージング API が組み込まれています。このトピックでは、メッセージを送受信する Flex アプリケーションを作成するための、クライアントサイドのメッセージング API の使用方法について説明します。サーバーサイドの Message Service の設定に関する詳細については、1357 ページ、第 49 章の「Message Service の設定」を参照してください。

目次

Flex アプリケーションでのメッセージングの使用

Flex クライアントアプリケーションでは、クライアントサイドのメッセージング API を使用して、サーバーサイドの宛先とメッセージを送受信します。メッセージは、プロトコルに固有のメッセージチャネルを経由して送受信されます。

クライアントサイドの主要なメッセージコンポーネントとして、Producer コンポーネントと Consumer コンポーネントの 2 つがあります。Producer コンポーネントは、メッセージをサーバーサイドの宛先に送信します。Consumer コンポーネントは、サーバーサイドの宛先にサブスクライブし、Producer コンポーネントからその宛先に送信されたメッセージを受信します。Producer コンポーネントと Consumer コンポーネントは MXML または ActionScript で作成できます。

Producer コンポーネントと Consumer コンポーネントではいずれも、有効なメッセージ宛先が必要になります。有効なメッセージ宛先は Flex サービス設定ファイルの中で設定します。メッセージ宛先の詳細については、1357 ページ、第 49 章の「Message Service の設定」を参照してください。

多くの場合、Flex アプリケーションには、Producer コンポーネントと Consumer コンポーネントのペアが少なくとも 1 つ含まれます。これにより、各アプリケーションからメッセージを宛先に送信でき、他のアプリケーションからその宛先に送信されたメッセージを受信できます。この種のアプリケーションのテストはたいへん容易であり、アプリケーションを異なるブラウザウィンドウで実行し、互いにメッセージを送信し合うことによってテストできます。

Producer コンポーネントの操作

Producer コンポーネントは MXML または ActionScript で作成できます。メッセージを送信するには、AsyncMessage オブジェクトを作成し、Producer コンポーネントの `send()` メソッドを呼び出して、メッセージを宛先に送信します。

Producer コンポーネント用の acknowledge および fault イベントハンドラを指定することもできます。acknowledge イベントは、Producer コンポーネントから送信されたメッセージを宛先が正常に受信したときにブロードキャストされます。fault イベントは、接続レベル、サーバーレベル、またはアプリケーションレベルの障害のために宛先がメッセージを正常に処理できなかったときに送出されます。

Producer コンポーネントは、メッセージアダプタを使用せずに Flex Message Service を使用する宛先にメッセージを送信でき、メッセージアダプタを使用する宛先にもメッセージを送信できます。メッセージアダプタは、Flex Message Service およびその他のメッセージングシステム間のパイプ役として機能するサーバーサイドコードです。たとえば、JMS (Java Message Service) アダプタを使用すると、Producer コンポーネントから JMS トピックにメッセージを送信できます。クライアントサイドでは、アダプタ非対応の宛先でも、アダプタ対応の宛先でも、API はまったく同じです。Producer コンポーネントがメッセージを宛先に送信したとき、または Consumer コンポーネントが宛先にサブスクライブしたときに、何が実行されるかは、Flex サービス設定ファイル内の特定の宛先の定義によって決まります。宛先の設定の詳細については、1357 ページ、第 49 章の「Message Service の設定」を参照してください。

Producer クラスの詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

MXML での Producer コンポーネントの作成

MXML で Producer コンポーネントを作成するには、`<mx:Producer>` タグを使用します。タグには `id` 値を設定する必要があります。また、サーバーサイドの services-config.xml ファイルで定義されている `destination` も指定してください。

次のコードは、宛先、acknowledge イベントハンドラ、および fault イベントハンドラを指定した `<mx:Producer>` タグです。

```
<?xml version="1.0"?>
<!-- fds\messaging\CreateProducerMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
    <mx:Script>
        <![CDATA[
            import mx.rpc.events.FaultEvent;
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private function messageHandler(event:MessageEvent):void {
                // Handle message event.
            }
            private function acknowledgeHandler(event:MessageAckEvent):void{
                // Handle message event.
            }
            private function faultHandler(event:MessageFaultEvent):void {
            // Handle fault event.
            }
        ]]>
    </mx:Script>
    <mx:Producer id="producer"
        destination="ChatTopicJMS"
        acknowledge="acknowledgeHandler(event)"
        fault="faultHandler(event)"/>
</mx:Application>
```

ActionScript での Producer コンポーネントの作成

ActionScript のメソッドで Producer コンポーネントを作成できます。次のコードは、<mx:Script> タグ内のメソッドで作成された Producer コンポーネントです。import ステートメントによって、Producer コンポーネントの作成、Message オブジェクトの作成、イベントリスナーの追加、およびメッセージハンドラの作成に、それぞれ必要なクラスが読み込まれます。

```
<?xml version="1.0"?>
<!-- fds\messaging\CreateProducerAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private var producer:Producer;
            private function acknowledgeHandler(event:MessageAckEvent):void{
                // Handle message event.
            }
            private function faultHandler(event:MessageFaultEvent):void{
            // Handle fault event.
            }
            private function logon():void {
                producer = new Producer();
                producer.destination = "ChatTopicJMS";
```

```
                producer.addEventListener(MessageAckEvent.ACKNOWLEDGE,
acknowledgeHandler);
                producer.addEventListener(MessageFaultEvent.FAULT, faultHandler);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

宛先へのメッセージの送信

Producer コンポーネントから宛先にメッセージを送信するには、mx.messaging.messages.AsyncMessage オブジェクトを作成し、AsyncMessage オブジェクトのボディを設定した後、コンポーネントの send() メソッドを呼び出します。テキストメッセージ、およびオブジェクトを含むメッセージを、作成できます。

次のコードは、メッセージを作成し、メッセージのボディにテキストを設定した後、Producer コンポーネントの send() メソッドを呼び出してメッセージを送信します。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private var producer:Producer;

            private function sendMessage():void {
                var message:AsyncMessage = new AsyncMessage();
                message.body = userName.text + ": " + input.text;
                producer.send(message);
            }
        ]]>
    </mx:Script>
    <mx:TextInput id="userName"/>
    <mx:TextInput id="input"/>
    <mx:Button label="Send" click="sendMessage();"/>
</mx:Application>
```

次のコードは、メッセージを作成し、メッセージのボディにオブジェクトを設定した後、Producer コンポーネントの send() メソッドを呼び出してメッセージを送信します。TypedObject() メソッドは、ソースパスで使用できる TypedObject という名前の ActionScript オブジェクトのコンストラクタを参照します。

```
<?xml version="1.0"?>
<!-- fds\messaging\CreateProducerMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
```

```
            import mx.rpc.events.FaultEvent;
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private function messageHandler(event:MessageEvent):void {
                // Handle message event.
            }
            private function acknowledgeHandler(event:MessageAckEvent):void{
                // Handle message event.
            }
            private function faultHandler(event:MessageFaultEvent):void {
            // Handle fault event.
            }
        ]]>
    </mx:Script>
    <mx:Producer id="producer"
        destination="ChatTopicJMS"
        acknowledge="acknowledgeHandler(event)"
        fault="faultHandler(event)"/>
</mx:Application>
```

メッセージへのその他の情報の追加

メッセージの追加情報をメッセージヘッダーの形式で含めることができます。メッセージヘッダーではストリングおよび数値を送信できます。メッセージのヘッダーは結合配列に格納されます。結合配列ではヘッダー名がキーとなります。メッセージクラスの headers プロパティを使用すると、特定のメッセージインスタンスに対してヘッダーを追加できます。

次のコードでは、prop1 というメッセージヘッダーを追加してその値を設定しています。

```
<?xml version="1.0"?>
<!-- fds\messaging\SendMessageHeader.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private var producer:Producer;

            private function sendMessage():void {
                var message:AsyncMessage = new AsyncMessage();
                message.headers = new Array();
                message.headers["prop1"] = 5;
                message.body =input.text;
                producer.send(message);
            }
        ]]>
    </mx:Script>
```

```
    <mx:TextInput id="input"/>
</mx:Application>
```

Consumer コンポーネントの selector プロパティを使用すると、メッセージヘッダーの値に応じて、コンポーネントが受信するメッセージをフィルタ処理できます。詳細については、1351 ページの「メッセージセレクタを使用したメッセージのフィルタ処理」を参照してください。

メモ

メッセージのヘッダー名を JMS または DS で始めないでください。これらの接頭辞は予約されています。

メッセージの再送信と要求のタイムアウト

Producer コンポーネントはメッセージを 1 回だけ送信します。Producer コンポーネントから送信されたメッセージが到着していない可能性がある場合は、Producer コンポーネントから fault イベントが送出されます。fault イベントは、指定されたメッセージの acknowledgment を Producer コンポーネントがまったく受信しなかったことを示します。このイベントが送出された場合、ハンドラコードでは、処理を中止するか、失敗したメッセージの再送信を試みるかを決定できます。到着していない可能性を示す fault イベントは、2 つのイベントによって発生する可能性があります。つまり、requestTimeout プロパティの値を超えた場合、または acknowledgment メッセージを受信する前に基になるメッセージチャネルが切断された場合に、fault イベントが発生する可能性があります。fault ハンドラコードでは、関連する ErrorMessage の faultCode プロパティで ErrorMessage.MESSAGE_DELIVERY_IN_DOUBT コードを調べることによって、この状態を検出できます。

Consumer コンポーネントの操作

Consumer コンポーネントは MXML または ActionScript で作成できます。宛先にサブスクライブするには、Consumer コンポーネントの subscribe() メソッドを呼び出します。

Consumer コンポーネント用の message および fault イベントハンドラを指定することもできます。message イベントは、Consumer コンポーネントのサブスクライブ先である宛先がメッセージを受信したときにブロードキャストされます。fault イベントは、Consumer コンポーネントのサブスクライブ先であるチャネルが宛先への接続を確立できないとき、サブスクリプション要求が拒否されたとき、または Consumer コンポーネントの receive() メソッドが呼び出された時点で障害が発生した場合に、ブロードキャストされます。receive() メソッドは、すべての保留中のメッセージを宛先から取得します。

Consumer クラスの詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

MXML での Consumer コンポーネントの作成

MXML で Consumer コンポーネントを作成するには、<mx:Consumer> タグを使用します。タグには id 値を設定する必要があります。また、サーバーサイドの services-config.xml ファイルで定義されている destination も指定してください。

次のコードは、宛先、acknowledge イベントハンドラ、および fault イベントハンドラを指定した <mx:Consumer> タグです。

```
<?xml version="1.0"?>
<!-- fds\messaging\CreateConsumerMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private function messageHandler(event:MessageEvent):void {
                // Handle message event.
            }
            private function faultHandler(event:MessageFaultEvent):void {
            // Handle fault event.
            }
        ]]>
    </mx:Script>
    <mx:Consumer id="consumer"
        destination="ChatTopicJMS"
        message="messageHandler(event)"
        fault="faultHandler(event)"/>
</mx:Application>
```

ActionScript での Consumer コンポーネントの作成

ActionScript のメソッドで Consumer コンポーネントを作成できます。次のコードは、<mx:Script> タグ内のメソッドで作成された Consumer コンポーネントです。import ステートメントによって、Consumer コンポーネントの作成、AsyncMessage オブジェクトの作成、イベントリスナーの追加、およびメッセージハンドラの作成に、それぞれ必要なクラスが読み込まれます。

```
<?xml version="1.0"?>
<!-- fds\messaging\CreateConsumerAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private var consumer:Consumer;
            private function acknowledgeHandler(event:MessageAckEvent):void{
                // Handle message event.
```

```
            }
            private function faultHandler(event:MessageFaultEvent):void{
            // Handle fault event.
            }
            private function logon():void {
                consumer = new Consumer();
                consumer.destination = "ChatTopicJMS";
                consumer.addEventListener
                    (MessageAckEvent.ACKNOWLEDGE,  acknowledgeHandler);
                consumer.addEventListener
                    (MessageFaultEvent.FAULT, faultHandler);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

宛先へのサブスクライブ

Consumer コンポーネントを MXML または ActionScript のどちらで作成する場合でも、コンポーネントの subscribe() メソッドを呼び出して宛先にサブスクライブし、その宛先からメッセージを受信する必要があります。

Consumer コンポーネントは、メッセージアダプタを使用せずに Flex Message Service を使用する宛先にサブスクライブでき、メッセージアダプタを使用する宛先にもサブスクライブできます。たとえば、JMS (Java Message Service) アダプタを使用すると、Consumer コンポーネントから JMS トピックにサブスクライブできます。クライアントサイドでは、アダプタ非対応の宛先でも、アダプタ対応の宛先でも、API はまったく同じです。Consumer コンポーネントが宛先にサブスクライブしたとき、または Producer コンポーネントがメッセージを宛先に送信したときに、何が実行されるかは、Flex サービス設定ファイル内の特定の宛先の定義によって決まります。宛先の設定の詳細については、1357 ページ、第 49 章の「Message Service の設定」を参照してください。

次のコードは、Consumer コンポーネントの subscribe() メソッド呼び出しです。

```
<?xml version="1.0"?>
<!-- fds\messaging\Subscribe.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private var consumer:Consumer;

            private function logon():void {
                consumer = new Consumer();
                consumer.subscribe();
            }
        ]]>
```

```
    </mx:Script>
</mx:Application>
```

Consumer コンポーネントを宛先からサブスクライブ解除するには、コンポーネントの unsubscribe() メソッドを呼び出します。

メッセージセレクタを使用したメッセージのフィルタ処理

Consumer コンポーネントの selector プロパティを使用すると、コンポーネントで受信する必要のあるメッセージをフィルタ処理できます。メッセージセレクタは、SQL92 条件式のシンタックスをベースとした SQL 条件式を含むストリングです。Consumer コンポーネントは、セレクタの条件に一致するヘッダーを持つメッセージだけを受信します。メッセージヘッダーの作成の詳細については、1347 ページの「メッセージへのその他の情報の追加」を参照してください。

次のコードでは、Consumer コンポーネントの selector プロパティを <mx:Consumer> タグで設定しています。

```
<?xml version="1.0"?>
<!-- fds\messaging\CreateConsumerMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;
            private function messageHandler(event:MessageEvent):void {
                // Handle message event.
            }
            private function faultHandler(event:MessageFaultEvent):void {
            // Handle fault event.
            }
        ]]>
    </mx:Script>
    <mx:Consumer id="consumer"
        destination="ChatTopicJMS"
        selector="prop1 > 5"
        message="messageHandler(event)"
        fault="faultHandler(event)"/>
</mx:Application>
```

次のコードでは、Consumer コンポーネントの selector プロパティを ActionScript で設定しています。

```
<?xml version="1.0"?>
<!-- fds\messaging\SelectorAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.events.*;
```

```
            private var consumer:Consumer;

            private function logon():void {
                consumer = new Consumer();
                consumer.destination = "ChatTopic";
                consumer.selector = "prop1 > 5";
                consumer.subscribe();
            }
        ]]>
    </mx:Script>
</mx:Application>
```

サブトピックの使用

サブトピック機能を利用すると、Producer コンポーネントが宛先に送信するメッセージを、宛先で特定のカテゴリに分割することができます。宛先にサブスクライブする Consumer コンポーネントを、特定のサブトピックまたは一連のサブトピックに送信されたメッセージだけを受信するように設定できます。ワイルドカード文字 (*) を使用して、複数のサブトピックとの間でメッセージを送受信します。

JMS 宛先ではサブトピックを使用できません。しかし、メッセージヘッダーと Consumer セレクタ式を使用すると、JMS の使用時でも同様の機能を実現できます。詳細については、1351 ページの「メッセージセレクタを使用したメッセージのフィルタ処理」を参照してください。

Producer コンポーネントの subtopic プロパティで、コンポーネントがメッセージを送信するサブトピックを指定します。Consumer コンポーネントの subtopic プロパティで、Consumer のサブスクライブ先のサブトピックを指定します。

Producer コンポーネントから宛先とサブトピックにメッセージを送信するには、次の例に示すように destination および subtopic プロパティを設定し、send() メソッドを呼び出します。

```
<?xml version="1.0"?>
<!-- fds\messaging\Subtopic1.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;

            private function acknowledgeHandler(event:MessageAckEvent):void {
                // Handle message acknowledgement event.
            }
            private function faultHandler(event:MessageFaultEvent):void {
            // Handle fault event.
            }
            private function useSubtopic():void {
                var message:AsyncMessage = new AsyncMessage();
```

```
                producer.subtopic = "chat.fds.newton";
                // Generate message.
                producer.send(message);
            }
        ]]>
    </mx:Script>
    <mx:Producer id="producer"
        destination="ChatTopicJMS"
        acknowledge="acknowledgeHandler(event)"
        fault="faultHandler(event)"/>
</mx:Application>
```

Consumer コンポーネントで宛先とサブトピックにサブスクライブするには、次の例に示すように destination および subtopic プロパティを設定し、subscribe() メソッドを呼び出します。この例では、ワイルドカード文字 (*) を使用して、chat.fds サブトピックの下のすべてのサブトピックに送信されるすべてのメッセージを受信します。

```
<?xml version="1.0"?>
<!-- fds\messaging\Subtopic2.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;

            private function messageHandler(event:MessageEvent):void {
                // Handle message event.
            }
            private function faultHandler(event:MessageFaultEvent):void {
            // Handle fault event.
            }
            private function useSubtopic():void {
                consumer.destination = "ChatTopic";
                consumer.subtopic = "chat.fds.*";
                consumer.subscribe();
            }
        ]]>
    </mx:Script>
    <mx:Consumer id="consumer" destination="ChatTopicJMS"
         message="messageHandler(event)"
         fault="faultHandler(event)"/>
</mx:Application>
```

宛先のサブトピックを許可するには、"services-config.xml" ファイルまたは参照によって含められるファイル ("messaging-config.xml" ファイルなど) で、allow-subtopics エレメントを true に設定する必要があります。subtopic-separator エレメントはオプションで、デフォルト値は "." (ピリオド) です。

```
<destination id="ChatTopic">
  <properties>
```

```
    <network>
      <session-timeout>0</session-timeout>
    </network>
    <server>
      <max-cache-size>1000</max-cache-size>
      <message-time-to-live>0</message-time-to-live>
      <durable>false</durable>
      <allow-subtopics>true</allow-subtopics>
      <subtopic-separator>.</subtopic-separator>
    </server>
  </properties>
  <channels>
    <channel ref="my-rtmp"/>
  </channels>
</destination>
```

次の例は、サブトピックを使用する実行可能なアプリケーションです。この場合は、Producer および Consumer コンポーネントは MXML タグで宣言されます。Consumer コンポーネントは、ワイルドカード文字 (*) を使用して、chat.fds サブトピックの下のすべてのメッセージを受信します。

```
<?xml version="1.0"?>
<!-- fds\messaging\Subtopic3.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.messaging.*;
            import mx.messaging.messages.*;
            import mx.messaging.events.*;

            private function sendMessage():void {
                myCon.subscribe();
                var message:AsyncMessage = new AsyncMessage();
                message.body = userName.text;
                myPub.send(message);
            }
            private function messageHandler(event: MessageEvent):void {
                ta.text += event.message.body + "\n";
            }
        ]]>
    </mx:Script>

    <mx:Producer id="myPub" destination="mike" subtopic = "chat.fds.blue"/>
    <mx:Consumer id="myCon" destination="mike" subtopic="chat.fds.*"
        message="messageHandler(event);"/>


    <mx:TextInput id="userName" width="20%"/>
    <mx:TextInput id="msg" width="20%"/>
    <mx:Button label="Send" click="sendMessage();"/>
    <mx:TextArea id="ta"/>
</mx:Application>
```

アプリケーションでの Producer コンポーネントと Consumer コンポーネントのペアの使用

多くの場合、Flex アプリケーションには、Producer コンポーネントと Consumer コンポーネントのペアが少なくとも 1 つ含まれます。これにより、各アプリケーションからメッセージを宛先に送信でき、他のアプリケーションからその宛先に送信されたメッセージを受信できます。

ペアとして機能するためには、アプリケーション内の Producer コンポーネントと Consumer コンポーネントで同じメッセージ宛先を使用する必要があります。Producer コンポーネントのインスタンスはメッセージを宛先に送信し、Consumer コンポーネントのインスタンスはその宛先からメッセージを受信します。

次のコードは簡単なチャットアプリケーションからの抜粋で、Producer コンポーネントと Consumer コンポーネントのペアが含まれています。ユーザーは `msg` という TextInput コントロールにメッセージを入力します。ユーザーがキーボードの Enter キーを押すか、"Send" というラベルの付いた Button コントロールをクリックすると、Producer コンポーネントによってメッセージが送信されます。`ta` という TextArea コントロールには、他のユーザーからのメッセージが表示されます。ユーザーはトピック List コントロール内のアイテムを選択することで、JMS 非対応の宛先と JMS 対応の宛先を切り替えることができます。

```
<?xml version="1.0"?>
<!-- fds\messaging\ProducerConsumer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[

            import mx.messaging.messages.*;
            import mx.messaging.events.*;

            private function logon():void {
                producer.destination = topics.selectedItem.toString();
                consumer.destination = topics.selectedItem.toString();
                consumer.subscribe();
                topics.selectedIndex = 1;
            }

            private function messageHandler(event: MessageEvent):void {
                ta.text += event.message.body + "\n";
            }

            private function sendMessage():void {
                var message: AsyncMessage = new AsyncMessage();
                message.body = userName.text + ": " + msg.text;
                producer.send(message);
                msg.text = "";
            }
```

```
        ]]>
    </mx:Script>

    <mx:Producer id="producer" destination="ChatTopicJMS"/>
    <mx:Consumer id="consumer" destination="ChatTopicJMS"
        message="messageHandler(event)"/>

    <mx:List id="topics">
        <mx:dataProvider>
            <mx:ArrayCollection>
                <mx:String>chat-topic</mx:String>
                <mx:String>chat-topic-jms</mx:String>
            </mx:ArrayCollection>
        </mx:dataProvider>
    </mx:List>

    <mx:TextArea id="ta" width="100%" height="100%"/>
    <mx:TextInput id="userName" width="100%"/>
    <mx:TextInput id="msg" width="100%"/>
    <mx:Button label="Send" click="sendMessage()"/>
</mx:Application>
```

メモ

Flex のサンプルアプリケーションには、完成版のチャットアプリケーションのコードが付属しています。

第 49 章

Message Service の設定

49

Adobe Flex クライアントアプリケーションでメッセージングを可能にするには、<mx:Producer> および <mx:Consumer> タグで、または対応する ActionScript API で、サーバーサイド宛先への接続を宣言することによって、Flex Message Service に接続します。Message Service の宛先は、パブリッシュ / サブスクライブメッセージングまたはポイントツーポイントメッセージングを実行する際にメッセージの送信先および受信元となるエンドポイントです。宛先は、Flex サービス設定ファイルの中で Message Service 定義の一部として設定します。このトピックでは、宛先の設定方法について説明します。

MXML または ActionScript でのクライアントサイドのメッセージング API の使用と宛先への接続に関する詳細については、1343 ページ、第 48 章の「Flex メッセージングの使用」を参照してください。

目次

Message Service の設定について

Message Service を設定する際に実行する通常の作業としては、メッセージ宛先の定義、メッセージ宛先へのセキュリティの適用、およびログ設定の変更があります。Message Service 宛先は、Producer コンポーネントと Consumer コンポーネントを使用して接続する接続先のサーバーサイドコードです。Message Service 宛先は、Flex サービス設定ファイル、またはそのファイルを参照によってインクルードするファイルの、Message Service セクションで設定します。デフォルトでは、Flex サービス設定ファイルの名前は services-config.xml であり、Adobe Flex データサービスのある Web アプリケーションの WEB_INF/flex ディレクトリにあります。Adobe から提供されている Flex サービス設定ファイルでは、通常は、Flex データサービス用、Flex Message Service 用、および RPC (リモートプロシージャコール) サービス用の、個々の設定ファイルが参照されています。

次の例は、Flex サービス設定ファイルでの Message Service の基本的な設定を示しています。この設定には、デフォルトのメッセージアダプタ (この例では ActionScript アダプタ) を使用するチャットアプリケーション用に設定された宛先が 1 つあります。

```
...
<service id="message-service"
  class="flex.messaging.services.MessageService"
  messageTypes="flex.messaging.messages.AsyncMessage">
  <adapters>
    <adapter-definition id="actionscript"
    class="flex.messaging.services.messaging.
      adapters.ActionScriptAdapter" default="true"/>
    <adapter-definition id="jms"
      class="flex.messaging.services.messaging.adapters.JMSAdapter"/>
   </adapters>
...
  <destination id="chat-topic">
    <properties>
      <network>
        <session-timeout>0</session-timeout>
        <throttle-inbound policy="ERROR" max-frequency="50"/>
        <throttle-outbound policy="REPLACE" max-frequency="500"/>
      </network>
      <server>
        <max-cache-size>1000</max-cache-size>
        <message-time-to-live>0</message-time-to-live>
        <durable>true</durable>
        <durable-store-manager>
          flex.messaging.durability.FileStoreManager
        </durable-store-manager>
      </server>
    </properties>
    <channels>
      <channel ref="samples-rtmp"/>
      <channel ref="samples-amf-polling"/>
    </channels>
```

```
    </destination>
...
</service>
...
```

Message Service 宛先

宛先を定義するときは、メッセージを伝達するメッセージチャネルを 1 つまたは複数参照します。また、メッセージアダプタを参照するか、デフォルトアダプタとして設定されたアダプタを使用します。ActionScript アダプタを使用すると、Flex クライアントとのメッセージングを、メッセージの単独のプロデューサおよびコンシューマとして使用できます。JMS (Java Message Service) メッセージアダプタを使用すると、JMS の実装とやり取りすることができます。詳細については、1359 ページの「Message Service アダプタ」を参照してください。

また、ネットワーク関連プロパティとサーバー関連プロパティも設定します。宛先に対するセキュリティ制限を参照または定義することもできます。詳細については、1360 ページの「Message Service 宛先の設定」を参照してください。

メッセージチャネル

宛先では 1 つまたは複数のメッセージチャネルを参照します。メッセージチャネルは services-config.xml 設定ファイルで定義します。メッセージングで最もよく使用される 2 つのチャネルとして、RTMP (Realtime Message Protocol) チャネルと、メッセージのポーリングが有効にされた AMF (Action Message Format) チャネルがあります。

RTMP チャネルではクライアントとサーバー間の接続が管理されるため、クライアントでサーバーをポーリングする必要がありません。ポーリングが有効にされた AMF チャネルでは、サーバーがポーリングされ、新しいメッセージがないかが調べられます。

メッセージチャネルのセキュリティ保護の詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。

Message Service アダプタ

Message Service アダプタは、アプリケーションで ActionScript オブジェクトのみを使用する場合や、JMS の実装などの別のシステムとやり取りする場合に、メッセージングを容易にするサーバーサイドコードです。アダプタは宛先定義の中で参照します。アダプタ固有の設定を指定することもできます。メッセージアダプタの詳細については、1360 ページの「Message Service 宛先の設定」を参照してください。

セキュリティ

宛先をセキュリティ保護する方法の1つとして、宛先に対するアクセス権限を定義する“セキュリティ制限”の使用があります。

セキュリティ制限は、ユーザーが宛先にアクセスできるようにする前にユーザーの認証および承認を行う場合に使用します。認証では、基本認証を使用するかカスタム認証を使用するかを指定できます。承認では、承認に必要なロールを指定できます。

セキュリティ制限は宛先定義内でインラインで宣言できます。または、セキュリティ制限をグローバルに宣言し、宛先定義内でその id によって参照できます。

セキュリティの詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。

Message Service 宛先の設定

このセクションでは、Message Service 宛先の作成について説明します。

メッセージチャネルの参照

メッセージは、メッセージチャネルを経由して宛先に、または宛先から、伝達されます。宛先では1つまたは複数のメッセージチャネルを参照します。メッセージチャネルは fds.config.xml 設定ファイルでも定義されます。宛先へは、列挙されたチャネルのうち1つだけを使用して接続できます。

列挙されていないチャネルで宛先に接続しようとした場合はエラーが発生します。Flex アプリケーションではこの設定情報をもとに、どのチャネルを使用して指定された宛先に接続するのかを判断します。Flex アプリケーションは、参照されているチャネルの使用を、指定された順序で試行します。デフォルトのメッセージチャネルを設定することもできます。この場合は、宛先でチャネルを明示的に参照する必要はありません。

次の例は、宛先のチャネル参照を示しています。samples-rtmp チャネルが最初に列挙されているので、宛先はそのチャネルの使用を最初に試みます。

```
...
  <destination id="chat-topic">
...
    <channels>
      <channel ref="samples-rtmp"/>
      <channel ref="samples-amf-polling"/>
    </channels>
...
  </destination>
...
```

メッセージチャネルの詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。

ネットワークプロパティの設定

宛先には、クライアントとサーバー間のメッセージング動作を定義するための一連のプロパティがあります。次の例は、宛先のネットワーク関連プロパティを示しています。

```
...
  <destination id="chat-topic">
    <properties>
      <network>
        <session-timeout>0</session-timeout>
        <throttle-inbound policy="ERROR" max-frequency="50"/>
        <throttle-outbound policy="REPLACE" max-frequency="500"/>
      </network>
  ...
    </properties>
  </destination>
...
```

Message Service 宛先では次のネットワーク関連プロパティが使用されます。

プロパティ	説明
session-timeout	サブスクライバがサブスクライブ解除されるまでのアイドル時間 (分数)。値が 0 に設定された場合、サブスクライバでは自動的にサブスクライブ解除が強制されることがなくなります。
throttle-inbound	max-frequency 属性は、サーバーが受信できる毎秒のメッセージ数を制御します。policy 属性は、メッセージの上限に達したときに実行する動作を指定します。 policy の値が ERROR の場合は、上限に達したときにエラーを返すことを指定します。policy の値が IGNORE の場合は、上限に達したときにエラーを返さないことを指定します。
throttle-outbound	max-frequency 属性は、サーバーが送信できる毎秒のメッセージ数を制御します。policy 属性は、メッセージの上限に達したときに実行する動作を指定します。 policy の値が ERROR の場合は、上限に達したときにエラーを返すことを指定します。policy の値が IGNORE の場合は、上限に達したときにエラーを返さないことを指定します。policy の値が REPLACE の場合は、上限に達したときに前のメッセージを置き換えることを指定します。

サーバープロパティの設定

宛先には、サーバー関連のパラメータを制御するための一連のプロパティがあります。次の例は、宛先のサーバー関連プロパティを示しています。

```
...
  <destination id="chat-topic">
    <properties>
...
      <server>
        <max-cache-size>1000</max-cache-size>
        <message-time-to-live>0</message-time-to-live>
        <durable>true</durable>
        <durable-store-manager>
          flex.messaging.durability.FileStoreManager
        </durable-store-manager>
      </server>
    </properties>
  </destination>
...
```

Message Service 宛先では次のサーバー関連プロパティが使用されます。

プロパティ	内容
max-cache-size	メモリキャッシュに維持するメッセージの最大数。
message-time-to-live	メッセージをサーバー上で保持するミリ秒数。 値 0 は、メッセージを永続的に保持することを示します。
durable	メッセージを永続的メッセージストアに保存するかどうかを示すブール値。メッセージストアへの保存によって、接続障害が発生してもメッセージが残り、メッセージが宛先のサブスクライバに到達することが保証されます。 JMS アダプタを使用している場合は、アダプタで durable の値が継承されます。
durable-store-manager	JMS アダプタを使用しない場合に使用する永続的ストアマネージャクラス。デフォルトでは、Flex データサービスに組み込まれている flex.messaging.durability.FileStoreManager によって、Flex Web アプリケーションの /WEB-INF/flex/message_store/< 宛先名 > にあるファイルにメッセージが格納されます。この場所は、file-store-root プロパティを設定することによって変更できます。 **メモ :** Flex データサービスの FileStoreManager を永続的なメッセージングに使用した場合、クラスタセーフではなくなります。
batch-write-size	(オプション) 各バッチで書き込む永続メッセージファイルの数。
file-store-root	(オプション) 永続メッセージファイルの格納場所。
max-file-size	(オプション) 永続メッセージファイルの最大ファイルサイズ (キロバイト単位)。

Message Service アダプタの参照

Flex データサービスに組み込まれている ActionScript アダプタ、JMS アダプタ、ColdFusion Event Gateway Adapter などの Message Service アダプタを使用するには、宛先定義でアダプタを参照するか、デフォルトアダプタとして設定されたアダプタを使用します。アダプタの参照に加えて、宛先定義内でアダプタのプロパティを設定することもできます。JMS アダプタの詳細については、1363 ページの「JMS アダプタの設定」を参照してください。

次の例は、JMS アダプタの定義と、宛先での JMS アダプタへの参照を示しています。

```
...
<service id="message-service"
  class="flex.messaging.services.MessageService"
  messageTypes="flex.messaging.messages.AsyncMessage">
...
  <adapters>
    <adapter-definition id="actionscript"
    class="flex.messaging.services.messaging.
      adapters.ActionScriptAdapter" default="true"/>
    <adapter-definition id="jms"
      class="flex.messaging.services.messaging.adapters.JMSAdapter"/>
   </adapters>

...

  <destination id="chat-topic-jms">
...
    <adapter ref="jms"/>
...
  </destination>
...
</service>
```

JMS アダプタの設定

JMS アダプタは、JMS の実装で設定された JMS トピックまたはキューにサブスクライブするために使用します。このアダプタを使用することで、Java の JMS 発行者およびサブスクライバが Flex クライアントアプリケーションと同じ宛先を共有できるようになります。Java コードでは、Flex アプリケーションにメッセージをパブリッシュでき、Flex アプリケーションからパブリッシュされたメッセージに応答できます。

JMS アダプタは、アダプタを使用する宛先ごとに個別に設定します。JNDI (Java Naming and Directory Interface) 情報と、JNDI 内の接続ファクトリを参照するための JMS ConnectionFactory 情報を、それぞれ適切に使用してアダプタを設定する必要があります。そして、アダプタからの接続を作成し、Flex クライアントに代わってセッションを作成し、Flex クライアントに代わって発行者とサブスクライバを作成する必要があります。

次の例は、JMS アダプタを使用する宛先を示しています。

```
...
<destination id="chat-topic-jms">
  <properties>
...
     <jms>
       <destination-type>Topic</destination-type>
       <message-type>javax.jms.TextMessage</message-type>
       <connection-factory>jms/flex/TopicConnectionFactory
       </connection-factory>
       <destination-jndi-name>jms/topic/flex/simpletopic
       </destination-jndi-name>
       <destination-name>FlexTopic
       </destination-name>
       <delivery-mode>NON_PERSISTENT</delivery-mode>
       <message-priority>DEFAULT_PRIORITY</message-priority>
       <acknowledge-mode>AUTO_ACKNOWLEDGE</acknowledge-mode>
       <transacted-sessions>false</transacted-sessions>
  <!-- ( オプション ) JNDI 環境。JMS をリモートの JNDI サーバーで使用する場合に使用します。
  これを使用して、外部の JMS プロバイダにアクセスするための JNDI 環境を指定します。 -->
  <initial-context-environment>
    <property>
      <name>Context.SECURITY_PRINCIPAL</name>
      <value>anonymous</value>
    </property>
    <property>
      <name>Context.SECURITY_CREDENTIALS</name>
      <value>anonymous</value>
    </property>
    <property>
      <name>Context.PROVIDER_URL</name>
      <value>http://{server.name}:1856</value>
    </property>
    <property>
       <name>Context.INITIAL_CONTEXT_FACTORY</name>
       <value>fiorano.jms.runtime.naming.FioranoInitialContextFactory
       </value>
    </property>
  </initial-context-environment>

     </jms>
  </properties>
...
<adapter ref="jms"/>
</destination>
...
```

JMS アダプタは次の設定プロパティを受け入れます。JMS の詳細については、Java Message Service の仕様書、またはアプリケーションサーバーのマニュアルを参照してください。

プロパティ	内容
destination-type	(オプション) アダプタが実行しているメッセージングのタイプ。有効な値は、パブリッシュ / サブスクライブメッセージングの場合は topic、ポイントツーポイントメッセージングの場合は queue です。デフォルト値は topic です。
message-type	Flex メッセージを JMS メッセージに変換する際に使用するメッセージのタイプ。サポートされているタイプは、javax.jms.TextMessage および javax.jms.ObjectMessage です。クライアントサイドの Publisher コンポーネントからメッセージをオブジェクトとして送信する場合は、message-type を javax.jms.ObjectMessage に設定する必要があります。
connection-factory	JNDI 内の JMS 接続ファクトリの名前。
destination-jndi-name	JNDI レジストリ内の宛先の名前。
destination-name	(オプション) JMS 内の宛先の名前。デフォルト値は Flex の宛先の id です。
delivery-mode	プロデューサの JMS DeliveryMode。 有効な値は PERSISTENT および NON_PERSISTENT です。 PERSISTENT モードでは、送信されたメッセージがすべて JMS サーバーによって格納されてから、コンシューマに転送されます。これにより処理のオーバーヘッドが増加しますが、配信を保証する場合には必要になります。NON_PERSISTENT モードでは、JMS サーバーでメッセージを格納してからコンシューマに転送する必要はなくなります。そのため、メッセージの処理中に JMS サーバーが障害を起こした場合、メッセージが失われる可能性があります。この設定は、配信の保証を必要としない通知メッセージの場合に適しています。
message-priority	Flex プロデューサから送信されるメッセージの JMS 優先順位。有効な値は、DEFAULT_PRIORITY、または必要な優先順位を示す整数値です。JMS API では 10 レベルの優先順位値が定義されています。0 は最低の優先順位、9 は最高の優先順位です。さらに、クライアントでは 0 ～ 4 を通常の優先順位の等級、および 5 ～ 9 を効率的な優先順位の等級として、それぞれ扱ってください。
acknowledge-mode	JMS メッセージ確認モード。有効な値は、AUTO_ACKNOWLEDGE、DUPS_OK_ACKNOWLEDGE、および CLIENT_ACKNOWLEDGE です。

プロパティ	内容
transacted-sessions	JMS セッショントランザクションモード。
initial-context-environment	JNDI での ConnectionFactory および Destination の参照に使用される InitialContext を設定するための一連の JNDI プロパティ。リモートの JNDI サーバーを JMS に使用できるようにします。詳細については、1366 ページの「リモート JMS プロバイダの使用」を参照してください。

リモート JMS プロバイダの使用

JMS アダプタを使用するメッセージ宛先の jms セクションにあるオプションの initial-context-environment エレメントを設定することによって、リモートの JNDI サーバーで JMS を使用できます。initial-context-environment エレメントは property サブエレメントを受け取ります。このサブエレメントはさらに name サブエレメントと value サブエレメントを受け取ります。name サブエレメントと value サブエレメントでは、javax.naming.Context 定数名と対応する値を指定するか、またはストリングリテラル名と対応する値を指定することによって、必要な JNDI 環境を確立できます。

次の例では、initial-context-environment の設定をボールド体のコードで示しています。

```
...
<destination id="chat-topic-jms">
  <properties>
...
     <jms>
       <destination-type>Topic</destination-type>
       <message-type>javax.jms.TextMessage</message-type>
       <connection-factory>jms/flex/TopicConnectionFactory
       </connection-factory>
       <destination-jndi-name>jms/topic/flex/simpletopic
       </destination-jndi-name>
       <destination-name>FlexTopic
       </destination-name>
       <delivery-mode>NON_PERSISTENT</delivery-mode>
       <message-priority>DEFAULT_PRIORITY</message-priority>
       <acknowledge-mode>AUTO_ACKNOWLEDGE</acknowledge-mode>
       <transacted-sessions>false</transacted-sessions>
  <!-- ( オプション ) JNDI 環境。JMS をリモートの JNDI サーバーで使用する場合に使用します。
  -->
  <initial-context-environment>
    <property>
      <name>Context.SECURITY_PRINCIPAL</name>
      <value>anonymous</value>
    </property>
    <property>
      <name>Context.SECURITY_CREDENTIALS</name>
      <value>anonymous</value>
```

```
    </property>
    <property>
      <name>Context.PROVIDER_URL</name>
      <value>http://{server.name}:1856</value>
    </property>
    <property>
      <name>Context.INITIAL_CONTEXT_FACTORY</name>
      <value>fiorano.jms.runtime.naming.FioranoInitialContextFactory</value>
    </property>
  </initial-context-environment>

    </jms>
  </properties>
...
<adapter ref="jms"/>
</destination>
...
```

Flex では、テキスト `Context.` で始まる `name` エレメントの値が、javax.naming.Context によって定義された定数として扱われ、指定した定数が Context クラスで定義されているかが確認されます。JMS プロバイダによってはカスタムプロパティを最初のコンテキストの中で設定できるものもあり、プロバイダで必要となるストリングリテラル名と対応する値を使用することによって、それらのカスタムプロパティを指定できます。たとえば、FioranoMQ JMS プロバイダでは次のプロパティによってバックアップサーバーへのフェイルオーバーが設定されます。

```
<property>
   <name>BackupConnectURLs</name>
   <value>http://backup-server:1856;http://backup-server-2:1856</value>
</property>
```

`initial-context-environment` プロパティを宛先定義の `jms` セクションで指定しない場合は、デフォルトの JNDI 環境が使用されます。デフォルトの JNDI 環境は、jndiprovider.properties アプリケーションリソースファイルで、または jndi.properties ファイルで設定されます。

使用する JNDI 環境によっては、connection-factory および destination-jndi-name 設定エレメントで、ターゲット名を持つインスタンスをディレクトリ内で正しく参照する必要があります。トピック接続ファクトリおよび宛先に対する JNDI プロバイダ間の命名規則はそれぞれ異なります。また、JMS プロバイダのクライアントライブラリ JAR ファイルを、Flex Web アプリケーションの WEB-INF/lib ディレクトリか、またはクラスローダーによってロードされる別の場所に、インクルードする必要もあります。外部の JMS プロバイダを使用する場合でも、Flex データサービスは `connection-factory` および `destination-jndi-name` 設定エレメントを使用して、必要な接続ファクトリおよび宛先のインスタンスを参照します。

カスタムの Message Service アダプタの作成

どの標準のアダプタでも提供されていない機能を必要とする場合には、カスタムの Message Service アダプタを作成できます。Message Service アダプタクラスでは flex.messaging.services.ServiceAdapter クラスを拡張する必要があります。アダプタは、flex.messaging.MessageService オブジェクトのインスタンスのメソッドを呼び出します。ServiceAdapter および MessageService は、公開されている Flex データサービス Javadoc マニュアルに含まれています。

どの Message Service アダプタクラスでも、その主要なメソッドは `invoke()` メソッドです。このメソッドは、クライアントから宛先にメッセージが送信されたときに呼び出されます。`invoke()` メソッドには、サブスクライブしているすべてのクライアントにメッセージを送信するコードを含めることができます。または、クライアントからのメッセージに含まれているセレクタステートメントを評価することによって、特定のクライアントにメッセージを送信することもできます。

クライアントにメッセージを送信するには、アダプタの `invoke()` メソッドで `MessageService.pushMessageToClients()` メソッドを呼び出します。このメソッドは、第 1 パラメータとしてメッセージオブジェクトを受け取ります。第 2 パラメータは、メッセージセレクタステートメントを評価するかどうかを示すブール値です。アダプタの `invoke()` メソッドで `MessageService.sendPushMessageFromPeer()` メソッドを呼び出すと、クラスタ化環境でピアサーバーノードにメッセージをブロードキャストできます。

```
package customclasspackage;

import flex.messaging.services.ServiceAdapter;
import flex.messaging.services.MessageService;
import flex.messaging.messages.Message;
import flex.messaging.Destination;

public class SimpleCustomAdapter extends ServiceAdapter {

    public Object invoke(Message message) {
        MessageService msgService = (MessageService)service;
        msgService.pushMessageToClients(message, true);
        msgService.sendPushMessageFromPeer(message, true);
        return null;
    }
}
```

オプションで、Message Service アダプタで独自のサブスクリプションを管理することもできます。これには、`ServiceAdapter.handlesSubscriptions()` メソッドをオーバーライドし、`true` を返します。また、`ServiceAdapter.manage()` メソッドをオーバーライドする必要もあります。このメソッドには、サブスクライブ操作およびサブスクライブ解除操作のための CommandMessages が渡されます。

ServiceAdapter クラスの getAdapterState() メソッドと setAdapterState() メソッドは、クラスタ全体で複製する必要のあるメモリ内状態を維持するアダプタで使用します。アダプタの起動時に、実行中の別のノードが存在する場合、アダプタは別のクラスタノードから状態のコピーを取得します。

アダプタクラスを使用するには、次の例のように、Message Services 設定の adapter-definition エレメントでアダプタクラスを指定する必要があります。

```
<adapters>
...
   adapter-definition id="cfgateway" class="foo.bar.SampleMessageAdapter"/>
...
</adapters>
```

アダプタで実装できるもう1つのオプション機能として、MBean コンポーネントの管理があります。これを使用すると、管理コンソールを取得および設定するためのプロパティを公開できます。詳細については、1270 ページ、第 43 章の「サービスの管理」を参照してください。

第 50 章

Flex Data Management Service について

50

Adobe Flex Data Management Service はクライアント層とサーバー層に及ぶ機能で、Flex アプリケーション内で分散データを利用するための最上位の機能を提供します。この機能を使用してアプリケーションを構築すると、データの同期、データの複製、および状況に応じて接続するアプリケーションサービスを実現できます。また、大量のデータのコレクションやネストされたデータの関係 (1 対 1 や 1 対多の関係など) を管理したり、Data Management Service を使用してデータリソースを統合したりできます。

クライアントサイドの DataService コンポーネントは MXML または ActionScript で作成できます。クライアントサイドの DataService コンポーネントは、サーバーサイドの Data Management Service で設定された宛先のメソッドを呼び出します。クライアントサイドコンポーネントは、クライアントサイドのオブジェクトにリモートデータソースのデータを設定する、複数のクライアントインスタンスのデータとサーバーサイドの宛先のデータとの同期をとる、などのアクティビティを実行します。このトピックでは、Data Management Service の概要について紹介します。

目次

Data Management Service 機能について

Data Management Service 機能は、クライアントサイド機能とサーバーサイド機能を組み合わせて使用し、複数のクライアントとサーバー層との間でデータを分散します。クライアントサイドの DataService コンポーネントは、サーバーサイドの Data Management Service と連携して分散データの管理機能を提供する ActionScript オブジェクトです。クライアントサイドの DataService コンポーネントはクライアントのデータを管理し、サーバーサイドの Data Management Service は複数のクライアントとサーバーサイドデータリソースとの間におけるデータの分散を管理します。DataService コンポーネントによって管理されるデータは、多くの場合、Data Management Service を通じて、リモートデータソースまたは別のクライアントアプリケーションから複製されます。

Data Management Service 機能と RPC 機能との比較

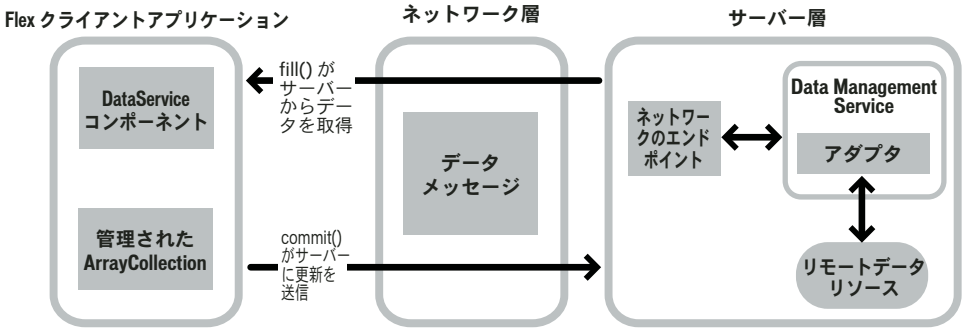
Data Management Service 機能では、Flex RPC 機能の RPC (リモートプロシージャコール) におけるアプローチとは根本的に異なるデータアプローチが利用されています。RPC 機能とは異なり、Data Management Service 機能では、複数のクライアント上のデータとサーバーサイドデータリソース上のデータからなる共通のデータセットについて、自動および手動の同期がサポートされています。また、状況に応じて接続するクライアントに対して、オフラインでのクライアントサイドデータの永続性もサポートされています。

RPC コンポーネントとは異なり、DataService コンポーネントでは、オブジェクトにバインドするための静的な結果を返すリモートサービスの処理やメソッドを直接呼び出すことはありません。そうではなく、Data Management Service から複数のクライアントに一連のデータが複製され、データが変更されるたびに、変更が自動的に Data Management Service に設定されます。そして、Data Management Service によって、基になるデータリソースとクライアントアプリケーションが更新されます。

データの流れについて

Data Management Service 機能の中心となるのが Flex メッセージングフレームワークです。このフレームワークは、Flex クライアントアプリケーションと Data Management Service との間でデータメッセージを受け渡します。データメッセージはメッセージチャネルを経由してネットワーク層間で受け渡されます。メッセージチャネルによって、メッセージのプロデューサとコンシューマが宛先に接続されます。Data Management Service では複数の異なるメッセージチャネルがサポートされており、HTTP、AMF (Action Message Format)、RTMP (Realtime Message Protocol) の各プロトコルをサポートするチャネルがサポートされています。Data Management Service のクライアントアプリケーションは、データを Data Management Service 宛先に送信するメッセージプロデューサとしても機能し、宛先にサブスクライブしてデータ更新を受信するメッセージコンシューマとしても機能します。

次の図は、バックエンドのデータリソースと ArrayCollection オブジェクト間のデータの流れを示しています。ArrayCollection オブジェクトは Flex アプリケーション内で DataService コンポーネントによって管理されます。



Data Management Service では "Data Management Service アダプタ " を使用して、特定のデータストアタイプに適した方法でデータストアを更新します。いずれかのクライアント側のデータか、または Data Management Service 側のデータが変更されると、クライアント側とサーバー側のすべてのデータが Data Management Service によって自動的に同期されます。データベースやディレクトリなどのバッキングデータストアに対する変更は、Data Management Service やクライアントアプリケーションには自動的には伝播されません。また、Data Management Service 機能では、データ同期の競合を処理するためのサーバーサイドおよびクライアントサイドの競合解決 API も提供されています。

クライアントサイドでは、DataService コンポーネントの `fill()` メソッドを呼び出し、サーバーサイドの Data Management Service 宛先からデータを取得して、ArrayCollection に設定します。また、DataService コンポーネントの `getItem()` メソッドを呼び出して個々のデータアイテムを取得することもできます。クライアント側のデータが変更されると、DataService コンポーネントの `commit()` メソッドがトリガされます。このメソッドは、データ変更を Data Management Service 宛先に送信します。そして、これらの変更が他のクライアントに伝播されます。Data Management Service 宛先では、クライアントとサーバーサイドデータリソース間のデータの同期を管理します。データアイテムの追加または削除を実行するには、DataService コンポーネントの `createItem()` または `deleteItem()` メソッドを呼び出します。

Data Management Service 機能では、ドメインモデルを構造化して各層間でデータを分散するための J2EE のベストプラクティスがサポートされています。

データ同期の競合の解決

Data Management Service 機能では、アプリケーション間において同期された方法で分散データを共有および更新できます。複数のクライアントで同じデータ部分が同時に変更された場合、データ同期の競合が発生する可能性があります。Data Management Service 機能は、データ同期の競合を検出し処理するための競合解決 API を備えており、競合が解決されるまで以降の変更を制限することによって、分散データの整合性を保ちます。

Data Management Service 宛先は、クライアントからのデータ更新が古いことを検出して、競合をクライアントに知らせることができます。競合解決の実装に必要なサーバーサイドコードの量は、使用するアダプタのタイプによって異なります。競合の例外を処理するクライアントの ActionScript コードは、アプリケーションにとって最適な方法で記述できます。

第 51 章

51 アプリケーションでのデータの分散

Adobe Flex データサービス に組み込まれているクライアントサイドの DataService コンポーネントは、サーバーサイドの Flex Data Management Service と組み合わせて使用し、複数のクライアントアプリケーション間でのデータの分散と同期を行います。このトピックでは、分散データの共有と同期を実行できるクライアントサイドの Flex アプリケーションを作成する方法について説明します。サーバーサイドの Data Management Service の設定に関する詳細については、1389 ページ、第 52 章の「Data Management Service の設定」を参照してください。

目次

分散データアプリケーションの作成

Flex クライアントアプリケーションはクライアントサイドの DataService コンポーネントを使用して、サーバーサイドの Data Management Service との間でデータの送受信を行います。データは、プロトコル固有のメッセージチャネルを経由して Data Management Service 宛先との間で受け渡されます。DataService コンポーネントでは、クライアントサイドの ArrayCollection オブジェクトにデータを設定できます。また、他のクライアント上およびサーバー上にあるデータとの間で、ArrayCollection オブジェクトのデータの同期を管理できます。DataService コンポーネントは MXML または ActionScript で作成できます。

DataService コンポーネントでは有効な Data Management Service 宛先が必要になります。宛先は、services-config.xml 設定ファイル、または services-config.xml 設定ファイルを参照によってインクルードしているファイルで、設定します。Data Management Service 宛先の詳細については、1389 ページ、第 52 章の「Data Management Service の設定」を参照してください。

DataService コンポーネントの作成

DataService コンポーネントは、サーバーサイドの Data Management Service 宛先とのやり取りを管理します。DataService コンポーネントは MXML または ActionScript で作成できます。

次の例は、DataService コンポーネントを作成するための MXML コードです。DataService コンポーネントの宛先プロパティでは、サーバーサイドの有効な Data Management Service 宛先を参照する必要があります。

```
<?xml version="1.0"?>
<!-- fds\datamanagement\DataServiceMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:DataService id="ds" destination="contact"/>
</mx:Application>
```

次の例は、同じ DataService コンポーネントを作成するための ActionScript コードです。

```
<?xml version="1.0"?>
<!-- fds\datamanagement\DataServiceAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp();">
    <mx:Script>
        <![CDATA[
            import mx.data.DataService;
            public var ds:DataService;

            public function initApp():void {
                ds = new DataService("contact");
            }
        ]]>
    </mx:Script>
</mx:Application>
```

DataService コンポーネントを ActionScript で作成する場合は、mx.data.DataService クラスを読み込み、DataService 型の変数を宣言する必要があります。この変数の値を、新しい DataService オブジェクトに設定します。

ArrayCollection へのデータの設定と ArrayCollection からのデータの解放

DataService コンポーネントの fill() メソッドを呼び出すときは、Data Management Service 宛先からのデータを ArrayCollection オブジェクトに設定します。ArrayCollection オブジェクトは MXML または ActionScript で作成できます。ArrayCollection API では、データセットを操作するための一連のメソッドとプロパティが提供されています。詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

設定した ArrayCollection オブジェクトを解放するには、DataService コンポーネントの releaseCollection() メソッドを呼び出します。同じ ArrayCollection オブジェクトに対して同じパラメータで再度 fill() メソッドを呼び出した場合は、データの最新のコピーが取得されます。別のパラメータで再度 fill() メソッドを呼び出した場合には、最初の設定が解放された後、新しい設定が取得されます。

fill() メソッドの第 1 パラメータは、設定する ArrayCollection の id 値です。その他のパラメータの値は、呼び出すサーバーサイド宛先の種類によって異なります。

たとえば、Java アダプタとカスタムアセンブラを使用する宛先を呼び出す場合は、ArrayCollection の id 以降の引数として、宛先で宣言されている、対応するサーバーサイドメソッドの引数を指定できます。Hibernate オブジェクトリレーショナルマッピングシステムを使用する宛先の場合には、ArrayCollection の id 以降の引数は Hibernate 固有の値になります。これらの引数は、宛先で使用するように設定されている Hibernate の機能によって異なります。次の例は、HQL (Hibernate Query Language) を使用する Hibernate 宛先を呼び出すための fill() メソッドです。

```
myService.fill(myCollection, "flex:hql", ["from Person p where p.firstName =
  :firstName", parameterMap]);
```

詳細については、『1389 ページ、第 52 章の「Data Management Service の設定」』を参照してください。

次の例は、ArrayCollection オブジェクトの作成と設定を MXML で行うコードです。Application コンポーネントの creationComplete イベントリスナーを DataService コンポーネントの fill() メソッドに設定しているので、ArrayCollection である contacts はアプリケーションのロード時に設定されます。

```
<?xml version="1.0"?>
<!-- fds\datamanagement\FillArrayCollectionMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="ds.fill(contacts);">
    <mx:DataService id="ds" destination="contact"/>
    <mx:ArrayCollection id="contacts"/>
</mx:Application>
```

次の例は、ArrayCollection の作成と設定を ActionScript で行うコードです。DataService コンポーネントの fill() メソッドを initApp() メソッドの中で呼び出しています。このメソッドは Application コンポーネントの creationComplete イベントリスナーとして指定しています。

```
<?xml version="1.0"?>
<!-- fds\datamanagement\FillArrayCollectionAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp();">
    <mx:Script>
        <![CDATA[
            import mx.data.DataService;
            import mx.collections.ArrayCollection;

            public var ds:DataService;

            [Bindable]
            public var contacts:ArrayCollection;

            public function initApp():void {
                contacts = new ArrayCollection();
                ds = new DataService("contact");
                ds.fill(contacts);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

データプロバイダコントロールへの分散データの設定

データプロバイダコントロールに分散データを設定するには、データバインディングを使用し、管理された ArrayCollection オブジェクトを、データプロバイダコントロールの `dataProvider` プロパティにバインドします。関連付けられた DataService コンポーネントの `autoCommit` プロパティを `true` に設定すると、DataGrid でのデータの変更が自動的に Data Management Service 宛先に送信されます。

次の例は、DataGrid コントロールの `dataProvider` プロパティにバインドされた ArrayCollection オブジェクトです。

```
<?xml version="1.0"?>
<!-- fds\datamanagement\PopulateDataGrid.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="ds.fill(contacts);">
    <mx:DataService id="ds" destination="contact"/>
    <mx:ArrayCollection id="contacts"/>
    <mx:DataGrid id="dg" dataProvider="{contacts}" editable="true">
        <mx:columns>
            <mx:DataGridColumn dataField="contactId" headerText="Id"
                editable="false"/>
            <mx:DataGridColumn dataField="firstName" headerText="First Name"/>
            <mx:DataGridColumn dataField="lastName" headerText="Last Name"/>
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

管理された ArrayCollection オブジェクトからの変更の送信

デフォルトでは、DataService コンポーネントによって管理されている ArrayCollection オブジェクトのデータが変更されると、DataService コンポーネントの commit() メソッドが自動的に呼び出されます。または、commit() メソッドを手動で呼び出し、DataService コンポーネントの autoCommit プロパティを false に設定して、commit() メソッドへの手動呼び出しだけを許可することもできます。

次の例は、DataService コンポーネントによって管理されている ArrayCollection オブジェクト内のアイテムに対する手動更新処理です。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="initApp();">
  <mx:Script>
    <![CDATA[
      import mx.data.DataService;
      import mx.collections.ArrayCollection;
      import samples.customer.Customer;
      ...
      public function initApp():void {
               var customers:ArrayCollection = new ArrayCollection();
               var customerService:DataService = new
                  DataService("customers");
               customerService.autoCommit = false;
               var customer:Customer = customers.getItemAt(4);
               customer.name = "CyberTech Enterprises";
               customerService.commit();
          }
          ...
    ]]>
  </mx:Script>
</mx:Application>
```

DataService コンポーネントによって、customer.name プロパティへの変更が含まれている単一の更新メッセージが作成されます。DataService コンポーネントの commit() メソッドが呼び出されると、このメッセージが宛先に送信されます。

次の例は、DataService コンポーネントによって管理されている ArrayCollection オブジェクト内のアイテムに対する、2 つの手動更新処理です。

```
...
var customer:Customer = customers.getItemAt(4);
var oldName:String = customer.name;
customer.name = "CyberTech Enterprises";
customer.name = oldName;
customerService.commit();
...
```

DataService コンポーネントによって、互いの処理をキャンセルする 2 つの更新のログ記録が試みられます。customer.name プロパティの値が "CyberTech Enterprises" に変更されると、DataService コンポーネントによって単一の更新メッセージが作成されます。その後、customer.name = oldName を使用して古い名前に戻すと、顧客の名前に対応する元の更新メッセージが削除されます。その結果、commit() メソッドが呼び出されても何も送信されません。

次の例では、DataService コンポーネントによって管理されている ArrayCollection オブジェクトに対して顧客を追加または削除します。

```
<?xml version="1.0"?>
<!-- fds\datamanagement\AddRemoveItem.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="initApp();">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.data.DataService;
            import samples.customer.Customer;
            public function initApp():void {
                var customers:ArrayCollection = new ArrayCollection();
                var customerService:DataService =
                  new DataService("customers");
                customerService.autoCommit = false;
                customers.addItemAt(new Customer(), 4);
                // Remove the previously added customer.
                customers.removeItemAt(4);
                customerService.commit();
            }
        ]]>
    </mx:Script>
</mx:Application>
```

DataService コンポーネントによって、互いの処理をキャンセルする 2 つの更新のログ記録が試みられます。addItemAt() メソッドが呼び出されると、DataService コンポーネントによって customers 宛先に対する作成メッセージが作成されます。その後、removeItemAt(4) を呼び出すと、以前の作成メッセージが削除され、commit() メソッドが呼び出されても何も送信されません。

メモ

ArrayCollection オブジェクトの setItemAt(x) メソッドを呼び出すことは、removeItemAt(x) メソッドを呼び出してから addItemAt(..., x) メソッドを呼び出すことと同じです。

ここまでの例ではすべて、指定したアイテムの ID のみを DataService コンポーネントで使用し、新しい更新によって保留中の更新をキャンセルするかどうかを決めています。次の例もこの種の動作を示しています。

```
...
var customer:Customer = new Customer("1099");
customer.name = "CyberTech Enterprises";
// リスト内の位置 4 に、1099 という ID を持つ顧客オブジェクトが存在します。
customers.setItemAt(customer, 4);
```

```
customerService.commit();
...
```

この例では、新しい Customer オブジェクトが作成され、その name プロパティの値が変更されても、DataService コンポーネントからサーバーに変更が送信されません。

単一のデータアイテムの操作

mx.data.DataService クラスには、個々のデータアイテムを操作するためのメソッドがいくつかあります。これらのメソッドについて次の表で説明します。

メソッド	説明
createItem()	ArrayCollection を操作せずに新しいデータアイテムを作成できます。たとえば、コールセンターアプリケーションで顧客がフォームに入力して単一のチケットアイテムを作成する場合などに、このメソッドが役立ちます。一方、コールセンターの従業員はすべてのチケットアイテムを確認する必要があるため、従業員用のアプリケーションでは ArrayCollection にデータアイテムを設定します。
getItem()	アイテムの ID から単一のデータアイテムを取得できます。たとえば、注文と注文アイテムの関係などの、マスター詳細関係で、このメソッドが役立ちます。 DataGrid 内の注文アイテムの 1 つをクリックしたときに、別の宛先に戻り、そこからより多くのプロパティを含んだ詳細なアイテムを取得したい、という場合があります。こうすることで、注文のための最低限のデータを最初の ArrayCollection に設定し、詳細なアイテムは必要なときだけ個別に取得することが可能になります。
deleteItem()	createItem()、getItem()、または fill() メソッド呼び出しを使用して管理されているアイテムを削除します。削除は、トランザクションが確定された時点で直ちにサーバーに送信されます。
releaseItem()	アイテムを管理から解放します。このメソッドは、完了した getItem() または createItem() 呼び出しごとに呼び出してください。そうすることで、クライアントでのメモリリークを防ぎ、autoSyncEnabled を true に設定している場合はサーバーでのメモリリークも防ぎます。

次の例は、DataGrid コントロールが変更されたときに特定のデータアイテムを取得するメソッドです。アイテムは ResultEvent.result イベントから取得します。宛先に送られる ID 値は companyId で、現在 DataGrid コントロール内で選択されているアイテムの companyId に設定されます。宛先ではアイテムの ID に基づいてアイテムを取得します。ID は設定ファイルの宛先定義の中で指定します。

```
  <mx:Script>
    <![CDATA[
...
       private function companyChange() {
         dsCompany.getItem({companyId: dg.selectedItem.companyId});
```

```
        }
    ]]>
  </mx:Script>
...
```

接続と切断

DataService コンポーネントは、サーバーサイド宛先から切断された状態で開始されます。初めて処理を実行するときに、DataService コンポーネントによって宛先への接続が試みられます。処理が成功した場合は、result イベントが DataService コンポーネントに送られます。処理が失敗した場合は、fault イベントが送られます。`DataService.disconnect()` メソッドを呼び出すと、接続中のクライアントで切断を強制できます。この場合、DataService コンポーネントには自身の管理されたデータが保持され、再接続時には DataService コンポーネントによって自動的に再サブスクライブが行われ、変更が再開されます。

また、`DataService.release()` メソッドを呼び出すと、DataService コンポーネントによって取得された、管理されたオブジェクトを、すべて解放することができます。

DataStore オブジェクトについて

Data Management Service では、発着信するデータの変更が DataStore と呼ばれるオブジェクトに保持されています。このオブジェクトは複数の DataService コンポーネントで共有できます。デフォルトでは、2 つの DataService コンポーネントについて、両者の間に関連付けが存在する場合、または一方の DataService コンポーネントの `dataStore` プロパティが他方の DataService コンポーネントの DataStore オブジェクトを参照するように手動で設定している場合に、それら 2 つの DataService コンポーネントが同じ DataStore オブジェクトを共有します。

DataService コンポーネントの `commit()` メソッドを呼び出すと、DataStore オブジェクトを共有しているすべての DataService コンポーネントに対する変更も含めて、DataStore オブジェクトのすべての変更が確定します。同様に、ある DataService コンポーネントの `autoCommit` プロパティを変更すると、同じ DataStore オブジェクトを共有しているすべての DataService コンポーネントの値が変更されます。

エラー処理

DataService コンポーネントは、処理の実行中に発生したエラーに対して fault イベントを送出します。これには、サーバーへの接続時に発生したエラーの他、処理の実行に応答してアセンブラクラスから送られるエラーも含まれます。

詳細については、『Adobe Flex 2 リファレンスガイド』の mx.data.errors package を参照してください。

DataService コンポーネントは、最初の処理の実行時に自動的に ChannelSet に接続します。クライアントで commit() 要求の処理中にエラーが発生した場合は、commit 内の変更が、確定解除された変更に戻されます。この場合、DataService コンポーネントの revertChanges() メソッドをアイテム引数付きで呼び出してそれらの変更を復帰するか、または、引数を指定せずに revertChanges() メソッドを呼び出してすべての変更を復帰するかを選択できます。

プッシュされた変更をクライアントが受信するかどうかの管理

デフォルトでは、Data Management Service は Flex クライアントとサーバープッシュ API からのデータ変更を検出し、それらの変更を他のクライアントに伝播します。特定の宛先についてこのデフォルトの動作を変更するには、宛先設定で auto-sync-enabled を false に設定します。

クライアントサイドで特定の ArrayCollection について自動同期動作を有効にするには、DataService コンポーネントの autoSyncEnabled プロパティを true に設定してから、コンポーネントの fill() メソッドを呼び出します。同様に、管理されたアイテムを個々の参照から取得するには、DataService コンポーネントの autoSyncEnabled プロパティを true に設定してから、コンポーネントの getItem() または createItem() メソッドを呼び出します。

autoSyncEnabled プロパティの値を変更しても、そのクライアント上にある既存の管理されたオブジェクトには影響はありません。影響を受けるのは、値を変更した後の fill()、getItem()、および createItem() メソッド呼び出しのみです。これにより、同じクライアントにある同じ宛先からの管理されたインスタンスについて、autoSyncEnabled プロパティが true に設定されたインスタンスと、autoSyncEnabled プロパティが false に設定されたインスタンスを持つことができます。

プッシュされた変更の管理

デフォルトでは、クライアントで変更が検出されると、それらの変更が直ちに Data Management Service に適用されます。ある宛先についてこの動作を無効にするには、DataService コンポーネントの autoMerge プロパティを false に設定します。保留中の変更が着信すると、DataStore オブジェクトで mergeRequired プロパティが true に設定されます。保留中の変更を結合するには、DataService.dataStore.merge() メソッドを呼び出します。競合を避けるため、クライアントでデータをローカルに変更する場合は、その前に変更を結合してください。

メモ

DataService コンポーネントの autoMerge プロパティが false に設定されていて、ページングが有効であり、ページサイズが 0 以外の値の場合は、結合されていない更新が存在している間、クライアントではページ要求は実行されません。この条件が成り立っているかどうかを調べるには、DataService コンポーネントまたは関連付けられた DataStore オブジェクトの mergeRequired プロパティの値を調べます。

承認情報の提供

DataService コンポーネントの `setCredentials()` メソッドを使用すると、サーバーサイドのセキュリティ制限で必要になる承認情報を提供できます。これらの資格情報は `logout()` メソッドによって削除されます。承認情報の提供の詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。

クライアントサイドオブジェクトから Java オブジェクトへのマッピング

サーバーサイドの Java オブジェクトをクライアントアプリケーション内で表現するには、`[RemoteClass(alias=" ")]` メタデータタグを使用し、Java オブジェクトに直接マップされる ActionScript オブジェクトを作成します。`alias` の値として、Java クラスの完全修飾クラス名を指定します。これは、RemoteObject コンポーネントを使用する場合に Java オブジェクトへのマッピングに使用する方法と同じです。

サーバー上の Java オブジェクトにマップせず、サーバーから独自のオブジェクト型を送り返す場合は、エイリアスを指定しない `[RemoteClass]` メタデータタグを使用できます。独自の ActionScript オブジェクトは、サーバーに送られるときは Map オブジェクトに直列化されますが、サーバーからクライアントに返されるときは元の ActionScript 型になります。

クライアントサイドオブジェクトとサーバーサイドオブジェクト間で、管理された関連付けを作成する場合は、同時に `[Managed]` メタデータタグを使用するか、または mx.data.IManaged インターフェイスを明示的に実装します。

Flex データサービス サンプルアプリケーションに含まれる CRM アプリケーションは、管理された関連付けの例です。次の例は、クライアントサイドの ActionScript Company クラスのソースコードを示しています。これには、サーバーサイドの Java Company クラスとの管理された関連付けが含まれます。

```
package samples.crm
{
import mx.collections.ArrayCollection;
  [Managed]
  [RemoteClass(alias="samples.crm.Company")]
  public class Company
  {
    public var companyId:int;

    public var name:String = "";

    public var address:String = "";

    public var city:String = "";
```

```
    public var state:String = "";

    public var zip:String = "";

        public var industry:String = "";

    public function Company()
    {
    }

  }
}
```

次の例は、対応するサーバーサイドの Java Company クラスのソースコードです。

```
package samples.crm;

import java.util.Set;

public class Company
{
  private int companyId;

  private String name;

  private String address;

  private String city;

  private String zip;

  private String state;

   private String industry;

  public String getAddress()
  {
    return address;
  }

  public void setAddress(String address)
  {
    this.address = address;
  }

  public String getCity()
  {
    return city;
  }
```

```
public void setCity(String city)
{
  this.city = city;
}

public int getCompanyId()
{
  return companyId;
}

public void setCompanyId(int companyId)
{
  this.companyId = companyId;
}

public String getName()
{
  return name;
}

public void setName(String name)
{
  this.name = name;
}

public String getState()
{
  return state;
}

public void setState(String state)
{
  this.state = state;
}

public String getZip()
{
  return zip;
}

public void setZip(String zip)
{
  this.zip = zip;
}

 public String getIndustry()
 {
     return this.industry;
 }
```

```
    public void setIndustry(String industry)
    {
        this.industry = industry;
    }

    public String toString()
    {
       return "Company(companyId=" + companyId + ", name=" + name + ", address="
  + address +
                      ", state" + state + ", zip=" + zip + " industry=" +
  industry + ")";
    }
}
```

データ同期の競合の処理

分散データアプリケーションを操作する場合、しばしば、既に完了済みの変更と競合するデータ変更がクライアントで試みられることがあります。データ同期の競合が発生する主な事例を次のシナリオに示します。

競合の種類	内容
古いデータによる更新の場合。	クライアントがデータを受信して以降、既にデータが変更されているために、クライアントが変更を確定するときにサーバーが競合を検出します。
競合していた変更についてクライアントが既に確定を解除したときに、クライアントに変更がプッシュされた場合。	クライアントはデータ変更の最中であり、その変更は、サーバーからクライアントにプッシュされた変更と競合します。
古いデータによる削除の場合。	クライアントがアイテムを削除しようとしたときに、そのオブジェクトの古いものを使用して削除しようとした場合に、サーバーが競合を検出します。
既にローカルに削除されたアイテムについて、クライアントに変更がプッシュされた場合。	データに対する更新が発生した直後の時点で、同じデータを削除する要求についてクライアントが既に確定を解除しており、データメッセージがサーバーからクライアントに既にプッシュされています。
クライアントのプッシュがない状態で、削除されたデータを更新する場合。	サーバーから既に削除されたデータをクライアントが更新しようとしています。
クライアントのプッシュがある状態で、削除されたデータを更新する場合。	既に削除されたデータをクライアントが更新しようとしており、データメッセージがサーバーからクライアントに既にプッシュされています。クライアントは更新を送信しようとしますが、クライアント上で競合が存在します。

次の例は、競合が発生したときにクライアントでアラートボックスを表示し、サーバー側のデータを受け入れる、簡単なイベントハンドラです。

```
...
public function useDS:void {
...

   ds.addEventListener(DataConflictEvent.CONFLICT, conflictHandler);
...
}

public function conflictHandler(event:DataConflictEvent):void {
   var conflicts:Conflicts = ds.conflicts;
   var c:Conflict;
   for (var i:int=0; i<conflicts.length; i++) {
      c = Conflict(conflicts.getItemAt(i));
      Alert.show("Reverting to server value", "Conflict");
      c.acceptServer();
   }
}
```

mx.data.Conflicts と mx.data.Conflict、および mx.data.events.DataConflictEvent クラスの詳細については、『Adobe Flex 2 リファレンスガイド』を参照してください。

宛先で Java アダプタを使用する場合は、データの同期を処理し、以前のデータが現在データソースにあるデータと一致しない場合に DataSyncException をスローする、sync メソッドを、アセンブラクラスに記述します。DataSyncException の結果、DataConflictEvent がクライアントで発生します。sync メソッドの詳細については、1405 ページの「fill-method および sync-method によるアプローチの使用」を参照してください。

次のコードに示すように、競合が発生した場合にどのデータを使用するのかをユーザーに決定してもらうこともできます。PersonForm は、その dataSource プロパティに割り当てられた Person オブジェクトのすべてのプロパティを表示するカスタムコンポーネントを表しています。

```
// 競合イベントハンドラ
function resolveConflictsHandler():void {
displayConflictsScreen();
}
...
<!-- 競合画面の MXML コード -->
<PersonForm id="serverValue" editable="false"
   dataSource="{ds.conflicts.current.serverObject}"/>

<PersonForm id="clientValue" dataSource="{ds.conflicts.current.clientObject}"/>

<PersonForm id="originalValue"
   dataSource="{ds.conflicts.current.originalObject}" editable="false"/>

<mx:Button label="Accept Server" click="ds.conflicts.current.acceptServer" />
<mx:Button label="Accept Client" click="ds.conflicts.current.acceptClient" />
...
```

第 52 章

Data Management Service の設定 52

Adobe Flex クライアントアプリケーションで分散データを可能にするには、<mx:DataService> タグで、または対応する ActionScript API で、サーバーサイドの Data Management Service 宛先への接続を宣言することによって、Flex Data Management Service に接続します。宛先は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしているファイルで、Data Management Service 定義の一部として設定します。このトピックでは、宛先および宛先で使用されるデータアダプタを設定する方法について説明します。データをサーバーからクライアントにプッシュする方法についても説明します。

MXML または ActionScript でのクライアントサイドの DataService コンポーネントの使用と Data Management Service 宛先への接続に関する詳細については、1375 ページ、第 51 章の「アプリケーションでのデータの分散」を参照してください。

目次

Data Management Service の設定について

Data Management Service を設定する際に実行する通常の作業としては、Data Management Service 宛先の定義、宛先へのセキュリティの適用、およびログ設定の変更があります。"Data Management Service 宛先 " は、Data Management Service を使用してアプリケーションでデータの分散と同期を実現する際にデータの受け渡し相手となるエンドポイントです。Data Management Service 宛先は、"services-config.xml" ファイル、または "services-config.xml" ファイルを参照によってインクルードしているファイルの、データサービスセクションで設定します。デフォルトでは、"services-config.xml" ファイルは Adobe Flex データサービスのある Web アプリケーションの WEB_INF/flex ディレクトリにあります。Flex データサービスに付属の "services-config.xml" ファイルでは、Adobe によって、Data Management Service、Message Service、Remoting Service、Proxy Service の各設定ファイルが参照されています。

次の例は、基本的な Data Management Service 設定です。Java アダプタを使用してデータリソースとやり取りする宛先が 1 つあります。ActionScript アダプタはデフォルトアダプタとして設定されています。

```
<service id="data-service" class="flex.data.DataService"
  messageTypes="flex.data.messages.DataMessage">

  <adapters>
    <adapter-definition id="actionscript"
      class="flex.data.adapters.ASObjectAdapter" default="true"/>
    <adapter-definition id="java-adapter"
      class="flex.data.adapters.JavaAdapter"/>
  </adapters>

  <default-channels>
    <channel ref="my-rtmp"/>
  </default-channels>

  <destination id="contact">
    <adapter ref="java-adapter" />
      <properties>
        <source>dev.contacts.ContactAssembler</source>
        <scope>application</scope>
        <cache-items>true</cache-items>
        <metadata>
          <identity property="contactId"/>
        </metadata>

        <network>
          <session-timeout>20</session-timeout>
          <paging enabled="true" pageSize="10" />
          <throttle-inbound policy="ERROR" max-frequency="500"/>
          <throttle-outbound policy="REPLACE" max-frequency="500"/>
        </network>
```

```
            <server>
              <fill-method>
                <name>loadContacts</name>
              </fill-method>
              <fill-method>
                <name>loadContacts</name>
                <params>java.lang.String</params>
              </fill-method>
              <sync-method>
                <name>syncContacts</name>
              </sync-method>

            </server>
          </properties>
      </destination>
</service>
```

Data Management Service 宛先

Data Management Service 宛先では、メッセージを伝達するメッセージチャネルを 1 つまたは複数参照し、ネットワーク関連プロパティとサーバー関連プロパティを格納します。また、" データアダプタ " も参照できます。データアダプタは、宛先で Java オブジェクトなどの特定種類のインターフェイスを通じてデータを操作できるようにするサーバーサイドコードです。特定のデータアダプタをデフォルトデータアダプタとして設定することもできます。この場合は宛先の中でデータアダプタを参照する必要はありません。宛先に対するセキュリティ制限を宛先で参照または定義することもできます。

Java アダプタを使用する場合は、他のアダプタの場合の作業とは異なり、アセンブラクラスを記述します。" アセンブラクラス " は、データリソースと間接的または直接的にやり取りする Java クラスです。通常のデザインパターンでは、データリソースを呼び出す DAO (データアクセスオブジェクト) をアセンブラで呼び出します。

メッセージチャネル

Data Management Service 宛先ではメッセージチャネルを 1 つまたは複数参照します。メッセージチャネルは設定ファイル内の任意の場所で定義します。パブリッシュ / サブスクライブメッセージングで最もよく使用される 2 つのチャネルとして、RTMP (Realtime Message Protocol) チャネルと、メッセージのポーリングが有効にされた AMF (Action Message Format) チャネルがあります。

RTMP チャネルではクライアントとサーバー間の接続が管理されるため、クライアントでサーバーをポーリングする必要がありません。ポーリングが有効にされた AMF チャネルでは、サーバーがポーリングされ、新しいメッセージがないかが調べられます。メッセージチャネルの詳細については、1240 ページの「メッセージチャネルの設定」を参照してください。

Data Management Service データアダプタ

データアダプタはデータリソースとやり取りするためのインフラストラクチャを提供します。データアダプタは宛先で指定します。アダプタ固有の設定を指定することもできます。

データアダプタの詳細については、1392 ページの「Data Management Service 宛先の設定」を参照してください。

セキュリティ

宛先に対するアクセス権限を定義する " セキュリティ制限 " を使用すると、宛先をセキュリティ保護できます。セキュリティ制限は、ユーザーが宛先にアクセスできるようにする前にユーザーの認証および承認を行う場合に使用します。認証では、基本認証を使用するかカスタム認証を使用するかを指定できます。承認では、承認に必要なロールを指定できます。

セキュリティ制限は宛先でインラインで宣言できます。または、セキュリティ制限をグローバルに宣言し、宛先でその `id` 値によって参照できます。

セキュリティの詳細については、1259 ページの「宛先のセキュリティ保護」を参照してください。

Data Management Service 宛先の設定

Data Management Service 宛先の設定の要素は、すべての Data Management Service 宛先に対する全般的なものや、宛先で使用される特定のデータアダプタによって決まるものがあります。このセクションでは、Data Management Service 宛先の全般的な設定について説明します。

メッセージチャネルの参照

メッセージは、メッセージチャネルを経由して宛先に、または宛先から、伝達されます。宛先では 1 つまたは複数のメッセージチャネルを参照します。メッセージチャネルは "services-config.xml" ファイルで定義します。宛先では、参照されたチャネルのうち、使用可能な最初のチャネルが使用されます。チャネルに到達できない場合は、次に使用可能なチャネルが宛先で使用されます。デフォルトのメッセージチャネルを設定することもできます。この場合は、宛先でチャネルを参照する必要はありません。

次の例は、宛先のチャネル参照を示しています。`samples-rtmp` チャネルが最初に列挙されているので、宛先ではそのチャネルの使用を最初に試みます。

```
...
  <destination id="contact">
...
     <channels>
       <channel ref="samples-rtmp"/>
       <channel ref="samples-amf-polling"/>
```

```
    </channels>
...
  </destination>
...
```

メッセージチャネルの詳細については、1240 ページの「メッセージチャネルの設定」を参照してください。

ネットワークエレメントの設定

宛先定義の properties セクションには、クライアントとサーバー間のメッセージング動作を定義するための network セクションがあります。次の例は、宛先定義の network セクションを示しています。

```
...
  <destination id="contact">
    <properties>
...
          <network>
            <session-timeout>20</session-timeout>
            <paging enabled="true" pageSize="10"/>
            <throttle-inbound policy="ERROR" max-frequency="500"/>
            <throttle-outbound policy="REPLACE" max-frequency="500"/>
            <!--<cluster ref="default-cluster"/>-->
          </network>
  ...
    </properties>
  </destination>
...
```

Data Management Service 宛先定義では次のネットワーク関連エレメントを使用します。

エレメント	説明
cluster	cluster エレメントの ref 属性は、ソフトウェアクラスタの名前を参照します。この名前は "services-config.xml" ファイルの clusters セクションで設定します。クラスタの詳細については、1268 ページの「ソフトウェアクラスタ化の使用」を参照してください。
paging	paging エレメントの enabled 属性は、データページングを宛先で有効にするかどうかを指定します。ページングを有効にした場合、pageSize は、クライアントサイドの DataService.fill() メソッドが呼び出されたときにクライアントに送信されるレコード数を示します。 このエレメントはオプションです。デフォルト値は false です。
session-timeout	(オプション) session-timeout エレメントは、サブスクライバがサブスクライブ解除されるまでのアイドル時間を分数で指定します。値を 0 に設定した場合、サブスクライバでは自動的にサブスクライブ解除が強制されることがなくなります。デフォルト値は 20 です。

エレメント	説明
throttle-inbound	throttle-inbound エレメントの max-frequency 属性は、サーバーが受信できる毎秒のメッセージ数を制御します。policy 属性は、メッセージの上限に達したときに実行する動作を指定します。policy 属性には、次の値を設定できます。 • ERROR は、上限に達したときにエラーを返すことを指定します。 • IGNORE は、上限に達したときにエラーを返さないことを指定します。
throttle-outbound	throttle-outbound エレメントの max-frequency 属性は、サーバーが送信できる毎秒のメッセージ数を制御します。policy 属性は、メッセージの上限に達したときに実行する動作を指定します。policy 属性には、次の値を設定できます。 • ERROR は、上限に達したときにエラーを返すことを指定します。 • IGNORE は、上限に達したときにエラーを返さないことを指定します。 • REPLACE は、上限に達したときに前のメッセージを置き換えることを指定します。

トランザクションの使用

Flex のデフォルトでは、クライアントで確定した処理は単一の J2EE 分散トランザクションにカプセル化されます。クライアントのデータ変更はすべて 1 単位として処理されます。ある値を変更できない場合、他のすべての値の変更はロールバックされます。分散トランザクションは、J2EE (Java 2 Enterprise Edition) サーバーで JTA (J2EE Transaction) がサポートされている限り、サポートされます。

次の例では、ボールド体のテキストの部分で use-transactions プロパティをデフォルト値の true に設定しています。

```
...
  <destination id="contact">
       <properties>
         <use-transactions>true</use-transactions>
    </properties>
  </destination>
...
```

データアイテムの一意の識別

Data Management Service 宛先には 1 つまたは複数の identity エレメントがあります。このエレメントを使用することで、データのコレクション内のアイテムを一意に識別することを保証するためにデータプロパティを使用することを指定できます。

次の例では、identity プロパティをボールド体のテキストで示しています。

```
...
  <destination id="contact">
    <properties>
...
    <metadata>
      <identity property="name"/>
    </metadata>
  </properties>
  </destination>
...
```

identity エレメントはオプションで type 属性を受け取ります。この属性は、指定した identity プロパティの Java クラスです。この属性は、identity の型が ActionScript の型と 1 対 1 に対応していない可能性がある場合に使用する必要があります。よくある問題として、ActionScript の Number がその値によって long または integer のいずれかに変換されることがあります。次の例のコードは、type 属性を指定した identity エレメントです。

```
<identity property="id" type="java.lang.Long">
```

よくある間違いとして、次の例のように、すべてのプロパティに対して 1 つの identity エレメントを使用しようとする事例があります。

```
<metadata>
  <!– Don't do this. -->
  <identity property="firstName,lastname"/>
</metadata>
```

そうではなく、次の例のように、複数のフィールド identity に対して複数の identity エレメントを指定できます。

```
<metadata>
  <identity property="firstName"/>
  <identity property="lastName"/>
</metadata>
```

データアイテムのキャッシュ

Data Management Service のデフォルトでは、fill() および getItem() 呼び出しから返されたアイテムがキャッシュされ、キャッシュされたアイテムを使用して、ページングの実装や、遅延関連付けの実装時におけるオブジェクトグラフの構築が行われます。これにより、すべてのアクティブなクライアントについて、それらの管理された状態の完全なコピーが各サーバーのメモリに維持されます。このメモリキャッシュを無効にするには、次の例のように、cache-items を false に設定します。

```
...
  <destination id="contact">
    <properties>
...
      <cache-items>false</cache-items>
...
  </properties>
  </destination>
...
```

cache-items を false に設定した場合は、この宛先を使用するときに、ページングや遅延関連付けをサポートするためのメソッドを get-method エレメントで指定する必要があります。

データの自動同期

auto-sync-enabled エレメントは、宛先を使用しているクライアントのクライアントサイドの DataService.autoSyncEnabled プロパティについて、そのデフォルト値を制御します。このエレメントのデフォルト値は true です。

次の例は、auto-sync-enabled エレメントを示しています。

```
...
  <destination id="contact">
    <properties>
...
      <auto-sync-enabled>false</auto-sync-enabled>
...
  </properties>
  </destination>
...
```

データアダプタの操作

Data Management Service は、ArrayCollection オブジェクト、単一の管理されたオブジェクト、および、エンタープライズ中間層またはリソース層の永続データについて、Flex クライアントアプリケーションによって実行される処理に基づいて、データメッセージを作成および更新します。データリソースへのアクセス処理は、Data Management Service とデータアダプタとの取り決めを通じて発生します。" データアダプタ " は、特定のデータストアタイプに適した方法で永続データストアを更新する役割を果たします。Java アダプタの場合は、この更新の役割をアセンブラクラスに委ねる役割を果たします。

Java アダプタと ActionScript オブジェクトアダプタは Flex データサービスに組み込まれています。

Java アダプタは、Java アセンブラと呼ばれる、Java クラスで使用できるメソッドにデータ変更を渡します。このアダプタにより、Java 開発者は、DTO (データ転送オブジェクト) デザインパターンを使用できます。Hibernate アセンブラは、Data Management Service から Hibernate オブジェクトリレーショナルマッピングシステムへのブリッジを提供する特別なアセンブラです。詳細については、1418 ページの「Hibernate 宛先の使用」を参照してください。

ActionScript オブジェクトアダプタは、Flex クライアントアプリケーションが一時データオブジェクトのただ 1 つの作成者および使用者であり、バックエンドのデータリソースが存在しない場合に使用します。Data Management Service は、ActionScript オブジェクトアダプタを使用してメモリ内のオブジェクトを管理します。

以降のセクションでは、これらのアダプタとその使用方法について説明します。

Java アダプタの使用

Java アダプタを使用すると、サーバー上の Java オブジェクトとクライアント上の ActionScript オブジェクトを同期させることができます。このアダプタは、アセンブラと呼ばれる、Java クラスで使用できるメソッドにデータ変更を渡します。Java アダプタはカスタムアセンブラクラスや Hibernate アセンブラに使用します。Hibernate アセンブラを使用すると、Hibernate オブジェクトリレーショナルマッピングシステムを操作できます。

Hibernate を使用しないアプリケーションでは、まず、アプリケーションオブジェクトのサポートに必要な宛先について決定し、それから、各宛先のアセンブラクラスを定義する必要があります。アセンブラクラスでは 1 つまたは複数の fill メソッドをサポートする必要があります。これらのメソッドではクエリを実装します。クエリではそれぞれ、オブジェクトの Collection をクライアントコードに返します。クライアントでオブジェクトを更新する必要がある場合は、sync メソッドが必要になるか、または create、update、delete の各メソッドが個別に必要になります。

使用するサーバーアーキテクチャによっては、ドメインモデルオブジェクトをクライアントに直接転送することを選択するか、あるいは、クライアント層に公開するモデル状態だけを複製する別の DAO (データ転送オブジェクト) 層を実装することができます。

アセンブラについて

Data Management Service では、Java アセンブラクラスを記述する 2 つのアプローチがサポートされています。1 つは簡単なアプリケーションを対象としたアプローチで、Data Management Service の機能のうち必要なものだけをサポートできます。このアプローチでは、宛先定義で `fill-method` および `sync-method` という XML エレメントを使用して個々のメソッドを宣言し、アセンブラクラスのメソッドにマップします。クライアントで必要になる機能セットについて、それらに必要なメソッドを追加するだけで済みます。たとえば、クライアントアプリケーションが読み取り専用の場合は、クライアントでの必要に応じて get メソッドや fill メソッドを提供するだけで済む可能性があります。get メソッドは、クライアントで `DataService.getItem()` メソッドを使用する場合に、アセンブラで必要になります。また、`cache-items` を `false` に設定し、そのアイテムを指し示す遅延関連付けを設定する場合や、ページングを使用する場合にも、get メソッドが必要になります。

その後、アセンブラで更新をサポートする必要が生じた場合には、`sync-method` 定義を追加できます。このアプローチでは特別な Flex インターフェイスをクラスで実装する必要がありません。そのため、場合によっては、簡単な事例においてこのアプローチを使用することで、Java コードを記述しなくても機能を Flex アプリケーションに公開できる可能性があります。

もう 1 つのアプローチは Assembler インターフェイスによるアプローチと呼ばれ、従来の手法に近い方法で Java コンポーネントとのインターフェイスを提供します。これには、flex.data.assemblers.Assembler インターフェイスを実装する flex.data.assemblers.AbstractAssembler を拡張し、`getItem()`、`fill()`、`createItem()`、`updateItem()` などのメソッドをオーバーライドします。AbstractAssembler クラスは Assembler インターフェイスのすべてのメソッドのデフォルト実装を提供します。そのため、このアプローチでは最初のアプローチと比べていくつかの利点があります。AbstractAssembler クラスと Assembler インターフェイスの詳細については、公開されている『Flex データサービス Javadoc マニュアル』を参照してください。

両方のアプローチを同じアセンブラクラスで組み合わせることもできます。メソッドの XML 定義がある場合、Data Management Service は、Assembler インターフェイス内の等価なメソッドではなく XML 定義のほうのメソッドを呼び出します。宛先定義で XML 宣言を使用して fill および count メソッドを追加することにより、クエリをクライアントに公開できるので便利です。また、アセンブラの `fill()` メソッドでさまざまなタイプの fill メソッドについて送出を実行せずに済みます。一致する `fill-method` エレメントが宛先内で見つかった場合は、そのメソッドが呼び出されます。見つからなかった場合には、Data Management Service は、Assembler インターフェイスで定義されアセンブラで実装されている `fill()` メソッドを呼び出します。

オブジェクト型と関係

Java アダプタでは、プリミティブなフィールドおよびプロパティしか含まれていないフラットなオブジェクトだけでなく、複雑なオブジェクトをプロパティとして持つオブジェクトもサポートされます。プロパティとしての複雑なオブジェクトがあり、それらのオブジェクトが独自の identity プロパティを 1 つまたは複数持つ場合は、別の宛先を定義し、宛先定義の `identity` エレメントで指定した 1 つまたは複数の identity プロパティに基づいて、それらのインターフェイスの同期を管理するようにできます。複雑なオブジェクトのプロパティに対する identity プロパティがなく、各インスタンスがその親オブジェクトによって所有される場合は、親の宛先を使用してオブジェクトのプロパティの状態を管理できます。

オブジェクトの同期を管理するにはさまざまな方法がありますが、その一例として、ネストされた Address オブジェクトを持つ Customer オブジェクトがあります。Address が他の Customer インスタンスで使用されていない場合、Address インスタンスの変更に応じてそれらの状態を永続化する作業を Customer 宛先に任せることができます。Address が独自の ID を持っていて、複数の顧客間で共有される可能性がある場合は、別の Address 宛先を作成して Address インスタンスを永続化したほうが意味があると考えられます。この場合、Customer 宛先には、Customer が参照している住所の ID を 1 つまたは複数永続化する作業だけを任せることになります。プロパティを使用し、別の宛先によって管理されているオブジェクトに対する関係を表現する場合、そのプロパティのことを " 管理された関連付けプロパティ " と呼びます。このトピックの詳細については、1427 ページの「階層コレクションの管理」を参照してください。

場合によっては、オブジェクト間の関係を処理する方法として、管理された関連付けによるアプローチが最善ではないことがあります。代わりの方法として、クエリを使用して 2 つのオブジェクト間の関係を実装することができます。この方法では、アセンブラで定義された `fill()` メソッドのいずれかのパラメータを、関連するオブジェクトの ID にします。大量のオブジェクトコレクションを扱う場合には、Assembler インターフェイスによるこのアプローチのほうが、管理された関連付けによるアプローチよりも効率的です。たとえば、多数の Employee インスタンスを持つ Company オブジェクトがあるとします。これを、管理された関連付けプロパティ (company.employees など) を使用して実装した場合、従業員のリスト全体が、fill 要求に対するデータが返されるときはサーバーからクライアントに送信され、更新が実行されるときはクライアントからサーバーに送信されます。詳細については、1438 ページの「fill メソッドでのオブジェクト関係の実装」を参照してください。

管理された関連付けによるアプローチを使用した場合は、遅延ロード機能を使用したとしても、参照されたすべての従業員の ID が送信されることになります。遅延ロードの詳細については、1430 ページの「管理された関連付けの実装」を参照してください。

10,000 人の従業員を擁するような会社では、計算処理が膨大になる可能性があります。代わりに、会社の id をパラメータとして受け取る fill メソッドを従業員宛先で用意し、その会社に存在する従業員を返すようにすることができます。このアプローチでは、Data Management Service のページング機能を使用して、従業員の最初のページを送信できます。新しい従業員を会社に追加するときや、従業員を会社から削除するときは、その従業員のデータを送信するだけで済み、関係のない従業員がクライアントからサーバーに送信されることがなくなります。

メモ

Flex データサービス Java API を使用するコードをコンパイルするときは、"messaging.jar" ファイルと "flex-messaging-common.jar" ファイルをクラスパスにインクルードする必要があります。
コンパイルされたアセンブラクラスや、DAO クラスなどのその他の必要なクラスを、Web アプリケーションのクラスパスで使用できるようにする必要があります。

Java アダプタでの厳密な型および匿名型の使用

厳密に型指定されたオブジェクトのプロパティは、宣言されたフィールドと、get および set メソッドを使用することによって、コンパイル時に定義されます。匿名型は、通常は Java では java.util.Map によって表現され、ActionScript では Object 型のインスタンスによって表現されます。匿名オブジェクトの場合、型チェックはコンパイル時にではなく実行時に行われます。

Data Management Service では、クライアントとサーバーのいずれでも、厳密に型指定されたオブジェクトまたは匿名オブジェクトを使用できます。たとえば、匿名オブジェクトをクライアントとサーバーの両方で使用するには、クライアントで Object インスタンスに変換される java.util.Map インスタンスを、アセンブラから返します。クライアントでこれらの Object インスタンスを変更すると、それらのインスタンスが java.util.Map インスタンスとしてアセンブラに送り返されます。

厳密に型指定されたオブジェクトをクライアントとサーバーで使用するには、管理する必要のあるデータの明示的なパブリックプロパティを持つ ActionScript クラスを、`[Managed]` メタデータタグで定義します。`[RemoteClass(alias="`<Java クラス名 >`")]` メタデータタグを使用し、このクラスをサーバー上の Java クラスにマップします。この場合、いずれかのクラスを明示的に使用して Data Management Service 宛先を設定する必要がなくなり、クライアントから宛先に送信されるどのようなインスタンスも 1 つの宛先でサポートできるようになります。クライアントでは Java アセンブラから返されるインスタンスを想定するだけで済み、Java アセンブラではクライアントから返されるインスタンスを認識する必要があります。このため、1 つの宛先でインスタンスのクラス階層全体をサポートできるようになり、クラスごとに Data Management Service 宛先を明示的に設定する必要がなくなります。

Data Management Service ではまた、厳密に型指定された Java インスタンスを Java アセンブラから返し、返されたインスタンスをクライアントで匿名型に変換できるようにする手法もサポートされています。この場合、Java クラスごとに ActionScript クラスを維持する必要はありません。匿名インスタンスが変更されると、サーバーではそれらを java.util.Map インスタンスとして受信します。Data Management Service では、宛先定義の `properties` セクションで `item-class` エレメントがサポートされています。このエレメントによって、それらのインスタンスが、Java アセンブラで想定している、厳密に型指定された元の単一の Java クラスに自動的に変換されます。item-class エレメントは、アセンブラで想定している Java クラスを参照するように設定します。このパターンが利用できるのは、各宛先で単一の Java クラスのインスタンスしか返さない一般的な場合のみです。アセンブラでクラス階層を操作する必要があり、厳密に型指定されたインスタンスをクライアントで使用したくない場合は、アセンブラの中で、java.util.Map から厳密に型指定されたインスタンスへの変換を自分で行う必要があります。

サーバーで匿名オブジェクトを使用する Java アセンブラを記述し、それらの匿名オブジェクトを、クライアントで厳密に型指定された ActionScript クラスに直列化することもできます。これには、java.util.Map インターフェイスを実装する flex.messaging.io.amf.ASObject クラスのインスタンスを作成します。このインスタンスの type プロパティには、そのインスタンスをクライアントで直列化する必要のある ActionScript クラスの完全修飾クラス名を設定します。そして、ActionScript インスタンスで想定しているプロパティ値を、キーと値のペアとして java.util.Map に格納します。このオブジェクトがクライアントから返されると、直列化コードによって正しい型のインスタンスが作成されます。

Java アダプタを使用する宛先の設定

既に説明したように、Java アダプタを操作する際は、アセンブラの記述方法を次の中から選択できます。

- 宛先で `fill-method` および `sync-method` XML エレメントを使用し、任意のクラスのメソッドにマップします。
- flex.data.assemblers.Assembler インターフェイスを実装するアセンブラクラスを記述し、`fill()` メソッドの実装を提供します。
- 上記の 2 つのアプローチを組み合わせて使用します。2 つのアプローチを組み合わせる場合、クライアントからの fill 要求のシグネチャが `fill-method` の宣言と一致しなければ、アセンブラで定義された `fill()` メソッドが Data Management Service によって呼び出されます。

この選択によって、どの XML エレメントを宛先に含め、どのような Java コードを記述するかが決まります。詳細については、1398 ページの「アセンブラについて」を参照してください。

次の例の Data Management Service 宛先では、Java アセンブラを使用しており、`fill-method` エレメントと `sync-method` エレメントがあります。

```
<service id="data-service" class="flex.data.DataService"
  messageTypes="flex.data.messages.DataMessage">
...
```

```
  <adapters>
    <adapter-definition id="java-adapter"
  class="flex.data.adapters.JavaAdapter"/>
  </adapters>
...
  <destination id="contact">
    <adapter ref="java-adapter"/>
    <properties>
      <metadata>
        <identity property="contactId"/>
      </metadata>

      <source>samples.contact.ContactAssembler</source>
      <scope>application</scope>

      <network>
        <paging enabled="true" pageSize="10"/>
        <throttle-inbound policy="ERROR" max-frequency="500"/>
        <throttle-outbound policy="REPLACE" max-frequency="500"/>
      </network>

      <server>

        <fill-method>
          <name>loadContacts</name>
        </fill-method>

        <fill-method>
          <name>loadContacts</name>
          <params>java.lang.String</params>
        </fill-method>

        <sync-method>
          <name>syncContacts</name>
        </sync-method>

        <get-method>
          <name>getPerson</name>
        </get-method>

        <count-method>
          <name>getCount</name>
        </count-method>
      </server>
    </properties>
  </destination>
</service>
```

source エレメントでは、データ変更を処理するクラスであるアセンブラクラスを指定します。scope エレメントではアセンブラのスコープを指定します。有効な値は、application、session、request です。

fill-method および sync-method エレメントを使用し、アセンブラの任意のメソッドを、クライアントサイドの ArrayCollection オブジェクトでデータの設定と同期に使用する fill および sync メソッドとしてマップします。fill-method エレメントでは、クライアントサイドの ArrayCollection オブジェクトにデータを設定するために使用するアセンブラメソッドを指定します。sync-method エレメントでは、クライアント上のデータとサーバーサイドのデータリソースを同期させるために使用するメソッドを指定します。または、get-method エレメントを使用して単一のデータアイテムを取得するためのメソッドを指定することや、count-method エレメントを使用してデータリソースにあるすべてのデータアイテムの数を取得するメソッドを指定することもできます。

fill メソッドの結果に対する変更の検出

fill メソッドは、特定の宛先についてアイテムのコレクションを返すクエリを実装するために使用します。クライアントで fill メソッドを実行する際、DataService.autoSyncEnabled プロパティを true に設定した場合は、クライアントに表示されるリストを更新できるように、このクエリの結果に対する変更をクライアントでリッスンします。Data Management Service のデフォルトでは、自動更新と呼ばれる総合的な方法によってこの処理が実装されています。通常はこの自動更新によって、クライアントの結果とデータベースとの同期が保たれます。この方法を使用した場合は、クライアントまたはサーバープッシュ API によってアイテムが作成または更新されるたびに、アクティブクライアントで現在管理されているすべての fill() メソッドが Data Management Service によって再実行されます。返されるアイテムの ID と、キャッシュされている結果とが比較され、追加または削除されたものがある場合は、コレクション更新メッセージが作成され、その fill メソッドが既にキャッシュされているクライアントに送信されます。単純なアプリケーションではこれだけで必要な処理が済む場合があります。しかし、クライアントによって管理された一意のパラメータを持つ fill メソッドの数が多くなると、自動更新の結果、不必要なクエリが多数実行される可能性があります。

不必要な fill メソッドが実行されないようにするための方法として、2 つの方法があります。1 つは自動更新を無効にする方法です。flex.data.DataServiceTransaction クラス (サーバープッシュ API) の refreshFill() メソッドを使用し、更新する fill メソッドのパラメータに一致するパターンを指定することによって、1 つまたは複数の fill メソッドを明示的に無効化できます。もう 1 つは自動更新を有効のままにしておく方法です。指定したアイテムによって結果が変更される fill メソッドだけが再実行されるように、自動更新を設定できます。

後者のアプローチでは、アクティブクライアントで現在管理されている一意の fill パラメータセットについて、それぞれ作成または更新されるアイテムごとに呼び出されるメソッドを設定します。このメソッドからは、変更に対して fill をどのように扱うのかを指定する値を返します。

このメソッドの動作内容は、アセンブラの宛先で XML ベースの fill-method および sync-method エレメントを使用するか、またはアセンブラで Assembler インターフェイスを実装するかによって異なります。fill-method と sync-method によるアプローチの詳細については、1405 ページの「fill-method および sync-method によるアプローチの使用」を参照してください。Assembler インターフェイスによるアプローチの詳細については、1409 ページの「Assembler インターフェイスによるアプローチの使用」を参照してください。

fill-method エレメントを使用するときは、Assembler クラス内のメソッドを指し示す fill-contains-method エレメントを定義します。Assembler クラス内のメソッドは、変更または作成されたアイテムと、fill パラメータの List を受け取り、アイテムがその fill に存在するかどうかに応じて true または false を返します。fill が並べられておらず、メソッドが true を返した場合は、その fill に対して、管理されたアイテムのリストの末尾にアイテムが追加されます。fill が並べられていて、メソッドが true を返した場合は、fill メソッドが再実行され、リストの順序が正しく設定されます。メソッドが false を返した場合には、その更新に対して fill メソッドはそのままになります。

次の例は、並べられていない fill と並べられた fill に対する、fill-method エレメントと fill-contains-method エレメントです。

```
<fill-method>
  <name>loadUnorderedPatternGroups</name>
  <params>java.lang.String,java.lang.Boolean</params>
  <ordered>false</ordered>
  <fill-contains-method>
    <name>unorderedPatternContains</name>
  </fill-contains-method>
</fill-method>
<fill-method>
  <name>loadOrderedPatternGroups</name>
  <params>java.lang.Boolean,java.lang.String</params>
  <ordered>true</ordered>
  <fill-contains-method>
    <name>orderedPatternContains</name>
  </fill-contains-method>
</fill-method>
```

次の例は、対応するアセンブラの unorderedPatternContains() メソッドと orderedPatternContains() メソッドです。

```
...
  public boolean unorderedPatternContains(Object[] params, Object group) {
    AssocGroup g = (AssocGroup)group;
    if (g.getName().indexOf((String)params[0]) != -1)
    return true;
    return false;
}

  public boolean orderedPatternContains(Object[] params, Object group) {
    AssocGroup g = (AssocGroup)group;
    if (g.getName().indexOf((String)params[1]) != -1)
    return true;
    return false;
}
```

Assembler インターフェイスを実装すると、クライアントで管理されているアクティブな fill ごとに `refreshFill()` メソッドが呼び出されます。このメソッドには fill パラメータのリストと変更されたアイテムが提供されますが、`refreshFill()` メソッドは、この処理に対して fill を再実行するか、アイテムを fill の末尾に追加するか、または fill を変更しないままにするかを示す、3 つのコードのいずれか 1 つを返します。

Assembler インターフェイスを実装する際に fill メソッドが不必要に実行されないようにする両方の方法を示したコード例については、1409 ページの「Assembler インターフェイスによるアプローチの使用」を参照してください。

fill-method および sync-method によるアプローチの使用

Java アダプタを使用する宛先の `fill-method` および `sync-method` エレメントでは、Flex クライアントアプリケーションでデータ処理が発生したときに呼び出されるメソッドを宣言できます。これらのエレメントを使用して、アセンブラの任意のメソッドを fill および sync メソッドとしてマップします。これらのエレメントで指定するメソッド名は、アセンブラクラス内のメソッドに対応します。アイテムが読み取り専用の場合は、sync メソッドを使用することは決してありません。

クライアントに対して送信された fill 要求のシグネチャが `fill-method` エレメントと一致しない場合、`flex.data.assemblers.Assembler` インターフェイスが実装されていれば、アセンブラで定義されている `fill()` メソッドが Data Management Service によって呼び出されます。

`fill-method` エレメントを使用する場合、各 fill メソッドは、XML ファイルで受信のために宣言されたパラメータの型によって識別されます。2 つの fill メソッドを宣言する際、同じ型の引数を受け取ることがないようにする必要があります。そうしないと、fill メソッドを呼び出そうとしたときにエラーが発生します。もう 1 つ重要な点として、fill メソッドに渡された null 値は任意の型に一致するワイルドカードのように機能する、ということがあります。fill メソッドを多数準備する場合、fill メソッドで名前や ID を第 1 パラメータとして受け取り、実行するクエリを一意に識別することもできます。こうすると、fill メソッドの中で異なる名前を使用してクエリへの送出を実行できます。

クライアントサイドの ArrayCollection にデータアイテムを設定する fill メソッドはいくつでも実装できます。fill メソッドは宛先の `<fill-method>` セクションで指定します。これらのメソッドは、各種の型を持つ任意数のパラメータを受け取り、List オブジェクトを返すことができる fill メソッドとしてそれぞれ割り当てられます。型は実行時に使用できるようにする必要があります。クライアントサイドの DataService コンポーネントの `fill()` メソッドが呼び出されると、サーバーサイドの該当する Java fill メソッドが、クライアントから提供されたパラメータを伴って呼び出されます。

`sync-method` エレメントでは、変更リストをただ 1 つの入力パラメータとして受け取るメソッドを指定します。sync メソッドは ChangeObject のリストを受け取る単一のメソッドとして実装できます。または、AbstractAssembler クラスを拡張し、`updateItem()`、`createItem()`、および `deleteItem()` メソッドをオーバーライドできます。

変更リストは java.util.List の実装で、flex.data.ChangeObject 型のオブジェクトを格納します。リスト内の各 ChangeObject には、アプリケーション固有の変更された Object インスタンスにアクセスするためのメソッドと、発生した変更の種類に関する詳細情報を取得するための便利なメソッド、および変更されたデータメンバーにアクセスするための便利なメソッドを格納します。このリストでは、ChangeObject 全体を繰り返し処理する場合にのみイテレータを使用してください。このリストで get メソッドを使用することはできません。

flex.data.ChangeObject クラスは、公開されている『Flex データサービス Javadoc マニュアル』に含まれています。

次の例は、DAO を呼び出して SQL データベースを操作するアセンブラクラスです。2 つの `loadContacts()` メソッドが fill メソッドで、`syncContacts()` メソッドが sync メソッドです。

```
package samples.contact;

import java.util.Iterator;
import java.util.List;
import java.util.Map;

import flex.data.ChangeObject;
import flex.data.DataSyncException;

public class ContactAssembler{

  public List loadContacts(){
    ContactDAO dao = new ContactDAO();
    return dao.getContacts();
  }

  public List loadContacts(String name){
    ContactDAO dao = new ContactDAO();
    return dao.getContacts(name);
  }

  public Contact getContact(Map uid){
    ContactDAO dao = new ContactDAO();
    return dao.getContact(((Integer) uid.get("contactId")).intValue());
  }

  public List syncContacts(List changes){
    Iterator iterator = changes.iterator();
    ChangeObject co;
    while (iterator.hasNext()){
      co = (ChangeObject) iterator.next();
      if (co.isCreate()){
        doCreate(co);
      }
      else if (co.isUpdate()){
        doUpdate(co);
```

```
        }
        else if (co.isDelete()){
          doDelete(co);
        }
      }
      return changes;
    }

    private void doCreate(ChangeObject co){
      ContactDAO dao = new ContactDAO();
      Contact contact = dao.create((Contact) co.getNewVersion());
      co.setNewVersion(contact);
      }

    private void doUpdate(ChangeObject co){
      ContactDAO dao = new ContactDAO();
      try{
        dao.update((Contact) co.getNewVersion(), (Contact)
          co.getPreviousVersion());
      }
      catch (ItemNotFoundException e){
        throw new DataSyncException(co);
      }
    }

    private void doDelete(ChangeObject co){
      ContactDAO dao = new ContactDAO();
      try{
        dao.delete((Contact) co.getNewVersion(), (Contact)
          co.getPreviousVersion());
      }
      catch (ItemNotFoundException e){
        throw new DataSyncException(co);
      }
    }
}
```

見つかったデータ変更の種類に応じて、syncContacts() メソッドから doCreate()、doUpdate()、または doDelete() メソッドが呼び出され、データアイテムの作成、更新、または削除が試みられます。DAO では、クライアント側にある変更対象の以前のデータと、現在バックエンドのデータリソースにあるデータとが比較されます。以前のデータと現在データリソースにあるデータとが一致しない場合は、DataSyncException がスローされます。DataSyncException の結果、DataConflictEvent がクライアントで発生し、このイベントをクライアントの ActionScript コードで処理できます。クライアントでの競合の解決に関する詳細については、1387 ページの「データ同期の競合の処理」を参照してください。

次の例は DAO での update() および delete() メソッドです。これらのメソッドでは、クライアントからの以前のデータと、この宛先のデータリソースである SQL データベースに現在存在するデータとが、比較されます。クライアントからの以前のデータがデータベース内のデータと一致しない場合は、データは変更されず、例外がスローされます。その結果、クライアントで DataConflictEvent が発生します。

```
...
public void update(Contact newVersion, Contact previousVersion) throws
  DAOException, ItemNotFoundException{
    Connection c = null;
    try{
      c = ConnectionHelper.getConnection();
      PreparedStatement ps = c.prepareStatement("UPDATE contact SET
        first_name=?, last_name=?, address=?, city=?, zip=?, state=? WHERE
        contact_id=? AND first_name=? AND last_name=? AND address=? AND
        city=? AND zip=? AND state=?");
      ps.setString(1, newVersion.getFirstName());
      ps.setString(2, newVersion.getLastName());
      ps.setString(3, newVersion.getAddress());
      ps.setString(4, newVersion.getCity());
      ps.setString(5, newVersion.getZip());
      ps.setString(6, newVersion.getState());
      ps.setInt(7, newVersion.getContactId());
      ps.setString(8, previousVersion.getFirstName());
      ps.setString(9, previousVersion.getLastName());
      ps.setString(10, previousVersion.getAddress());
      ps.setString(11, previousVersion.getCity());
      ps.setString(12, previousVersion.getZip());
      ps.setString(13, previousVersion.getState());
      if (ps.executeUpdate() == 0){
        throw new ItemNotFoundException("Item not found");
      }
    }
    catch (Exception e){
      e.printStackTrace();
      throw new DAOException(e);
    }
    finally{
      ConnectionHelper.closeConnection(c);
    }
  }

  public void delete(Contact newVersion, Contact previousVersion) throws
  DAOException, ItemNotFoundException{
    Connection c = null;
    try{
      c = ConnectionHelper.getConnection();
```

```
        PreparedStatement ps = c.prepareStatement("DELETE FROM contact WHERE
    contact_id=? AND first_name=? AND last_name=? AND address=? AND city=? AND
    zip=? AND state=?");
        ps.setInt(1, newVersion.getContactId());
        ps.setString(2, previousVersion.getFirstName());
        ps.setString(3, previousVersion.getLastName());
        ps.setString(4, previousVersion.getAddress());
        ps.setString(5, previousVersion.getCity());
        ps.setString(6, previousVersion.getZip());
        ps.setString(7, previousVersion.getState());
        if (ps.executeUpdate() == 0){
          throw new ItemNotFoundException("Item not found");
        }
      }
      catch (Exception e){
        e.printStackTrace();
        throw new DAOException(e);
}
      finally{
        ConnectionHelper.closeConnection(c);
      }
    }
...
```

メモ

Flex データサービス Java API を使用するコードをコンパイルするときは、"messaging.jar" ファイルと "flex-messaging-common.jar" ファイルをクラスパスにインクルードする必要があります。
コンパイルされたアセンブラクラスや、DAO クラスなどのその他の必要なクラスを、Web アプリケーションのクラスパスで使用できるようにする必要があります。

Assembler インターフェイスによるアプローチの使用

アセンブラで Assembler インターフェイスによるアプローチを使用するには、flex.data.assemblers.Assembler インターフェイスを実装する必要があります。flex.data.assemblers.AbstractAssembler クラスはこのインターフェイスを拡張したものです。したがって、独自のアセンブラを記述するにはこのクラスを拡張するのが最も簡単な方法です。

クライアントから送信する fill 要求に応じて、`fill()` メソッドでは特定のクエリを実行して Collection オブジェクトを作成し、それをクライアントサイドに返します。Assembler インターフェイスを実装するアセンブラは `autoRefreshFill()` メソッドを持ちます。このメソッドは、データアイテムが変更されたときに fill を更新するかどうかを示すブール値を返します。AbstractAssembler クラスの `autoRefreshFill()` メソッドは、`true` を返します。

アセンブラの autoRefreshFill() メソッドが true を返した場合は、変更されたアイテムごとに refreshFill() メソッドが呼び出されます。autoRefreshFill() メソッドからは次の値を返すことができます。

値	使用方法
DO_NOT_EXECUTE_FILL	fill の内容を変更せずに残す必要がある場合に、この値を使用します。
EXECUTE_FILL	データサービスから自分の fill メソッドを呼び出す必要がある場合に、この値を使用します。
APPEND_TO_FILL	変更されたアイテムを、単に直前の fill 呼び出しから返されたリストの末尾に追加する場合に、この値を使用します。

次の例は CompanyAssembler クラスのソースコードです。このソースコードはサンプル Web アプリケーションに含まれている CRM アプリケーションの一部です。アプリケーションの Java アセンブラと DAO クラスのソースコードは、サンプル Web アプリケーションの WEB_INF/src/samples/crm ディレクトリにあります。CompanyAssembler クラスでは Assembler インターフェイスによるアプローチを使用しています。このクラスは flex.data.assemblers.AbstractAssembler クラスを拡張したもので、autoRefreshFill() メソッドはオーバーライドせず、true を返します。このクラスは refreshFill() メソッドを実装しています。このメソッドはデータ変更の種類に応じて、fill を実行するか、fill への追加を行うか、または fill を実行しません。

```
package samples.crm;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Collection;
import java.util.Arrays;

import flex.data.ChangeObject;
import flex.data.DataSyncException;
import flex.data.assemblers.AbstractAssembler;

public class CompanyAssembler extends AbstractAssembler
{
  public Object getItem(Map uid)
  {
    CompanyDAO dao = new CompanyDAO();
    return dao.getCompany(((Integer) uid.get("companyId")).intValue());
  }

   /**
    * このメソッドは、クライアントから呼び出される fill メソッドのうち、
    * 設定ファイルで fill-method タグを明示的に使用して設定されていない
    * fill メソッドに対して、Data Management Service によって呼び出されます。
    * これらの fill メソッドからは以下のメソッドが直接呼び出されます。
    * この方法は、メカニズムを置き換えるものとして、またはメカニズムを拡張
```

```
     * して fill メソッドを追加する場合に、便利です。
     * </p>
     * <p>
     * 想定する fill の数や種類によっては、第 1 パラメータをクエリの名前
     * として使用し、残りのパラメータをクエリが多数ある場合の値として
     * 使用するとよいでしょう。
     */
    public Collection fill(List fillParameters)
    {
     CompanyDAO dao = new CompanyDAO();
        if (fillParameters.size() == 0)
            return dao.findCcompanies(null, null);
        else if (fillParameters.size() == 1)
            return dao.findCcompanies((String) fillParameters.get(0), null);
        else if (fillParameters.size() == 2)
            return dao.findCompanies((String) fillParameters.get(0),
                                     (String) fillParameters.get(1));

        return super.fill(fillParameters);
    }
  /**
     * fill メソッドが自動更新される場合は、変更されたアイテム（isCreate パラメータで
     * 指定されたとおりに作成または更新されたもの）ごとに、このメソッドが
     * 呼び出されます。このメソッドからは、
     * fill の内容を変更せずに残す必要がある場合は DO_NOT_EXECUTE_FILL を、
     * データサービスから自分の fill メソッドを呼び出す必要がある場合は EXECUTE_FILL を、
     * この変更されたアイテムを、単に直前の fill 呼び出しから返されたリストの末尾に
     * 追加する場合は APPEND_TO_FILL を、返すことができます。
     *
     * @param fillParameters 1 つまたは複数のクライアントによって引き続き管理されている、
     * 管理されたコレクションを作成した fill メソッドへの
     * クライアントサイドパラメータ。
     * @param 作成または更新されようとしているアイテム。
     * @param これが作成処理の場合は true、更新処理の場合は false。
     * @return DO_NOT_EXECUTE_FILL - 何も実行しません。EXECUTE_FILL - fill メソッドを
     * 再実行して新しいリストを取得します。APPEND_TO_FILL - 単にアイテムを
     * 既存のリストに追加します。
     */
    public int refreshFill(List fillParams, Object item, boolean isCreate)
    {
        Company company = (Company) item;
        if (fillParams.size() == 0) // これは " すべてクエリする " 場合です。
            return APPEND_TO_FILL;

        if (fillParams.size() == 1)
        {
            String name = (String) fillParams.get(0);
            if (company.getName().indexOf(name) != -1)
                return APPEND_TO_FILL;
            return DO_NOT_EXECUTE_FILL;
```

```
        }
        else if (fillParams.size() == 2)
        {
            String name = (String) fillParams.get(0);
            String mseg = (String) fillParams.get(1);
            if ((name == null || company.getName().indexOf(name) != -1) &&
                (mseg == null || company.getMarketSegments().contains(mseg)))
                return APPEND_TO_FILL;
            return DO_NOT_EXECUTE_FILL;
        }
        return EXECUTE_FILL;
    }

  public void createItem(Object newVersion)
  {
    CompanyDAO dao = new CompanyDAO();
    dao.create((Company) newVersion);
  }

  public void updateItem(Object newVersion, Object prevVersion, List changes)
  {
    CompanyDAO dao = new CompanyDAO();
    try
    {
      dao.update((Company) newVersion, (Company) prevVersion, changes);
    }
    catch (ConcurrencyException e)
    {
          int companyId = ((Company) newVersion).getCompanyId();
      System.err.println("*** Throwing DataSyncException when trying to update
  company id=" + companyId);
          // 競合エラー。会社の現在の情報が格納されているので、
          // クライアントで調整を実行できます。
      throw new DataSyncException(dao.getCompany(companyId), changes);
    }
  }

  public void deleteItem(Object prevVersion)
  {
    CompanyDAO dao = new CompanyDAO();
    try
    {
      dao.delete((Company) prevVersion);
    }
    catch (ConcurrencyException e)
    {
          int companyId = ((Company) prevVersion).getCompanyId();
      System.err.println("*** Throwing DataSyncException when trying to delete
  company id=" + companyId);
            // 競合エラー。会社の現在の情報が格納されているので、
```

```
                version of the company so the client can reconcile.
            throw new DataSyncException(dao.getCompany(companyId), null);
        }
    }
}
```

次の例は、CRM アプリケーションの EmployeeAssembler の `fill()`、`createItem()`、`updateItem()` の各メソッドです。このアセンブラでも Assembler インターフェイスによるアプローチを使用していますが、ここでは `autoRefreshFill()` メソッドをオーバーライドして `false` を返しています。追加または更新されるデータアイテムに応じて、`createItem()` メソッドおよび `updateItem()` メソッドは、flex.data.DataServiceTransaction クラスのインスタンスの `refreshFill()` メソッドを呼び出す特定のクエリの結果を使用して、fill を更新します。flex.data.DataServiceTransaction クラスの詳細については、1439 ページの「サーバーからクライアントへのデータ変更のプッシュ」を参照してください。

```
public Collection fill(List fillParameters)
{
    EmployeeDAO dao = new EmployeeDAO();
    if (fillParameters.size() == 0)
        return dao.getEmployees();

    String queryName = (String) fillParameters.get(0);

    if (queryName.equals("match"))
        return dao.findMatchingEmployees((String) fillParameters.get(1));
    if (queryName.equals("byCompany"))
        return dao.findEmployeesByCompany((Integer) fillParameters.get(1));

        return super.fill(fillParameters); // エラーをスローします。
}

public void createItem(Object newItem)
{
    Employee newEmployee = (Employee) newItem;
    DataServiceTransaction dtx =
        DataServiceTransaction.getCurrentDataServiceTransaction();
    EmployeeDAO dao = new EmployeeDAO();
    dao.create(newEmployee);

    // " すべての従業員 " クエリを更新します。
    dtx.refreshFill("crm.employee", new ArrayList(0));
    // この従業員の会社に対する従業員を参照している
    // 人物をすべて更新します。
    dtx.refreshFill("crm.employee", Arrays.asList(
        new Object[] {"byCompany",
            newEmployee.getCompany() == null ? null : new
            Integer(newEmployee.getCompany().getCompanyId())}));
}

public void updateItem(Object newVersion, Object prevVersion, List changes)
{
    EmployeeDAO dao = new EmployeeDAO();
```

```
   DataServiceTransaction dtx =
     DataServiceTransaction.getCurrentDataServiceTransaction();
   try
   {
     Employee newEmployee = (Employee) newVersion;
     Employee prevEmployee = (Employee) prevVersion;
     dao.update(newEmployee, prevEmployee, changes);

/*
* 更新の実行が完了したので、実行された変更に基づいてどの fill を
* 更新する必要があるのかを決定します。
*/
/*
* ユーザーが会社を変更した場合は、会社の古いデータの fill と新しいデータの fill の
* 2 つを更新する必要があります。
*/
     if (changes == null || changes.contains("company"))
     {
       Company prevCompany = prevEmployee.getCompany();
       Company newCompany = newEmployee.getCompany();
       dtx.refreshFill("crm.employee",
         Arrays.asList(new Object[] {"byCompany",
         prevCompany == null ? null :
         new Integer(prevCompany.getCompanyId())}));
         dtx.refreshFill("crm.employee",
         Arrays.asList( new Object[] {"byCompany",
         newCompany == null ? null : new
         Integer(newCompany.getCompanyId())}));
       }

/*
* ユーザーがたまたまパターン一致の結果を名前順に表示していた
* 場合は、検索結果をリアルタイムに更新することは簡単なので
* それを実行します。
*/
       if (changes == null || changes.contains("firstName") ||
         changes.contains("lastName"))
       {
       // 名前 ( 名 ) を変更した場合は、従業員に対する
       // すべての検索クエリを更新します。
         dtx.refreshFill("crm.employee", Arrays.asList
           (new Object[] {"match", null}));
       }
     }
   catch (ConcurrencyException e)
   {
     int employeeId = ((Employee) newVersion).getEmployeeId();
     System.err.println("*** Throwing DataSyncException when trying
     to update employee id=" + employeeId );
     throw new DataSyncException(dao.getEmployee(employeeId), changes);
   }
}
```

メモ

Flex データサービス Java API を使用するコードをコンパイルするときは、"messaging.jar" ファイルと "flex-messaging-common.jar" ファイルをクラスパスにインクルードする必要があります。
コンパイルされたアセンブラクラスや、DAO クラスなどのその他の必要なクラスを、Web アプリケーションのクラスパスで使用できるようにする必要があります。

設定ファイルの設定

次の表では、Java アダプタに固有の宛先エレメントについてそれぞれ説明します。特に説明がない限り、これらのエレメントは properties エレメントまたは server エレメントのサブエレメントです。Hibernate アセンブラはこの表のエレメントとは異なる特別なエレメントを持ちます。詳細については、1421 ページの「Hibernate 設定ファイル」を参照してください。

エレメント	説明

`<source>`
`samples.contact.ContactAssembler`
`</source>`
`<scope>application</scope>`
	source エレメントでは、データ変更を処理するクラスであるアセンブラクラスを指定します。scope エレメントではアセンブラのスコープを指定します。有効な値は、application、session、および request です。
	これらのエレメントは properties エレメントのサブエレメントです。

エレメント	説明
<fill-method> <name>loadOrderedPatternGroups</name> <params> java.lang.Boolean,java.lang.String </params> <ordered>true</ordered> <fill-contains-method> <name>orderedPatternContains</name> </fill-contains-method> <security-constraint ref="sample-users"/> </fill-method>	fill-method エレメントでは、fill 要求が行われたときに呼び出すメソッドを指定します。 必須の name エレメントでは、呼び出すメソッドの名前を指定します。 オプションの params エレメントでは、fill 処理の入力パラメータ型を指定し、メソッドのオーバーロードに対応します。 オプションの fill-contains-method エレメントでは、Assembler クラス内のメソッドを参照します。このメソッドは、変更または作成されたアイテムと、fill パラメータの List を受け取り、アイテムがその fill に存在するかどうかに応じて true または false を返します。fill が並べられておらず、メソッドが true を返した場合は、その fill に対して、管理されたアイテムのリストの末尾にアイテムが単に追加されます。fill が並べられていて、メソッドが true を返した場合は、fill メソッドが再実行され、リストの順序が正しく設定されます。メソッドが false を返した場合には、その更新に対して fill メソッドはそのままになります。 オプションの security-constraint および security-run-as エレメントでは、"services-config.xml" ファイル内のセキュリティ設定を参照します。 宛先の fill-method および sync-method エレメントで fill および sync メソッドを宣言する別の方法として、flex.data.assemblers.Assembler インターフェイスを実装し、fill() メソッドの実装をアセンブラに含めることができます。詳細については、1409 ページの「Assembler インターフェイスによるアプローチの使用」を参照してください。

エレメント	説明
<sync-method> <name>syncCustomer</name> <security-constraint ref="admins"/> </sync-method>	sync-method エレメントでは、更新、削除、挿入の各処理に対して呼び出されるメソッドを指定します。必須の name エレメントでは、呼び出すメソッドの名前を指定します。 パラメータについては ChangeObject の List としてあらかじめ定義されているため、params エレメントはありません。 security-constraint および security-run-as エレメントでは、"services-config.xml" ファイル内のセキュリティ設定を参照します。 宛先の fill-method および sync-method エレメントで fill および sync メソッドを宣言する別の方法として、flex.data.assemblers.Assembler インターフェイスを実装し、fill() メソッドの実装をアセンブラに含めることができます。詳細については、1409 ページの「Assembler インターフェイスによるアプローチの使用」を参照してください。
<get-method> <name>getCustomer</name> <security-constraint ref="admins"/> </get-method>	get-method エレメントでは、アイテムインスタンスの List ではなく単一のアイテムインスタンスを取得するために呼び出すメソッドを指定します。このエレメントが存在する場合は、常に単一のパラメータ、つまり、オブジェクトの identity を示すために使用される値の Map を受け取ります。
<count-method> <name>numCustomers</name> <security-constraint ref="admins"/> </count-method>	count-method エレメントでは、アセンブラから整数を取得するために呼び出すメソッドを指定します。この整数を使用することで、先にすべてのデータセットをメモリにロードしなくてもアイテムの数を取得できます。fill-method エレメントと同様に、count-method エレメントでもパラメータを受け取ることができます。 このメソッドでは fill されるシーケンスのサイズは取得されません。つまり、どのようなシーケンスでもデータのサブセットを表すことができ、シーケンスは常に完全にサーバーにロードされます。count メソッドの実装では、データベースに対して COUNT ステートメントを実行することもできます。
<properties> <item-class> mypackage.MyClass </item-class> ... </properties>	クライアントサイドから受信した匿名の ActionScript オブジェクトを、Java アセンブラで想定されている、厳密に型指定された元の単一の Java クラスに変換します。 item-class エレメントは properties エレメントのサブエレメントです。

Hibernate 宛先の使用

Hibernate アセンブラは、Java アダプタと一緒に使用することによって Hibernate オブジェクトリレーショナルマッピングシステムへのブリッジを提供する、特別な Java アセンブラクラスです。Hibernate アセンブラでは Hibernate 3.0 以降がサポートされています。このアセンブラを使用すると、既存の Hibernate バックエンドと統合するために必要な、データ管理に固有の Java コードを記述する必要がなくなります。代わりに、Hibernate の XML ベースのマッピングファイルと設定ファイルを設定し、それからそれらのファイルを参照する Data Management Service 宛先を設定します。Hibernate アセンブラは実行時に Hibernate 設定ファイルを使用し、Hibernate API を使用することによってデータ変更をリレーショナルデータベースに永続化します。Hibernate の処理はアセンブラの中にカプセル化されます。Hibernate の設定内の関連付けは、Flex の宛先定義内の関連付けとしてミラーリングしてください。

Data Management Service の count、get、create、update、delete の各メソッドはすべて、同様の名前と動作を持つ Hibernate 処理に対応しており、Hibernate マッピングファイル内の設定以外の設定は必要ありません。ただし、Hibernate のオブジェクト取得メソッドは洗練されており、Data Management Service の fill メソッドを Hibernate のいずれかのオブジェクト取得メソッドにマップする場合は追加の設定を含める必要があります。

次の例は、Hibernate アセンブラを使用する宛先です。この HibernatePerson 宛先は HibernateGroup 宛先との多対 1 関係を持っています。

```
...
<destination id="HibernatePerson" channels="rtmp-ac">
  <adapter ref="java-adapter" />
  <properties>
    <source>flex.data.assemblers.HibernateAssembler</source>
    <scope>application</scope>
    <metadata>
      <identity property="id"/>
      <many-to-one property="group"
        destination="HibernateGroup" lazy="true"/>
      </metadata><network>
      <paging enabled="true" pageSize="10" />
      <throttle-inbound policy="ERROR" max-frequency="500"/>
      <throttle-outbound policy="REPLACE" max-frequency="500"/>
    </network>
    <server>
      <!-- このエレメントが存在しない場合、アダプタでは Hibernate エンティティが
      宛先 ID と同じ名前を持っているものと見なされます。
      -->
      <hibernate-entity>contacts.hibernate.Person</hibernate-entity>
      <!-- 競合モードによって、競合が 1 つでもあるかどうかを調べ、競合がある場合は、
      変更されたプロパティだけを調べるか、クライアントがオブジェクト全体の
      正しいデータを持っていたことを検証するかどうかが決まります。
      update-conflict-mode に対する有効な値は、NONE、PROPERTY、および OBJECT です。
      delete-conflict-mode に対する有効な値は、NONE および OBJECT です。
```

```
        -->
        <update-conflict-mode>PROPERTY</update-conflict-mode>
        <delete-conflict-mode>OBJECT</delete-conflict-mode>
        <fill-configuration>
          <use-query-cache>false</use-query-cache>
          <allow-hql-queries>true</allow-hql-queries>
        </fill-configuration>
      </server>
    </properties>
</destination>
```

次の例は、HibernatePerson 宛先と HibernateGroup 宛先を操作するための、対応する ActionScript メソッドです。

```
...
private function createDataCollection():void {
  groupDS = new DataService("HibernateGroup");
  groupDS.addEventListener(MessageFaultEvent.FAULT, faultHandler);
  groupDS.addEventListener(MessageEvent.RESULT, resultHandler);
  groupDS.addEventListener(DataConflictEvent.CONFLICT, conflictHandler);
  groupDS.autoCommit = false;
  personDS = new DataService("HibernatePerson");
  personDS.autoCommit = false;
  personDS.fill(allPersons,"flex:hql","From Person where groupId is null");
  groupDS.fill(groups1,"flex:hql","From Group");
  personDS.dataStore = groupDS.dataStore;
}
...
```

サポートされている Hibernate 機能

現在、Hibernate アセンブラではデータを操作するための次の Hibernate 機能がサポートされています。

Hibernate 機能	説明
load と get	単一のオブジェクトを identity によって取得します。 Data Management Service の `getItem()` 機能は Hibernate の get 機能に対応しています。Hibernate の load メソッドは Hibernate アセンブラでは使用されません。

Hibernate 機能	説明
HQL (Hibernate Query Language)	アイテムの List を HQL クエリに基づいて取得します。 `fill` の引数を HQL クエリストリングとしてパラメータのマップと一緒に認識するよう、Data Management Service の fill 機能を設定できます。 クライアントコードから HQL クエリを使用するには、まず `fill-configuration` セクションで `<allow-hql-queries>true</allow-hql-queries>` を設定する必要があります。 **メモ：**クライアントで任意の HQL クエリの指定を許可した場合、それによって、潜在的に危険な処理を、信頼性のないクライアントコードに公開することになります。クライアントでは、クエリの取得や実行に対して意図しないデータを取得できる可能性があり、それが原因でデータベースにパフォーマンスの問題が生じることがあります。HQL クエリはプロトタイプ作成の場合に使用することをお勧めします。ただし、アプリケーションを展開する前に名前付きのクエリに置き換え、運用環境ではデフォルトの設定である `<allow-hql-queries>false</allow-hql-queries>` を使用することをお勧めします。 HQL を使用する場合、クライアントサイドの `DataService.fill()` メソッドは次の例のようなものになります。 `myService.fill(myCollection, "flex:hql", ["from Person p where p.firstName = :firstName", parameterMap]);` `parameterMap` パラメータには、クエリのパラメータ名と値のバインディングを格納します。前の例では名前付きパラメータを使用しています。 位置パラメータもサポートされています。つまり、`"from Person p where p.firstName = ?"` のような HQL ストリングを記述できます。この場合、`parameterMap` パラメータは値のマップではなく値の配列になります。通常、位置パラメータを使用すると名前付きパラメータを使用するよりも信頼性が落ちると考えられていますが、いずれのアプローチもサポートされています。
名前付き Hibernate クエリ	Hibernate で名前付きクエリを設定するには、`query` という名前の Hibernate 設定エレメントを使用します。そして、Hibernate クエリの名前を `DataService.fill()` メソッドへの第 2 引数として渡すことにより、このクエリを Flex クライアントから呼び出すことができます。HQL クエリの場合と同様に、Hibernate クエリで名前付きパラメータを使用している場合は、それらのパラメータを含む匿名オブジェクトを使用して、クライアントでこのクエリを指定します。Hibernate クエリで位置パラメータを使用している場合には、パラメータの配列を設定します。次の例は、名前付きクエリを使用している `fill()` メソッドです。 `myDS.fill(myCollection, "eg.MyHibernateEntity.myQueryName", myQueryParameterObjectOrArray);`

Hibernate 設定ファイル

Hibernate では 2 種類の設定ファイルを使用します。グローバル設定ファイルには、データベースの設定および動作のための宣言を格納します。マッピングファイルには、Java のクラスとフィールドをデータベースのテーブルと列にマップする方法を示した宣言を格納します。Data Management Service は実行時にこれらのファイルにアクセスする必要があります。Hibernate では、Web アプリケーションのクラスパス内でこれらのファイルにアクセスできることが必要になります。hibernate.cfg.xml ファイルを WEB_INF/classes ディレクトリに置き、Hibernate JAR ファイルを Web アプリケーションの WEB_INF/lib ディレクトリに置いてください。

Hibernate アセンブラは Hibernate マッピングファイルを使用して、名前付きクエリをクライアントサイドコードに公開します。クライアントコードでは、fill パラメータ引数を使用してこれらのクエリにパラメータを提供できます。また、宛先設定で `allow-hql-queries` エレメントを `true` に設定することによって、クライアントでの HQL クエリの使用を許可することもできます。

Flex では、Hibernate エンティティのマッピングでの同時競合を自動検出するために、Hibernate の `version` エレメントまたは `timestamp` エレメントがサポートされています。Hibernate 設定ファイルにこれらの設定が使用されていると、FDS の update または delete 競合モードが定義されている場合でも、それより優先されます。FDS の競合モードが定義されている場合は、Hibernate の自動競合検出を使用することを示す警告がログに記録されます。

注	このオプションを使用した場合は、信頼性のないコードによって実行される任意のクエリを公開することになります。このオプションはプロトタイプ作成の場合にのみ使用し、運用環境用のアプリケーションを展開する場合は展開前に名前付きクエリに切り替えてください。

現在、Hibernate アセンブラでは条件ベースのクエリはサポートされていませんが、flex.data.assemblers.HibernateAssembler クラスを拡張し、追加の Java コードを記述することによって、任意のクエリを公開することができます。このクラスの詳細については、公開されている『Flex データサービス Javadoc マニュアル』を参照してください。

宛先のサーバーセクションで設定をしなくても、Hibernate アセンブラを使用してデータをリレーショナルデータベースに永続化することが可能です。このシナリオでは、fill 要求が HQL クエリにマップされ、Hibernate アセンブラでクエリキャッシュや Hibernate ページングが使用されません。ただし、実行時に Flex データサービス Web アプリケーションから Hibernate 設定ファイルを参照できるようにする必要があります。

クエリパラメータバインディングでの型マッピングのカスタム処理を公開するため、Flex では、Hibernate 設定ファイルで提供されている永続 Java オブジェクト用の Hibernate 型マッピングが優先されます。`type` 属性の値は、サポートされているすべてのデータベースにおいて Java 型を SQL 型にマップする、Hibernate 型として扱われます。属性を省略した場合、Hibernate ではパラメータの Java 型に基づいて型が選択されます。Java の Date 型をデータベースの日付型やタイムスタンプ型にマッピングする処理など、場合によっては、このアプローチを Hibernate マッピング内のパラメータ宣言にオーバーライドする必要性が高くなります。

次の例は、Group という名前の Java クラスにマップする、Group.hbm.xml という Hibernate マッピングファイルです。このファイルでは、Group オブジェクトと Person オブジェクトとの間で 1 対多の関係を宣言しています。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="dev.contacts.hibernate.Group" table="simplegroup">
        <id name="id" column="id">
            <generator class="identity"/>
        </id>
        <property name="name" />
        <many-to-one name="leader"
          class="dev.contacts.hibernate.Person" column="leaderId"
          cascade="save-update"/>
        <set name="members" inverse="true">
        <key column="groupId" />
        <one-to-many class="dev.contacts.hibernate.Person" />
        </set>
    </class>
</hibernate-mapping>
```

次の例は、対応する Person クラスのソースコードです。

```
package contacts.hibernate;
import java.util.List;
import java.util.HashSet;
import java.util.Set;

public class Group {
    private String name = null;
    private Set members = null;
    private Person leader = null;
    private String id;
    public String getName() {
        return name;
    }
    public void setName(String n) {
        name = n;
    }
    public Set getMembers() {
        return members;
    }
    public void setMembers(Set s) {
        members = s;
    }
    public Person getLeader() {
        return leader;
```

```
    }
    public void setLeader(Person sp) {
        leader = sp;
    }

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }

    public int hashCode() {
        if (id == null)
            return 0;
        return id.hashCode();
    }

    public boolean equals(Object o) {
        if (o instanceof Group) {
            Group e = (Group) o;
            if (e.getId() == getId())
                return true;
            if (e.getId() == null)
                return false;
            return e.getId().equals(getId());
        }
        return false;
    }
}
```

設定ファイルの設定

次の表では、Hibernate アセンブラに固有の宛先エレメントについてそれぞれ説明します。これらのエレメントは server エレメントの子エレメントです。

エレメント	説明
<source> flex.data.assemblers.HibernateAssembler </source> <scope>application</scope>	source エレメントでは、データ変更を処理するクラスであるアセンブラクラスを指定します。scope エレメントではアセンブラのスコープを指定します。有効な値は、application、session、および request です。 これらのエレメントは properties エレメントのサブエレメントです。
<hibernate-entity> Person </hibernate-entity>	クエリパラメータバインディングでの型マッピングのカスタム処理を公開します。Flex では、Hibernate の設定で提供されている永続 Java オブジェクト用の Hibernate 型マッピングが優先されます。このようなプログラムからのクエリの作成とパラメータの型マッピングで必要となる、クラスメタデータにアクセスできるようにするには、永続 Java オブジェクトを表す Hibernate エンティティの名前を使用して宛先を設定します。 設定から省略した場合、Hibernate エンティティの型は宛先の id と同じであると見なされます。
<update-conflict-mode> PROPERTY </update-conflict-mode>	Flex クライアントアプリケーションからの更新データが古くないことを検証します。省略した場合、データ検証は行われません。有効な値は、NONE、PROPERTY、および OBJECT です。データの競合が生じた場合は、DataConflictEvent がクライアントで発生します。
<delete-conflict-mode> OBJECT </delete-conflict-mode>	Flex クライアントアプリケーションからの削除データが古くないことを検証します。省略した場合、データ検証は行われません。 有効な値は、NONE および OBJECT です。 データの競合が生じた場合は、DataConflictEvent がクライアントで発生します。

エレメント	説明
<fill-configuration> <allow-hql-queries>true</allow-hql-queries> <use-query-cache>true</use-query-cache> </fill-configuration>	allow-hql-queries エレメントでは、HQL クエリを使用できるかどうかを指定します。デフォルト値は false です。 use-query-cache エレメントは、Hibernate でのクエリのキャッシュを可能にする、Hibernate クエリメソッドのオプションです。デフォルト値は false です。
<create-security-constraint ref="sample-users"/>	ref 属性で参照されるセキュリティ制限を、create 要求に適用します。
<update-security-constraint ref="sample-users"/>	ref 属性で参照されるセキュリティ制限を、update 要求に適用します。
<delete-security-constraint ref="sample-users"/>	ref 属性で参照されるセキュリティ制限を、delete 要求に適用します。

ActionScript オブジェクトアダプタの使用

ActionScript オブジェクトアダプタは、一時データのみを必要とし、バックエンドのデータリソースを必要とせず、Flex クライアントアプリケーションがデータオブジェクトのただ1つの作成者および使用者であるシナリオでの使用を目的としています。Data Management Service は、ActionScript オブジェクトアダプタを使用してメモリ内のオブジェクトを管理します。ActionScript オブジェクト内の任意のフィールドを使用し、宛先のメタデータセクションの identity エレメントに一意の identity を提供できます。組み合わせた identity を設定する場合は、フィールドを複数指定します。identity を指定しない場合は、デフォルトで ActionScript オブジェクトの uid によってオブジェクトの identity が提供されます。

次の表では、ActionScript オブジェクトアダプタに固有の宛先エレメントについてそれぞれ説明します。これらのエレメントは server エレメントの子エレメントです。

エレメント	説明
<create-security-constraint ref="sample-users"/>	ref 属性で参照されるセキュリティ制限を、create 要求に適用します。
<read-security-constraint ref="sample-users"/>	ref 属性で参照されるセキュリティ制限を、fill 要求に適用します。
<update-security-constraint ref="sample-users"/>	ref 属性で参照されるセキュリティ制限を、update 要求に適用します。
<delete-security-constraint ref="sample-users"/>	ref 属性で参照されるセキュリティ制限を、delete 要求に適用します。

次の例のクライアントアプリケーションは、ActionScript アダプタを使用し、すべてのクライアントにわたって単一のデータアイテムがまだ TextArea コントロールに存在しない場合に、そのデータアイテムを永続化します。バックエンドのデータリソースが存在しないため、サーバーが停止した場合はデータが失われます。

```
<?xml version="1.0"?>
<!-- fds\datamanagement\ASAdapter.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
 height="100%" width="100%"
 creationComplete="initApp();">

    <mx:Script>
        <![CDATA[
            import mx.data.DataService;
            import mx.rpc.AsyncToken;

            public var noteObj:Object = new Object();
            [Bindable]
            public var getToken:AsyncToken;
            private var ds:DataService;

            private function initApp():void {
            ds = new DataService("notes");
            ds.autoCommit = false;
            noteObj.noteId = 1;
            noteObj.noteText =
                "Type your notes here and share them with other clients!";
            getToken = ds.getItem(noteObj, noteObj);
            }
      ]]>
    </mx:Script>
    <mx:Binding source="log.text" destination="getToken.result.noteText"/>
    <mx:TextArea id="log" width="100%" height="100%"
text="{getToken.result.noteText}"/>
    <mx:Button label="Send" click="ds.commit();"/>
</mx:Application>
```

サーバーサイドの設定は、"data-management_config.xml" ファイルにある次の宛先のみです。

```
<destination id="notes">
  <adapter ref="actionscript"/>
    <properties>
     <metadata>
       <identity property="noteId"/>
     </metadata>
    </properties>
</destination>
```

アイテムにはそれぞれ一意の ID を持たせる必要があります。ID を自動的に作成するバックエンドのデータリソースはありません。したがって、ArrayCollection のデータアイテムを必要とするアプリケーションでは、管理されたオブジェクトの作成時にクライアントでアイテム ID を生成する必要があります。次の例のコードでは、Math.random() メソッドを使用して customer.custId を生成しています。実際のアプリケーションでは、custId は設定ファイル内の宛先で識別子として指定するプロパティになります。

```
private function newCustomer():void {
  dg.selectedIndex = -1;
  customer = new Customer();
  customer.custId = Math.random() * 1000;
}
```

階層コレクションの管理

" 階層コレクション " は、単純なプリミティブ値以外の型の複雑なプロパティを持つオブジェクトを含んだコレクションです。たとえば、Person オブジェクトを含むコレクションで、Person オブジェクトが Address プロパティを持ち、その Address プロパティが番地、都市、州などのプロパティを持つことが考えられます。

階層コレクションを Data Management Service と組み合わせて管理する場合、3 つの方法があります。

- " 階層値 " によるアプローチでは、単一の Data Management Service 宛先でデータのツリー全体を管理します。
- " 管理された関連付け " によるアプローチでは、子宛先への関連付けを宣言した親宛先を定義し、これらの子宛先で子オブジェクトのプロパティの値を管理します。
- "クエリ" によるアプローチでは、クエリを使用して 2 つのオブジェクト間の関係を実装できます。クエリでは、アセンブラで定義された fill メソッドのいずれかのパラメータが、関連するオブジェクトの ID になります。

これらのアプローチは、プロパティに応じて、単独で使用することも、組み合わせて使用することもできます。

階層値によるアプローチ、および管理された関連付けによるアプローチでは、fill 呼び出しから得られる同じオブジェクトグラフを親宛先で返します。階層値によるアプローチでは、親オブジェクトのアダプタが、複雑なプロパティ全体の状態を保存する役割を担います。これはデフォルトの動作です。親オブジェクトのアダプタはまた、sync 呼び出しに応答してグラフ全体を更新する役割も担います。何らかの変更が発生した場合は、オブジェクトの新しい完全なグラフを取得し、そのグラフで発生した変更をすべて更新する役割を担います。

管理された関連付けによるアプローチでは、親宛先の宛先設定に、独自の宛先を持つ子オブジェクトの関連付けを定義した追加のタグを含めます。この関連付けタグからは、子オブジェクトの状態を管理する役割を担う別の宛先を参照します。

階層値によるアプローチや管理された関連付けによるアプローチを使用する場合は、管理されたクラスの最上位オブジェクトのグラフにある各クラスを [Managed] メタデータタグを使用してマークするか、または mx.data.IManaged インターフェイスを明示的に実装する必要があります。これにより、管理されたオブジェクトで正しい変更イベントがサポートされ、遅延ロードされる 1 対 1 関連付けプロパティの getter メソッドから ItemPendingError をスローできるようになります。遅延ロードと ItemPendingError の詳細については、1430 ページの「管理された関連付けの実装」を参照してください。

次の例は、[Managed] メタデータタグを使用した ActionScript クラスのコードです。この Employee クラスは Employees という ArrayCollection で使用されます。Employees は Company クラスのプロパティとして使用されます。

```
package samples.crm
{

   import mx.data.IManaged;
   import mx.data.utils.Managed;
   import mx.core.mx_internal;

   [Managed]
   [RemoteClass(alias="samples.crm.Employee")]
   public class Employee{
      public var employeeId:int;

      public var firstName:String = "";

      public var lastName:String = "";

      public var title:String = "";

      public var phone:String = "";

      public var email:String = "";
   }
}
```

次の例は、[Managed] メタデータタグを使用する代わりに mx.data.IManaged インターフェイスを実装したクラスです。implement mx.data.IManaged を実装する場合は mx.core.IUID インターフェイスをサポートする必要もあります。mx.core.IUID では uid プロパティが必要になります。また、このクラスの最後のコメントブロックの下で宣言されている referenceIds 変数と destination 変数を含める必要もあります。

```
import mx.core.mx_internal;
import mx.data.Managed;
import mx.data.IManaged;
import mx.utils.UIDUtil;

   [RemoteClass(alias="foo.bar.Customer")]
```

```
  public class Customer implements IManaged {
    public function Customer() {
    super();
  }

  [Bindable(event="propertyChange")]
  public function get firstName():String {
    _firstName = Managed.getProperty(this, "firstName", _firstName);
    return _firstName;
  }

  public function set firstName(value:String):void {
    var oldValue:String = this._firstName;
    _firstName = value;
    Managed.setProperty(this, "firstName", oldValue, _firstName);
  }
// IManaged のすべてのインプリメンタで IUID をサポートする必要があります。
// IUID には uid が必要になります。
  [Bindable(event="propertyChange")]
  public function get uid():String {
    // uid にまだ値が割り当てられていない場合は、新しい値を作成します。
    if (_uid == null) {
      _uid = UIDUtil.createUID();
    }
    return _uid;
  }

  public function set uid(value:String):void {
    _uid = value;
  }
// これらは、DataService によって " 管理される " ことを必要とするどのようなオブジェクトでも
// 必要になる、特別な要件です。
// referencedIds は、遅延ロードされるプロパティに属するオブジェクト ID の
// リストです。
// 宛先を使用し、リモート宛先にある "flex-dataservices.xml" ファイルの中で
// このオブジェクトによって設定された関連付けについて、それらに関する
// メタデータ情報を参照します。

  mx_internal var referencedIds:Object = {};
  mx_internal var destination:String;

  private var _firstName:String;
  private var _uid:String;
}
```

管理された関連付けの実装

管理された関連付けによるアプローチでは、親宛先は、子オブジェクトのプロパティに対する変更について更新を処理することはなく、親から参照される子の id 値に対する変更を処理する役割だけを担います。ただし、Person オブジェクトが Address プロパティを持っている例のような場合は、階層値によるアプローチを使用して、人物の住所の都市に対する変更を Person 宛先で処理します。

管理された関連付けによるアプローチでは、Address 宛先で都市に対する変更を処理します。ユーザーが、異なる id を持つ Address を Person オブジェクトの Address プロパティに割り当てた場合は、Person 宛先で変更を処理します。関係において、サーバーからクライアントへのオブジェクトグラフの遅延ロードをサポートするかどうかを設定することもできます。

単方向の 1 対 1、1 対多、または多対 1 の関係、あるいは双方向の多対多の関係となる、宛先間の関係を設定できます。管理された関連付けは、宛先の one-to-one、one-to-many、および many-to-many エレメントで設定します。これらのエレメントは metadata エレメントのサブエレメントであり、metadata は properties エレメントのサブエレメントです。

データベーステーブルに依存する管理された関連付けにおいて、多対多の関係で作業を行う場合は、いずれかのテーブルの外部キーを使用するのではなく、結合テーブルを作成して、データベースが使用するテーブル間の関係を確立する必要があります。

サーバーからクライアントへのオブジェクトグラフの遅延ロードを有効にするには、one-to-one、one-to-many、または many-to-many エレメントの lazy 属性を true に設定します。クライアントから初めて値にアクセスすると、複数値の関連付けの場合は ArrayCollection.getItem() メソッドから、または単一値の関連付けの場合は Managed.getProperty() メソッドから、ItemPendingError がスローされます。このスローによって、参照されたオブジェクトを取得する要求が発生し、ItemPendingError 内のイベントハンドラが呼び出されて、アイテムが使用可能になったときにクライアントに通知されます。Flex データバインディングを使用しているときは、lazy 属性を true に設定している場合に、この動作が自動的に実行され、データは必要になった時点で出現します。

データバインディングを遅延ロードと組み合わせて使用しない場合は、コードの中で ItemPendingError をキャッチして適切に処理できます。アイテムを ArrayCollection から取得するために非同期呼び出しを必要とする場合は、ItemPendingError がスローされます。このエラーの受信側で、要求されたアイテムが使用可能になったことの通知を非同期呼び出しの完了時に必要とする場合は、受信側で addResponder() メソッドを使用し、mx.rpc.IResponder インターフェイスをサポートするオブジェクトを指定して、アイテムが使用可能になったときに応答する必要があります。IResponder インターフェイスは mx.collections.ItemResponder クラスで実装されており、data プロパティがサポートされています。

次の例では、クライアントの groupCollection オブジェクトでグループオブジェクトがまだ使用できないときに、ItemPendingError がスローおよびキャッチされます。groupCollection オブジェクトはサーバーの AssocGroup 宛先から設定されます。宛先には admin プロパティがあります。このプロパティは AssocPerson インスタンスへの参照であり、遅延ロードされます。printAdminName() メソッドは、groupCollection オブジェクト内の最初のグループに対して呼び出されます。グループが使用できない場合は、メソッドから ItemPendingError がスローされます。ItemPendingError を処理するために ItemResponder が登録されています。ItemResponder は、グループが使用可能になったときに printAdminName() メソッドを呼び出します。失敗が生じた場合、ItemResponder は fetchAdminError() メソッドを呼び出します。

```
...
import mx.collections.ItemResponder;
import mx.messaging.messages.ErrorMessage;
import mx.collections.errors.ItemPendingError;
...

public function printAdminName (data:Object, group:Object):void {
  trace(group.admin.firstName);
}

public function fetchAdminError(message:ErrorMessage):void {
  trace("error occurred fetching admin: " + message.faultString);
}

// この関数は、遅延ロードされる Person への参照を admin プロパティに持つ
// "group" オブジェクトを取得するために呼び出されます。
// printAdminName 関数は、このオブジェクトがクライアントでまだ
// ロードされていない場合に、group.admin プロパティを取得するときに
// ItemPendingError をスローする場合があります。
// 関数は、値が使用可能になったときに再び printAdminName を呼び出す
// リスナーを登録します。

public function dumpAdminName():void {
  try {
    printAdminName(null, groupCollection.getItemAt(0));
  }
  catch (ipe:ItemPendingError) {
    trace("item pending error fetching admin.");
    ipe.addResponder(new ItemResponder(printAdminName, fetchAdminError,
    groupCollection.getItemAt(0)));
  }
}
...
```

まとめると、階層値によるアプローチでは、単一の宛先でグラフの状態全体を管理します。しかし、管理された関連付けによるアプローチでは、親宛先は、参照される子オブジェクトの ID、および親オブジェクトと子オブジェクト間の関係を管理します。管理された関連付けによるアプローチでは、子アセンブラが子オブジェクト自身の状態を所有します。管理された関連付けによるアプローチを使用する場合、あるアプリケーションでは各宛先を互いに独立して使用し、別のアプリケーションではそれらの宛先をまとめて使用できる、という利点もあります。

管理された関連付けの例について

このセクションでは、管理された関連付けによるアプローチを使用して会社と従業員との間の 1 対多関係をモデル化するアプリケーションについて、いくつかの重要な側面を説明します。

アプリケーションのクライアントサイドでは、dsCompany という名前の DataService コンポーネントが、Company オブジェクトの ArrayCollection を管理します。dsCompany コンポーネントでは、company という名前のサーバーサイド宛先を使用します。

Company にはそれぞれ employees プロパティがあります。このプロパティは Employee オブジェクトの ArrayCollection です。これらのクライアントサイドの Company および Employee オブジェクトはサーバーサイドの同じ名前の Java クラスにマップされます。Java クラスは会社宛先へのメソッド呼び出しによって設定されます。

もう 1 つ、employee という宛先があり、各会社の従業員データを提供します。会社宛先には、会社宛先と従業員宛先との間の 1 対多関係があります。宛先はそれぞれ独自のアセンブラおよび DAO クラスを持っています。

1 対多関係のため、従業員宛先は会社宛先とは独立して、独自のアセンブラと DAO を使用して、従業員データに対するすべての変更を管理します。Company および Employee Java クラスは会社と従業員を表し、これらのクラスはクライアントサイドにある対応する Company および Employee ActionScript クラスにマップされます。

宛先の設定

次の例は、会社宛先と従業員宛先の関連するセクションを示しています。宛先どうしの関係を定義する one-to-many エレメントもあります。one-to-many エレメントでは、会社宛先の employees プロパティと従業員宛先との間の 1 対多関係を定義しています。1 対多関係の設定の部分をボールド体のテキストで示しています。

```
...
<destination id="company">
  <adapter ref="java-adapter"/>
  <properties>
    <source>samples.crm.CompanyAssembler</source>
    <scope>application</scope>

    <metadata>
      <identity property="companyId"/>
      <one-to-many property="employees" destination="employee"/>
    </metadata>
...
      <server>
        <fill-method>
          <name>loadCompanies</name>
        </fill-method>
        <sync-method>
          <name>syncCompanies</name>
```

```
          </sync-method>
...
  </properties>
</destination>
...
<destination id="employee">
  <adapter ref="java-adapter"/>
  <properties>
       <source>samples.crm.EmployeeAssembler</source>
       <scope>application</scope>

    <metadata>
      <identity property="employeeId"/>
    </metadata>

    <server>
      <fill-method>
        <name>loadEmployees</name>
      </fill-method>

      <sync-method>
        <name>syncEmployees</name>
      </sync-method>
...
    </server>
  </properties>
</destination>
```

ActionScript コード

次の例は、ArrayCollection に Company オブジェクトを設定する、クライアントサイドアプリケーションの ActionScript コードの一部です。Company という名前の ActionScript クラスには employees プロパティがあります。このプロパティは Employee オブジェクトの ArrayCollection です (例の `var company` を参照)。

Company および Employee ActionScript オブジェクトは、サーバーサイドの対応する Company および Employee Java オブジェクトにマップされます。Employee ActionScript クラスでは、クラス定義の上に `[Managed]` メタデータタグが含まれており、管理されたクラスであることを示しています。`conflictHandler()` メソッドでは、データストアでの競合イベントをリッスンすることにより、会社宛先と従業員宛先の 2 つの宛先間の 1 対多関係のために両方の宛先から生じるデータ同期の競合を処理します。会社宛先と従業員宛先との間には関連付けがあるため、デフォルトではこれらは同じ dataStore プロパティを共有します。競合と、確定解除された変更を、データストアに保持することで、関連付けがある場合のデータの整合性が確保されます。

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
  creationComplete="initApp()">
  <mx:Script>
```

```
    <![CDATA[
...
    [Bindable]
    public var companies:ArrayCollection;
    [Bindable]
    public var company:Company;
    private var dsCompany:DataService;

    private function initApp() {
      companies = new ArrayCollection();
      dsCompany = new DataService("company");
...
      dsCompany.dataStore.addEventListener(DataConflictEvent.CONFLICT,
        conflictHandler);
      dsCompany.autoCommit = false;
...
    }
...
    private function conflictHandler(event:DataConflictEvent):void {
      Hourglass.remove();
      var conflicts:Conflicts = dsCompany.dataStore.conflicts;
      var c:Conflict;
      for (var i:int=0; i<conflicts.length; i++) {
        c = Conflict(conflicts.getItemAt(i));
        Alert.show("Reverting to server value", "Conflict");
        c.acceptServer();
      }
    }
  </mx:Script>
</mx:Application>
```

MXML コード

アプリケーションが初めてロードされるとき、dsCompany.fill() メソッドによってすべての会社オブジェクトがロードされます。employees プロパティの one-to-many 関連付けエレメントでは lazy 属性が true に設定されていないため、各会社でもその会社のすべての従業員がロードされます。つまり、会社のリスト全体について、および各会社について、従業員のリスト全体が fill 呼び出しの一部として1回の要求で取得されます。

従業員の関連付けに対して lazy 属性の値を true に設定した場合は、従業員の ID だけが会社と一緒にロードされ、個々の従業員のデータは company.employees プロパティのそのエレメントへのアクセスが生じたときに取得されます。効率をよくするため、1人の従業員が要求された場合は従業員のページ全体が同時に取得されます。pageSize をデフォルトの 0 に設定した場合は、最初の従業員が要求されたときに、その ArrayCollection にあるすべての従業員が取得されます。pageSize を 1 に設定した場合には、各従業員が要求時に個別にロードされます。

次の例は、会社の概要データを表示する DataGrid コントロールの MXML コードです。

```
...
<mx:DataGrid id="dg" dataProvider="{companies}"
  width="100%" height="100%" change="companyChange()">
  <mx:columns>
    <mx:DataGridColumn dataField="companyId" headerText="Id"/>
    <mx:DataGridColumn dataField="name" headerText="Company Name"/>
    <mx:DataGridColumn dataField="city" headerText="City"/>
  </mx:columns>
</mx:DataGrid>
...
```

次の例は、会社の詳細を表示する MXML コードです。会社の従業員の編集可能な DataGrid が独自のタブに表示されます。会社宛先と従業員宛先との間の 1 対多関係のため、従業員データに対する変更は従業員宛先に送信されます。

```
...
<mx:TabNavigator width="100%" height="100%">
  <mx:Form width="100%" height="100%" label="General">
    <mx:FormItem label="Company Name" required="true">
      <mx:TextInput id="companyName" width="250" text="{company.name}"/>
   </mx:FormItem>
    <mx:FormItem label="Address">
     <mx:TextInput id="address" width="250" text="{company.address}"/>
    </mx:FormItem>
    <mx:FormItem label="City">
      <mx:TextInput id="city" width="250" text="{company.city}"/>
    </mx:FormItem>
    <mx:FormItem label="State">
    <mx:TextInput id="state" width="250" text="{company.state}"/>
    </mx:FormItem>
    <mx:FormItem label="Zip">
      <mx:TextInput id="zip" width="250" text="{company.zip}"/>
    </mx:FormItem>
  </mx:Form>

  <mx:VBox label="Employees" width="100%" height="100%">
    <mx:HBox>
      <mx:Button label="+" click="addEmployee()"/>
      <mx:Button Label="-"
        click="company.employees.removeItemAt
        (dgEmployees.selectedIndex)"/>
    </mx:HBox>
    <mx:DataGrid id="dgEmployees" dataProvider="{company.employees}"
      width="100%" height="100%" editable="true">
      <mx:columns>
        <mx:DataGridColumn dataField="firstName"
          headerText="First Name"/>
        <mx:DataGridColumn dataField="lastName" headerText="Last Name"/>
        <mx:DataGridColumn dataField="title" headerText="Title"/>
```

```
            <mx:DataGridColumn dataField="email" headerText="Email"/>
            <mx:DataGridColumn dataField="phone" headerText="Phone"/>
        </mx:columns>
    </mx:DataGrid>
  </mx:VBox>
</mx:TabNavigator>
```

アセンブラおよび DAO コード

アセンブラを実装する場合は、関係内のデータを返す方法として 2 つの選択肢があります。遅延関連付けを使用する場合は、ID のみが設定された子オブジェクトのインスタンスを親アセンブラから返すことができます。遅延関連付けを使用しない場合は、完全に設定されたインスタンスをアセンブラから返す必要があります。次の例は、完全に設定された従業員のコレクションを返す getCompany() メソッドの一例を示しています。この例では company_employee という結合テーブルを使用しています。この結合テーブルでは、会社が従業員のプロパティを所有し、従業員には会社プロパティはありません。

より従来に近いスキーマでは、従業員リストが大きいという理由から、company_id を従業員テーブルに格納し、fill メソッドによって従業員リストを取得することが考えられます。このような方法でスキーマを実装した場合は、従業員宛先で会社との関連付けが設定され、company_id 列の値を保存する役割を担うことになります。

双方向の関連付けを設定することもできます。この場合、クライアントコードでは関係の両側を更新する必要があります。たとえば、従業員を company.employees プロパティに追加し、会社を employee.company プロパティに設定します。アセンブラでは、どの宛先が変更を更新する役割を担うのかを示す関連付けプロパティの所有者宛先を選択する必要があります。この場合、通常は company.employees プロパティに対する変更を無視し、read-only 属性を使用して読み取り専用としてマークすることができます。

次の例は、CompanyAssembler クラスの getCompany() メソッドです。

```
  public Company getCompany(Map uid)
  {
    CompanyDAO dao = new CompanyDAO();
    return dao.getCompany(((Integer) uid.get("companyId")).intValue());
  }
```

getCompany() メソッドは、ユーザーが会社概要データの DataGrid 内の行をクリックしたときに、クライアントから呼び出されます。getCompany() メソッドからは CompanyDAO.getCompany() メソッドが呼び出されます。CompanyDAO.getCompany() メソッドでは、会社データベーステーブルからの SQL クエリを実行して会社情報を取得します。そして、company_employee テーブルと従業員テーブルを結合する別のクエリを実行し、その会社の従業員を取得します。次の例は CompanyDAO.getCompany() メソッドを示しています。

```
// CompanyDAO.java
...
  public Company getCompany(int companyId) throws DAOException
```

```
{
   Company company = null;
   Connection c = null;
   try
   {
      c = ConnectionHelper.getConnection();
      Statement s = c.createStatement();
      ResultSet rs = s.executeQuery("SELECT * FROM company WHERE
         company_id=" + companyId);
      if (rs.next())
      {
         company = new Company();
         company.setCompanyId(rs.getInt("company_id"));
         company.setName(rs.getString("name"));
         company.setAddress(rs.getString("address"));
         company.setCity(rs.getString("city"));
         company.setZip(rs.getString("zip"));
         company.setState(rs.getString("state"));
         List employees = new ArrayList();
         rs = s.executeQuery("SELECT a.employee_id, first_name,
            last_name, title, email, phone FROM company_employee AS a,
            employee AS b WHERE a.employee_id=b.employee_id AND
            company_id=" + companyId);
         while (rs.next())
         {
            employees.add(new Employee(
                 rs.getInt("employee_id"),
                 rs.getString("first_name"),
                 rs.getString("last_name"),
                 rs.getString("title"),
                 rs.getString("email"),
                 rs.getString("phone")));
         }
         company.setEmployees(employees);
      }
   }
```

会社宛先と従業員宛先にはいずれも sync メソッドがあり、それぞれ、会社データまたは従業員データのサーバーサイドでのデータ同期を処理します。これらの sync メソッドは、ネストされた関係を持たない宛先の場合の sync メソッドとまったく同じです。ただし、いずれかの宛先でデータの競合が発生した場合には、コードで同じ競合ハンドラを使用し、宛先間で依存関係がある場合に競合が正しい順序で処理されるようにします。sync メソッドでは変更の種類によって、アセンブラの doCreate()、doUpdate()、または doDelete() メソッドを呼び出します。doCreate() メソッドは、オブジェクトが所有するすべての関連付けについて、関連付けられたすべてのオブジェクトの ID の初期セットを永続化します。doUpdate() メソッドは、同じオブジェクトの古いものと新しいものが与えられた場合に ID のリストを更新します。また、このメソッドは関連付けられた ID での競合を検出できます。doDelete() メソッドは、オブジェクトが所有する関連付けについて、関連付けられたすべてのオブジェクトの ID を削除します。

fill メソッドでのオブジェクト関係の実装

Java アダプタを使用するときは、クエリを使用して 2 つのオブジェクト間の関係を実装できます。クエリでは、アセンブラで定義された fill メソッドのいずれかのパラメータが、関連するオブジェクトの ID になります。たとえば Company オブジェクトが Employee インスタンスを多数持つ場合など、オブジェクトのコレクションが多数ある場合には、クエリによるアプローチのほうが、管理された関連付けによるアプローチよりも効率的になります。

クエリによるアプローチで関係を実装する方法にはいくつかの特徴があり、すべての状況に適しているというわけではありません。ユーザーが従業員の `companyId` を変更した場合、Data Management Service では、従業員のリストを表示するクライアントが自動的に更新されるように、その変更の影響を受ける fill メソッドの結果を更新する必要があります。Data Management Service のデフォルトでは、自動更新機能が使用され、従業員を返す保留中の fill メソッドがすべて再実行されます。これらのクエリが過度に再実行されないようにこの動作を調整することができますが、そのためにはアセンブラにさらにコードを追加する必要があります。自動更新の使用の詳細については、1403 ページの「fill メソッドの結果に対する変更の検出」を参照してください。

また、クエリによるアプローチでは、関連付けを更新するときに競合が検出されません。2 つのクライアントから同じ会社に、ほぼ同時に従業員を追加したという場合には、競合を検出する必要はないと考えられますが、2 つのクライアントから同じ顧客の住所リストを同時に更新したという場合には、競合を検出する必要があると考えられます。管理された関連付けプロパティを使用する場合は、Data Management Service の競合検出メカニズムを利用して、コレクションの完全な内容についてデータ競合を検出することができます。しかし、fill メソッドを使用する場合は、アイテムのプロパティについてしか競合が検出されず、アイテムが返された fill 済みのコレクションについては競合が検出されません。

このセクションでは、Flex データサービスに付属のサンプル Web アプリケーションに含まれている CRM アプリケーションからの抜粋コードを示します。このアプリケーションでは、会社と従業員との間の関係を EmployeeAssembler クラスの `fill()` メソッドで実装しています。アプリケーションの Java アセンブラと DAO クラスのソースコードは、サンプル Web アプリケーションの WEB_INF/src/samples/crm ディレクトリにあります。

次の例は、CRM アプリケーションにある EmployeeAssembler クラスの `fill()` メソッドです。メソッドの中で、クライアントの fill 要求で提供された数値の会社 ID に基づいて、従業員を会社によって検索する部分を、ボールド体のテキストで示しています。

```
...
public Collection fill(List fillParameters)
{
  EmployeeDAO dao = new EmployeeDAO();
    if (fillParameters.size() == 0)
      return dao.getEmployees();

  String queryName = (String) fillParameters.get(0);
  if (queryName.equals("match"))
```

```
    return dao.findMatchingEmployees((String) fillParameters.get(1));
  if (queryName.equals("byCompany"))
    return dao.findEmployeesByCompany((Integer)
      fillParameters.get(1));
  return super.fill(fillParameters); // 適切なエラーをスローします。
}
```

次の例では、従業員を会社によって取得する、対応するクライアントサイドの fill 要求を、ボールド体のテキストで示しています。

```
private function companyChange():void {
  if (dg.selectedIndex > -1)
  {
    if (company != companies.getItemAt(dg.selectedIndex))
    {
      company = Company(companies.getItemAt(dg.selectedIndex));
      dsEmployee.fill(employees, "byCompany",
      company.companyId);
    }
  }
}
```

サーバーからクライアントへのデータ変更のプッシュ

サーバーサイドのデータ変更をクライアントにプッシュするには、flex.data.DataServiceTransaction クラスを使用します。このクラスの詳細については、公開されている『Flex データサービス Javadoc マニュアル』を参照してください。DataServiceTransaction クラスのインスタンスは、Data Management Service によって管理されているオブジェクトの状態を変更する処理ごとに、作成されます。このオブジェクトをサーバーサイドコードから使用し、DataService コンポーネントの `autoSyncEnabled` プロパティが `true` に設定されたクライアントに格納されている、管理されたデータに、変更をプッシュできます。

変更を sync メソッドの中から追加した場合は、Data Management Service によって DataServiceTransaction クラスのインスタンスが作成されます。そのため、このクラスの静的な `getCurrentDataServiceTransaction()` メソッドを呼び出し、その次に `updateItem()`、`deleteItem()`、または `createItem()` メソッドを呼び出して、追加の変更をトリガすることができます。これらのメソッドは、このトランザクションで永続化が完了した変更を適用する場合、または永続化しようとしている変更を適用する場合に、呼び出してください。現在のトランザクションがロールバックされた場合、これらの変更はクライアントにプッシュされません。

トランザクションをロールバックするには、通常の J2EE アプリケーションのように javax.transaction.UserTransaction インスタンスをロールバック済みとしてマークします。または、DataServiceTransaction の `setRollbackOnly()` を呼び出すこともできます。

DataServiceTransaction クラスによって、スレッドローカル状態にある現在のトランザクションにアクセスでき、トランザクションの完了時にクライアントにプッシュされるメッセージが保持されます。また、このクラスを使用して、トランザクションの完了前および完了後の同期イベントを登録します。

DataServiceTransaction インスタンスはそれぞれスレッドローカル状態に格納され、一度に 1 つのスレッドだけで処理されるものと見なされます。このインスタンスはスレッドセーフではありません。

DataServiceTransaction クラスをアセンブラの sync、update、create、または delete メソッドの中から使用する場合は、このトランザクションにおいて既に変更途中のアイテムに対する変更を指定することは避けてください。そのような変更を指定した場合、現在アクティブな変更が変えられるのではなく、そのオブジェクトに対する追加の変更がキューに置かれるため、競合が生じる可能性があります。代わりに、変更済みプロパティ値を使用して NewVersion インスタンスを更新し、新しく変更されたすべてのプロパティ値を、送信された変更済みプロパティのリストに追加してください。たとえば、インスタンスに対するすべての更新が完了した後に、そのインスタンスの `versionId` を変更する必要がある場合は、`versionId` を変更リストに追加でき、newVersion インスタンスで `versionId` の値を更新することもできます。

ChangeObject インスタンスを使用している場合は、`addChangedPropertyName()` メソッドを呼び出して `versionId` プロパティを追加します。`updateItem()` メソッドを使用している場合は、単にそのプロパティを、`updateItem()` メソッドに提供されたリストに追加します。

`DataServiceTransaction.refreshFill()` メソッドを使用すると、アセンブラの sync メソッドの一部として、または他のサーバーサイドコードから、サーバーコードの fill または一致する fill のセットを手動で更新できます。このメソッドでは fill パラメータのリストを指定します。このリストを使用して、アクティブなクライアントによって現在キャッシュされている、一致する fill のリストが作成されます。このリストには null を指定できます。この場合は、その宛先のすべての fill が一致することになります。リストが null ではない場合、`refreshFill()` では、次の表に示す規則に基づいてすべてのスロットが一致した場合に、同じ数のパラメータを持つクライアントによる fill について、比較が実行されます。

値	規則
null 値	そのスロットを無条件で比較します。
クラス値	その種類のそのスロットにあるパラメータを比較します。
上記以外の値	equals メソッドを使用して fill パラメータを比較します。

メモ

Flex データサービス Java API を使用するコードをコンパイルするときは、"messaging.jar" ファイルと "flex-messaging-common.jar" ファイルをクラスパスにインクルードする必要があります。

第 7 部

チャートコンポーネント

7

第VII部では、Adobe Flex 2 でチャートコンポーネントを使用する方法について説明します。

次のトピックが含まれます。

第 53 章

チャートの概要

53

Adobe Flex 2 製品を使用して開発しているアプリケーションでチャートやグラフを使用してデータを表示すると、アプリケーションのユーザーにデータをわかりやすく提示することができます。数値を単純な表形式で表示するだけでなく、棒グラフ、円グラフ、折れ線グラフなどのチャートを、色分けし、キャプションを付けて作成して、データを 2 次元で表示することができます。

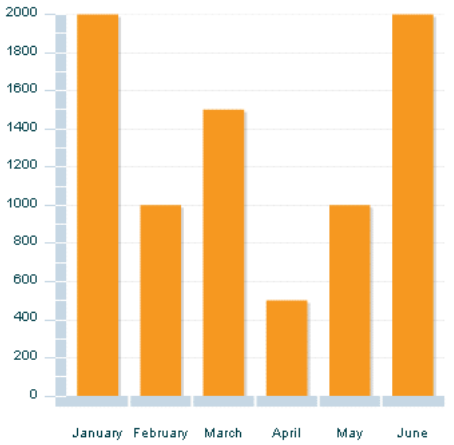
このトピックでは、Adobe Flex チャートコントロールを使用する際の考え方を説明します。

目次

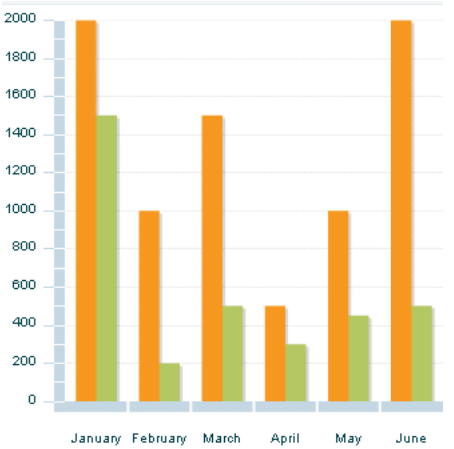
チャートの作成について

データを視覚化することで、データの解釈が容易になり、データの関連性がわかりやすくなります。データを 2 次元で提示する視覚化の方法の 1 つに、チャート (グラフ) があります。Flex では、横棒グラフ、縦棒グラフ、円グラフなどの一般的な 2 次元のチャートがサポートされており、チャートの外観をさまざまに変更することができます。

簡単なチャートの場合、データ系列は 1 つだけです。関連するひとまとまりのデータの値を系列 (データ系列) と呼びます。たとえば、月ごとの売上高や、ホテルの日ごとの客室占有率などがあります。次のチャートは、数か月間の売上高を 1 つのデータ系列で示しています。



データ系列が 2 つのチャートもあります。たとえば、上の 4 つの四半期のチャートに、収益成長率のデータを重ねる場合です。次のチャートでは、売上高と収益の 2 つのデータ系列を示しています。



Flex チャートのインストールの詳細については、Flex チャートインストーラの "readme.txt" ファイルを参照してください。

チャートコントロールの使用

Flex チャートを使用して、数種類の一般的な形式のチャート (グラフ) を作成できます。作成したチャートの外観をカスタマイズすることもできます。チャートコントロールは、mx.charts.* パッケージに含まれています。

次の表は、サポートされるチャートのタイプと、それぞれのチャートに表示するデータの定義に使用するコントロールのクラス名と系列クラス名の一覧です。

チャートタイプ	チャートコントロールクラス	チャート系列クラス
面グラフ	AreaChart	AreaSeries
横棒グラフ	BarChart	BarSeries
バブルチャート	BubbleChart	BubbleSeries
ローソク足チャート	CandlestickChart	CandlestickSeries
縦棒グラフ	ColumnChart	ColumnSeries
HighLowOpenClose チャート	HLOCChart	HLOCSeries
折れ線グラフ	LineChart	LineSeries
円グラフ	PieChart	PieSeries
プロットチャート	PlotChart	PlotSeries

チャートコントロールは、PieChart クラス以外はすべて CartesianChart クラスのサブクラスです。Cartesian チャートは一般的に、長方形の 2 次元領域に一連のデータポイントを表すチャートです。PieChart クラスは PolarChart クラスのサブクラスです。PolarChart クラスは円形の領域にデータを表します。

チャートコントロールはすべて、ChartBase クラスの基本的なチャート特性を継承しています。

MXML でチャートコントロールを記述する場合、通常は次のような構成になります。

```
<mx:ChartName>
  <!-- 系列を定義します。 -->
  <mx:SeriesName/>
  <mx:SecondSeriesName/>

  <!-- 軸を定義します。 -->
  <mx:horizontalAxis>
    <mx:AxisType/>
  </mx:horizontalAxis>
  <mx:verticalAxis>
```

```
      <mx:AxisType/>
   </mx:verticalAxis>

   <!-- 軸と目盛りのスタイルを指定します。 -->
   <mx:horizontalAxisRenderer>
      <mx:AxisRenderer/>
   </mx:horizontalAxisRenderer>
   <mx:verticalAxisRenderer>
      <mx:AxisRenderer/>
   </mx:verticalAxisRenderer/>

   <!-- グリッド線などのエレメントをチャートに追加します。 -->
   <mx:annotationElements>
      <mx:Array/>
   </mx:annotationElements>
   <mx:backgroundElements>
      <mx:Array/>
   </mx:backgroundElements/>
</mx:ChartName>

<!-- オプションで凡例を定義します。 -->
<mx:Legend/>
```

次の表で、チャートの各部分を詳細に説明します。

各部のタイトル	説明
チャート	(必須) チャートのデータプロバイダを 1 つまたは 2 つ定義します。チャートのタイプも定義し、データヒント、マウス感度、ふち取りのスタイル、および軸のスタイルを設定します。 チャートコントロールではトップレベルのタグです。他のタグはすべて、このタグの子タグです。
系列	(必須) チャートに表示するデータ系列を 1 つまたは複数定義します。線、塗り、データ系列のレンダラー (スキン) を設定し、各系列のチャート凡例で使用される線と塗りも設定します。 それぞれのチャートに 2 番目の系列を定義して、1 つのチャートに複数のデータ系列を表示することもできます。 チャートの各系列には、独自のデータプロバイダを設定できます。
軸	軸のタイプ (数値またはカテゴリ) を設定します。軸のラベル、タイトル、追加スペースなどのスタイルプロパティも定義します。 2 番目の系列セットがある場合は、その系列の軸を定義することもできます。
軸レンダラー	(オプション) 目盛りの配置とスタイルの設定、ラベルの有効 / 無効の切り替え、軸線、ラベルの回転、ラベルの隙間の定義を行います。 2 番目の系列がある場合は、その系列の軸レンダラーを定義することもできます。
エレメント	(オプション) チャートに表示されるグリッド線などのエレメントを定義します。

Flex では、チャートタイプごとに、対応するチャートコントロールおよびチャート系列があります。チャートコントロールでは、チャートタイプ、チャートに表示するデータプロバイダ、グリッド線、チャートの軸に表示するテキストなど、チャートタイプに応じたプロパティを定義します。チャートコントロールの dataProvider プロパティは、チャートでどのデータを使用するかを指定します。

" データプロバイダ " とは、オブジェクトのコレクションです。オブジェクトの配列や Collection API を実装するオブジェクトもデータプロバイダになります。また、E4X クエリの結果など、XML ノードを持つ XMLList オブジェクトもデータプロバイダになります。

チャートコンポーネントでは、1 次元配列と同様のフラットな (リストベースの) データプロバイダを使用します。データプロバイダには、ストリング、数値などのオブジェクトや他のオブジェクトを含めることができます。チャートのデータプロバイダの詳細については、1458 ページの「チャートデータの定義」を参照してください。

チャート系列では、チャートにデータプロバイダのどのデータを表示するかを指定します。データプロバイダにはチャートに不要なデータも含まれている場合があるため、チャート系列を使用して、データプロバイダから使用するポイントを指定します。データ系列を 1 つだけ指定することも、2 番目の系列を指定することもできます。また、チャート内のデータの外観も、チャート系列で定義することができます。

明示的に設定されたデータプロバイダがない場合、すべてのチャート系列は、データプロバイダをチャートから継承します。dataProvider プロパティの値をチャートコントロールに設定している場合は、プロパティの値を系列に設定する必要はありません。ただし、チャートのそれぞれの系列に異なるデータプロバイダを定義することもできます。

たとえば、円グラフを作成する場合は、PieChart コントロールと PieSeries チャート系列を使用します。面グラフを作成する場合は、次の例のように、AreaChart コントロールと AreaSeries チャート系列を使用します。

```
<?xml version="1.0"?>
<!-- charts/BasicAreaOneSeries.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Area Chart">
     <mx:AreaChart id="myChart" dataProvider="{expenses}"
     showDataTips="true">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
```

```
                categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:AreaSeries
                yField="Profit"
                displayName="Profit"
            />
        </mx:series>
    </mx:AreaChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

この例では、<mx:AreaSeries> タグを1つだけ含む配列を定義します。<mx:AreaSeries> タグで、面グラフに表示するデータ系列を1つ指定しています。

次の例のように、<mx:AreaSeries> タグをもう1つ追加すると、面グラフに2つのデータ系列を表示できます。

```
<?xml version="1.0"?>
<!-- charts/BasicArea.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Area Chart">
     <mx:AreaChart id="myChart" dataProvider="{expenses}"
     showDataTips="true">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:AreaSeries
                yField="Profit"
                displayName="Profit"
           />
           <mx:AreaSeries
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
```

```
    </mx:AreaChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

チャートコントロールにデータプロバイダを定義する必要はありません。次の例のように、それぞれの系列には独自のデータプロバイダを設定できます。

```
<?xml version="1.0"?>
<!-- charts/MultipleDataProviders.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var profit04:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000},
        {Month:"Feb", Profit:1000},
        {Month:"Mar", Profit:1500}
     ]);
     [Bindable]
     public var profit05:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2200},
        {Month:"Feb", Profit:1200},
        {Month:"Mar", Profit:1700}
     ]);
     [Bindable]
     public var profit06:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2400},
        {Month:"Feb", Profit:1400},
        {Month:"Mar", Profit:1900}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{profit04}">
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="Month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                dataProvider="{profit04}"
                yField="Profit"
                xField="Month"
                displayName="2004"
           />
           <mx:ColumnSeries
                dataProvider="{profit05}"
                yField="Profit"
                xField="Month"
                displayName="2005"
           />
           <mx:ColumnSeries
```

```
                dataProvider="{profit06}"
                yField="Profit"
                xField="Month"
                displayName="2006"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

チャートに 2 番目のデータプロバイダ、系列、軸、または軸レンダラーを追加するには、secondDataProvider プロパティ、secondSeries プロパティ、secondVerticalAxis/secondHorizontalAxis プロパティ、または secondVerticalAxisRenderer/secondHorizontalAxisRenderer プロパティを使用します。詳細については、1524 ページの「複数のデータ系列の使用」および 1526 ページの「複数の軸の使用」を参照してください。

ブラウザウィンドウのサイズに合わせてチャートのサイズが動的に変更されるようにするには、次の例のように width と height の属性をパーセントで指定します。

```
<?xml version="1.0"?>
<!-- charts/BasicBarSize.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Bar Chart" height="1000" width="1000">
     <mx:BarChart id="myChart"
        dataProvider="{expenses}"
        height="100%"
        width="100%"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                yField="Month"
                xField="Profit"
                displayName="Profit"
           />
```

```
            <mx:BarSeries
                 yField="Month"
                 xField="Expenses"
                 displayName="Expenses"
            />
        </mx:series>
    </mx:BarChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

チャートコントロールの親コンテナもパーセントで指定する必要があります。パーセント値で指定されていない場合、ウィンドウサイズが変更されてもチャートのサイズは変更されません。

軸について

チャートのデータは、チャートのデータ系列に指定されているフィールドで定義されます。" 軸ラベル " の概観と内容は、x 軸については <mx:horizontalAxis> タグで、y 軸については <mx:verticalAxis> タグで、これらのタグのレンダラー (<mx:horizontalAxisRenderer> および <mx:verticalAxisRenderer>) と共に定義します。これらのタグでは、チャートに表示されるデータ範囲を定義するだけではなく、データポイントと、名前やラベルのマッピングも行います。ラベルのマッピングは、Data Management Service チャートでの DataTip のラベルの値、軸ラベル、および目盛りのレンダリング方法に大きく影響します。

Flex チャートでは、次のタイプの軸を使用できます。

CategoryAxis　CategoryAxis クラスは、株の銘柄コードや、国名、人口統計のカテゴリのような、連続していない値の集合を軸にマッピングする場合に使用します。<mx:CategoryAxis> タグを使用して、論理的な関連性でグループ化した軸ラベルを定義します。数値である必要はありません。たとえば、1444 ページの「チャートの作成について」のチャートで使用されている月名を CategoryAxis クラスとして定義できます。

LinearAxis　LinearAxis クラスは、数値データをマッピングする軸に使用します。<mx:horizontalAxis> タグまたは <mx:verticalAxis> タグの子タグ <mx:LinearAxis> で、軸に表示する値の範囲や、目盛りの軸ラベル間の幅を指定します。

LogAxis　LogAxis クラスは、数値データを対数的にマッピングする軸に使用します。<mx:horizontalAxis> タグまたは <mx:verticalAxis> タグの <mx:LogAxis> 子タグを使用します。対数軸のラベルは 10 の偶数乗です。

DateTimeAxis　DateTimeAxis クラスは、時間、日、週、年などの時間ベースの値をマッピングする軸に使用します。<mx:DateTimeAxis> タグを使用して、軸ラベルを定義します。

DateTimeAxis、LogAxis、および LinearAxis は数値の表示に使用されるので、すべて NumericAxis タイプです。多くの場合、一方の軸だけを NumericAxis または CategoryAxis として定義すれば十分です。Flex では、明示的にタイプを定義されていない軸はすべて LinearAxis と見なされます。ただし、DataTip のラベルや凡例などの装飾を使用する場合は、両方の軸を明示的に定義することが必要となることがあります。

PlotChart コントロールの場合は、2 つの座標の交点がデータポイントになるため、両方の軸が LinearAxis として扱われます。このため、どちらの軸も指定する必要はありません。ただし、最小値や最大値などの設定を追加するために、軸を指定することもできます。

それぞれの軸には対応する AxisRenderer オブジェクト (`horizontalAxisRenderer` プロパティまたは `verticalAxisRenderer` プロパティで指定) があり、これで軸ラベルと目盛りの外観を定義します。フォーマットの定義だけでなく、AxisRenderer クラスを使用して軸ラベルの値をカスタマイズすることもできます。詳細については、1533 ページ、第 55 章の「チャートの書式設定」を参照してください。

Flex のデフォルトでは、チャートのタイプと方向を基にして、チャートの x 軸および y 軸に沿って表示するラベルが計算されます。たとえば、縦棒グラフには次のデフォルトの値があります。

X 軸　最小値は 0 で、最大値はチャート内のデータ系列の項目数です。

Y 軸　Y 軸の最小値は、チャートのデータに対して十分小さな値となるように、最大値は、チャートのデータに対して十分大きな値となるように計算されます。

チャートの軸の詳細については、1549 ページの「軸の操作」を参照してください。

チャートのイベントについて

チャートコントロールには、チャート内のデータポイントへのユーザーの操作に対応するための新しいイベントが導入されています。それらのイベントについては 1643 ページ、第 56 章の「チャートにおけるイベントと エフェクトの使用」で説明します。

ActionScript でのチャートの作成

ActionScript でも、他の Flex コンポーネントと同じようにチャートを作成、破棄、および操作できます。

スクリプトブロックや別の ActionScript クラスファイルで操作する場合は、適切なクラスをすべて読み込む必要があります。最も一般的な例は、次の import ステートメントで定義されます。

```
import mx.collections.*;
import mx.charts.*;
import mx.charts.series.*;
import mx.charts.renderers.*;
import mx.charts.events.*;
```

ActionScript でチャートを作成するには、new キーワードを使用します。MXML での場合と同じようにチャートオブジェクトにプロパティを設定できます。dataProvider プロパティにデータプロバイダを割り当てます。チャートにデータ系列を追加するには、適切なタイプの新しいデータ系列を定義します。チャートに系列を適用するには、チャートの series プロパティを使用します。CategoryAxis クラスを使用して、カテゴリを軸に設定できます。次の例では、系列を 2 つ含む BarChart コントロールを定義します。

```
<?xml version="1.0"?>
<!-- charts/CreateChartInActionScript.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.BarChart;
     import mx.charts.series.BarSeries;
     import mx.charts.CategoryAxis;
     import mx.charts.Legend;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);

     public var myChart:BarChart;
     public var series1:BarSeries;
     public var series2:BarSeries;
     public var legend1:Legend;

     public function init():void {
        // Create the chart object and set some
        // basic properties.
        myChart = new BarChart();
        myChart.showDataTips = true;
        myChart.dataProvider = expenses;

        // Define the category axis.
        var vAxis:CategoryAxis = new CategoryAxis();
        vAxis.categoryField = "Month" ;
        vAxis.dataProvider =  expenses;
        myChart.verticalAxis = vAxis;

        // Add the series.
        var mySeries:Array=new Array();
        series1 = new BarSeries();
        series1.xField="Profit";
        series1.yField="Month";
        series1.displayName = "Profit";
```

```
        mySeries.push(series1);

        series2 = new BarSeries();
        series2.xField="Expenses";
        series2.yField="Month";
        series2.displayName = "Expenses";
        mySeries.push(series2);

        myChart.series = mySeries;

        // Create a legend.
        legend1 = new Legend();
        legend1.dataProvider = myChart;

        // Attach chart and legend to the display list.
        p1.addChild(myChart);
        p1.addChild(legend1);
    }
  ]]></mx:Script>
  <mx:Panel id="p1" title="Bar Chart Created in ActionScript"/>
</mx:Application>
```

この例では、既存の系列の配列を新しい配列で置き換えます。

同様の方法で、既存の配列を置き換えるのではなく、チャートにデータ系列を追加することができます。次の例では、2 つの ColumnSeries を作成し、それらのデータプロバイダを設定します。次に既存のチャート系列を保持する配列を作成し、その配列に新しい系列を読み込みます。最後に、チャートの series プロパティの値がその系列の新しい配列になるように設定します。

```
<?xml version="1.0"?>
<!-- charts/AddingSeries.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.series.ColumnSeries;

     [Bindable]
     public var profit04:ArrayCollection = new ArrayCollection([
        {Month: "Jan", Profit: 2000},
        {Month: "Feb", Profit: 1000},
        {Month: "Mar", Profit: 1500}
     ]);
     [Bindable]
     public var profit05:ArrayCollection = new ArrayCollection([
        {Month: "Jan", Profit: 2200},
        {Month: "Feb", Profit: 1200},
        {Month: "Mar", Profit: 1700}
     ]);
     [Bindable]
     public var profit06:ArrayCollection = new ArrayCollection([
        {Month: "Jan", Profit: 2400},
```

```
        {Month: "Feb", Profit: 1400},
        {Month: "Mar", Profit: 1900}
    ]);

    public var series1:ColumnSeries;
    public var series2:ColumnSeries;

    public function addMoreSeries():void {
        if (!series1 || !series2) {
            series1 = new ColumnSeries();
            series1.dataProvider = profit05;
            series1.yField = "Profit";
            series1.xField = "Month";
            series1.displayName = "2005";

            series2 = new ColumnSeries();
            series2.dataProvider = profit06;
            series2.yField = "Profit";
            series2.xField = "Month";
            series2.displayName = "2006";

            var currentSeries:Array = myChart.series;

            currentSeries.push(series1);
            currentSeries.push(series2);

            myChart.series = currentSeries;
        }
    }
  ]]></mx:Script>
  <mx:Panel title="Column Chart">
    <mx:ColumnChart id="myChart" dataProvider="{profit04}">
      <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="Month"/>
      </mx:horizontalAxis>
      <mx:series>
        <mx:ColumnSeries dataProvider="{profit04}"
            yField="Profit"
            xField="Month"
            displayName="2004"
        />
      </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
  <mx:Button id="b1" label="Add More Series To Chart" click=
  "addMoreSeries()"/>
</mx:Application>
```

ActionScript を使用すると、チャートの系列の変数を定義することもできます。次の例は、E4X シンタックスを使用して、データから一意の名前の配列を抽出します。次に、この配列を反復処理して、それぞれの名前用の新しい LineSeries を作成します。

```
<?xml version="1.0"?>
<!-- charts/VariableSeries.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp();">
  <mx:Script><![CDATA[
        import mx.charts.series.LineSeries;
        import mx.charts.DateTimeAxis;

        [Bindable]
        private var myXML:XML =
            <dataset>
            <item>
                <who>Tom</who>
                <when>08/22/2006</when>
                <hours>5.5</hours>
            </item>
            <item>
                <who>Tom</who>
                <when>08/23/2006</when>
                <hours>6</hours>
            </item>
            <item>
                <who>Tom</who>
                <when>08/24/2006</when>
                <hours>4.75</hours>
            </item>
            <item>
                <who>Dick</who>
                <when>08/22/2006</when>
                <hours>6</hours>
            </item>
            <item>
                <who>Dick</who>
                <when>08/23/2006</when>
                <hours>8</hours>
            </item>
            <item>
                <who>Dick</who>
                <when>08/24/2006</when>
                <hours>7.25</hours>
            </item>
            <item>
                <who>Jane</who>
                <when>08/22/2006</when>
                <hours>6.5</hours>
            </item>
```

```
    <item>
        <who>Jane</who>
        <when>08/23/2006</when>
        <hours>9</hours>
    </item>
    <item>
        <who>Jane</who>
        <when>08/24/2006</when>
        <hours>3.75</hours>
    </item>
    </dataset>;

public function initApp():void {
    var wholist:Array = new Array();
    for each(var property:XML in myXML.item.who) {
        // Create an Array of unique names.
        if (wholist[property] != property)
            wholist[property] = property;
    }

    // Iterate over names and create a new series
    // for each one.
    for (var s:String in wholist) {
        // Use all items whose name matches s.
        var localXML:XMLList = myXML.item.(who==s);

        // Create the new series and set its properties.
        var localSeries:LineSeries = new LineSeries();
        localSeries.dataProvider = localXML;
        localSeries.yField = "hours";
        localSeries.xField = "when";

        // Set values that show up in dataTips and Legend.
        localSeries.displayName = s;

        // Back up the current series on the chart.
        var currentSeries:Array = myChart.series;
        // Add the new series to the current Array of series.
        currentSeries.push(localSeries);
        // Add the new Array of series to the chart.
        myChart.series = currentSeries;
    }

    // Create a DateTimeAxis horizontal axis.
    var hAxis:DateTimeAxis = new DateTimeAxis();
    hAxis.dataUnits = "days";
    // Set this to false to display the leftmost label.
    hAxis.alignLabelsToUnits = false;
    // Take the date in its current format and create a Date
    // object from it.
```

```
            hAxis.parseFunction = createDate;
            myChart.horizontalAxis = hAxis;
        }

        public function createDate(s:String):Date {
            // Reformat the date input to create Date objects
            // for the axis.
            var a:Array = s.split("/");

            // The existing String s is in the format "MM/DD/YYYY".
            // To create a Data object, you pass "YYYY,MM,DD",
            // where MM is zero-based, to the Date() constructor.
            var newDate:Date = new Date(a[2],a[0]-1,a[1]);
            return newDate;
        }
    ]]></mx:Script>

  <mx:Panel title="Line Chart with Variable Number of Series">
     <mx:LineChart id="myChart" showDataTips="true"/>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

チャートデータの定義

すべてのチャートコントロールで、チャートのデータを定義する dataProvider プロパティが使用されます。データプロバイダでは、Flex コンポーネントとそのコンポーネントの生成に使用するデータのと間に抽象レベルが作成されます。データプロバイダを使用するすべてのチャートで変更が反映されるように、同じデータプロバイダから複数のチャートを生成し、実行時にチャート用のデータプロバイダを切り替えることや、データプロバイダを修正することができます。

チャートデータの使用

チャートコントロールでデータプロバイダのデータを使用するには、チャート系列の xField プロパティと yField プロパティにデータプロバイダのフィールドを指定します。xField プロパティでは水平軸のデータを定義し、yField プロパティでは垂直軸のデータを定義します。

たとえば、データプロバイダに次の構造が含まれるとします。

```
{Month: "Feb", Profit: 1000, Expenses: 200, Amount: 60}
```

この構造から Profit フィールドと Expenses フィールドを使用し、Month フィールドを無視する場合は、次の例のように、系列オブジェクトの xField プロパティに1つのフィールドを指定し、yField プロパティにもう1つのフィールドを指定します。

```
<mx:PlotSeries xField="Profit" yField="Expenses"/>
```

この結果として得られる各データポイントは、データプロバイダの Profit フィールドと Expenses フィールドの交点です。

データポイントを目的に合わせてグループ化するために、データプロバイダのプロパティのうち 1 つを categoryField として選択できます。この例でデータポイントを月別に並び替えるには、次のように、Month フィールドを水平軸の categoryField プロパティに割り当てます。

```
<mx:horizontalAxis>
  <mx:CategoryAxis dataProvider="{expenses}" categoryField="Month"/>
</mx:horizontalAxis>
```

場合によっては、チャートのタイプや表示するデータ型に応じて、xField プロパティまたは yField プロパティを使用してデータ系列を定義します。たとえば、ColumnChart コントロールでは、yField プロパティを使用して縦棒の高さを定義します。xField プロパティを指定する必要はありません。個々の縦棒に対する軸ラベルが必要な場合は、horizontalAxis の categoryField プロパティを指定します。

チャートデータを使用する場合、次の事項に注意します。

- 系列を表示する必要がある場合、通常は系列とデータプロバイダフィールドを一致させます。ただし例外もあります。ColumnSeries の xField を指定しないと、インデックスが値であると見なされます。yField を指定しないと、データプロバイダは、y 値を持つオブジェクトのコレクションではなく、y 値のコレクションであると見なされます。たとえば、次の系列は ColumnChart コントロールで正しくレンダリングされます。

  ```
  <mx:ColumnSeries dataProvider="{[1,2,3,4,5]}"/>
  ```

- 系列によっては、データプロバイダのフィールドを 1 つのみ使用するものと、複数使用できるものがあります。たとえば、PieSeries オブジェクトの場合は field プロパティのみを指定しますが、PlotSeries オブジェクトの場合は xField および yField、BubbleSeries オブジェクト場合は xField、yField、および radiusField を指定できます。
- ほとんどの系列では、主要な次元以外についてフィールドが指定されていない場合は適切なデフォルト値が選択されます。たとえば、ColumnSeries 系列、LineSeries 系列、および AreaSeries 系列に対して xField を明示的に設定しない場合、データプロバイダに格納されている順に、データがチャートのカテゴリに割り当てられます。BarSeries でも同様に、yField を設定しない場合はデータがカテゴリに割り当てられます。

各データ系列で使用できるすべてのフィールドの一覧については、『Adobe Flex 2 リファレンスガイド』のデータ系列の項目を参照してください。データプロバイダの詳細については、244 ページの「データプロバイダコントロールの使用」を参照してください。

チャートデータの種類

次の方法でデータプロバイダにデータを指定できます。

- <mx:Script> ブロックで定義する。
- 外部 XML、ActionScript、またはテキストファイルで定義する。
- WebService 呼び出しを使用して返す。
- RemoteObject コンポーネントを使用して返す。
- HTTPService コンポーネントを使用して返す。
- MXML で定義する。

データプロバイダの詳細については、151 ページ、第 7 章の「データプロバイダおよびコレクションの使用」を参照してください。

データプロバイダとしての静的配列の使用

データプロバイダとしてオブジェクトの静的配列を使用するのは、非常に簡単な方法です。通常、次の例のように、オブジェクトの配列を作成します。

```
<?xml version="1.0"?>
<!-- charts/ArrayOfObjectsDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     private var expenses:Array = [
        {Month:"January",Profit:2000,Expenses:1500,Amount:450},
        {Month:"February",Profit:1000,Expenses:200,Amount:600},
        {Month:"March",Profit:1500,Expenses:500,Amount:300},
        {Month:"April",Profit:500,Expenses:300,Amount:500},
        {Month:"May",Profit:1000,Expenses:450,Amount:250},
        {Month:"June",Profit:2000,Expenses:500,Amount:700}
     ];

  ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
```

```
            />
            <mx:ColumnSeries
                 xField="Month"
                 yField="Expenses"
                 displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

次の例のように、MXML を使用して配列のコンテンツを定義することもできます。

```
<?xml version="1.0"?>
<!-- charts/ArrayOfMXMLObjectsDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Array id="expenses">
    <mx:Object
       Month="January"
       Profit="2000"
       Expenses="1500"
       Amount="450"
    />
    <mx:Object
       Month="February"
       Profit="1000"
       Expenses="200"
       Amount="600"
    />
    <mx:Object
       Month="March"
       Profit="1500"
       Expenses="500"
       Amount="300"
    />
    <mx:Object
       Month="April"
       Profit="500"
       Expenses="300"
       Amount="500"
    />
    <mx:Object
       Month="May"
       Profit="1000"
       Expenses="450"
       Amount="250"
    />
    <mx:Object
       Month="June"
       Profit="2000"
```

```
        Expenses="500"
        Amount="700"
     />
  </mx:Array>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

また、次の例に示すように、MXML でさらに多くのシンタックスを使用してオブジェクトを定義することもできます。

```
<?xml version="1.0"?>
<!-- charts/ArrayOfVerboseMXMLObjects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Array id="expenses">
     <mx:Object>
       <mx:Month>January</mx:Month>
       <mx:Profit>2000</mx:Profit>
       <mx:Expenses>1500</mx:Expenses>
       <mx:Amount>450</mx:Amount>
     </mx:Object>
     <mx:Object>
       <mx:Month>February</mx:Month>
       <mx:Profit>1000</mx:Profit>
       <mx:Expenses>200</mx:Expenses>
       <mx:Amount>600</mx:Amount>
     </mx:Object>
     <mx:Object>
       <mx:Month>March</mx:Month>
```

```
      <mx:Profit>1500</mx:Profit>
      <mx:Expenses>500</mx:Expenses>
      <mx:Amount>300</mx:Amount>
    </mx:Object>
    <mx:Object>
      <mx:Month>April</mx:Month>
      <mx:Profit>500</mx:Profit>
      <mx:Expenses>300</mx:Expenses>
      <mx:Amount>300</mx:Amount>
    </mx:Object>
    <mx:Object>
      <mx:Month>May</mx:Month>
      <mx:Profit>1000</mx:Profit>
      <mx:Expenses>450</mx:Expenses>
      <mx:Amount>250</mx:Amount>
    </mx:Object>
    <mx:Object>
      <mx:Month>June</mx:Month>
      <mx:Profit>2000</mx:Profit>
      <mx:Expenses>500</mx:Expenses>
      <mx:Amount>700</mx:Amount>
    </mx:Object>
  </mx:Array>

  <mx:Panel title="Column Chart">
    <mx:ColumnChart id="myChart" dataProvider="{expenses}">
      <mx:horizontalAxis>
        <mx:CategoryAxis
            dataProvider="{expenses}"
            categoryField="Month"
        />
      </mx:horizontalAxis>
      <mx:series>
        <mx:ColumnSeries
            xField="Month"
            yField="Profit"
            displayName="Profit"
        />
        <mx:ColumnSeries
            xField="Month"
            yField="Expenses"
            displayName="Expenses"
        />
      </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

チャートのデータプロバイダとして単純な配列を使用することの短所は、データを操作するのに Array クラスのメソッドしか使用できないことです。さらに、データプロバイダとして配列を使用する場合、その中のデータは静的にする必要があります。配列をバインド可能にしても、配列のデータの変更はチャートに反映されません。より堅牢なデータ操作とデータバインディングを行うには、チャートのデータプロバイダにコレクションを使用します。1464 ページの「データプロバイダとしてのコレクションの使用」を参照してください。

データプロバイダとしてのコレクションの使用

コレクションは配列よりも堅牢なデータプロバイダとなります。オブジェクトの挿入や削除だけでなく、ソートやフィルタリングの操作も可能です。コレクションは変更通知もサポートします。ArrayCollection オブジェクトを使用すると、ICollectionView または IList インターフェイスとして配列を簡単に公開できます。

次の例のように、配列と同じ方法で MXML を使用してコレクションのコンテンツを定義できます。

```
<?xml version="1.0"?>
<!-- charts/ArrayCollectionOfMXMLObjectsDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ArrayCollection id="expenses">
     <mx:Object
        Month="January"
        Profit="2000"
        Expenses="1500"
        Amount="450"
     />
     <mx:Object
        Month="February"
        Profit="1000"
        Expenses="200"
        Amount="600"
     />
     <mx:Object
        Month="March"
        Profit="1500"
        Expenses="500"
        Amount="300"
     />
     <mx:Object
        Month="April"
        Profit="500"
        Expenses="300"
        Amount="500"
     />
     <mx:Object
        Month="May"
        Profit="1000"
        Expenses="450"
```

```
        Amount="250"
     />
     <mx:Object
        Month="June"
        Profit="2000"
        Expenses="500"
        Amount="700"
     />
  </mx:ArrayCollection>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

また、MXML で複雑なオブジェクトを定義することもできます。

```
<?xml version="1.0"?>
<!-- charts/ArrayOfVerboseMXMLObjects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:ArrayCollection id="expenses">
     <mx:Object>
       <mx:Month>January</mx:Month>
       <mx:Profit>2000</mx:Profit>
       <mx:Expenses>1500</mx:Expenses>
       <mx:Amount>450</mx:Amount>
     </mx:Object>
     <mx:Object>
       <mx:Month>February</mx:Month>
       <mx:Profit>1000</mx:Profit>
       <mx:Expenses>200</mx:Expenses>
```

```
      <mx:Amount>600</mx:Amount>
    </mx:Object>
    <mx:Object>
      <mx:Month>March</mx:Month>
      <mx:Profit>1500</mx:Profit>
      <mx:Expenses>500</mx:Expenses>
      <mx:Amount>300</mx:Amount>
    </mx:Object>
    <mx:Object>
      <mx:Month>April</mx:Month>
      <mx:Profit>500</mx:Profit>
      <mx:Expenses>300</mx:Expenses>
      <mx:Amount>300</mx:Amount>
    </mx:Object>
    <mx:Object>
      <mx:Month>May</mx:Month>
      <mx:Profit>1000</mx:Profit>
      <mx:Expenses>450</mx:Expenses>
      <mx:Amount>250</mx:Amount>
    </mx:Object>
    <mx:Object>
      <mx:Month>June</mx:Month>
      <mx:Profit>2000</mx:Profit>
      <mx:Expenses>500</mx:Expenses>
      <mx:Amount>700</mx:Amount>
    </mx:Object>
  </mx:ArrayCollection>

  <mx:Panel title="Column Chart">
    <mx:ColumnChart id="myChart" dataProvider="{expenses}">
       <mx:horizontalAxis>
          <mx:CategoryAxis
               dataProvider="{expenses}"
               categoryField="Month"
          />
       </mx:horizontalAxis>
       <mx:series>
          <mx:ColumnSeries
               xField="Month"
               yField="Profit"
               displayName="Profit"
          />
          <mx:ColumnSeries
               xField="Month"
               yField="Expenses"
               displayName="Expenses"
          />
       </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
```

```
  </mx:Panel>
</mx:Application>
```

ActionScript では ArrayCollection オブジェクトを作成できます。この方法で ArrayCollection を定義する場合は、次の例のように、mx.collections.ArrayCollection クラスを読み込む必要があります。

```
<?xml version="1.0"?>
<!-- charts/ArrayCollectionOfObjects.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     private var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ]);

  ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

データを配列に格納している場合は、配列を ArrayCollection のコンストラクタに渡すことによって、ArrayCollection に変換することができます。次の例では、配列を作成し、それを ArrayCollection に変換しています。

```
<?xml version="1.0"?>
<!-- charts/ArrayConvertedToArrayCollection.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     private var expenses:Array = [
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ];

     [Bindable]
     public var expensesAC:ArrayCollection =
        new ArrayCollection(expenses);

  ]]></mx:Script>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expensesAC}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expensesAC}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

同様に、<mx:ArrayCollection> タグを使用して変換することもできます。

```
<?xml version="1.0"?>
<!-- charts/ArrayConvertedToArrayCollectionMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     private var expenses:Array = [
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ];
  ]]></mx:Script>

  <mx:ArrayCollection id="expensesAC" source="{expenses}"/>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expensesAC}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expensesAC}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

ArrayCollections のデータはチャートのデータプロバイダにバインドできるので、リアルタイムで更新できます。次の例では、経過時間と1秒ごとの総メモリ使用量を保持するオブジェクトを作成します。次に、折れ線グラフのデータプロバイダとして使用する ArrayCollection に、作成した新しいオブジェクトをプッシュします。結果として、チャート自体が1秒ごとに更新され、時間経過に伴う Flash Player のメモリ使用量を表示します。

```
<?xml version="1.0"?>
<!-- charts/RealTimeArrayCollection.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize=
"initTimer()">
  <mx:Script><![CDATA[
  import flash.utils.Timer;
  import flash.events.TimerEvent;
  import mx.collections.ArrayCollection;

  [Bindable]
  public var memoryUsage:ArrayCollection = new ArrayCollection();

  public function initTimer():void {
     // The first parameter in the Timer constructor
     // is the interval, in milliseconds.
     // The second parameter is how many times to run (0 is
     // infinity).
     var myTimer:Timer = new Timer(1000, 0);

     // Add the listener for the timer event.
     myTimer.addEventListener("timer", timerHandler);
     myTimer.start();
  }

  public function timerHandler(event:TimerEvent):void {
     var o:Object = new Object();

     // Get the number of milliseconds since Flash Player started.
     o.time = getTimer();

     // Get the total memory Flash Player is using.
     o.memory = flash.system.System.totalMemory;
     trace(o.time + ":" + o.memory);

     // Add new object to the ArrayCollection, which is bound
     // to the chart's data provider.
     memoryUsage.addItem(o);
  }
  ]]></mx:Script>

  <mx:LineChart id="chart" dataProvider="{memoryUsage}"
     showDataTips="true">
     <mx:horizontalAxis>
```

```
        <mx:LinearAxis/>
      </mx:horizontalAxis>
      <mx:verticalAxis>
         <mx:LinearAxis minimum="5000000"/>
      </mx:verticalAxis>
      <mx:series>
         <mx:LineSeries yField="memory"/>
      </mx:series>
   </mx:LineChart>
</mx:Application>
```

データコレクションでは、ページング、つまり、アプリケーション要求に応じてデータをまとめてクライアントに送信することが可能です。しかしデフォルトでは、Flex チャートコンポーネントには常時すべてのデータが表示されます。このため、データコレクションをチャートに使用する場合は、ページング機能を無効にするか、チャートデータとして使用するデータコレクションをページングビューで表示しないようにしてください。コレクションの使用の詳細については、157 ページの「コレクションについて」を参照してください。

データプロバイダとしての XML ファイルの使用

構造化ファイルでデータプロバイダのデータを定義できます。次の例は、"data.xml" ファイルの内容です。

```
<data>
<result month="Jan-04">
<apple>81768</apple>
<orange>60310</orange>
<banana>43357</banana>
</result>
<result month="Feb-04">
<apple>81156</apple>
<orange>58883</orange>
<banana>49280</banana>
</result>
</data>
```

次の例に示すように、このファイルはモデルのソースとして直接ロードできます。

```
<?xml version="1.0"?>
<!-- charts/XMLFileDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Model id="results" source="../assets/data.xml"/>
  <mx:Panel title="Line Chart">
     <mx:LineChart id="chart" dataProvider="{results.result}">
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:LineSeries yField="banana" displayName="Banana"/>
           <mx:LineSeries yField="apple" displayName="Apple"/>
```

```
            <mx:LineSeries yField="orange" displayName="Orange"/>
        </mx:series>
    </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

ArrayCollection をチャートのデータプロバイダとして使用するには、次の例に示すように、モデルを ArrayCollection に変換します。

```
<?xml version="1.0"?>
<!-- charts/XMLFileToArrayCollectionDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%">
  <mx:Script>
     import mx.utils.ArrayUtil;
  </mx:Script>

  <mx:Model id="results" source="../assets/data.xml"/>
  <mx:ArrayCollection id="myAC"
        source="{ArrayUtil.toArray(results.result)}"
  />

  <mx:Panel title="Line Chart">
     <mx:LineChart id="chart" dataProvider="{myAC}">
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:LineSeries yField="banana" displayName="Banana"/>
           <mx:LineSeries yField="apple" displayName="Apple"/>
           <mx:LineSeries yField="orange" displayName="Orange"/>
        </mx:series>
     </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

また、次の例のように、XML ファイルを HTTPService コンポーネントの URL として定義し、HTTPService の結果をチャートのデータプロバイダに直接バインドすることもできます。

```
<?xml version="1.0"?>
<!-- charts/HTTPServiceDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%" creationComplete="srv.send()">
  <mx:HTTPService id="srv" url="../assets/data.xml"/>

  <mx:Panel title="Line Chart">
     <mx:LineChart id="chart"
        dataProvider="{srv.lastResult.data.result}"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="month"/>
        </mx:horizontalAxis>
```

```
        <mx:series>
           <mx:LineSeries yField="apple" name="Apple"/>
           <mx:LineSeries yField="orange" name="Orange"/>
           <mx:LineSeries yField="banana" name="Banana"/>
        </mx:series>
     </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

ArrayCollection を使用するには、次の例のように、HTTPService の結果を ArrayCollection に変換します。

```
<?xml version="1.0"?>
<!-- charts/HTTPServiceToArrayCollectionDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="srv.send()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var myData:ArrayCollection;
  ]]></mx:Script>

  <mx:HTTPService
     id="srv"
     url="../assets/data.xml"
     useProxy="false"
     result="myData=ArrayCollection(srv.lastResult.data.result)"
  />
  <mx:Panel title="Line Chart">
     <mx:LineChart id="chart" dataProvider="{myData}">
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:LineSeries yField="apple" name="Apple"/>
           <mx:LineSeries yField="orange" name="Orange"/>
           <mx:LineSeries yField="banana" name="Banana"/>
        </mx:series>
     </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

また、次の例のように、HTTPService の結果の形式を E4X に設定し、E4X を XMLListCollection オブジェクトのソースとして使用することもできます。

```
<?xml version="1.0"?>
<!-- charts/HTTPServiceToXMLListCollection.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="srv.send()">
  <mx:Script><![CDATA[
     import mx.utils.ArrayUtil;
  ]]></mx:Script>
```

```
  <mx:HTTPService id="srv"
    url="../assets/data.xml"
    resultFormat="e4x"
  />

  <mx:XMLListCollection id="myAC"
    source="{srv.lastResult.result}"
  />

  <mx:Panel title="Line Chart">
     <mx:LineChart id="chart" dataProvider="{myAC}">
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:LineSeries yField="apple" name="Apple"/>
           <mx:LineSeries yField="orange" name="Orange"/>
           <mx:LineSeries yField="banana" name="Banana"/>
        </mx:series>
     </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

チャートデータのランダムな生成

サンプルチャートで使用するデータを作成するには、ランダムにデータを生成する方法が便利です。次の例では、チャートコントロールで使用するテストデータを生成しています。

```
<?xml version="1.0"?>
<!-- charts/RandomDataGeneration.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.collections.*;

     // Define data provider array for the chart data.
     [Bindable]
     public var dataSet:ArrayCollection;

     // Define the number of elements in the array.
     public var dsLength:Number = 10;

     public function initApp():void {
        // Initialize data provider array.
        dataSet = new ArrayCollection(genData());
     }

     public function genData():Array {
        var result:Array = [];
```

```
        for (var i:int=0;i<dsLength;i++) {
           var localVals:Object = {
                valueA:Math.random()*100,
                valueB:Math.random()*100,
                valueX:Math.random()*100,
                valueY:Math.random()*100
           };

           // Push new object onto the data array.
           result.push(localVals);
        }
        return result;
     }
  ]]></mx:Script>

  <mx:Panel title="Plot Chart">
     <mx:PlotChart id="myChart" dataProvider="{dataSet}">
        <mx:series>
           <mx:PlotSeries
                xField="valueX"
                yField="valueY"
                displayName="Series 1"
           />
           <mx:PlotSeries
                xField="valueA"
                yField="valueB"
                displayName="Series 2"
           />
        </mx:series>
     </mx:PlotChart>
     <mx:Legend id="l1" dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

実行時におけるチャートデータの変更

ActionScript を使用すると、さまざまな方法で実行時にチャートコントロールのデータを変更できます。

チャートまたは系列のデータプロバイダを変更できます。次の例では、データプロバイダをローカル変数にバインドしています。次に、ユーザーがボタンをクリックしたときに、そのローカル変数を使用してチャートのデータプロバイダを切り替えています。

```
<?xml version="1.0"?>
<!-- charts/ChangeDataProvider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
```

```
    public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
    ]);

    [Bindable]
    public var expenses2:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2400, Expenses:1509, Amount:950},
        {Month:"Feb", Profit:3000, Expenses:2200, Amount:400},
        {Month:"Mar", Profit:3500, Expenses:1200, Amount:200}
    ]);

    [Bindable]
    public var dp:ArrayCollection = expenses;

    public function changeDataProvider():void {
        if (dp==expenses) {
            dp = expenses2;
        } else {
            dp = expenses;
        }
    }
]]></mx:Script>
<mx:Panel title="Line Chart">
    <mx:LineChart id="myChart" dataProvider="{dp}">
        <mx:horizontalAxis>
            <mx:CategoryAxis
                 dataProvider="{dp}"
                 categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:LineSeries
                 yField="Profit"
                 displayName="Profit"
            />
            <mx:LineSeries
                 yField="Expenses"
                 displayName="Expenses"
            />
            <mx:LineSeries
                 yField="Amount"
                 displayName="Amount"
            />
        </mx:series>
    </mx:LineChart>
    <mx:Legend dataProvider="{myChart}"/>
    <mx:Button label="Change Data Provider"
        click="changeDataProvider()"
```

```
    />
  </mx:Panel>
</mx:Application>
```

系列のデータポイントを追加または削除できます。次の例では、ユーザーがボタンをクリックしたときに、既存のデータプロバイダにアイテムを追加します。

```
<?xml version="1.0"?>
<!-- charts/AddDataItem.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*">
    <mx:Script><![CDATA[
        import mx.collections.ArrayCollection;
        [Bindable]
        public var dpac:ArrayCollection =  new ArrayCollection([
           {A:2000},
           {A:3000},
           {A:4000},
           {A:4000},
           {A:3000},
           {A:2000},
           {A:6000}
        ]);

        public function addDataItem():void {
            var o:Object = {"A":2000};
            dpac.addItem(o);
        }
    ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart"
        height="400"
        width="800"
        dataProvider="{dpac}"
     >
        <mx:series>
            <mx:ColumnSeries yField="A" displayName="Series 1"/>
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
     <mx:Button label="Add Data Item" click="addDataItem();"/>
  </mx:Panel>
</mx:Application>
```

系列のフィールドを変更して、実行時にチャートデータを変更することもできます。これは、系列オブジェクトに対してデータプロバイダフィールド (xField や yField など) の値を変更することによって行います。系列への参照を取得するには、系列の id プロパティまたはチャートコントロールの series インデックスを使用します。次の例では、ユーザーが [Change Series] ボタンをクリックしたときにデータ系列を切り替えています。

```
<?xml version="1.0"?>
<!-- charts/ToggleSeries.mxml -->
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize=
"initApp();">
  <mx:Script><![CDATA[
     [Bindable]
     public var dataSet:Array;
     public var myStates:Array =
        ["Wisconsin","Ohio","Arizona","Penn"];
     public var curSeries:String;
     public function initApp():void {
        var newData:Array = [];
        for(var i:int=0;i<myStates.length;i++) {
           newData[i] = {
            Apples: Math.floor(Math.random()*150),
            Oranges: Math.floor(Math.random()*150),
            myState: myStates[i]
           }
        }
        dataSet = newData;
        curSeries = "apples";
     }
     public function changeData():void {
        var series:Object = myChart.series[0];
        if (curSeries == "apples") {
           curSeries="oranges";
           series.yField = "Oranges";
           series.displayName = "Oranges";
           series.setStyle("fill",0xFF9933);
        } else {
           curSeries="apples";
           series.yField = "Apples";
           series.displayName = "Apples";
           series.setStyle("fill",0xFF0000);
        }
     }
  ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart"
        dataProvider="{dataSet}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{dataSet}"
                categoryField="myState"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                yField="Apples"
                displayName="Apples"
```

```
            >
            <mx:fill>
                <mx:SolidColor color="0xFF0000"/>
            </mx:fill>
           </mx:ColumnSeries>
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
     <mx:Button id="b1"
        label="Change Series"
        click="changeData()"
     />
  </mx:Panel>
</mx:Application>
```

データバインディングを利用すると、データの変更をリアルタイムにチャートに反映できます。次の例では、Timer を使用して、新しいデータをチェックする時間間隔を定義します。チャートのデータプロバイダが ArrayCollection にバインドされているため、コレクションに新しいデータポイントを追加するたびに、チャートが自動的に更新されます。

```
<?xml version="1.0"?>
<!-- charts/WatchingCollections.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    initialize="initData();">
    <mx:Script><![CDATA[
        import mx.collections.ArrayCollection;

        [Bindable]
        public var dataSet:ArrayCollection;

        [Bindable]
        public var revenue:Number = 100;

        private var t:Timer;

        private function initData():void {
            dataSet = new ArrayCollection();
            t = new Timer(500);
            t.addEventListener(TimerEvent.TIMER, addData);
            t.start();
        }

        private function addData(e:Event):void {
            dataSet.addItem( { revenue: revenue } );
            revenue += Math.random() * 10 - 5;
        }
    ]]></mx:Script>

    <mx:SeriesInterpolate id="interp"
        elementOffset="0"
```

```
        duration="300"
        minimumElementDuration="0"
    />

    <mx:Panel title="Line Chart">
        <mx:LineChart id="myChart" dataProvider="{dataSet}">
            <mx:series>
                <mx:LineSeries
                    yField="revenue"
                    showDataEffect="{interp}"
                />
            </mx:series>
            <mx:horizontalAxis>
                <mx:LinearAxis autoAdjust="false"/>
            </mx:horizontalAxis>
        </mx:LineChart>
    </mx:Panel>
</mx:Application>
```

また、システム値を使用して、実行時にチャートを更新することもできます。次の例では、LineChart コントロールの totalMemory プロパティの値をリアルタイムに追跡します。

```
<?xml version="1.0"?>
<!-- charts/MemoryGraph.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize=
"initTimer()">
  <mx:Script><![CDATA[
  import flash.utils.Timer;
  import flash.events.TimerEvent;
  import mx.collections.ArrayCollection;

  [Bindable]
  public var memoryUsage:ArrayCollection = new ArrayCollection();

        public function initTimer():void {
           // The first parameter in the Timer constructor
           // is the interval, in milliseconds. The second
           // parameter is how many times to run (0 is
           // infinity).
           var myTimer:Timer = new Timer(1000, 0);

           // Add the listener for the timer event.
           myTimer.addEventListener("timer", timerHandler);
           myTimer.start();
        }

        public function timerHandler(event:TimerEvent):void {
           var o:Object = new Object();

           // Get the number of milliseconds since Flash
           // Player started.
```

```
            o.time = getTimer();

            // Get the total memory Flash Player is using.
            o.memory = flash.system.System.totalMemory;

            // Add new object to the ArrayCollection, which
            // is bound to the chart's data provider.
            memoryUsage.addItem(o);
        }
    ]]></mx:Script>

    <mx:LineChart id="chart"
          dataProvider="{memoryUsage}"
          showDataTips="true"
    >
       <mx:horizontalAxis>
          <mx:LinearAxis/>
       </mx:horizontalAxis>

       <mx:verticalAxis>
          <mx:LinearAxis minimum="5000000"/>
       </mx:verticalAxis>
       <mx:series>
          <mx:Array>
             <mx:LineSeries yField="memory"/>
          </mx:Array>
       </mx:series>
    </mx:LineChart>
</mx:Application>
```

第 54 章

チャートタイプ

54

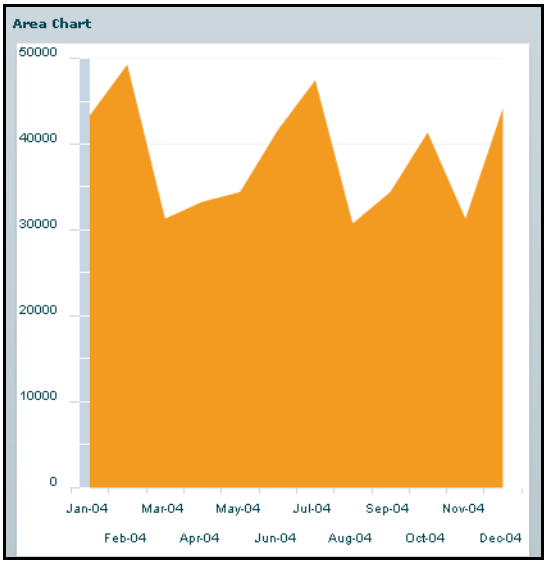
Adobe Flex には、さまざまなタイプのチャートコントロールが用意されています。

目次

面グラフの使用

AreaChart コントロールを使用すると、データ内の値を結ぶ線を境界とする面としてデータを表すことができます。線の下の面は色またはパターンで塗りつぶされます。線上の各データポイントをアイコンやシンボルで表すことも、アイコンなしの単純な線を表示することもできます。

次の図は、面グラフの例を示しています。



次の例では、AreaChart コントロールを作成します。

```
<?xml version="1.0"?>
<!-- charts/BasicArea.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Area Chart">
     <mx:AreaChart id="myChart" dataProvider="{expenses}"
     showDataTips="true">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
```

```
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:AreaSeries
                yField="Profit"
                displayName="Profit"
            />
            <mx:AreaSeries
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:AreaChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

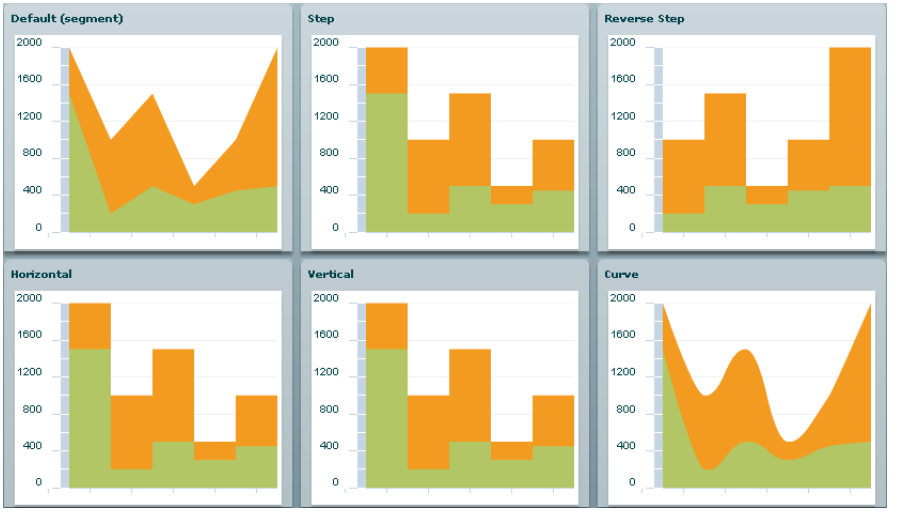
面グラフは基本的に線の下の範囲が塗りつぶされた折れ線グラフであるため、面グラフと折れ線グラフには共通する特性が多数あります。詳細については、1503 ページの「折れ線グラフの使用」を参照してください。

面グラフのデータを定義するには、AreaChart コントロールと共に AreaSeries チャート系列を使用します。次の表で、面グラフの定義によく使用する AreaSeries チャート系列のプロパティについて説明します。

プロパティ	説明
yField	各データポイントの y 軸の位置を決定するデータプロバイダのフィールドを指定します。
xField	各データポイントの x 軸の位置を決定するデータプロバイダのフィールドを指定します。このフィールドを省略した場合、データプロバイダにおけるデータの順序でデータポイントが配置されます。
minField	領域の下端の y 軸位置を決定するデータプロバイダのフィールドを指定します。このプロパティはオプションです。このフィールドを省略すると、面の下端は x 軸に揃えられます。このプロパティは、積み上げグラフ、積み重ねグラフ、または構成比グラフには影響しません。minField プロパティの詳細については、1576 ページの「minField プロパティの使用」を参照してください。

プロパティ	説明
form	データ系列をグラフに表示する方法を指定します。使用できる値は次のとおりです。 • segment　データ系列の各データポイントを線でつなぎ、角度を付けたセグメントを描画します。これはデフォルトの設定です。 • step　水平方向と垂直方向のセグメントを成す線を描画します。最初のデータポイントで横線の描画を始め、2 番目のデータポイントにさしかかったところで縦線を描画します。この操作を各データポイントについて繰り返します。 • reverseStep　水平方向と垂直方向のセグメントを成す線を描画します。最初のデータポイントで縦線の描画を始め、2 番目のデータポイントにさしかかったところで横線を描画します。この操作を各データポイントについて繰り返します。 • vertical　最初のデータポイントの y 座標を起点とし、2 番目のデータポイントの x 座標上にある、2 番目のデータポイントの y 座標まで縦線のみを描画します。この操作を各データポイントについて繰り返します。 • horizontal　最初のデータポイントの x 座標を起点とし、最初のデータポイントの y 座標上にある、2 番目のデータポイントの x 座標まで横線のみを描画します。この操作を各データポイントについて繰り返します。 • curve　データポイント間に曲線を描画します。

次の例は、使用可能な AreaSeries 系列の形式を示しています。



AreaChart コントロールの type プロパティを使用すると、積み上げ、積み重ね、構成比、高低など、チャートをさまざまな方法で表すことができます。詳細については、1636 ページの「チャートの積み重ね」を参照してください。

横棒グラフの使用

BarChart コントロールを使用すると、データ内の値によって長さが決まる一連の横棒でデータを表すことができます。BarChart コントロールを使用すると、集合横棒、積み上げ、積み重ね、構成比、高低など、チャートをさまざまな方法で表すことができます。詳細については、1636 ページの「チャートの積み重ね」を参照してください。

横棒グラフのデータを定義するには、BarSeries コントロールと共に BarSeries チャート系列を使用します。次の表で、横棒グラフの定義に使用する BarSeries チャート系列のプロパティについて説明します。

プロパティ	説明
yField	チャート内の各横棒の下端の y 軸の位置を決定するデータプロバイダのフィールドを指定します。このプロパティを省略すると、データプロバイダ内でのデータの順序に従って横棒が並べられます。
xField	各横棒の終端の x 軸の位置を決定するデータプロバイダのフィールドを指定します。
minField	横棒の下端の x 軸の位置を決定するデータプロバイダのフィールドを指定します。このプロパティは、積み上げグラフ、積み重ねグラフ、または構成比グラフには影響しません。minField プロパティの詳細については、1576 ページの「minField プロパティの使用」を参照してください。

次の例では、単純な BarChart コントロールを作成します。

```
<?xml version="1.0"?>
<!-- charts/BasicBar.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Bar Chart">
     <mx:BarChart id="myChart" dataProvider="{expenses}">
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                yField="Month"
```

```
                xField="Profit"
                displayName="Profit"
            />
            <mx:BarSeries
                yField="Month"
                xField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:BarChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

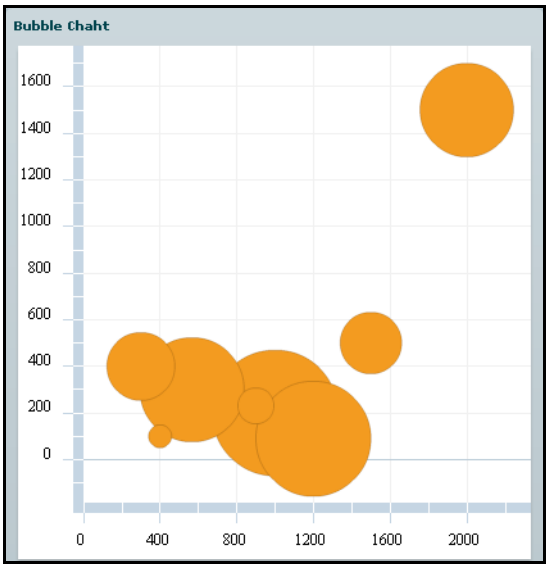
横棒グラフは本質的には、時計回りに 90 度回転させた縦棒グラフであるため、横棒グラフと縦棒グラフには共通する特性が多数あります。詳細については、1494 ページの「縦棒グラフの使用」を参照してください。

バブルチャートの使用

BubbleChart コントロールを使用すると、各データポイントに関して 3 つの値を使用してデータを表すことができます。この 3 つの値とは、データポイントの x 軸上の位置を決定する値、y 軸上の位置を決定する値、さらにチャート記号の大きさを決定するチャート内の他のデータポイントからの相対値です。

<mx:BubbleChart> タグには、maxRadius という追加のプロパティがあります。このプロパティにより、最大のチャートエレメントの最大半径をピクセル単位で指定します。この半径は、最大値を持つデータポイントに割り当てられます。他のすべてのデータポイントには、最大値からの相対値に基づき、最大値より小さい半径が割り当てられます。デフォルト値は 30 ピクセルです。

次の例はバブルチャートです。



バブルチャートのデータを定義するには、BubbleChart コントロールと共に BubbleSeries チャート系列を使用します。次の表で、バブルチャートの定義によく使用する BubbleSeries チャート系列のプロパティについて説明します。

プロパティ	説明
yField	各データポイントの y 軸の位置を決定するデータプロバイダのフィールドを指定します。このプロパティは必須です。
xField	各データポイントの x 軸の位置を決定するデータプロバイダのフィールドを指定します。このプロパティは必須です。
radiusField	グラフ内の他のデータポイントを基準にして各シンボルの半径を決定するデータプロバイダのフィールドを指定します。このプロパティは必須です。

次の例では、バブルエレメントの最大半径を 50 ピクセルに設定して BubbleChart コントロールを描画します。

```
<?xml version="1.0"?>
<!-- charts/BasicBubble.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:120, Amount:45},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:60},
```

```
        {Month:"Mar", Profit:1500, Expenses:500, Amount:30}
    ]);
  ]]></mx:Script>
  <mx:Panel title="Bubble Chart">
    <mx:BubbleChart id="myChart"
      maxRadius="50"
      dataProvider="{expenses}"
      showDataTips="true"
    >
      <mx:series>
        <mx:BubbleSeries
            xField="Profit"
            yField="Expenses"
            radiusField="Amount"
            displayName="Profit"
        />
      </mx:series>
    </mx:BubbleChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

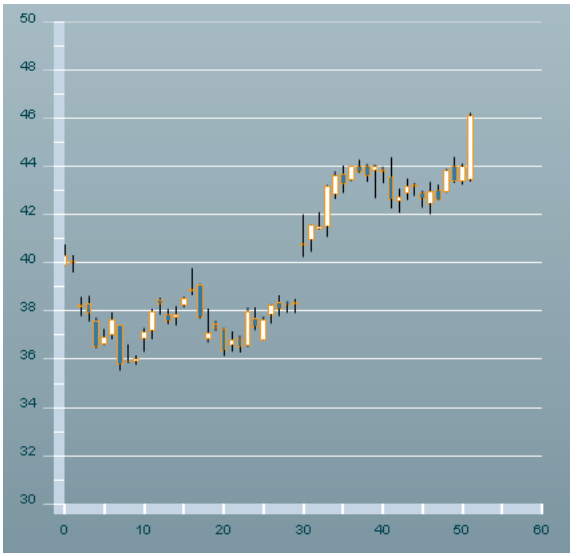
ローソク足チャートの使用

CandlestickChart コントロールは、データ系列の高値、安値、始値、終値などの財務データを一連のローソク足で示します。各ローソク足の垂直線の上部と下部は、データポイントの高値と安値を示しています。一方、塗りつぶされたボックスの上部と下部は始値と終値を示しています。ローソク足の塗りつぶされ方は、データポイントの終値が始値よりも高いかどうかによって異なります。

CandlestickChart コントロールの CandlestickSeries では、高値、安値、始値、終値の 4 つのデータポイントすべてが必要です。始値のデータポイントを使用しない場合は HighLowOpenClose チャートを使用します。このチャートでは、始値を示すデータポイントは必要ありません。詳細については、1499 ページの「HighLowOpenClose チャートの使用」を参照してください。

データを定義するには、CandlestickSeries チャート系列と CandlestickChart コントロールを使用します。

次は CandlestickChart チャートの例です。

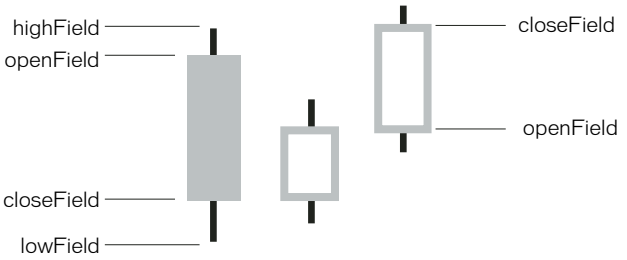


次の表で、ローソク足チャートの定義によく使用する CandlestickSeries チャート系列のプロパティについて説明します。

プロパティ	説明
closeField	エレメントの終値の y 軸位置を決定するデータプロバイダのフィールドを指定します。このプロパティによって、ローソク足の上部または下部が定義されます。
highField	エレメントの高値の y 軸位置を決定するデータプロバイダのフィールドを指定します。このプロパティによって、ローソク足内部の線の上部が定義されます。
lowField	エレメントの安値の y 軸位置を決定する、データプロバイダのフィールドを指定します。このプロパティによって、ローソク足内部の線の下部が定義されます。
openField	エレメントの始値の y 軸位置を決定する、データプロバイダのフィールドを指定します。このプロパティによって、ローソク足の上部または下部の位置が定義されます。
xField	エレメントの x 軸位置を決定するデータプロバイダのフィールドを指定します。空のストリング ("") に設定されている場合、縦棒はデータプロバイダに表示される順序でレンダリングされます。デフォルト値は空のストリングです。

closeField が openField よりも低い場合は、ローソクが単色で塗りつぶされます。デフォルトでは、この塗りつぶしに使用される色はボックスの輪郭の色と同じです。この色は、declineFill スタイルプロパティで定義します。closeField が openField よりも高い場合、デフォルトではローソクが白で塗りつぶされます。

次の図はこれらのプロパティを示しています。この図からわかるように、closeField プロパティは、openField プロパティよりも高いかどうかによって、ローソク足の上部または下部に配置されます。



次の例では、CandlestickChart コントロールを作成します。

```
<?xml version="1.0"?>
<!-- charts/BasicCandlestick.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     [Bindable]
     public var TICKER:Array = [
        {date:"1-Aug-05",open:42.57,high:43.08,low:42.08,close:42.75},
        {date:"2-Aug-05",open:42.89,high:43.5,low:42.61,close:43.19},
        {date:"3-Aug-05",open:43.19,high:43.31,low:42.77,close:43.22},
        {date:"4-Aug-05",open:42.89,high:43,low:42.29,close:42.71},
        {date:"5-Aug-05",open:42.49,high:43.36,low:42.02,close:42.99},
        {date:"8-Aug-05",open:43,high:43.25,low:42.61,close:42.65},
        {date:"9-Aug-05",open:42.93,high:43.89,low:42.91,close:43.82},
        {date:"10-Aug-05",open:44,high:44.39,low:43.31,close:43.38},
        {date:"11-Aug-05",open:43.39,high:44.12,low:43.25,close:44},
        {date:"12-Aug-05",open:43.46,high:46.22,low:43.36,close:46.1}
     ];
  ]]></mx:Script>

  <mx:Panel title="Candlestick Chart">
     <mx:CandlestickChart id="mychart"
        dataProvider="{TICKER}"
        showDataTips="true"
        height="400"
        width="400"
     >
        <mx:series>
           <mx:CandlestickSeries
                dataProvider="{TICKER}"
                openField="open"
                highField="high"
                lowField="low"
                closeField="close"
                displayName="TICKER"
           />
```

```
        </mx:series>
    </mx:CandlestickChart>
    <mx:Legend dataProvider="{mychart}"/>
  </mx:Panel>
</mx:Application>
```

ローソクの塗りつぶしの色を変更するには、系列の fill プロパティと declineFill プロパティを使用します。fill プロパティによって、closeField の値が openField の値よりも高い場合のローソクの色が定義されます。declineFill プロパティによって、その逆の場合のローソクの色が定義されます。次の例に示すように、Stroke クラスを使用して、上部および下部の線とローソクの境界線のプロパティを定義することもできます。

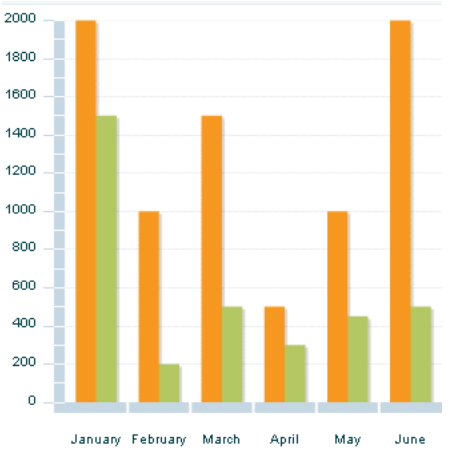
```
<?xml version="1.0"?>
<!-- charts/CandlestickStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var TICKER:ArrayCollection = new ArrayCollection([
        {date:"1-Aug-05",open:42.57,high:43.08,low:42.08,close:42.75},
        {date:"2-Aug-05",open:42.89,high:43.5,low:42.61,close:43.19},
        {date:"3-Aug-05",open:43.19,high:43.31,low:42.77,close:43.22},
        {date:"4-Aug-05",open:42.89,high:43,low:42.29,close:42.71},
        {date:"5-Aug-05",open:42.49,high:43.36,low:42.02,close:42.99},
        {date:"8-Aug-05",open:43,high:43.25,low:42.61,close:42.65},
        {date:"9-Aug-05",open:42.93,high:43.89,low:42.91,close:43.82},
        {date:"10-Aug-05",open:44,high:44.39,low:43.31,close:43.38},
        {date:"11-Aug-05",open:43.39,high:44.12,low:43.25,close:44},
        {date:"12-Aug-05",open:43.46,high:46.22,low:43.36,close:46.1}
     ]);
  ]]></mx:Script>

  <mx:Panel title="Candlestick Chart">
     <mx:CandlestickChart id="mychart"
        dataProvider="{TICKER}"
        showDataTips="true"
        height="400"
        width="400"
     >
        <mx:verticalAxis>
           <mx:LinearAxis title="linear axis" minimum="40" maximum=
           "50"/>
        </mx:verticalAxis>

        <mx:series>
           <mx:CandlestickSeries
                dataProvider="{TICKER}"
                openField="open"
                highField="high"
```

```
                lowField="low"
                closeField="close"
                displayName="TICKER"
            >
             <mx:fill>
                <mx:SolidColor color="green"/>
             </mx:fill>
             <mx:declineFill>
                <mx:SolidColor color="red"/>
             </mx:declineFill>
             <mx:stroke>
                <mx:Stroke weight="1" color="black"/>
             </mx:stroke>
            </mx:CandlestickSeries>
        </mx:series>
    </mx:CandlestickChart>
    <mx:Legend dataProvider="{mychart}"/>
  </mx:Panel>
</mx:Application>
```

縦棒グラフの使用

ColumnChart コントロールは、データを一連の縦棒で表します。縦棒の高さはデータの値によって決定します。ColumnChart コントロールを使用すると、単純な縦棒、集合縦棒、積み上げ、積み重ね、構成比、高低などの種類の縦棒グラフを作成することができます。詳細については、1636 ページの「チャートの積み重ね」を参照してください。

次の例は、系列を 2 つ含む ColumnChart コントロールを示しています。



縦棒グラフのデータを定義するには、ColumnChart コントロールと共に ColumnSeries チャート系列を使用します。次の表で、縦棒グラフを定義する ColumnSeries チャート系列のプロパティについて説明します。

プロパティ	説明
yField	縦棒上端の y 軸の位置を決定するデータプロバイダのフィールドを指定します。このフィールドは縦棒の高さを定義します。
xField	縦棒の x 軸の位置を決定するデータプロバイダのフィールドを指定します。このプロパティを省略すると、データプロバイダ内でのデータの順序に従って縦棒が並べられます。
minField	縦棒下端の y 軸の位置を決定するデータプロバイダのフィールドを指定します。このプロパティは、積み上げグラフ、積み重ねグラフ、または構成比グラフには影響しません。minField プロパティの詳細については、1576 ページの「minField プロパティの使用」を参照してください。

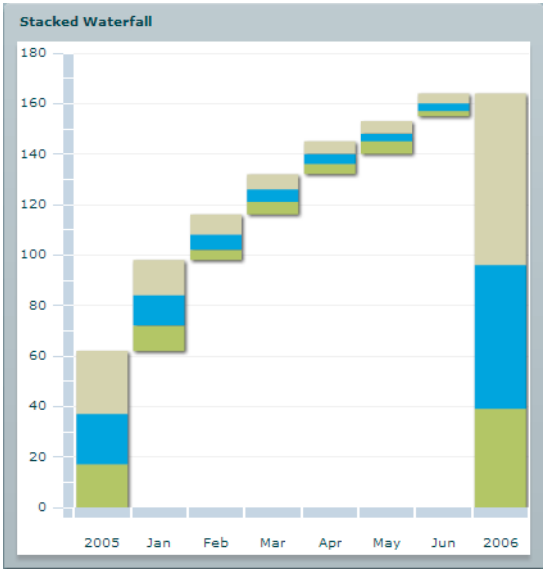
次の例では、系列を 2 つ含む ColumnChart コントロールを作成します。

```
<?xml version="1.0"?>
<!-- charts/BasicColumn.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
```

```
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

ColumnChart コントロールの type プロパティを設定して、チャート内の系列データの積み重ねやグループ化を行えます。詳細については、1636 ページの「チャートの積み重ね」を参照してください。

ColumnChart コントロールを使用すると、カスケーディング (ウォーターフォール) 縦棒グラフを作成できます。これを行う 1 つの方法は、非表示の系列を作成し、それを使用して他の縦棒の可変高さを設定することです。これにより、ウォーターフォール効果が得られます。次の図は、ウォーターフォール縦棒グラフの例を示しています。

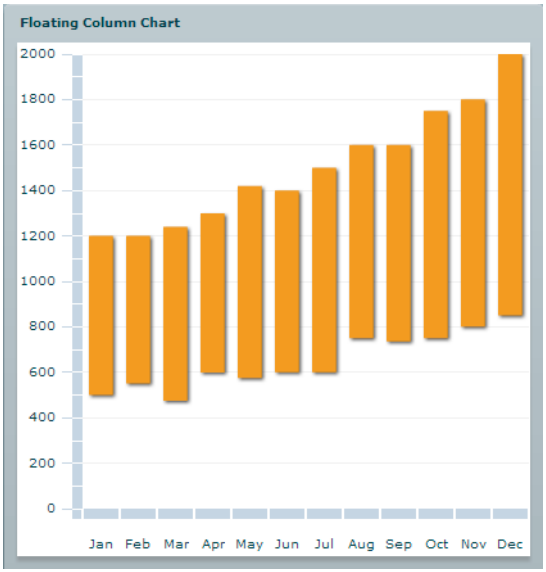


このグラフを作成するコードは次のとおりです。

```
<?xml version="1.0"?>
<!-- charts/WaterfallStacked.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"2005", top:25, middle:20, bottom:17, Invisible:0},
        {Month:"Jan", top:14, middle:12, bottom:10, Invisible:62},
        {Month:"Feb", top:8, middle:6, bottom:4, Invisible:98},
        {Month:"Mar", top:6, middle:5, bottom:5, Invisible:116},
        {Month:"Apr", top:5, middle:4, bottom:4, Invisible:132},
```

```
        {Month:"May", top:5, middle:3, bottom:5, Invisible:140},
        {Month:"Jun", top:4, middle:3, bottom:2, Invisible:155},
        {Month:"2006", top:68, middle:57, bottom:39, Invisible:0}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Stacked Waterfall">
     <mx:ColumnChart id="myChart"
        dataProvider="{expenses}"
        columnWidthRatio=".9"
        showDataTips="true"
        type="stacked"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                yField="Invisible"
                displayName="Invisible"
           >
                <mx:fill>
                    <!--Set alpha to 0 to hide invisible column.-->
                    <mx:SolidColor color="0xFFFFFF" alpha="0"/>
                </mx:fill>
           </mx:ColumnSeries>
           <mx:ColumnSeries yField="bottom" displayName="Profit"/>
           <mx:ColumnSeries yField="middle" displayName="Expenses"/>
           <mx:ColumnSeries yField="top" displayName="Profit"/>
        </mx:series>
     </mx:ColumnChart>
  </mx:Panel>
</mx:Application>
```

チャートのデータ系列の minField プロパティを使用して、浮動縦棒グラフを作成することもできます。このプロパティによって、縦棒の下部の値を設定できます。次に浮動 ColumnChart コントロールの例を示します。



このグラフを作成するコードは次のとおりです。

```
<?xml version="1.0"?>
<!-- charts/MinFieldColumn.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Revenue:1200, Expenses:500},
        {Month:"Feb", Revenue:1200, Expenses:550},
        {Month:"Mar", Revenue:1240, Expenses:475},
        {Month:"Apr", Revenue:1300, Expenses:600},
        {Month:"May", Revenue:1420, Expenses:575},
        {Month:"Jun", Revenue:1400, Expenses:600},
        {Month:"Jul", Revenue:1500, Expenses:600},
        {Month:"Aug", Revenue:1600, Expenses:750},
        {Month:"Sep", Revenue:1600, Expenses:735},
        {Month:"Oct", Revenue:1750, Expenses:750},
        {Month:"Nov", Revenue:1800, Expenses:800},
        {Month:"Dec", Revenue:2000, Expenses:850}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Floating Column Chart">
```

```
    <mx:ColumnChart
       dataProvider="{expenses}"
       showDataTips="true"
    >
       <mx:horizontalAxis>
          <mx:CategoryAxis
               dataProvider="{expenses}"
               categoryField="Month"
           />
       </mx:horizontalAxis>
       <mx:series>
          <mx:ColumnSeries
               yField="Revenue"
               minField="Expenses"
               displayName="Revenue"
           />
       </mx:series>
    </mx:ColumnChart>
  </mx:Panel>
</mx:Application>
```

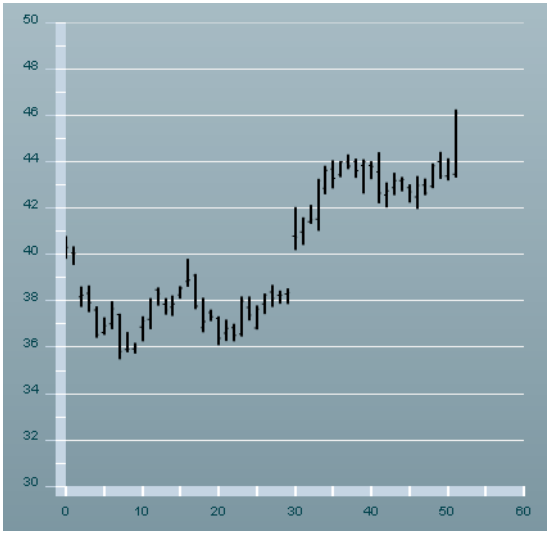
詳細については、1576 ページの「minField プロパティの使用」を参照してください。

HighLowOpenClose チャートの使用

HLOCChart コントロールは、データ系列の高値、安値、始値、終値などの財務データを一連の線で示します。垂直線の上部および下部は、データポイントの高値と安値を示しています。一方、左の目盛りは始値を、右の目盛りは終値を示しています。

HLOCChart コントロールでは、始値を示すデータポイントは必要ありません。関連チャートには CandlestickChart コントロールがあります。このコントロールは、ローソク足と同様のデータを示します。詳細については、1490 ページの「ローソク足チャートの使用」を参照してください。

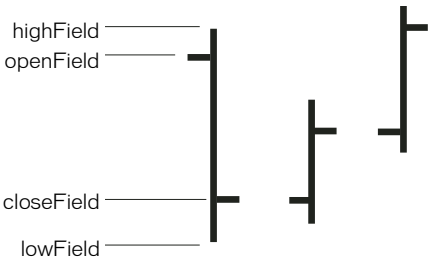
HighLowOpenClose チャートのデータを定義するには、HLOCSeries と共に HLOCChart コントロールを使用します。HLOC チャートの例を次に示します。



次の表で、HighLowOpenClose チャートの定義によく使用する HLOCChart コントロールの系列のプロパティについて説明します。

プロパティ	説明
closeField	エレメントの終値の y 軸位置を決定するデータプロバイダのフィールドを指定します。このプロパティによって、垂直線の右目盛りの位置が定義されます。
highField	エレメントの高値の y 軸位置を決定するデータプロバイダのフィールドを指定します。このプロパティによって、垂直線の上部が定義されます。
lowField	エレメントの安値の y 軸位置を決定するデータプロバイダのフィールドを指定します。このプロパティによって、垂直線の下部が定義されます。
openField	エレメントの始値の y 軸位置を決定する、データプロバイダのフィールドを指定します。このプロパティによって、垂直線の左目盛りの位置が定義されます。このプロパティはオプションです。
xField	エレメントの x 軸位置を決定するデータプロバイダのフィールドを指定します。空のストリング ("") に設定されている場合、縦棒はデータプロバイダに表示される順序でレンダリングされます。デフォルト値は空のストリングです。

HLOCChart コントロールのデータポイントでは、openField プロパティを設定する必要はありません。openField プロパティを指定しない場合、データポイントは、右側を指す終値のインジケータが1つ付加された水平な線でレンダリングされます。openField プロパティを指定した場合、次の図に示すように、データポイントは左側を指す別のインジケータでレンダリングされます。



次の例では、HLOCChart コントロールを作成します。

```
<?xml version="1.0"?>
<!-- charts/BasicHLOC.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var TICKER:ArrayCollection = new ArrayCollection([
        {date:"1-Aug-05",open:42.57,high:43.08,low:42.08,close:42.75},
        {date:"2-Aug-05",open:42.89,high:43.5,low:42.61,close:43.19},
        {date:"3-Aug-05",open:43.19,high:43.31,low:42.77,close:43.22},
        {date:"4-Aug-05",open:42.89,high:43,low:42.29,close:42.71},
        {date:"5-Aug-05",open:42.49,high:43.36,low:42.02,close:42.99},
        {date:"8-Aug-05",open:43,high:43.25,low:42.61,close:42.65},
        {date:"9-Aug-05",open:42.93,high:43.89,low:42.91,close:43.82},
        {date:"10-Aug-05",open:44,high:44.39,low:43.31,close:43.38},
        {date:"11-Aug-05",open:43.39,high:44.12,low:43.25,close:44},
        {date:"12-Aug-05",open:43.46,high:46.22,low:43.36,close:46.1}
     ]);
  ]]></mx:Script>

  <mx:Panel title="HighLowOpenClose Chart">
     <mx:HLOCChart id="myChart"
        dataProvider="{TICKER}"
        showDataTips="true"
     >
        <mx:verticalAxis>
           <mx:LinearAxis minimum="30" maximum="50"/>
        </mx:verticalAxis>
        <mx:series>
           <mx:HLOCSeries
                dataProvider="{TICKER}"
```

```
                openField="open"
                highField="high"
                lowField="low"
                closeField="close"
                displayName="TICKER"
            >
            </mx:HLOCSeries>
        </mx:series>
    </mx:HLOCChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

垂直線の太さを変更するには、データ系列で Stroke オブジェクトを使用します。垂直線の始値と終値を示す目盛りの外観を変更するには、openTickStroke スタイルプロパティと closeTickStroke スタイルプロパティを使用します。次の例では、垂直線の太さをデフォルト値の 1 から 2 に変更し、線の色をすべて黒に設定しています。

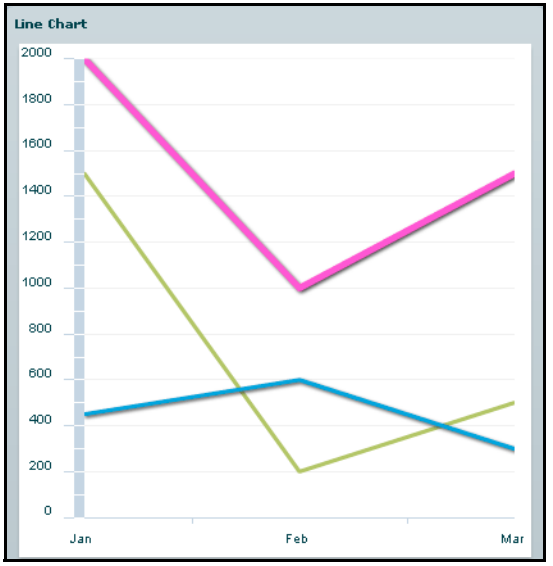
```
<?xml version="1.0"?>
<!-- charts/HLOCStyled.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var TICKER:ArrayCollection = new ArrayCollection([
        {date:"1-Aug-05",open:42.57,high:43.08,low:42.08,close:42.75},
        {date:"2-Aug-05",open:42.89,high:43.5,low:42.61,close:43.19},
        {date:"3-Aug-05",open:43.19,high:43.31,low:42.77,close:43.22},
        {date:"4-Aug-05",open:42.89,high:43,low:42.29,close:42.71},
        {date:"5-Aug-05",open:42.49,high:43.36,low:42.02,close:42.99},
        {date:"8-Aug-05",open:43,high:43.25,low:42.61,close:42.65},
        {date:"9-Aug-05",open:42.93,high:43.89,low:42.91,close:43.82},
        {date:"10-Aug-05",open:44,high:44.39,low:43.31,close:43.38},
        {date:"11-Aug-05",open:43.39,high:44.12,low:43.25,close:44},
        {date:"12-Aug-05",open:43.46,high:46.22,low:43.36,close:46.1},
     ]);
  ]]></mx:Script>

  <mx:Panel title="HLOC Chart">
     <mx:HLOCChart id="myChart"
        dataProvider="{TICKER}"
        showDataTips="true"
     >
        <mx:verticalAxis>
           <mx:LinearAxis minimum="30" maximum="50"/>
        </mx:verticalAxis>
        <mx:series>
           <mx:HLOCSeries
                dataProvider="{TICKER}"
                openField="open"
```

```
                highField="high"
                lowField="low"
                closeField="close"
                displayName="TICKER"
            >
            <mx:stroke>
                <mx:Stroke color="#000000" weight="2"/>
            </mx:stroke>
            <mx:closeTickStroke>
                <mx:Stroke color="#000000" weight="1"/>
            </mx:closeTickStroke>
            <mx:openTickStroke>
                <mx:Stroke color="#000000" weight="1"/>
            </mx:openTickStroke>
          </mx:HLOCSeries>
        </mx:series>
    </mx:HLOCChart>
    </mx:Panel>
    <mx:Legend dataProvider="{myChart}"/>
</mx:Application>
```

折れ線グラフの使用

LineChart コントロールは、直交座標上の一連の点を直線で結んでデータを表します。線上の各データポイントをアイコンやシンボルで表すことも、アイコンなしの単純な線を表示することもできます。

単純な折れ線グラフの例を次に示します。

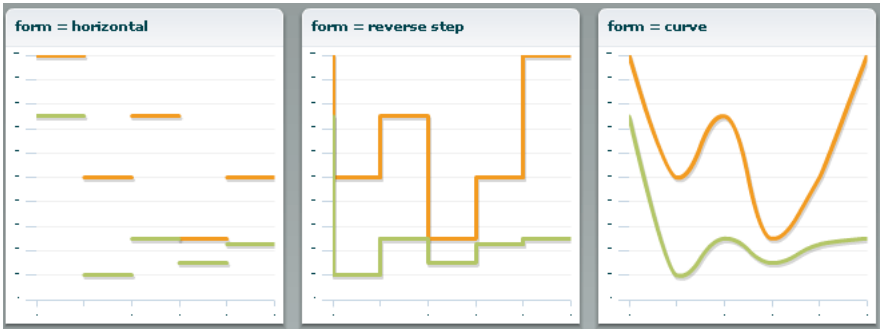


折れ線グラフのデータを定義するには、LineChart コントロールと共に LineSeries チャート系列を使用します。次の表で、折れ線グラフの定義によく使用する LineSeries チャート系列のプロパティについて説明します。

プロパティ	説明
yField	各データポイントの y 軸の位置を決定するデータプロバイダのフィールドを指定します。これは、その位置での線の高さを示します。
xField	各データポイントの x 軸の位置を決定するデータプロバイダのフィールドを指定します。このフィールドを省略した場合、データプロバイダにおけるデータの順序でデータポイントが配置されます。
interpolateValues	存在しないデータの表現方法を指定します。このプロパティの値を false に設定した場合、値の欠落している箇所で線の描画が途切れます。true に設定した場合、欠落した値が補完されて連続した線が描画されます。デフォルト値は false です。
form	データ系列をグラフに表示する方法を指定します。使用できる値は次のとおりです。 • segment　データ系列の各データポイントを線でつなぎ、角度を付けたセグメントを描画します。これはデフォルトの設定です。 • step　水平方向と垂直方向のセグメントを成す線を描画します。最初のデータポイントで横線の描画を始め、2 番目のデータポイントにさしかかったところで縦線を描画します。これを各データポイントについて繰り返します。 • reverseStep　水平方向と垂直方向のセグメントを成す線を描画します。最初のデータポイントで縦線の描画を始め、2 番目のデータポイントにさしかかったところで横線を描画します。これを各データポイントについて繰り返します。 • vertical　最初のデータポイントの y 座標を起点とし、2 番目のデータポイントの x 座標上にある、2 番目のデータポイントの y 座標まで縦線のみを描画します。これを各データポイントについて繰り返します。 • horizontal　最初のデータポイントの x 座標を起点とし、最初のデータポイントの y 座標上にある、2 番目のデータポイントの x 座標まで横線のみを描画します。これを各データポイントについて繰り返します。 • curve　データポイント間に曲線を描画します。

次の図に示すのは、LineSeries チャートで使用できる形式です。

segment　　step　　vertical



horizontal　　reverseStep　　curve

次の例では、LineChart コントロールを作成します。

```
<?xml version="1.0"?>
<!-- charts/BasicLine.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Line Chart">
     <mx:LineChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
```

```
            <mx:CategoryAxis
                 dataProvider="{expenses}"
                 categoryField="Month"
             />
         </mx:horizontalAxis>
         <mx:series>
            <mx:LineSeries
                 yField="Profit"
                 displayName="Profit"
            />
            <mx:LineSeries
                 yField="Expenses"
                 displayName="Expenses"
            />
         </mx:series>
      </mx:LineChart>
      <mx:Legend dataProvider="{myChart}"/>
   </mx:Panel>
</mx:Application>
```

線のフォーマット

各系列の線の幅と色を変更するには、<mx:lineStroke> タグを使用します。デフォルトの線の太さは 3 ピクセルで、影が付いています。次の例では、Stroke 系列オブジェクトにカスタムの色と幅を設定しています。

```
<?xml version="1.0"?>
<!-- charts/BasicLineStroke.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Line Chart With Strokes">
     <mx:LineChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
            />
        </mx:horizontalAxis>
```

```
        <mx:series>
           <mx:LineSeries
            yField="Profit"
            displayName="Profit"
           >
            <mx:lineStroke>
                <mx:Stroke
                    color="0x0099FF"
                    weight="20"
                    alpha=".2"
                />
            </mx:lineStroke>
           </mx:LineSeries>
           <mx:LineSeries
            yField="Expenses"
            displayName="Expenses"
           >
            <mx:lineStroke>
                <mx:Stroke
                    color="0x0044EB"
                    weight="20"
                    alpha=".8"
                />
            </mx:lineStroke>
           </mx:LineSeries>
        </mx:series>
     </mx:LineChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

折れ線グラフでの Stroke クラスの使用の詳細については、1577 ページの「線の使用」を参照してください。

LineChart コントロールの線には、デフォルトでドロップシャドウが付いています。ドロップシャドウは、次の例のように LineChart コントロールの seriesFilters プロパティに空の配列を設定すると削除できます。

```
<?xml version="1.0"?>
<!-- charts/LineChartNoShadows.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
```

```
  <mx:Panel title="Line Chart with No Shadows">
     <mx:LineChart id="myChart" dataProvider="{expenses}">
        <mx:seriesFilters>
           <mx:Array/>
        </mx:seriesFilters>
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:LineSeries
                yField="Profit"
                displayName="Profit"
           />
           <mx:LineSeries
                yField="Expenses"
                displayName="Expenses"
           />
           <mx:LineSeries
                yField="Amount"
                displayName="Amount"
           />
        </mx:series>
     </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

次の例に示すように、seriesFilters プロパティの値をプログラムで設定することもできます。

```
myLineChart.seriesFilters = [];
```

LineSeries の lineSegmentRenderer プロパティを設定して、各系列のプログラムレンダラー (スキン) クラスを指定することもできます。デフォルトのレンダラーは LineRenderer ですが、すべての系列にシャドウフィルタも適用されます。前の例のようにシャドウフィルタを削除した後で、グラフの線にドロップシャドウを付けるには、次の例のように lineSegmentRenderer を ShadowLineRenderer クラスに設定します。

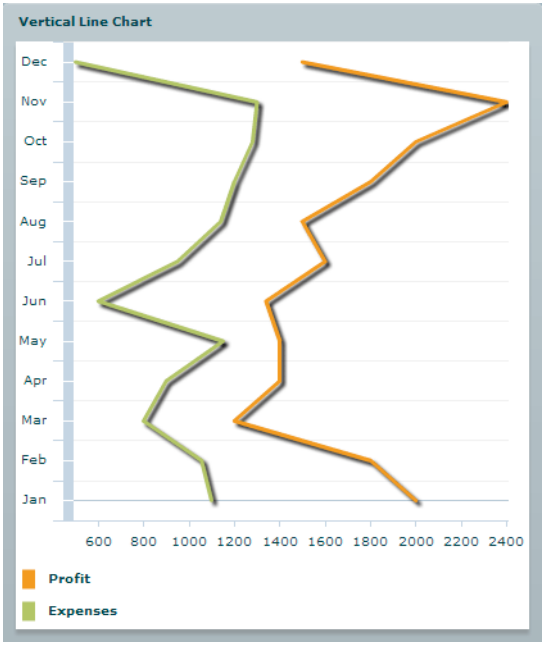
```
<?xml version="1.0"?>
<!-- charts/LineChartOneShadow.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
```

```
    ]);
  ]]></mx:Script>

  <mx:Panel title="Line Chart with One Shadow">
     <mx:LineChart id="myChart" dataProvider="{expenses}">
        <mx:seriesFilters>
           <mx:Array/>
        </mx:seriesFilters>
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:LineSeries
                yField="Profit"
                displayName="Profit"
           />
           <mx:LineSeries
                yField="Expenses"
                displayName="Expenses"
           />
           <mx:LineSeries
                yField="Amount"
                displayName="Amount"
                lineSegmentRenderer=
                "mx.charts.renderers.ShadowLineRenderer"
            />
        </mx:series>
     </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

LineChart コントロールの線のセグメントなどの ChartItem オブジェクトの外観を変更するためにレンダラークラスを使用する際の詳細については、1617 ページの「ChartItem オブジェクトへのスキンの適用」を参照してください。

LineChart コントロールでの垂直線の使用

垂直方向に進行を示す LineChart コントロールを作成することもできます。次の例は、水平方向ではなく垂直方向に表示された、LineChart コントロールの 2 つの LineSeries を示しています。



LineChart コントロールの線が水平方向ではなく垂直方向に表示されるようにするには、次の操作を実行する必要があります。

- LineSeries オブジェクトの `xField` プロパティと `yField` プロパティを明示的に定義します。
- LineSeries オブジェクトの `sortOnXField` プロパティを `false` に設定します。

デフォルトでは、レンダリングの前に系列のデータポイントが (x 軸上の) 左から右にソートされます。この設定では、LineSeries は水平方向に描画します。`xField` ソートを無効にして `yField` プロパティを明示的に定義すると、線が水平方向ではなく垂直方向に描画されます。

Flex では、垂直方向のデータのソートは実行されません。そのため、データプロバイダ内のデータが正しい順になっていることを確認する必要があります。正しい順になっていない場合、点がデータプロバイダ内の位置に従って結ばれるため、上下にジグザグした線がレンダリングされます。

次の例では、垂直線を表示する LineChart コントロールを作成します。

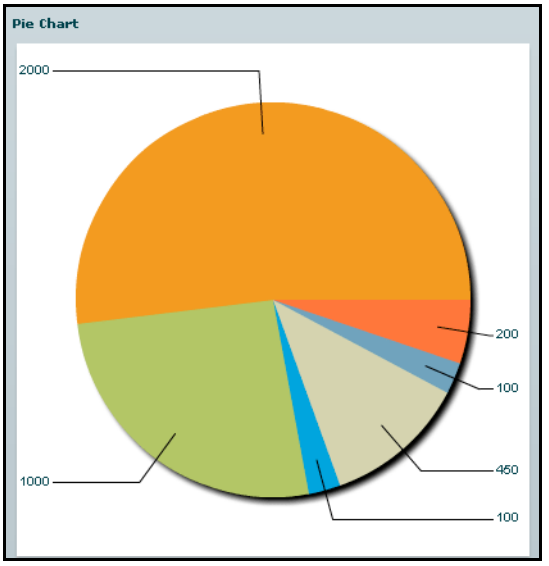
```
<?xml version="1.0"?>
<!-- charts/VerticalLineChart.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
```

```
    import mx.collections.ArrayCollection;
    [Bindable]
    public var expenses:ArrayCollection = new ArrayCollection([
       {Month:"Jan", Profit:2000, Expenses:1100},
       {Month:"Feb", Profit:1800, Expenses:1055},
       {Month:"Mar", Profit:1200, Expenses:800},
       {Month:"Apr", Profit:1400, Expenses:900},
       {Month:"May", Profit:1400, Expenses:1150},
       {Month:"Jun", Profit:1340, Expenses:600},
       {Month:"Jul", Profit:1600, Expenses:950},
       {Month:"Aug", Profit:1500, Expenses:1140},
       {Month:"Sep", Profit:1800, Expenses:1200},
       {Month:"Oct", Profit:2000, Expenses:1280},
       {Month:"Nov", Profit:2400, Expenses:1300},
       {Month:"Dec", Profit:1500, Expenses:500}
    ]);
  ]]></mx:Script>
  <mx:Panel title="Vertical Line Chart">
    <mx:LineChart id="myChart"
       dataProvider="{expenses}"
       showDataTips="true"
    >
       <mx:verticalAxis>
          <mx:CategoryAxis
               dataProvider="{expenses}"
               categoryField="Month"
          />
       </mx:verticalAxis>
       <mx:series>
          <mx:LineSeries
               xField="Profit"
               yField="Month"
               displayName="Profit"
               sortOnXField="false"
          />
          <mx:LineSeries
               xField="Expenses"
               yField="Month"
               displayName="Expenses"
               sortOnXField="false"
          />
       </mx:series>
    </mx:LineChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

円グラフの使用

標準的な円グラフを定義するには、PieChart コントロールを使用します。円グラフ内の各区分の大きさは、データプロバイダのデータにより、他の区分との関連で決定されます。

円グラフの例を次に示します。



円グラフのデータを定義するには、PieChart コントロールと共に PieSeries チャート系列を使用します。PieSeries では、標準的な円グラフまたはドーナツグラフを作成できます。また PieChart コントロールでは、データポイントを示すラベルを使用できます。

次の表で、円グラフの定義によく使用する PieChart コントロールの PieSeries チャート系列のプロパティについて説明します。

プロパティ	説明
`field`	円グラフの各区分のデータを決定するデータプロバイダのフィールドを指定します。
`labelPosition`	区分に使用するラベルのレンダリング方法を指定します。
`nameField`	データヒントと凡例で、各区分の名前として使用するデータプロバイダのフィールドを指定します。

次の例では、PieChart コントロールを定義します。

```
<?xml version="1.0"?>
<!-- charts/BasicPie.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1000},
        {Expense:"Bills", Amount:100},
        {Expense:"Car", Amount:450},
        {Expense:"Gas", Amount:100},
        {Expense:"Food", Amount:200}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Pie Chart">
     <mx:PieChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:series>
           <mx:PieSeries
                field="Amount"
                nameField="Expense"
                labelPosition="callout"
           />
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

PieChart コントロールでのラベルの使用

PieChart コントロールでは、各データポイントについての情報を表示するラベルを使用できます。すべてのチャートは、ユーザーがマウスを合わせたときにデータポイントの値を表示するデータヒントをサポートしています。ラベルは、データヒントとは異なり、マウスの操作に反応することはなく、常に表示されます。ラベルを使用できるチャートは、PieChart コントロールだけです。

PieChart コントロールにラベルを追加するには、データ系列の `labelPosition` プロパティを、`none` 以外の有効な値に設定します。円グラフからラベルを削除するには、`labelPosition` プロパティを `none` に設定します。デフォルト値は `none` です。

次の表で、labelPosition プロパティの有効な値について説明します。

値	説明
callout	PieChart コントロールのいずれかの側で、2 つの縦のスタック内にラベルを描画します。ラベル用の領域を空ける必要がある場合は PieChart を縮小します。各ラベルから関連するスライスに引き出し線を描画します。ラベルは、領域の大きさに合わせ、必要に応じて縮小されます。
inside	チャートの中にラベルを描画します。ラベルは、互いに重ならないように縮小されます。描画されたラベルが小さすぎる場合は、insideLabelSizeLimit プロパティでの定義に応じて、表示されないようになります。
insideWithCallout	ラベルを円の中に描画しますが、ラベルが読み取りできないサイズまで縮小された場合は、コールアウトラベルに変換されます。チャートの実際のサイズが変更可能で、ユーザーがサイズを変更する可能性がある場合、通常 labelPosition にはこの値を設定します。
none	ラベルを描画しません。これがデフォルト値です。
outside	PieChart コントロールの境界のまわりにラベルを描画します。

次の表では、ラベルの外観の操作に使用できる、PieChart コントロールのプロパティについて説明します。

プロパティ	説明
calloutGap	コールアウトを描画するときに円グラフの端とラベルとの間に挿入する隙間の幅を、ピクセル単位で定義します。デフォルト値は 10 ピクセルです。
calloutStroke	コールアウトの線の描画に使用される線のスタイルを定義します。線のデータポイントの定義の詳細については、1577 ページの「線の使用」を参照してください。
insideLabelSizeLimit	チャート内のラベルが何ポイント以下になったときに読み取り不可能と見なすかを定義します。このしきい値を下回った場合、ラベルは、データ系列の labelPosition プロパティの設定に基づき、完全に削除されるか、コールアウトに変化します。

ラベルテキストを変更するには、labelFunction プロパティを使用してコールバック関数を指定します。labelFunction で指定された関数は、円グラフの区分上にラベルとして表示されるストリングを返します。コールバック関数に必要な形式を以下に示します。

```
function function_name ( data, field, index, percentValue ): return_type { }
```

次の表で、labelFunction コールバック関数のパラメータについて説明します。

パラメータ	説明
data	この円グラフの区分によって表現されるデータポイントへの参照です。Object 型です。
field	データプロバイダのフィールド名です。String 型です。
index	データプロバイダに含まれるデータポイントの数です。Number 型です。
percentValue	円グラフ全体に対する区分のサイズです。区分のサイズが円グラフの 4 分の 1 の場合、この値は 25 です。Number 型です。

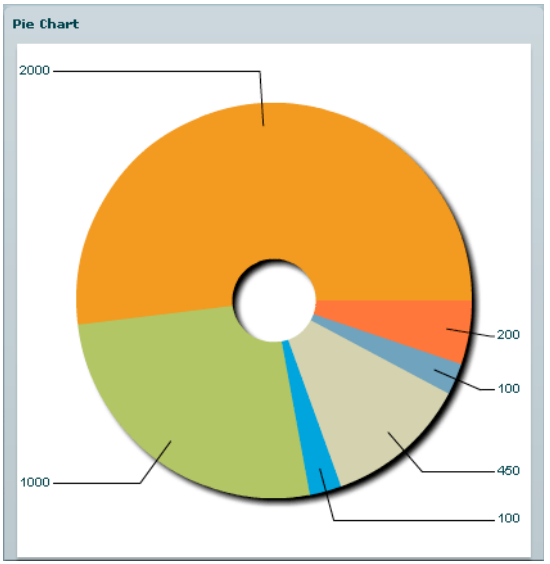
次の例では、データと書式設定を含むラベルを生成します。display() メソッドを labelFunction として定義し、ラベルテキストの書式設定を処理します。

```
<?xml version="1.0"?>
<!-- charts/PieLabelFunction.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.formatters.*;
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:1900000},
        {Expense:"Salaries", Amount:1350000},
        {Expense:"Building Rent", Amount:300000},
        {Expense:"Insurance", Amount:750000},
        {Expense:"Benefits", Amount:800000},
        {Expense:"Miscellaneous", Amount:900000}
     ]);
     public function display(
        data:Object,
        field:String,
        index:Number,
        percentValue:Number):String
     {
            return data.Expense + ":$" + data.Amount +
            "\n" + round(percentValue,2) + "%";
     }
     // Rounds to 2 places:
     public function round(num:Number, precision:Number):Number {
        var result:String;
        var f:NumberFormatter = new NumberFormatter();
        f.precision = precision;
        result = f.format(num);
        return Number(result);
     }
  ]]></mx:Script>
  <mx:Panel title="Expenditures for FY04">
```

```
      <mx:PieChart id="chart"
         dataProvider="{expenses}"
         showDataTips="false"
      >
         <mx:series>
            <mx:PieSeries
                  labelPosition="callout"
                  field="Amount"
                  labelFunction="display"
            />
         </mx:series>
      </mx:PieChart>
   </mx:Panel>
</mx:Application>
```

ドーナツグラフの作成

Flex では、PieChart コントロールからドーナツグラフを作成できます。ドーナツグラフは、円グラフと非常によく似ていますが、中央が白抜きになっていて、それを囲むように塗りつぶされた輪が存在するという点が異なります。ドーナツグラフの例を次に示します。



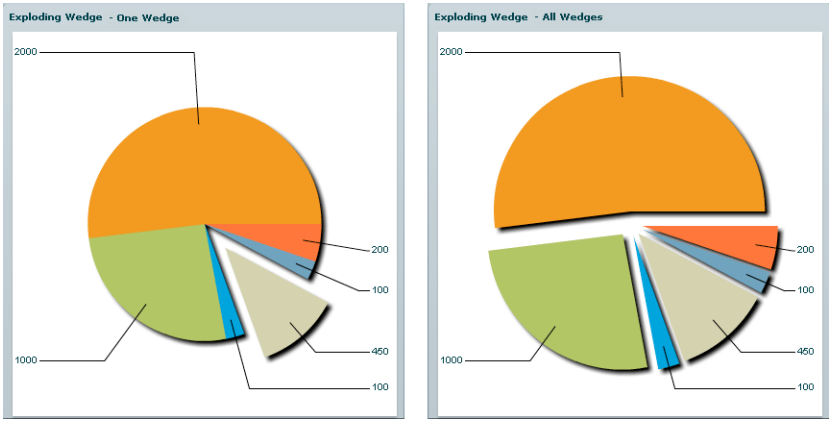
ドーナツグラフを作成するには、次の例のように、PieChart コントロールに対して innerRadius プロパティを指定します。

```
<?xml version="1.0"?>
<!-- charts/DoughnutPie.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1000},
        {Expense:"Bills", Amount:100},
        {Expense:"Car", Amount:450},
        {Expense:"Gas", Amount:100},
        {Expense:"Food", Amount:200}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Pie Chart">
     <mx:PieChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
        innerRadius=".3"
     >
        <mx:series>
           <mx:PieSeries
                field="Amount"
                nameField="Expense"
                labelPosition="callout"
           />
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

innerRadius プロパティの値は、円グラフ全体の半径に対する、" 白抜き部分 " のパーセント値です。有効な値の範囲は 0 から 1 です。

分離円グラフの作成

PieSeries チャート系列では、区分を一様に分離することも、区分ごとに分離することもできるので、次のようなチャートを作成することができます。



次の表で、分離円グラフをサポートするプロパティについて説明します。

プロパティ	説明
`explodeRadius`	0 ～ 1 までの値を指定します。円グラフの区分を分離する際に指定可能な半径を示すパーセント値です。
`perWedgeExplodeRadius`	0 ～ 100 までの値の配列を指定します。この配列の *N* 番目の値が `explodeRadius` の値に加算され、円グラフの各区分をどの程度分離するかが決定されます。それぞれの値を定義しないことも可能です。この場合、区分は `explodeRadius` プロパティに従って分離します。
`reserveExplodeRadius`	0 ～ 1 までの値を指定します。分離区分をアニメーション化する際に予約可能な半径を示しています。

円グラフの全区分を一様に分離するには、PieSeries で `explodeRadius` プロパティを使用します。次に例を示します。

```
<?xml version="1.0"?>
<!-- charts/ExplodingPie.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1000},
        {Expense:"Bills", Amount:100},
```

```
        {Expense:"Car", Amount:450},
        {Expense:"Gas", Amount:100},
        {Expense:"Food", Amount:200}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Exploding Pie Chart">
     <mx:PieChart id="pie"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:series>
           <!--explodeRadius is a number between 0 and 1.-->
           <mx:PieSeries
                field="Amount"
                nameField="Expense"
                explodeRadius=".12"
           />
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{pie}"/>
  </mx:Panel>
</mx:Application>
```

円グラフの区分を分離するには、explodeRadius 値の配列を使用します。配列の各値が、対応するデータポイントに適用されます。次の例では、4 番目のデータポイントである自動車 (Car) 費用の区分が分離されます。

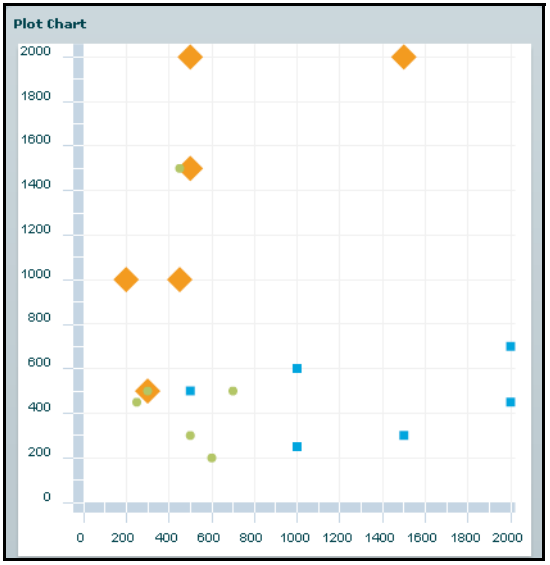
```
<?xml version="1.0"?>
<!-- charts/ExplodingPiePerWedge.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1000},
        {Expense:"Bills", Amount:100},
        {Expense:"Car", Amount:450},
        {Expense:"Gas", Amount:100},
        {Expense:"Food", Amount:200}
     ]);

     // Create a bindable Array of explode radii.
     [Bindable]
     public var explodingArray:Array = [0,0,0,.2,0,0]
  ]]></mx:Script>
  <mx:Panel title="Exploding Pie Chart Per Wedge">
     <mx:PieChart id="pie"
        dataProvider="{expenses}"
        showDataTips="true"
     >
```

```
        <mx:series>
           <!--Apply the Array of radii to the PieSeries.-->
           <mx:PieSeries
                field="Amount"
                nameField="Expense"
                perWedgeExplodeRadius="{explodingArray}"
                labelPosition="callout"
           />
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{pie}"/>
  </mx:Panel>
</mx:Application>
```

プロットチャートの使用

PlotChart コントロールを使用して、直交座標上でデータを表します。各データポイントには、x 軸上の位置を示す値と y 軸上の位置を示す値が 1 つずつあります。各データポイントに表示されるシェイプは、データ系列のレンダラーを使用して定義できます。

次の図は、プロットチャートの例を示しています。



プロットチャートのデータを定義するには、PlotChart コントロールと共に PlotSeries クラスを使用します。次の表で、プロットチャートの定義によく使用する PlotSeries チャート系列のプロパティについて説明します。

プロパティ	説明
yField	各データポイントの y 軸の位置を決定するデータプロバイダのフィールドを指定します。
xField	各データポイントの x 軸の位置を決定するデータプロバイダのフィールドを指定します。
radius	各データポイントに表示するシンボルの半径をピクセル単位で指定します。デフォルト値は 5 ピクセルです。

メモ

PlotChart コントロールでは、それぞれの PlotSeries に xField プロパティと **yField** プロパティの両方を指定する必要があります。

次の例では、PlotChart コントロールでデータ系列を 3 つ定義します。

```
<?xml version="1.0"?>
<!-- charts/BasicPlot.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Plot Chart">
     <mx:PlotChart id="myChart" dataProvider="{expenses}"
     showDataTips="true">
        <mx:series>
           <mx:PlotSeries
                xField="Expenses"
                yField="Profit"
                displayName="Plot 1"
           />
           <mx:PlotSeries
                xField="Amount"
                yField="Expenses"
                displayName="Plot 2"
           />
```

```
            <mx:PlotSeries
                 xField="Profit"
                 yField="Amount"
                 displayName="Plot 3"
            />
        </mx:series>
    </mx:PlotChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

デフォルトでは、チャートにおける最初のデータ系列の各ポイントがダイアモンドで表示されます。チャートにデータ系列を複数定義した場合、データ系列のシェイプが、ダイアモンド、円、四角形の順で循環して使用されます。データ系列がデフォルトのレンダラーより多い場合は、再度ダイアモンドから使用されます。

ダイアモンドのシェイプは、他のシェイプと同様にレンダラークラスで定義されています。これらのシェイプを定義しているレンダラークラスは、mx.charts.renderers パッケージの中にあります。円は CircleItemRenderer クラスによって定義されています。次のデフォルトのレンダラークラスは、データポイントの外観を定義します。

- BoxItemRenderer
- CircleItemRenderer
- CrossItemRenderer
- DiamondItemRenderer
- ShadowBoxItemRenderer
- TriangleItemRenderer

チャートに表示される各データポイントのイメージは、系列の itemRenderer スタイルプロパティを設定することによって制御できます。次の例では、系列のデフォルトのレンダラーをオーバーライドします。

```
<?xml version="1.0"?>
<!-- charts/PlotWithCustomRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ]);
```

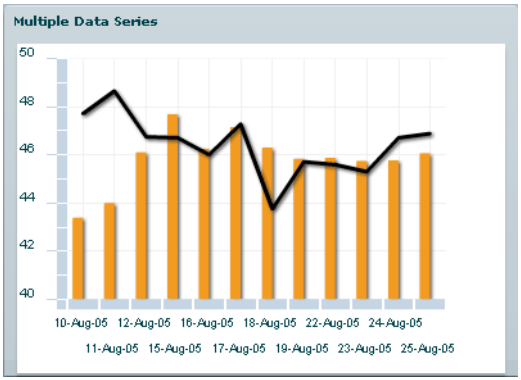
```
    ]]>
  </mx:Script>

  <mx:Panel title="Plot Chart With Custom Item Renderer">
     <mx:PlotChart id="myChart" dataProvider="{expenses}"
     showDataTips="true">
        <mx:series>
           <mx:PlotSeries
                xField="Expenses"
                yField="Profit"
                displayName="Plot 1"
                itemRenderer="mx.charts.renderers.CrossItemRenderer"
                radius="10"
           />
           <mx:PlotSeries
                xField="Amount"
                yField="Expenses"
                displayName="Plot 2"
                itemRenderer="mx.charts.renderers.DiamondItemRenderer"
                radius="10"
           />
           <mx:PlotSeries
                xField="Profit"
                yField="Amount"
                displayName="Plot 3"
                itemRenderer=
                "mx.charts.renderers.TriangleItemRenderer"
                radius="10"
           />
        </mx:series>
     </mx:PlotChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

グラフィックやカスタムクラスを使用して、プロットの各ポイントを定義することもできます。詳細については、1619 ページの「カスタムレンダラーの作成」を参照してください。

複数のデータ系列の使用

CartesianChart クラスのサブクラスであるチャートを使用すると、同じチャートコントロール内で、異なるデータ系列を混在させることができます。縦棒グラフに傾向線を重ねて表示することや、あらゆるデータ系列をそれ以外の類似する系列と混在させることができます。

次のグラフでは 2 つの株を追跡しています。1 つは縦棒で、もう 1 つは折れ線で示しています。



CartesianChart コントロールでは、次の系列オブジェクトを自由に組み合わせて使用できます。

- AreaSeries
- BarSeries
- BubbleSeries
- CandlestickSeries
- ColumnSeries
- HLOCSeries
- LineSeries
- PlotSeries

次の例では、LineSeries と ColumnSeries が混在しています。

```
<?xml version="1.0"?>
<!-- charts/MultipleSeries.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="500"
height="600">
  <mx:Script>
    <![CDATA[
        [Bindable]
        public var SMITH:Array = [
           {date:"22-Aug-05", close:45.87},
           {date:"23-Aug-05", close:45.74},
           {date:"24-Aug-05", close:45.77},
           {date:"25-Aug-05", close:46.06},
```

```
        ];
        [Bindable]
        public var DECKER:Array = [
           {date:"22-Aug-05", close:45.59},
           {date:"23-Aug-05", close:45.3},
           {date:"24-Aug-05", close:46.71},
           {date:"25-Aug-05", close:46.88},
        ];
    ]]>
  </mx:Script>

  <mx:Panel title="Multiple Data Series" width="400" height="400">
     <mx:ColumnChart id="mychart"
        dataProvider="{SMITH}"
        showDataTips="true"
        height="250"
        width="350"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="date"/>
        </mx:horizontalAxis>
        <mx:verticalAxis>
           <mx:LinearAxis minimum="40" maximum="50"/>
        </mx:verticalAxis>
        <mx:series>
           <mx:ColumnSeries
                dataProvider="{SMITH}"
                xField="date"
                yField="close"
                displayName="SMITH"
           >
           </mx:ColumnSeries>
           <mx:LineSeries
                dataProvider="{DECKER}"
                xField="date"
                yField="close"
                displayName="DECKER"
           >
           </mx:LineSeries>
        </mx:series>
     </mx:ColumnChart>
  </mx:Panel>
</mx:Application>
```

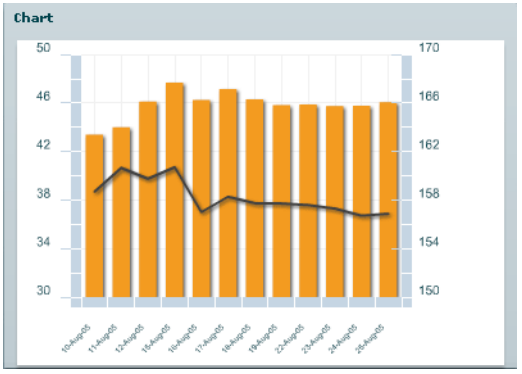
株価とその移動平均を示す場合など、データポイントの数値の範囲がそれほど変動しないときに、同じチャート内で複数のデータ系列を使用することをお勧めします。データポイントの数値の範囲が大幅に異なる場合は、データが1本の軸で表示されるため、チャートがわかりにくくなることがあります。この問題は、複数の軸を使用して、各軸でそれぞれの範囲を示すことによって解決できます。同じチャート内で、各データ系列をそれぞれの軸で描画するには、1526 ページの「複数の軸の使用」で説明している手法を使用します。

複数の軸の使用

1つのチャートで2つのデータ系列を使用すると、データの縮尺が大幅に異なる場合に、それぞれのデータポイントがチャートキャンバス上で離れた領域に描画される可能性があります。たとえば、$100 ～ $150 の範囲で取引されている株価と、$2 ～ $2.50 で取引されている株価があるとします。これらの株価を同じチャート内に描画すると、対数軸を使用しても、価格間の相互関係がわかりにくくなります。

この問題を回避するには、各データ系列がそれぞれの軸に相対的に配置されるように、チャートで複数の軸を使用します。CartesianChart のサブクラスのチャートコントロールでは、水平軸か垂直軸、またはその両方に、2番目のデータセットを別の縮尺で追加することができます。この機能は、PieChart コントロール以外のすべてのチャートに適用されます。2番目の軸にある値を使用して、異なる範囲で取引されている株価など、縮尺が異なる2つのデータセットを比較することができます。

次の例では、$40 ～ $45 で取引されている株価と、$150 ～ $160 で取引されている株価を示します。左の軸の値は、最初の銘柄の株価の範囲を示しています。右側の軸の値は、2つ目の銘柄の株価の範囲を示しています。



CartesianChart のサブクラスであるチャートでは、デフォルトで1つのデータセットと、水平軸と垂直軸が1つずつ表示されます。`secondDataProvider`、`secondSeries`、`secondVerticalAxis`、`secondHorizontalAxis`、`secondVerticalAxisRenderer`、`secondHorizontalAxisRenderer` など、2つ目のプロパティを設定すると、チャートに次のような効果を与えることができます。

ColumnChart、LineChart、および AreaChart コントロール　デフォルトでは、1番目と2番目のデータ系列は `horizontalAxis` を共有しますが、それぞれの `verticalAxis` では異なる LinearAxis を取得します。デフォルトでは、複数の軸を使用する際に、2番目の `verticalAxisRenderer` が作成されます。

BarChart コントロール　1番目と2番目のデータ系列は `verticalAxis` を共有しますが、デフォルトではそれぞれのデータ系列に異なる `horizontalAxis` が存在します。デフォルトでは、2つの `horizontalAxisRenderers` が作成されます。

PlotChart および BubbleChart コントロール　1番目と2番目のデータ系列には、異なる水平軸と垂直軸、および4つの軸レンダラーすべてが存在します。バブルチャートでは、すべてのデータ系列で1つの radialAxis が共有されます。

CartesianChart コントロール　デフォルトでは、Cartesian チャートには追加の軸または軸レンダラーはありません。

PieChart コントロール　2番目のデータ系列をグラフに含めることはできません。

別の縮尺でレンダリングされた系列オブジェクトの配列を指定するには、<mx:secondSeries> タグを使用します。この配列のデータ系列に適用されるフィールドは、secondDataProvider のフィールドを参照します。

secondSeries プロパティで提供されるデータ系列によって使用されるデータプロバイダには、secondDataProvider プロパティを使用します。

次の例では、2つのデータ系列を使用して、株価が異なる範囲で取引される2つの銘柄を視覚的に比較しています。ここでは、水平値として1つの categoryField を horizontalAxis に使用しています。これにより、verticalAxis と secondVerticalAxis の目盛りの最小範囲と最大範囲が設定されます。また secondVerticalAxisRenderer を使用して、2番目の垂直軸の外観を変更しています。

```
<?xml version="1.0"?>
<!-- charts/MultipleAxes.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
      public var SMITH:ArrayCollection = new ArrayCollection([
        {date:"22-Aug-05", close:41.87},
        {date:"23-Aug-05", close:45.74},
        {date:"24-Aug-05", close:42.77},
        {date:"25-Aug-05", close:48.06},
     ]);

     [Bindable]
      public var DECKER:ArrayCollection = new ArrayCollection([
        {date:"22-Aug-05", close:157.59},
        {date:"23-Aug-05", close:160.3},
        {date:"24-Aug-05", close:150.71},
        {date:"25-Aug-05", close:156.88},
     ]);

  ]]></mx:Script>

  <mx:Panel title="Column Chart With Multiple Series">
     <mx:ColumnChart id="myChart"
        dataProvider="{SMITH}"
        secondDataProvider="{DECKER}"
```

```
        showDataTips="true"
    >

        <mx:horizontalAxis>
            <mx:CategoryAxis
                dataProvider="{SMITH}"
                categoryField="date"
            />
        </mx:horizontalAxis>

        <mx:verticalAxis>
            <mx:LinearAxis minimum="40" maximum="50"/>
        </mx:verticalAxis>

        <mx:series>
            <mx:ColumnSeries id="cs1"
                dataProvider="{SMITH}"
                xField="date"
                yField="close"
                displayName="SMITH"
            />
        </mx:series>

        <mx:secondVerticalAxis>
            <mx:LinearAxis minimum="150" maximum="170"/>
        </mx:secondVerticalAxis>

        <mx:secondSeries>
            <mx:LineSeries id="cs2"
                dataProvider="{DECKER}"
                xField="date"
                yField="close"
                displayName="DECKER"
            />
        </mx:secondSeries>
    </mx:ColumnChart>
  </mx:Panel>
</mx:Application>
```

この例には、凡例が含まれていません。複数の軸を使用するチャートを作成するときは、標準の <mx:Legend> タグを使用してチャートの凡例を作成することはできません。カスタムの凡例を作成する必要があります。詳細については、1628 ページの「Legend カスタムコントロールの作成」を参照してください。

2 番目の軸の使用

チャートタイプによっては、2 番目のデータ系列が 1 番目のデータ系列と同じ軸を共有したり、デフォルトである別の 2 番目の軸に対してレンダリングされることがあります。水平軸と垂直軸を含むチャートでは、secondHorizontalAxis プロパティと secondVerticalAxis プロパティを指定することで、2 番目のデータ系列がサポートされます。2 番目の水平軸または垂直軸を明示的に指定した場合、チャートタイプに関係なく、それらが 2 番目のデータ系列で使用されます。

次の例では、secondVerticalAxis を LogAxis グラフ軸として定義しています。

```
<mx:secondVerticalAxis>
  <mx:LogAxis/>
</mx:secondVerticalAxis>
```

1 番目の軸と同様に、2 番目の軸の外観もレンダラーで変更できます。この場合、<mx:secondVerticalAxisRenderer> タグと <mx:secondHorizontalAxisRenderer> タグを使用します。次の例では、2 番目の軸の目盛り位置を設定しています。

```
<mx:secondVerticalAxisRenderer>
  <mx:AxisRenderer placement="left" tickPlacement="inside"/>
</mx:secondVerticalAxisRenderer>
```

軸の書式設定の詳細については、1549 ページの「軸の操作」を参照してください。

複数の軸レンダラーの使用

チャートに垂直方向または水平方向の縮尺が異なるデータが複数表示されている場合、複数の軸もレンダリングする必要があります。特定の軸レンダラーを表示する場所を指定するには、軸レンダラーの placement プロパティを設定します。有効な値は、left、top、right、および bottom です。1 番目のレンダラーの placement プロパティのデフォルト値は、垂直レンダラーでは left、水平レンダラーでは bottom です。2 番目のレンダラーの場合、デフォルト値は right と top です。

2 番目の軸の外観を制御するには、次の例に示すように、secondHorizontalAxisRenderer プロパティおよび secondVerticalAxisRenderer プロパティを使用します。

```
<?xml version="1.0"?>
<!-- charts/MultipleAxesMultipleRenderers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="600">
  <mx:Script><![CDATA[
  [Bindable]
   public var SMITH:Array = [
     {date: "22-Aug-05", close: 45.87},
     {date: "23-Aug-05", close: 45.74},
     {date: "24-Aug-05", close: 45.77},
     {date: "25-Aug-05", close: 46.06},
  ];
  [Bindable]
  public var DECKER:Array = [
```

```
    {date: "22-Aug-05", close: 157.59},
    {date: "23-Aug-05", close: 157.3},
    {date: "24-Aug-05", close: 156.71},
    {date: "25-Aug-05", close: 156.88},
];
]]></mx:Script>
<mx:Panel title="Chart" width="400" height="400">
    <mx:ColumnChart id="mychart"
        dataProvider="{SMITH}"
        secondDataProvider="{DECKER}"
        showDataTips="true"
        height="250"
        width="350"
    >
        <mx:horizontalAxis>
            <mx:CategoryAxis
                 dataProvider="{SMITH}"
                 categoryField="date"
            />
        </mx:horizontalAxis>

        <mx:verticalAxis>
            <mx:LinearAxis minimum="30" maximum="50"/>
        </mx:verticalAxis>

        <mx:series>
            <mx:ColumnSeries
                 dataProvider="{SMITH}"
                 xField="date"
                 yField="close"
                 displayName="SMITH"
            />
        </mx:series>

        <mx:secondVerticalAxis>
            <mx:LinearAxis minimum="150" maximum="170"/>
        </mx:secondVerticalAxis>

        <mx:verticalAxisRenderer>
            <mx:AxisRenderer
             placement="right"
             tickPlacement="inside"
            >
             <mx:axisStroke>
                 <mx:Stroke color="#CC6699" weight="1"/>
             </mx:axisStroke>
             <mx:tickStroke>
                 <mx:Stroke color="#CC0099" weight="1"/>
             </mx:tickStroke>
             <mx:minorTickStroke>
```

```
                <mx:Stroke color="#990066" weight="1"/>
            </mx:minorTickStroke>
          </mx:AxisRenderer>
        </mx:verticalAxisRenderer>

        <mx:secondVerticalAxisRenderer>
          <mx:AxisRenderer
            placement="left"
            tickPlacement="inside"
          >
            <mx:axisStroke>
                <mx:Stroke color="#000080" weight="1"/>
            </mx:axisStroke>
            <mx:tickStroke>
                <mx:Stroke color="#000160" weight="1"/>
            </mx:tickStroke>
            <mx:minorTickStroke>
                <mx:Stroke color="#100040" weight="1"/>
            </mx:minorTickStroke>
          </mx:AxisRenderer>
        </mx:secondVerticalAxisRenderer>

        <mx:secondSeries>
          <mx:LineSeries
                dataProvider="{DECKER}"
                xField="date"
                yField="close"
                displayName="DECKER"
          />
        </mx:secondSeries>

    </mx:ColumnChart>
  </mx:Panel>
</mx:Application>
```

チャートでは、2番目の軸レンダラーの placement プロパティが、1番目の軸レンダラーの反対側に表示されるように自動的に設定されるため、軸レンダラーが互いに重なり合うことはありません。verticalAxisRenderer で placement プロパティを left に設定すると、secondVerticalAxisRenderer は右側に表示されます。逆の場合も同様です。

第 55 章

チャートの書式設定

55

Adobe Flex チャートコンポーネントの外観をカスタマイズできます。軸ラベルのフォントプロパティから凡例の線の幅まで、ほぼすべてのチャートエレメントの外観を変更することができます。

チャートコントロールのスタイルプロパティを設定するには、カスケーディングスタイルシート (CSS) のシンタックスを使用するか、スタイルプロパティをタグ属性としてインラインで設定します。他のすべてのスタイルと同様、`setStyle()` メソッドを呼び出して、チャートエレメントに任意のスタイルを設定できます。`setStyle()` メソッドの使用の詳細については、682 ページの「setStyle() メソッドと getStyle() メソッドの使用」を参照してください。

目次

チャートスタイルの適用

スタイルをチャートに適用するには、CSS またはインラインシンタックスを使用します。バインディングを使用しても、チャートにスタイルを適用できます。

CSS を使用したスタイルの適用

CSS 定義を使用して、Flex チャートコンポーネントにスタイルを適用することができます。フォントや目盛りなどのチャートプロパティや、ColumnChart のボックスの塗りつぶしなどの系列のプロパティを設定できます。

CSS を使用してチャートにスタイルを適用する場合の問題は、スタイル適用可能なチャートプロパティでは、線やグラデーションの塗りなどの CSS で表現できない複合値が多用されることです。そのため、CSS シンタックスではチャートスタイルのすべての値を表現できません。

チャートコントロールへの CSS の適用

CSS でコントロール名を使用することによって、コントロールのスタイルを定義できます。このスタイルはタイプセレクタと呼ばれます。これは、定義するスタイルがそのタイプのすべてのコントロールに適用されるためです。たとえば、次のスタイル定義では、すべての BubbleChart コントロールのフォントを定義しています。

```
<?xml version="1.0"?>
<!-- charts/BubbleStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Style>
      BubbleChart {
         fontFamily:Arial;
         fontSize:20;
         color:#FF0033;
      }
    </mx:Style>
    <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:120, Amount:45},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:60},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:30}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Bubble Chart">
     <mx:BubbleChart id="myChart"
        maxRadius="50"
        dataProvider="{expenses}"
        showDataTips="true"
```

```
    >
       <mx:series>
          <mx:BubbleSeries
               xField="Profit"
               yField="Expenses"
               radiusField="Amount"
          />
       </mx:series>
    </mx:BubbleChart>
  </mx:Panel>
</mx:Application>
```

fontSize や fontFamily など、継承可能なスタイルも一部存在します。つまり、チャートコントロール、軸ラベル、タイトルなど、チャート上のテキストエレメントにこれらのスタイルを設定した場合、これらの設定が継承されます。スタイルが継承可能かどうかについては、『Adobe Flex 2 リファレンスガイド』で該当するスタイルの説明を参照してください。

軸ラベルは、チャートの軸に沿った目盛りの隣に表示されます。タイトルは、軸線に並行に表示されます。デフォルトでは、軸上のテキストにはチャートコントロールのスタイルが適用されます。

軸のエレメントでは、チャートコントロールのタイプセレクタまたはクラスセレクタのあらゆるスタイルが使用されますが、軸レンダラーのクラスセレクタまたは定義済みの軸スタイルプロパティを使用して、異なるスタイルを指定することもできます。

各系列への異なるスタイルの適用

CSS を使用してチャートのそれぞれの系列に異なるスタイルを適用するには、chartSeriesStyles プロパティを使用します。このプロパティはストリングの配列を受け取ります。各ストリングにスタイルシート内のクラスセレクタの名前を指定すると、それらのクラスセレクタがそれぞれ系列に適用されます。

CSS を系列に適用するには、chartSeriesStyles プロパティを定義するチャートに、タイプセレクタまたはクラスセレクタを定義します。次に、chartSeriesStyles プロパティで名前を付けた各クラスセレクタを定義します。

実質的には、チャートの系列ごとに新しいスタイルを定義することになります。たとえば、系列が 2 つある ColumnChart コントロールを使用する場合、明示的に指定しなくても、各系列に異なるスタイルを適用できます。

次の例では、ColumnChart コントロールの 2 つの系列に色を定義しています。

```
<?xml version="1.0"?>
<!-- charts/SeriesStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
     ColumnChart {
        chartSeriesStyles: PCCSeries1, PCCSeries2;
     }
```

```
    .PCCSeries1 {
       fill: #CCFF66;
    }
    .PCCSeries2 {
       fill: #CCFF99;
    }
  </mx:Style>

  <mx:Script>
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month: "Jan", Profit: 2000, Expenses: 1500},
     {Month: "Feb", Profit: 1000, Expenses: 200},
     {Month: "Mar", Profit: 1500, Expenses: 500}
  ]);
  </mx:Script>

  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

Flex によって、各系列の styleName プロパティが chartSeriesStyles 配列の対応するセレクタに設定されます。各系列のスタイルを明示的に設定する必要はありません。使用できる chartSeriesStyles セレクタより系列の数の方が多い場合、再度最初のスタイルから適用されます。

特定の系列に styleName プロパティの値を手動で設定すると、そのスタイルは、chartSeriesStyles プロパティで指定しているスタイルより優先的に使用されます。

PieChart コントロールの場合、系列の fills プロパティを定義できます。PieChart は、円グラフの区分ごとにこの配列内で各値を適用します。配列内で定義されている値よりも多くの区分がある場合、PieChart は最初の値で始まります。次の例では、区分に赤、白、および青のみを使用する PieChart を作成します。

```
<?xml version="1.0"?>
<!-- charts/PieWedgeFills.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
      PieSeries {
         fills:#FF0000, #FFFFFF, #006699;
      }
  </mx:Style>

  <mx:Script>
    <![CDATA[
        import mx.collections.ArrayCollection;
        [Bindable]
        public var expenses:ArrayCollection = new ArrayCollection([
           {Expense:"Taxes", Amount:2000},
           {Expense:"Rent", Amount:1000},
           {Expense:"Bills", Amount:100},
           {Expense:"Car", Amount:450},
           {Expense:"Gas", Amount:100},
           {Expense:"Food", Amount:200}
        ]);
    ]]>
  </mx:Script>
  <mx:Panel>
     <mx:PieChart id="pie"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:series>
           <mx:PieSeries
                field="Amount"
                nameField="Expense"
                labelPosition="callout"
           />
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{pie}"/>
  </mx:Panel>
</mx:Application>
```

定義済みの軸スタイルプロパティを使用する

軸スタイルには、次の定義済みクラスセレクタを使用できます。

- horizontalAxisStyleName
- verticalAxisStyleName
- secondHorizontalAxisStyleName
- secondVerticalAxisStyleName

各スタイルが、対応する軸に適用されます。

軸スタイルプロパティのこれらのクラスセレクタ以外に、axisTitleStyleName および gridLinesStylesName クラスセレクタを使用して、軸タイトルとグリッド線にスタイルを適用することもできます。

次の例では、水平軸の目盛りを非表示にします。

```
<?xml version="1.0"?>
<!-- charts/PredefinedAxisStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     ColumnChart {
        horizontalAxisStyleName:myAxisStyles;
        verticalAxisStyleName:myAxisStyles;
     }

     .myAxisStyles {
        tickPlacement:none;
     }
  </mx:Style>

  <mx:Script>
    <![CDATA[
        import mx.collections.ArrayCollection;
        [Bindable]
        public var expenses:ArrayCollection = new ArrayCollection([
           {Month:"Jan", Profit:2000, Expenses:1500},
           {Month:"Feb", Profit:1000, Expenses:200},
           {Month:"Mar", Profit:1500, Expenses:500}
        ]);
    ]]>
  </mx:Script>

  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
```

```
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
               displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

クラスセレクタを使用して軸スタイルを設定する

クラスセレクタを使って軸のエレメントに使用するスタイルを定義するには、<mx:Style> ブロックまたは外部のスタイルシートにカスタムのクラスセレクタを定義し、AxisRenderer クラスの styleName プロパティを使用して、そのクラスセレクタを参照します。

次の例では、MyStyle スタイルを定義し、そのスタイルを水平軸のエレメントに適用しています。

```
<?xml version="1.0"?>
<!-- charts/AxisClassSelectors.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Style>
  .myStyle {
     fontSize:7;
     color:red;
  }
</mx:Style>

  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month:"Jan",Profit:2000,Expenses:1500},
     {Month:"Feb",Profit:1000,Expenses:200},
     {Month:"Mar",Profit:1500,Expenses:500}
  ]);
  ]]></mx:Script>

  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
```

```
        <mx:horizontalAxisRenderer>
           <mx:AxisRenderer styleName="myStyle"/>
        </mx:horizontalAxisRenderer>

        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>

        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

チャート特有のスタイルプロパティは、ほとんどの場合継承できません。これは、親オブジェクトでこのプロパティを設定しても、その値は子オブジェクトに引き継がれないということです。

CSS の使用の詳細については、647 ページの「スタイルについて」を参照してください。

インラインでのスタイルの適用

MXML タグの属性と同様に、スタイル適用可能なさまざまなエレメントをチャートエレメントにも設定できます。たとえば、軸に対してスタイル適用可能なプロパティを設定するには、CSS に新しいスタイルを定義する代わりに、<mx:AxisRenderer> タグを使用します。

次の例では、水平軸の fontSize プロパティを 7 に設定しています。

```
<mx:horizontalAxisRenderer>
  <mx:AxisRenderer fontSize="7"/>
</mx:horizontalAxisRenderer>
```

ActionScript でレンダラーのプロパティにアクセスすることによって、実行時に外観を変更することもできます。軸レンダラーの詳細については、1549 ページの「軸の操作」を参照してください。

タグ定義のバインディングによるスタイルの適用

MXML タグを使用してスタイルを定義することができます。その後、レンダラープロパティの値をそれらのタグにバインドできます。次の例では、線の太さと色を定義し、それらの線をチャートの AxisRenderer クラスに適用しています。

```
<?xml version="1.0"?>
<!-- charts/BindStyleValues.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  [Bindable]
  public var aapl:Array = [
     {date:"1-Aug-05",open:42.57,high:43.08,low:42.08,close:
     42.75},
     {date:"2-Aug-05",open:42.89,high:43.5,low:42.61,close:
     43.19},
     {date:"3-Aug-05",open:43.19,high:43.31,low:42.77,close:
     43.22},
     {date:"4-Aug-05",open:42.89,high:43,low:42.29,close:42.71},
     {date:"5-Aug-05",open:42.49,high:43.36,low:42.02,close:
     42.99},
     {date:"8-Aug-05",open:43,high:43.25,low:42.61,close:42.65},
     {date:"9-Aug-05",open:42.93,high:43.89,low:42.91,close:
     43.82},
     {date:"10-Aug-05",open:44,high:44.39,low:43.31,close:43.38},
     {date:"11-Aug-05",open:43.39,high:44.12,low:43.25,close:44},
     {date:"12-Aug-05",open:43.46,high:46.22,low:43.36,close:
     46.1},
  ];
  ]]></mx:Script>

  <mx:Stroke color="0x00FF00" weight="2" id="axis"/>
  <mx:Stroke color="0xFF0000" weight="1" id="ticks"/>
  <mx:Stroke color="0x0000FF" weight="1" id="mticks"/>

  <mx:HLOCChart id="mychart"
    dataProvider="{aapl}"
    showDataTips="true"
  >
     <mx:horizontalAxisRenderer>
        <mx:AxisRenderer
            axisStroke="{axis}"
            placement="bottom"
            minorTickPlacement="inside"
            minorTickLength="2"
            tickLength="5"
            tickPlacement="inside"
        >
           <mx:tickStroke>{ticks}</mx:tickStroke>
           <mx:minorTickStroke>{mticks}</mx:minorTickStroke>
```

```
        </mx:AxisRenderer>
    </mx:horizontalAxisRenderer>

    <mx:verticalAxis>
        <mx:LinearAxis minimum="30" maximum="50"/>
    </mx:verticalAxis>
    <mx:series>
        <mx:HLOCSeries
            dataProvider="{aapl}"
            openField="open"
            highField="high"
            lowField="low"
            closeField="close"
            displayName="AAPL"
        >
        </mx:HLOCSeries>
    </mx:series>
  </mx:HLOCChart>
  <mx:Legend dataProvider="{mychart}"/>
</mx:Application>
```

ChartElement オブジェクトの追加

ChartElement クラスは、チャートのデータ領域に表示されるあらゆる要素の基本クラスです。GridLines オブジェクトなどの系列オブジェクトはすべて ChartElement オブジェクトです。イメージ、グリッド線、線などの ChartElement オブジェクトをチャートに追加するには、チャートクラスの backgroundElements プロパティと annotationElements プロパティを使用します。

backgroundElements プロパティでは、チャートによってレンダリングされるデータ系列の下に表示される ChartElement オブジェクトの配列を指定します。annotationElements プロパティでは、チャートによってレンダリングされるデータ系列の上に表示される ChartElement オブジェクトの配列を指定します。

GIF、SVG、JPEG などのサポートされるイメージファイルも、ChartElement オブジェクトとしてチャートに追加できます。

次の例では、注釈エレメントとして新しいグリッド線、背景エレメントとしてイメージをチャートに追加しています。ユーザーがボタンをクリックすると、注釈エレメントが変化します。

```
<?xml version="1.0"?>
<!-- charts/AnnotationElements.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  import mx.graphics.SolidColor;
  import mx.charts.GridLines;

  [Bindable]
```

```
public var expenses:ArrayCollection = new ArrayCollection([
   {Month:"Jan", Profit:2000, Expenses:1500},
   {Month:"Feb", Profit:1000, Expenses:200},
   {Month:"Mar", Profit:1500, Expenses:500}
]);

[Embed(source="../assets/bird.gif")]
public var bird:Class;

public function updateGridLines():void {
   var bgi:GridLines = new GridLines();

   var s:Stroke = new Stroke(0xff00ff, 3);
   bgi.setStyle("horizontalStroke",s);

   var c:SolidColor = new SolidColor(0x990033, .2);
   bgi.setStyle("horizontalFill",c);

   var c2:SolidColor = new SolidColor(0x999933, .2);
   bgi.setStyle("horizontalAlternateFill",c2);

   myChart.annotationElements = [bgi]

   var b:Object = new bird();
   b.alpha = .2;
   b.height = 150;
   b.width = 150;

   myChart.backgroundElements = [ b ];
}
]]></mx:Script>
<mx:Panel>
   <mx:ColumnChart id="myChart" dataProvider="{expenses}">
      <mx:horizontalAxis>
         <mx:CategoryAxis
              dataProvider="{expenses}"
              categoryField="Month"
         />
      </mx:horizontalAxis>
      <mx:series>
         <mx:ColumnSeries
              xField="Month"
              yField="Profit"
              displayName="Profit"
         />
         <mx:ColumnSeries
              xField="Month"
              yField="Expenses"
              displayName="Expenses"
         />
```

```
        </mx:series>

        <mx:annotationElements>
           <mx:GridLines>
            <mx:horizontalStroke>
                <mx:Stroke
                    color="#191970"
                    weight="2"
                    alpha=".3"
                />
            </mx:horizontalStroke>
           </mx:GridLines>
        </mx:annotationElements>

        <mx:backgroundElements>
           <mx:Image
                source="@Embed('../assets/bird.gif')"
                alpha=".2"
           />
        </mx:backgroundElements>

     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
     <mx:Button id="b1"
        click="updateGridLines()"
        label="Update Grid Lines"
     />
  </mx:Panel>
</mx:Application>
```

ActionScript では、次の例のように、イメージを追加してそのプロパティを操作することができます。

```
<?xml version="1.0"?>
<!-- charts/BackgroundElementsWithActionScript.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="addBird()">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  import mx.graphics.SolidColor;
  import mx.charts.GridLines;

  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month:"Jan", Profit:2000, Expenses:1500},
     {Month:"Feb", Profit:1000, Expenses:200},
     {Month:"Mar", Profit:1500, Expenses:500}
  ]);

  [Embed(source="../assets/bird.gif")]
  public var bird:Class;
```

```
  public function addBird():void {
     var b:Object = new bird();
     b.alpha = .2;
     myChart.backgroundElements = [ b ];
  }
  ]]></mx:Script>
  <mx:Panel>
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

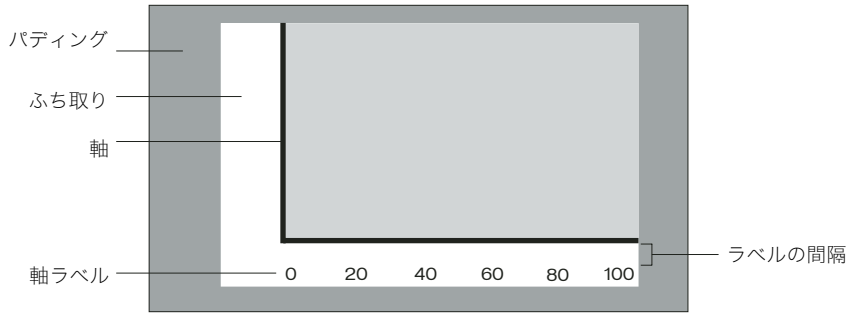
パディングプロパティの設定

他の Flex コンポーネントと同様、チャートのパディングプロパティは、チャートコントロールの外枠とコントロールの内容との間の余白を定義します。Flex では、パディング領域の背景の塗りのみを描画します。

チャートコントロールのパディングの値は、UIComponent クラスから継承される paddingLeft プロパティと paddingRight プロパティを使用して設定できます。ChartBase クラスから継承される paddingTop プロパティと paddingBottom プロパティを使用することもできます。

Flex チャートエレメントには " ふち取り " があります。ふち取りとは、パディング領域と実際の軸線との間の余白のことです。Flex の軸ラベル、タイトル、および目盛りは、チャートのふち取り部分に描画されます。チャートコントロールでは、これらの拡張機能が収まるようにふち取りの値を調整しますが、この値を明示的に指定することもできます。

次の例では、チャートのふち取りとパディングの位置を示しています。



チャートのふち取りのサイズは、次のスタイルプロパティで定義されます。

- gutterLeft
- gutterRight
- gutterTop
- gutterBottom

ふち取りスタイルのデフォルト値は、チャートによって適切な値が計算されることを意味する undefined になります。デフォルト値をオーバーライドし、ふち取りの値を明示的に設定すると、Flex で動的にふち取りのサイズを計算する必要がなくなるため、チャートのレンダリング時間を短縮できます。ただし、これを行うことにより軸ラベルの一部が表示されなくなるなどの望ましくない影響が生じることもあります。

ふち取りのスタイルはチャートコントロールで設定します。次の例では、gutterLeft、gutterRight、gutterBottom の各スタイルプロパティを明示的に設定することにより、軸ラベル、タイトル、および目盛りにそれぞれ 50 ピクセル幅を確保した領域を作成しています。また、paddingTop プロパティを 20 に設定しています。

```
<?xml version="1.0"?>
<!-- charts/GutterStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
     ColumnChart {
        gutterLeft:50;
        gutterRight:50;
        gutterBottom:50;
        paddingTop:20;
     }
  </mx:Style>

  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
```

```
    {Month:"Jan", Profit:2000, Expenses:1500},
    {Month:"Feb", Profit:1000, Expenses:200},
    {Month:"Mar", Profit:1500, Expenses:500}
  ]);
  ]]></mx:Script>
  <mx:Panel>
    <mx:ColumnChart id="column" dataProvider="{expenses}">
      <mx:horizontalAxis>
        <mx:CategoryAxis
            dataProvider="{expenses}"
            categoryField="Month"
         />
      </mx:horizontalAxis>
      <mx:series>
        <mx:ColumnSeries
            xField="Month"
            yField="Profit"
            displayName="Profit"
        />
        <mx:ColumnSeries
            xField="Month"
            yField="Expenses"
            displayName="Expenses"
        />
      </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

また、以下のようにふち取りのプロパティをインラインで設定することもできます。

```
<?xml version="1.0"?>
<!-- charts/GutterProperties.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
    {Month:"Jan", Profit:2000, Expenses:1500},
    {Month:"Feb", Profit:1000, Expenses:200},
    {Month:"Mar", Profit:1500, Expenses:500}
  ]);
  ]]></mx:Script>
  <mx:Panel>
    <mx:ColumnChart id="myChart"
      dataProvider="{expenses}"
      gutterLeft="50"
      gutterRight="50"
      gutterBottom="50"
      gutterTop="20"
```

```
    >
       <mx:horizontalAxis>
          <mx:CategoryAxis
               dataProvider="{expenses}"
               categoryField="Month"
          />
       </mx:horizontalAxis>
       <mx:series>
          <mx:ColumnSeries
               xField="Month"
               yField="Profit"
               displayName="Profit"
          />
          <mx:ColumnSeries
               xField="Month"
               yField="Expenses"
               displayName="Expenses"
          />
       </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

ふち取りとパディング領域のプロパティに加え、チャートの軸に対して labelGap プロパティを設定することもできます。labelGap プロパティは、目盛りと軸ラベル間の距離をピクセル単位で定義するものです。AxisRenderer タグの labelGap プロパティは、次のようにして設定します。

```
<?xml version="1.0"?>
<!-- charts/LabelGaps.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      public var expenses:ArrayCollection = new ArrayCollection([
         {Month:"Jan", Profit:2000, Expenses:1500},
         {Month:"Feb", Profit:1000, Expenses:200},
         {Month:"Mar", Profit:1500, Expenses:500}
      ]);
  ]]></mx:Script>
  <mx:Panel>
     <mx:ColumnChart id="myChart"
        dataProvider="{expenses}"
        gutterLeft="50"
        gutterRight="50"
        gutterBottom="50"
        gutterTop="20"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
```

```
                categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:horizontalAxisRenderer>
            <mx:AxisRenderer labelGap="20"/>
        </mx:horizontalAxisRenderer>
        <mx:verticalAxisRenderer>
            <mx:AxisRenderer labelGap="20"/>
        </mx:verticalAxisRenderer>
        <mx:series>
            <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
            />
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

軸の操作

軸タイトルとラベルの追加、目盛りと軸線の書式設定、および軸エレメントの回転が可能です。円グラフ以外のすべてのチャートに、垂直軸と水平軸があります。それぞれの軸上には、小さな印 (目盛り) と対応するラベルが描画されます。このラベルには、テキストストリングまたは数値を使用できます。

軸にはカテゴリと数値の 2 つのタイプがあります。一般的に " カテゴリ軸 " では、チャート内のアイテムのグループ分けを示すストリングを定義します。たとえば、家賃、公共料金、保険などの経費の種類や、従業員の名前などを定義します。" 数値軸 " では、経費や従業員の生産性上昇率など、継続的なデータを定義します。たとえばこれらのデータでは、縦棒グラフの高さや、円グラフの区分の幅などを定義します。

この 2 つの軸タイプは、軸の外観の定義に使用できる共通のプロパティをいくつか共有しています。たとえば、軸のラベルの値をカスタマイズするには、軸の `labelFunction` プロパティを使用します。このプロパティについては、1565 ページの「軸ラベルの定義」で説明します。軸のタイトルを設定するには、`title` プロパティを使用します。このプロパティについては、1561 ページの「軸タイトルの追加」で説明します。

以降のセクションで、CategoryAxis クラスと NumericAxis クラスについて説明します。

CategoryAxis クラスについて

CategoryAxis クラスは、国名、製品名、部署名などのように、連続しないひとまとまりのデータを軸にマッピングし、軸に対して均等に配置します。String で表現できる任意のデータ型を使用できます。

CategoryAxis オブジェクトの dataProvider プロパティでは、ラベルに使用するテキストが格納されているデータプロバイダを定義します。ほとんどの場合、このデータプロバイダは、そのチャートのデータプロバイダと同じです。チャートの dataProvider プロパティはそのチャート内で使用する CategoryAxis オブジェクトに継承されるため、dataProvider プロパティを CategoryAxis オブジェクトに明示的に設定する必要はありません。

CategoryAxis オブジェクトの dataProvider プロパティには、ラベルの配列またはオブジェクトの配列を格納できます。データプロバイダにオブジェクトが格納されている場合は、次の例のように、categoryField プロパティを使用して、軸のラベルのデータが格納されているデータプロバイダのフィールドを参照します。

```
<?xml version="1.0"?>
<!-- charts/BasicColumn.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
```

```
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

データプロバイダにラベルの配列のみが格納されている場合は、categoryField プロパティは指定しません。

データプロバイダの軸ラベルを使用する代わりに、CategoryAxis オブジェクトのラベルをカスタマイズする方法もあります。この方法では、カスタムデータプロバイダを使用します。カスタムデータプロバイダを提供するには、次の例のように、CategoryAxis オブジェクトの dataProvider プロパティの値にラベルのカスタム配列を設定します。

```
<?xml version="1.0"?>
<!-- charts/CategoryAxisLabels.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);

     [Bindable]
     public var months:Array = [
        {Month:"January", monthAbbrev:"Jan"},
        {Month:"February", monthAbbrev:"Feb"},
        {Month:"March", monthAbbrev:"Mar"}
     ];


  ]]></mx:Script>
  <mx:Panel title="Line Chart">
     <mx:AreaChart
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{months}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:AreaSeries
                yField="Profit"
                displayName="Profit"
           />
           <mx:AreaSeries
```

```
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:AreaChart>
  </mx:Panel>
</mx:Application>
```

チャートのデータプロバイダの詳細については、1458 ページの「チャートデータの定義」を参照してください。

CategoryAxis クラスの `labelFunction` プロパティを使用して、既存のデータプロバイダを基にラベルを定義する機能を指定しても、ラベルをカスタマイズできます。詳細については、1565 ページの「軸ラベルの定義」を参照してください。

NumericAxis クラスについて

NumericAxis クラスは、売上高や収益、利益などの連続する数値を画面の座標にマッピングします。通常、NumericAxis 基本クラスを直接使用することはありません。軸を定義する場合は、次のサブクラスを使用します。

- LinearAxis
- LogAxis
- DateTimeAxis

これらのクラスを使用すると、ラベルや目盛りなど、軸に表示されるエレメントの外観や値を自由に設定できるようになります。

`baseAtZero` プロパティを使用すると、軸に表示する目盛りの間隔や、軸の開始位置を 0 にするかどうかを設定することができます。このプロパティを `true` に設定すると、軸が 0 から開始します。`false` に設定すると、その軸に表示される値に対して適切な値が自動的に開始位置として設定されます。

`parseFunction` プロパティを使用すると、チャート内のデータポイントを書式設定するカスタムメソッドを指定できます。このプロパティは、NumericAxis クラスのすべてのサブクラスで使用できます。DateTimeAxis でのこのプロパティの使用の詳細については、1557 ページの「parseFunction プロパティの使用」を参照してください。

ラベルの値を変更する場合は、NumericAxis クラスの `labelFunction` プロパティを使用します。詳細については、1565 ページの「軸ラベルの定義」を参照してください。

LinearAxis サブクラスについて

LinearAxis サブクラスは、3 つの NumericAxis サブクラスの中で最も単純です。このサブクラスは、最小値から最大値の間の数値を、グラフの軸に沿って均等にマッピングします。デフォルトでは、画面上のすべてのチャートエレメントに適合するように、最小値、最大値、および間隔値がグラフのデータから決定されます。これらのプロパティに特定の値を明示的に設定することもできます。次の例では、最小値と最大値をそれぞれ 10 と 100 に設定しています。

```
<?xml version="1.0"?>
<!-- charts/LinearAxisSample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
  <![CDATA[
     import mx.collections.ArrayCollection;

    [Bindable]
    public var stocks:ArrayCollection = new ArrayCollection([
        {date:"2005/8/4", SMITH:37.23},
        {date:"2005/8/5", SMITH:56.53},
        {date:"2005/8/6", SMITH:17.67},
        {date:"2005/8/7", SMITH:27.72},
        {date:"2005/8/8", SMITH:85.23}
     ]);
  ]]>
  </mx:Script>

  <mx:LineChart id="mychart"
    dataProvider="{stocks}"
    showDataTips="true"
    height="300"
    width="400"
  >
     <mx:verticalAxis>
        <mx:LinearAxis
            title="linear axis"
            minimum="10"
            maximum="100"
            interval="10"
        />
     </mx:verticalAxis>

     <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="date"/>
     </mx:horizontalAxis>
     <mx:series>
        <mx:LineSeries
            yField="SMITH"
            displayName="SMITH close"
        />
     </mx:series>
  </mx:LineChart>
</mx:Application>
```

LogAxis サブクラスについて

LogAxis サブクラスは LinearAxis サブクラスに似ていますが、違いは、値を直線的ではなく対数的に軸にマッピングする点です。LogAxis オブジェクトを使用するのは、チャート内のデータの範囲が広すぎて、データポイントの集合を計測できないような場合です。また、LogAxis データは値が負の場合はレンダリングできません。たとえば、成功している企業の株価を 1929 年から追跡している場合、データを直線的ではなく対数的に表示したほうがチャートが読みやすく便利です。

LogAxis オブジェクトを使用する場合は、ラベルの値を軸に定義する乗数を設定します。乗数の設定には `interval` プロパティを使用します。値は 0 以上で 10 の偶数乗にする必要があります。値を 10 にすると、1、10、100、および 1000 でラベルが生成され、値を 100 にすると 1、100、および 10,000 でラベルが生成されます。`interval` プロパティのデフォルト値は 10 です。間隔は必要に応じて、LogAxis オブジェクトによって 10 の偶数乗に四捨五入されます。

垂直軸や水平軸と同じように、LogAxis オブジェクトの最小値と最大値を設定することもできます。次に例を示します。

```
<?xml version="1.0"?>
<!-- charts/LogAxisSample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
  <![CDATA[
     import mx.collections.ArrayCollection;

    [Bindable]
    public var stocks:ArrayCollection = new ArrayCollection([
        {date:"2005/8/4", SMITH:37.23, DECKER:1500},
        {date:"2005/8/5", SMITH:56.53, DECKER:1210},
        {date:"2005/8/6", SMITH:17.67, DECKER:1270},
        {date:"2005/8/7", SMITH:27.72, DECKER:1370},
        {date:"2005/8/8", SMITH:85.23, DECKER:1530}
     ]);
  ]]>
  </mx:Script>

  <mx:LineChart id="mychart"
    dataProvider="{stocks}"
    showDataTips="true"
    height="300"
    width="400"
  >
     <mx:verticalAxis>
        <mx:LogAxis title="log axis"
            interval="10"
            minimum="10"
            maximum="10000"
        />
     </mx:verticalAxis>
```

```
    <mx:horizontalAxis>
       <mx:CategoryAxis categoryField="date"/>
    </mx:horizontalAxis>
    <mx:series>
       <mx:LineSeries yField="DECKER" displayName="DECKER close"/>
       <mx:LineSeries yField="SMITH" displayName="SMITH close" />
    </mx:series>
  </mx:LineChart>
</mx:Application>
```

DateTimeAxis サブクラスについて

DateTimeAxis サブクラスは、チャートの軸に時間ベースの値をマッピングします。DateTimeAxis サブクラスは、論理的な日付および時刻の単位 (最も近い時間や週など) に合わせて最小値と最大値を計算します。また DateTimeAxis サブクラスでは間隔に使用する時間単位も選択できるので、適切な数のラベルがチャートでレンダリングされます。

DateTimeAxis サブクラスの dataUnits プロパティでは、Date オブジェクトの解釈方法を指定します。デフォルトではこのプロパティは Flex によって決定されますが、オーバーライドすることができます。日数に関連させてデータを表示するには、次の例のように dataUnits プロパティを days に設定します。

```
<mx:DateTimeAxis dataUnits="days"/>
```

dataUnits プロパティの有効な値は、milliseconds、seconds、minutes、hours、days、weeks、months、および years です。

適切なラベル単位を割り当てていれば、DateTimeAxis オブジェクトは、データによって表されている単位よりも小さな単位を割り当てません。dataUnits プロパティが days に設定されている場合は、最小範囲や最大範囲に関係なく、チャートでは時間ごとのラベルはレンダリングされません。これを行うには、値を明示的に設定する必要があります。

系列によっては、dataUnits プロパティの値を使用してレンダリングを変更します。具体的には、Column、Bar、Candlestick、HLOCS コントロールなどのほとんどの縦棒系列では、dataUnits の値を使用して、レンダリングされる縦棒の幅を決定します。たとえば、ColumnChart コントロールの水平軸で dataUnits が weeks に、dataUnits が days に設定されている場合、ColumnChart コントロールでは、各ラベル間の間隔が 1/7 ずつになるように各縦棒がレンダリングされます。

サポートされている型について

DateTimeAxis オブジェクトのデータポイントには、Date、String、Number の各データ型を使用できます。

- Date : データポイントの値が Date オブジェクトのインスタンスの場合、既に日付時刻の絶対値を示しており、解釈する必要はありません。Date オブジェクトをデータ値として渡すには、DateTimeAxis サブクラスの `parseFunction` プロパティを使用します。`parseFunction` プロパティは Date オブジェクトを戻します。詳細については、1557 ページの「parseFunction プロパティの使用」を参照してください。
- String : `Date.parse()` メソッドでサポートされているすべての形式を使用できます。次の形式がサポートされています。
  - *曜日 月 日 時:分:秒 GMT 年* (たとえば Tue Feb 1 12:00:00 GMT-0800 2005 のように表示されます)
  - *曜日 月 日 年 時:分:秒 AM|PM* (たとえば Tue Feb 1 2005 12:00:00 AM のように表示されます)
  - *曜日 月 日 年* (たとえば Tue Feb 1 2005 のように表示されます)
  - *月 / 日 / 年* (たとえば 02/01/2005 のように表示されます)

  DateTimeAxis の `parseFunction` プロパティを使用して、任意のデータ型を取得して Date を返すカスタムロジックを作成することもできます。詳細については、1557 ページの「parseFunction プロパティの使用」を参照してください。
- Number : 数字を使用すると、その数字は 1970 年 1 月 1 日 0 時からのミリ秒数であると見なされます。たとえば 543387600000 のような数字になります。この値を既存の Date オブジェクトで取得するには、Date オブジェクトの `getTime()` メソッドを使用します。

次の例では、月 / 日 / 年のパターンに一致する日付の String 値を使用して、日付を日単位で表示するように指定しています。

```
<?xml version="1.0"?>
<!-- charts/DateTimeAxisSample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="600">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection
     [Bindable]
     public var aapl:ArrayCollection = new ArrayCollection([
        {date:"08/01/2005", close:42.71},
        {date:"08/02/2005", close:42.99},
        {date:"08/03/2005", close:42.65}
     ]);
  ]]></mx:Script>
  <mx:Panel title="DateTimeAxis" width="100%" height="100%">
     <mx:LineChart id="mychart"
        dataProvider="{aapl}"
```

```
        showDataTips="true"
    >
        <mx:horizontalAxis>
            <mx:DateTimeAxis dataUnits="days"/>
        </mx:horizontalAxis>
        <mx:series>
            <mx:LineSeries
                yField="close"
                xField="date"
                displayName="AAPL"
            />
        </mx:series>
    </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

parseFunction プロパティの使用

データポイントの値をカスタマイズするメソッドを指定するには、DateTimeAxis の parseFunction プロパティを使用します。このプロパティを使用して、値を受け入れて Date オブジェクトを返すメソッドを指定します。その後で、Date オブジェクトをチャートの DateTimeAxis オブジェクトの中で使用します。これによって、カスタマイズ可能な日付入力ストリングを作成して、Date オブジェクトに変換できるようになります。Date オブジェクトはその後、Flex によって DateTimeAxis で使用できる形に解釈されます。

parseFunction プロパティで指定した解析メソッドは、DateTimeAxis の値の計算が必要になるたびに呼び出されます。つまり、ユーザーのチャート操作でデータポイントが見つかるたびに呼び出されます。その結果、解析メソッドが呼び出される頻度が高くなり、アプリケーションのパフォーマンスが低下することがあります。そのため、解析メソッド内のコードの量を最小に抑えるようにしてください。

Flex で解析メソッドに渡せるパラメータは 1 つだけです。このパラメータは、データ系列に指定したデータポイントの値です。通常これは、いずれかの日付形式を表しているストリングです。このパラメータをオーバーライドすることも、他のパラメータを追加することもできません。

次の例では、データプロバイダ内の “YYYY, MM, DD” (年、月、日) のパターンに一致する String 値から、Date オブジェクトを作成する解析メソッドを示しています。

```
<?xml version="1.0"?>
<!-- charts/DateTimeAxisParseFunction.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var aapl:ArrayCollection = new ArrayCollection([
        {date:"2005, 8, 1", close:42.71},
        {date:"2005, 8, 2", close:42.99},
        {date:"2005, 8, 3", close:44}
```

```
    ]);

    public function myParseFunction(s:String):Date {
       // Get an array of Strings from the
       // comma-separated String passed in.
       var a:Array = s.split(",");

       // Trace out year, month, and day values.
       trace("y:" + a[0]);
       trace("m:" + a[1]);
       trace("d:" + a[2]);

       // Create the new Date object.
       var newDate:Date = new Date(a[0],a[1],a[2]);
       return newDate;
    }
  ]]></mx:Script>
  <mx:LineChart id="mychart"
    dataProvider="{aapl}"
    showDataTips="true"
  >
     <mx:horizontalAxis>
        <mx:DateTimeAxis
            dataUnits="days"
            parseFunction="myParseFunction"
        />
     </mx:horizontalAxis>
     <mx:series>
        <mx:LineSeries
            yField="close"
            xField="date"
            displayName="AAPL"
        />
     </mx:series>
  </mx:LineChart>
</mx:Application>
```

DateTimeAxis ラベルの書式設定

軸に表示する単位を割り当てる場合、DateTimeAxis オブジェクトは、適切な数のラベルをレンダリングできる単位のうち最大の単位を使用します。次の表で、それぞれの単位のデフォルトラベルフォーマットと、最小範囲について説明します。

単位	ラベルのフォーマット	最小範囲
年	YYYY	最小値と最大値が最低でも 2 年間に渡る場合
月	MM/YY	2 か月間
週	DD/MM/YY	2 週間
日	DD/MM/YY	1 日
時	HH:MM	1 時間
分	HH:MM	1 分
秒	HH:MM:SS	1 秒
ミリ秒	HH:MM:SS:mmmm	1 ミリ秒

特定のチャートのインスタンスに使用する有効な単位のリストを、その使用法にとって意味のあるサブセットに制限することができます。LinearAxis オブジェクトと同様に、DateTimeAxis オブジェクトでも最小値、最大値、および間隔値を設定できます。

値を四捨五入する場合、渡された値をローカルタイムゾーンと UTC のどちらで表示するかを DateTimeAxis オブジェクトで決定します。DateTimeAxis オブジェクトで値をローカルタイムとして扱うには、`displayLocalTime` プロパティを `true` に設定します。デフォルト値は `false` です。

ラベルの値を変更するには、DateTimeAxis オブジェクトの `labelFunction` プロパティを使用します。このプロパティは NumericAxis クラスから継承されます。詳細については、1565 ページの「軸ラベルの定義」で説明します。

DateTimeAxis での最小値と最大値の設定

軸で `minimum` プロパティと `maximum` プロパティの値を設定することで、その軸が使用する値の範囲を定義できます。そのためには、次の例に示すように、バインド可能な Date オブジェクトを作成し、`minimum` プロパティと `maximum` プロパティの値をそのオブジェクトにバインドします。

```
<?xml version="1.0"?>
<!-- charts/DateTimeAxisRange.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="600">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var minDate:Date = new Date(2005, 11, 1);
```

```
    [Bindable]
    public var maxDate:Date = new Date(2007, 1, 1);

    [Bindable] public var myData:ArrayCollection = new
    ArrayCollection([
       {date: "01/01/2006", amt: 12345},
       {date: "02/01/2006", amt: 54331},
       {date: "12/31/2006", amt: 34343}
    ]);

  ]]></mx:Script>
  <mx:Panel title="DateTimeAxis" width="100%" height="100%">
     <mx:PlotChart id="mychart"
        dataProvider="{myData}"
        showDataTips="true">
        <mx:horizontalAxis>
           <mx:DateTimeAxis
                dataUnits="months"
                minimum="{minDate}"
                maximum="{maxDate}"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:PlotSeries
                yField="amt"
                xField="date"
                displayName="myData"
           />
        </mx:series>
     </mx:PlotChart>
  </mx:Panel>
</mx:Application>
```

次のシンタックスを使用して、日付の範囲を MXML で表すこともできます。

```
<mx:horizontalAxis>
  <mx:DateTimeAxis dataUnits="months">
    <mx:minimum>
      <mx:Date fullYear="2005" month="11" date="1"/>
    </mx:minimum>
    <mx:maximum>
      <mx:Date fullYear="2007" month="1" date="1"/>
    </mx:maximum>
  </mx:DateTimeAxis>
</mx:horizontalAxis>
```

軸タイトルの追加

チャートコントロールのそれぞれの軸には、その軸の意味をユーザーに示すタイトルを表示できます。Flex では明示的に設定しない限り、軸のタイトルは追加されません。チャートの軸にタイトルを追加するには、軸オブジェクトの `title` プロパティを使用します。これは CategoryAxis か、DateTimeAxis、LinearAxis、または LogAxis などの NumericAxis サブクラスの 1 つです。軸タイトルにスタイルを設定するには、チャートコントロールの `axisTitleStyleName` プロパティを使用します。

次の例では、MXML および ActionScript で水平軸と垂直軸のタイトルを設定し、そのタイトルにスタイルを適用します。

```
<?xml version="1.0"?>
<!-- charts/AxisTitles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="setTitles()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);

     private function setTitles():void {
        la1.title="Dollars";
     }

  ]]></mx:Script>

  <mx:Style>
     .myStyle {
        fontFamily:Verdana;
        fontSize:12;
        color:#4691E1;
        fontWeight:bold;
        fontStyle:italic;
     }
  </mx:Style>

  <mx:Panel>
     <mx:ColumnChart id="myChart"
        axisTitleStyleName="myStyle"
        dataProvider="{expenses}"
     >
        <mx:verticalAxis>
           <mx:LinearAxis id="la1"/>
        </mx:verticalAxis>
```

```
        <mx:horizontalAxis>
           <mx:CategoryAxis title="FY 2006"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

軸タイトルに埋め込みフォントを使用することもできます。次の例では、垂直軸のタイトルに使用するフォントを埋め込み、スタイルを設定しています。

```
<?xml version="1.0"?>
<!-- charts/AxisTitleEmbedFont.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month:"Jan", Profit:2000, Expenses:1500},
     {Month:"Feb", Profit:1000, Expenses:200},
     {Month:"Mar", Profit:1500, Expenses:500}
  ]);
  ]]></mx:Script>

  <mx:Style>
     @font-face{
        src:url("../assets/MyriadWebPro.ttf");
        fontFamily:myMyriad;
     }

     .myEmbeddedStyle {
        fontFamily:myMyriad;
        fontSize:20;
     }
  </mx:Style>

  <mx:Panel>
```

```
    <mx:ColumnChart id="column"
       dataProvider="{expenses}"
       axisTitleStyleName="myEmbeddedStyle"
    >
       <mx:horizontalAxis>
          <mx:CategoryAxis
               dataProvider="{expenses}"
               categoryField="Month"
               title="FY 2006"
          />
       </mx:horizontalAxis>
       <mx:series>
          <mx:ColumnSeries
               xField="Month"
               yField="Profit"
               displayName="Profit"
          />
          <mx:ColumnSeries
               xField="Month"
               yField="Expenses"
               displayName="Expenses"
          />
       </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

フォントの埋め込みの詳細については、708 ページの「埋め込みフォントの使用」を参照してください。

明示的に指定しなくても、チャートに axisTitleStyleName プロパティが適用されるという点を利用し、次のようにすることもできます。

```
<?xml version="1.0"?>
<!-- charts/CSSAxisTitle.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month:"Jan", Profit:2000, Expenses:1500},
     {Month:"Feb", Profit:1000, Expenses:200},
     {Month:"Mar", Profit:1500, Expenses:500}
  ]);
  ]]></mx:Script>

  <mx:Style>
     .axisTitles {
        color:red;
        fontWeight:bold;
        fontFamily:Arial;
```

```
        fontSize:20;
    }

    ColumnChart {
        axisTitleStyleName:axisTitles;
    }
  </mx:Style>

  <mx:Panel>
    <mx:ColumnChart id="column" dataProvider="{expenses}">
        <mx:horizontalAxis>
            <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month" title="FY 2006"
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
            />
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

次の例のように、軸にタイトルスタイルを適用することもできます。

```
<mx:CategoryAxis title="State" styleName="myEmbeddedStyle"/>
```

軸ラベルの定義

水平軸または垂直軸に軸ラベルの値を定義します。

次の例のように、AxisRenderer オブジェクトで showLabels プロパティの値を false に設定すると、ラベルを無効にできます。

```
<?xml version="1.0"?>
<!-- charts/DisabledAxisLabels.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month:"Jan", Profit:2000, Expenses:1500},
     {Month:"Feb", Profit:1000, Expenses:200},
     {Month:"Mar", Profit:1500, Expenses:500}
]);
  ]]></mx:Script>
  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>

        <mx:horizontalAxisRenderer>
           <mx:AxisRenderer showLabels="false"/>
        </mx:horizontalAxisRenderer>
        <mx:verticalAxisRenderer>
           <mx:AxisRenderer showLabels="false"/>
        </mx:verticalAxisRenderer>

        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

軸の labelFunction コールバック関数を使用すると、軸ラベルの値をカスタマイズできます。labelFunction に指定された関数は、軸ラベルとして表示されるストリングを返します。

コールバック関数に必要な形式を以下に示します。

```
function function_name(labelValue:Object, previousLabelValue:Object,
  axis:axis_type, labelItem:Object):return_type
```

次の表で、コールバック関数のパラメータについて説明します。

パラメータ	説明
labelValue	現在のラベルの値
previousLabelValue	このラベルの前のラベルの値。これが最初のラベルの場合、*previousLabelValue* の値は null になります。
axis	CategoryAxis または NumericAxis などの軸オブジェクト
labelItem	ラベルオブジェクトへの参照
return_type	コールバック関数が返すオブジェクトの型。どの型でも使用できますが、通常 CategoryAxis の軸には String、NumericAxis オブジェクトには Number、DateTimeAxis オブジェクトには Date オブジェクトを使用します。

labelFunction を使用する場合は、軸のクラスまたはチャートのパッケージ全体を次の例のように読み込む必要があります。

```
import mx.charts.*;
```

次の例では、水平な CategoryAxis オブジェクトに labelFunction を定義しています。この関数では、軸ラベルに '06 が追加され、ラベルが Jan '06、Feb '06、および Mar '06 と表示されます。ここでは月名を使用し、そのストリングに年を追加しているため、ラベルの書式設定関数の戻り値の型は String です。

```
<?xml version="1.0"?>
<!-- charts/CustomLabelFunction.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.charts.*;
  import mx.collections.ArrayCollection;

  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month: "Jan", Profit: 2000, Expenses: 1500},
     {Month: "Feb", Profit: 1000, Expenses: 200},
     {Month: "Mar", Profit: 1500, Expenses: 500}
  ]);

  public function defineLabel(
    cat:Object,
    pcat:Object,
```

```
    ax:CategoryAxis,
    catItem:Object):String
  {
        return cat + " '06";
  }
  ]]></mx:Script>
  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
                title="Expenses"
                labelFunction="defineLabel"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
  </mx:Panel>
</mx:Application>
```

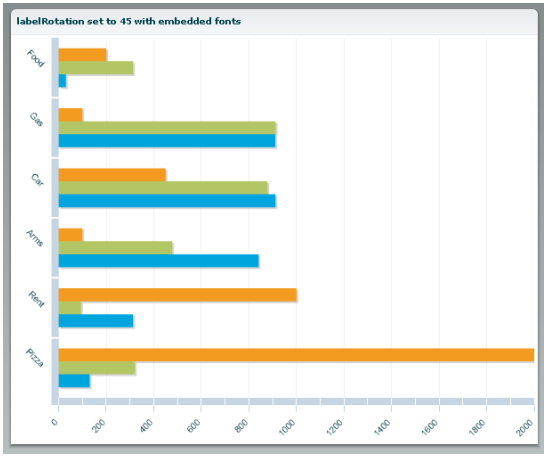
軸ラベルのカスタマイズには、ラベルにカスタムデータプロバイダを設定するという方法もあります。詳細については、1550 ページの「CategoryAxis クラスについて」を参照してください。

PieChart コントロールのラベルをカスタマイズするには、PieSeries クラスで定義したラベル関数を使用します。詳細については、1513 ページの「PieChart コントロールでのラベルの使用」を参照してください。

軸エレメントの回転

AxisRenderer オブジェクトの labelRotation プロパティを使用すると、軸ラベルを回転させることができます。度単位で、-90 から 90 までの値を指定します。labelRotation プロパティを null に設定すると、Flex によって最適な角度が計算された上でラベルがレンダリングされます。

次の例は、縦横の軸ラベルを 45 度回転させたグラフを示しています。



テキストを回転させるには、Flex アプリケーションにフォントを埋め込む必要があります。次の <mx:Style> ブロックは、Flex アプリケーションに Arial フォントを埋め込み、このフォントを、ラベルを 45 度回転させるチャートコントロールに適用しています。

```
<?xml version="1.0"?>
<!-- charts/RotateAxisLabels.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month: "Jan", Profit: 2000, Expenses: 1500},
     {Month: "Feb", Profit: 1000, Expenses: 200},
     {Month: "Mar", Profit: 1500, Expenses: 500}
  ]);
  ]]></mx:Script>

  <mx:Style>
     @font-face{
        src: url("../assets/MyriadWebPro.ttf");
        fontFamily: myMyriad;
     }

     ColumnChart {
```

```
        fontFamily: myMyriad;
        fontSize: 20;
    }
  </mx:Style>

  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
                title="FY 2006"
           />
        </mx:horizontalAxis>

        <mx:horizontalAxisRenderer>
           <mx:AxisRenderer labelRotation="45"/>
        </mx:horizontalAxisRenderer>
        <mx:verticalAxisRenderer>
           <mx:AxisRenderer labelRotation="45"/>
        </mx:verticalAxisRenderer>

        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
               displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
               displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

範囲の設定

軸の最小値、最大値、および間隔は NumericAxis オブジェクトの設定に基づいて計算されますが、Flex で自動的に設定された値は、変更することができます。チャートに表示されるデータの範囲を変更すると、目盛りの範囲も変更されます。

次の表で、軸の範囲を定義する軸のプロパティについて説明します。

プロパティ	内容
minimum	軸の最小値
maximum	軸の最大値
interval	軸の値の間隔 (軸の単位で指定)

次の例は、y 軸の範囲の定義です。

```
<?xml version="1.0"?>
<!-- charts/LinearAxisSample.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
  <![CDATA[
     import mx.collections.ArrayCollection;

    [Bindable]
    public var stocks:ArrayCollection = new ArrayCollection([
        {date:"2005/8/4", SMITH:37.23},
        {date:"2005/8/5", SMITH:56.53},
        {date:"2005/8/6", SMITH:17.67},
        {date:"2005/8/7", SMITH:27.72},
        {date:"2005/8/8", SMITH:85.23}
     ]);
  ]]>
  </mx:Script>

  <mx:LineChart id="mychart"
    dataProvider="{stocks}"
    showDataTips="true"
    height="300"
    width="400"
  >
     <mx:verticalAxis>
        <mx:LinearAxis
            title="linear axis"
            minimum="10"
            maximum="100"
            interval="10"
        />
     </mx:verticalAxis>
```

```
    <mx:horizontalAxis>
       <mx:CategoryAxis categoryField="date"/>
    </mx:horizontalAxis>
    <mx:series>
       <mx:LineSeries
           yField="SMITH"
           displayName="SMITH close"
       />
    </mx:series>
  </mx:LineChart>
</mx:Application>
```

この例では、y 軸の最小値は 10 で、最大値は 100、目盛りの間隔は 10 です。この場合、ラベルのテキストは、10、20、30、40、のように表示されます。

DateTimeAxis で最小値と最大値を設定するには、軸のタグで Strings または Numbers ではなく Date オブジェクトを使用する必要があります。詳細については、1559 ページの「DateTimeAxis での最小値と最大値の設定」を参照してください。

目盛りの長さや位置の設定の詳細については、1571 ページの「目盛りのフォーマット」を参照してください。

目盛りのフォーマット

Flex のチャートには、大目盛りと小目盛りの 2 種類の目盛りがあります。大目盛りは、軸ラベルに対応する軸に表示される印です。通常、軸ラベルのテキストは、チャートのデータプロバイダによって生成されます。小目盛りは、大目盛りの間に表示される印です。小目盛りが表示されることにより、ユーザーは大目盛り間の距離を容易に視覚化できます。

目盛りを表示するかどうかと、目盛りの表示位置は、AxisRenderer オブジェクトの tickPlacement プロパティと minorTickPlacement プロパティで指定します。

次の表で、tickPlacement プロパティと minorTickPlacement プロパティの有効な値について説明します。

値	内容
cross	軸の両側に目盛りを表示します。
inside	軸線の内側に目盛りを表示します。
none	目盛りを非表示にします。
outside	軸線の外側に目盛りを表示します。

目盛りとそのラベルは、tickAlignment プロパティを使用して整列させることができます。

Flex では、軸に表示される目盛りの長さと、小目盛りの数を指定することもできます。次の表で、チャートの軸における目盛りの長さを定義するプロパティについて説明します。

プロパティ	内容
tickLength	軸の大目盛りの長さをピクセル単位で設定します。
minorTickLength	軸の小目盛りの長さをピクセル単位で設定します。

小目盛りは、大目盛りの位置にも表示されます。そのため、小目盛りを表示させたまま大目盛りを非表示にすると、小目盛りが通常の目盛りの間隔で表示されます。

次の例では、軸線の内側に目盛りを表示します。目盛りの長さを 12 ピクセルに設定し、小目盛りを非表示にします。

```
<?xml version="1.0"?>
<!-- charts/TickStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  [Bindable]
  public var aapl:Array = [
     {date:"1-Aug-05",open:42.57,high:43.08,low:42.08,close:42.75},
     {date:"2-Aug-05",open:42.89,high:43.5,low:42.61,close:43.19},
     {date:"3-Aug-05",open:43.19,high:43.31,low:42.77,
     close:43.22},
     {date:"4-Aug-05",open:42.89,high:43,low:42.29,close:42.71},
     {date:"5-Aug-05",open:42.49,high:43.36,low:42.02,
     close:42.99},
     {date:"8-Aug-05",open:43,high:43.25,low:42.61,close:42.65},
     {date:"9-Aug-05",open:42.93,high:43.89,low:42.91,
     close:43.82},
     {date:"10-Aug-05",open:44,high:44.39,low:43.31,close:43.38},
     {date:"11-Aug-05",open:43.39,high:44.12,low:43.25,close:44},
     {date:"12-Aug-05",open:43.46,high:46.22,low:43.36,
     close:46.1},
  ];
  ]]></mx:Script>

  <mx:Style>
     .myAxisStyle {
        placement:bottom;
        minorTickPlacement:none;
        tickLength:12;
        tickPlacement:inside;
     }
  </mx:Style>

    <mx:HLOCChart id="mychart"
        dataProvider="{aapl}"
        showDataTips="true"
    >
```

```
        <mx:horizontalAxisRenderer>
        <mx:AxisRenderer styleName="myAxisStyle"/>
    </mx:horizontalAxisRenderer>

    <mx:verticalAxis>
        <mx:LinearAxis minimum="30" maximum="50"/>
    </mx:verticalAxis>

    <mx:series>
        <mx:HLOCSeries
            dataProvider="{aapl}"
            openField="open"
            highField="high"
            lowField="low"
            closeField="close"
            displayName="AAPL"
        />
     </mx:series>
  </mx:HLOCChart>
  <mx:Legend dataProvider="{mychart}"/>
</mx:Application>
```

軸線のフォーマット

軸には、目盛りが接する線が複数あります。スタイルプロパティを使用すると、これらの線を非表示にしたり、線幅を変えることができます。

軸線を非表示にするには、AxisRenderer オブジェクトの showLine プロパティを false に設定します。デフォルト値は true です。次の例では、showLine を false に設定しています。

```
<?xml version="1.0"?>
<!-- charts/DisableAxisLines.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Line Chart">
     <mx:LineChart dataProvider="{expenses}" showDataTips="true">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
```

```
        <mx:horizontalAxisRenderer>
           <mx:AxisRenderer showLine="false"/>
        </mx:horizontalAxisRenderer>
        <mx:verticalAxisRenderer>
           <mx:AxisRenderer showLine="false"/>
        </mx:verticalAxisRenderer>
        <mx:series>
           <mx:LineSeries
                yField="Profit"
                displayName="Profit"
           />
           <mx:LineSeries
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

showLine プロパティを CSS スタイルプロパティとして適用することもできます。

軸線の width、color、および alpha は、<mx:axisStroke> タグで変更できます。次の例に示すように、<mx:Stroke> 子タグを使用して、これらのプロパティを定義するか、線を定義して axisStroke オブジェクトにバインドします。

```
<?xml version="1.0"?>
<!-- charts/StyleAxisLines.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>

  <mx:Stroke id="axisStroke"
        color="#884422"
        weight="8"
        alpha=".75"
        caps="square"
  />

  <mx:Panel title="Line Chart">
     <mx:LineChart dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
```

```
            <mx:CategoryAxis
                 dataProvider="{expenses}"
                 categoryField="Month"
             />
        </mx:horizontalAxis>
        <mx:horizontalAxisRenderer>
            <mx:AxisRenderer>
             <mx:axisStroke>{axisStroke}</mx:axisStroke>
            </mx:AxisRenderer>
        </mx:horizontalAxisRenderer>
        <mx:verticalAxisRenderer>
            <mx:AxisRenderer>
             <mx:axisStroke>
                 <mx:Stroke color="#884422"
                     weight="8"
                     alpha=".75"
                     caps="square"
                 />
             </mx:axisStroke>
            </mx:AxisRenderer>
        </mx:verticalAxisRenderer>
        <mx:series>
            <mx:LineSeries
                 yField="Profit"
                 displayName="Profit"
            />
            <mx:LineSeries
                 yField="Expenses"
                 displayName="Expenses"
            />
        </mx:series>
    </mx:LineChart>
  </mx:Panel>
</mx:Application>
```

線の詳細については、1577 ページの「線の使用」を参照してください。

軸線にフィルタを適用して、外観をさらにカスタマイズすることもできます。詳細については、1594 ページの「フィルタの使用」を参照してください。

minField プロパティの使用

系列の種類によっては、画面上に描画するエレメントの最小値を指定できます。たとえば、ColumnChart コントロールでは、縦棒の基本の値を指定できます。縦棒の基本の値 (最小値) を指定するには、系列オブジェクトの minField プロパティに、データプロバイダのフィールドを設定します。

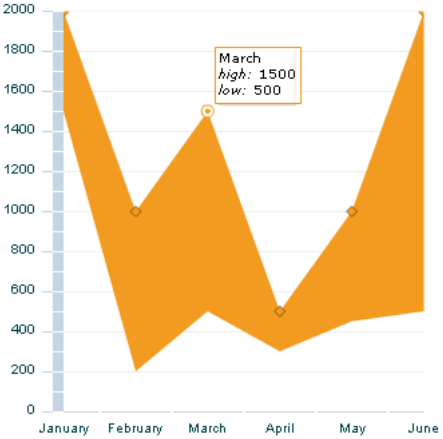
minField プロパティは、次のチャート系列タイプで指定できます。

- AreaSeries
- BarSeries
- ColumnSeries

minField プロパティの値を設定すると、領域内のそれぞれのデータポイントの軸に 2 つの値が作成されます。次に例を示します。

```
<?xml version="1.0"?>
<!-- charts/MinField.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Area Chart">
     <mx:AreaChart
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:AreaSeries
                yField="Profit"
                minField="Expenses"
                displayName="Profit"
            />
        </mx:series>
     </mx:AreaChart>
  </mx:Panel>
</mx:Application>
```

この場合、データヒントには、現在の値の “high” と、minField の値である “low” のラベルが表示されます。次の例は、縦軸の最小値が定義された AreaChart を示しています。



minField プロパティを使用して、ウォーターフォールを作成、または ColumnChart コントロールをカスケードする場合の例については、1494 ページの「縦棒グラフの使用」を参照してください。

線の使用

チャート系列やグリッド線で Stroke クラスを使用すると、チャートエレメントの描画に使用する線のプロパティを制御できます。

次の表で、線の外観の制御に使用するプロパティについて説明します。

プロパティ	内容
color	線の色を 16 進数値で指定します。デフォルト値は 0x000000 (黒) です。
weight	線幅をピクセル単位で指定します。デフォルト値は 0 (極細線) です。
alpha	線の透明度を指定します。有効な値は 0 (透明) ～ 100 (不透明) です。デフォルト値は 100 です。

次の例では、BarChart コントロールの BarSeries のアイテムの境界線に、2 ピクセルの線幅を定義します。色は一方が濃い灰色 (0x808080) で、もう一方は薄い灰色 (0xC0C0C0) です。

```
<?xml version="1.0"?>
<!-- charts/BasicBarStroke.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
```

```
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
    ]);
  ]]></mx:Script>
  <mx:Panel title="Bar Chart">
    <mx:BarChart id="myChart" dataProvider="{expenses}">
      <mx:verticalAxis>
        <mx:CategoryAxis
            dataProvider="{expenses}"
            categoryField="Month"
        />
      </mx:verticalAxis>
      <mx:series>
        <mx:BarSeries
            yField="Month"
            xField="Profit"
            displayName="Profit"
         >
         <mx:stroke>
            <mx:Stroke
                color="0x808080"
                weight="2"
                alpha=".8"
            />
         </mx:stroke>
        </mx:BarSeries>
        <mx:BarSeries
            yField="Month"
            xField="Expenses"
            displayName="Expenses"
        >
         <mx:stroke>
            <mx:Stroke
                color="0xC0C0C0"
                weight="2"
                alpha=".8"
            />
         </mx:stroke>
        </mx:BarSeries>
      </mx:series>
    </mx:BarChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

線を使用した AxisRenderer プロパティの定義

線を使用して、目盛りなどの AxisRenderer のプロパティを定義できます。次に例を示します。

```
<?xml version="1.0"?>
<!-- charts/AxisRendererStrokes.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  [Bindable]
  public var aapl:Array = [
     {date:"1-Aug-05",open:42.57,high:43.08,low:42.08,close:42.75},
     {date:"2-Aug-05",open:42.89,high:43.5,low:42.61,close:43.19},
     {date:"3-Aug-05",open:43.19,high:43.31,low:42.77,close:43.22},
     {date:"4-Aug-05",open:42.89,high:43,low:42.29,close:42.71},
     {date:"5-Aug-05",open:42.49,high:43.36,low:42.02,close:42.99},
     {date:"8-Aug-05",open:43,high:43.25,low:42.61,close:42.65},
     {date:"9-Aug-05",open:42.93,high:43.89,low:42.91,close:43.82},
     {date:"10-Aug-05",open:44,high:44.39,low:43.31,close:43.38},
     {date:"11-Aug-05",open:43.39,high:44.12,low:43.25,close:44},
     {date:"12-Aug-05",open:43.46,high:46.22,low:43.36,close:46.1},
  ];
  ]]></mx:Script>

  <mx:HLOCChart id="myChart"
    dataProvider="{aapl}"
    showDataTips="true"
  >
     <mx:horizontalAxisRenderer>
        <mx:AxisRenderer
            placement="bottom"
            canDropLabels="true"
            tickPlacement="inside"
            tickLength="10"
            minorTickPlacement="inside"
            minorTickLength="5"
        >
           <mx:axisStroke>
            <mx:Stroke color="#000080" weight="1"/>
           </mx:axisStroke>
           <mx:tickStroke>
            <mx:Stroke color="#000060" weight="1"/>
           </mx:tickStroke>
           <mx:minorTickStroke>
            <mx:Stroke color="#100040" weight="1"/>
           </mx:minorTickStroke>
        </mx:AxisRenderer>
     </mx:horizontalAxisRenderer>

     <mx:verticalAxis>
        <mx:LinearAxis minimum="30" maximum="50"/>
```

```
    </mx:verticalAxis>

    <mx:series>
       <mx:HLOCSeries
           dataProvider="{aapl}"
           openField="open"
           highField="high"
           lowField="low"
           closeField="close"
           displayName="AAPL"
       >
       </mx:HLOCSeries>
    </mx:series>
  </mx:HLOCChart>
  <mx:Legend dataProvider="{myChart}"/>
</mx:Application>
```

MXML タグで線オブジェクトを定義して、その線オブジェクトをチャートのレンダラープロパティにバインドできます。例については、1541 ページの「タグ定義のバインディングによるスタイルの適用」を参照してください。

ActionScript での線の使用

mx.graphics.Stroke クラスを使用すると、ActionScript の Stroke オブジェクトをインスタンス化して操作することができます。その後で `setStyle()` メソッドを使用して、Stroke オブジェクトをチャートに適用できます。次に例を示します。

```
<?xml version="1.0"?>
<!-- charts/ActionScriptStroke.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.graphics.Stroke;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month: "Jan", Profit: 2000, Expenses: 1500},
        {Month: "Feb", Profit: 1000, Expenses: 200},
        {Month: "Mar", Profit: 1500, Expenses: 500}
     ]);

     public function changeStroke(e:Event):void {
        var s:Stroke = new Stroke(0x001100,2);
        s.alpha = .5;
        s.color = 0x0000FF;
        har1.setStyle("axisStroke",s);
        var1.setStyle("axisStroke",s);
     }
  ]]></mx:Script>
```

```
  <mx:Stroke id="baseAxisStroke"
    color="0x884422"
    weight="10"
    alpha=".25"
    caps="square"
  />

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>

        <mx:horizontalAxisRenderer>
           <mx:AxisRenderer id="har1">
            <mx:axisStroke>{baseAxisStroke}</mx:axisStroke>
           </mx:AxisRenderer>
        </mx:horizontalAxisRenderer>

        <mx:verticalAxisRenderer>
           <mx:AxisRenderer id="var1">
            <mx:axisStroke>{baseAxisStroke}</mx:axisStroke>
           </mx:AxisRenderer>
        </mx:verticalAxisRenderer>

        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
  <mx:Button id="b1"
    click="changeStroke(event)"
    label="Change Stroke"
  />
</mx:Application>
```

LineSeries および AreaSeries の線の定義

チャート系列によっては、線に関連するスタイルプロパティが複数存在します。LineSeries では、チャートアイテムのレンダラーを定義する場合に、stroke スタイルプロパティを使用します。実際の線セグメントの線を定義するには、lineStroke プロパティを使用します。

次の例では、LineChart コントロールの線セグメントに青の太線を作成し、各データポイントに大きな赤い四角形を表示します。セグメントのレンダラーとして CrossItemRenderer オブジェクトを使用します。

```
<?xml version="1.0"?>
<!-- charts/LineSeriesStrokes.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Line Chart">
     <mx:LineChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:LineSeries
            yField="Profit"
            displayName="Profit"
           >
            <mx:itemRenderer>
                <mx:Component>
                    <mx:CrossItemRenderer
                        scaleX="1.5"
                        scaleY="1.5"
                    />
                </mx:Component>
            </mx:itemRenderer>
            <mx:fill>
                <mx:SolidColor
                    color="0x0000FF"
                />
```

```
            </mx:fill>
            <mx:stroke>
                <mx:Stroke
                    color="0xFF0066"
                    alpha="1"
                />
            </mx:stroke>
            <mx:lineStroke>
                <mx:Stroke
                    color="0x33FFFF"
                    weight="5"
                    alpha=".8"
                />
            </mx:lineStroke>
           </mx:LineSeries>
           <mx:LineSeries
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
    </mx:LineChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

同様に AreaSeries クラスでは、チャートアイテムのレンダラーのスタイルを設定する場合に stroke プロパティを使用します。領域を定義する線を定義するには、areaStroke スタイルプロパティを使用します。

塗りの使用

複数のデータ系列をチャートに表すとき、または単にチャートの外観を見やすくするために、チャートの各系列について塗りを制御することができます。塗りでは、チャートエレメントの描画方法を定義するパターンを指定することができます。また、塗りを使用して、チャートの背景色や、グリッド線で定義する背景色の領域を指定することもできます。塗りは単色にすることも、線状や放射状の塗りを使用することもできます。" グラデーション " では、塗りの色における段階的な色の変化を指定します。

塗りの特性を定義するには、チャート系列の fill プロパティを使用します。fill プロパティの設定は、LineSeries オブジェクトと LineRenderer オブジェクトには影響しません。PieSeries オブジェクトでは、単一の fill プロパティの代わりに fill の配列を使用できます。

塗りの最も一般的な使用法の 1 つは、チャートに複数のデータ系列があるときにチャートの色を制御することです。次の例では、fill プロパティを使用して、ColumnChart コントロールのそれぞれの ColumnSeries オブジェクトに色を設定しています。

```
<?xml version="1.0"?>
<!-- charts/ColumnFills.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           >
               <mx:fill>
                    <mx:SolidColor color="0x336699"/>
               </mx:fill>
           </mx:ColumnSeries>

           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           >
            <mx:fill>
                <mx:SolidColor color="0xFF99FF"/>
            </mx:fill>
           </mx:ColumnSeries>
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

複数のデータ系列に対して異なる塗りを明示的に定義しない場合は、自動的に単色が使用されます。

PieSeries では、塗りの配列を使用して個々の区分の描画方法を指定することができます。たとえば、次の例のように、PieSeries を表す各区分に独自の色を指定することができます。

```
<?xml version="1.0"?>
<!-- charts/PieFills.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1000},
        {Expense:"Bills", Amount:100},
        {Expense:"Car", Amount:450},
        {Expense:"Gas", Amount:100},
        {Expense:"Food", Amount:200}
     ]);
    ]]>
  </mx:Script>
  <mx:Panel title="Pie Chart">
     <mx:PieChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:series>
           <mx:PieSeries
                field="Amount"
                nameField="Expense"
                labelPosition="callout"
           >
                <mx:fills>
                    <mx:SolidColor color="0xCC66FF" alpha=".8"/>
                    <mx:SolidColor color="0x9966CC" alpha=".8"/>
                    <mx:SolidColor color="0x9999CC" alpha=".8"/>
                    <mx:SolidColor color="0x6699CC" alpha=".8"/>
                    <mx:SolidColor color="0x669999" alpha=".8"/>
                    <mx:SolidColor color="0x99CC99" alpha=".8"/>
                </mx:fills>
            </mx:PieSeries>
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

また、塗りを使用して、チャートの背景色を設定することもできます。これは、次の例に示すように、チャートのタグに <mx:fill> 子タグを追加することによって行います。

```
<?xml version="1.0"?>
<!-- charts/BackgroundFills.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1000},
        {Expense:"Bills", Amount:100}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Background Fill">
     <mx:BarChart
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Expense"
           />
        </mx:verticalAxis>
        <mx:fill>
           <mx:SolidColor
                color="0x66CCFF"
                alpha=".5"
           />
        </mx:fill>
        <mx:series>
           <mx:BarSeries xField="Amount"/>
        </mx:series>
     </mx:BarChart>
  </mx:Panel>
</mx:Application>
```

CSS を使用した塗りの設定

fill プロパティは、CSS シンタックスを使用して <mx:Style> 宣言で使用することができます。タイプセレクタとクラスセレクタのいずれかを使用することができます。次の例では、カスタムの myBarChartStyle クラスセレクタに #FF0000 の塗りを設定しています。

```
<?xml version="1.0"?>
<!-- charts/BackgroundFillsCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1100},
        {Expense:"Bills", Amount:100}
     ]);
  ]]></mx:Script>

  <mx:Style>
     .myBarChartStyle {
        fill:#FF0000;
     }
  </mx:Style>

  <mx:Panel title="Background Fill">
     <mx:BarChart
        dataProvider="{expenses}"
        showDataTips="true"
        styleName="myBarChartStyle"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Expense"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries xField="Amount"/>
        </mx:series>
     </mx:BarChart>
  </mx:Panel>
</mx:Application>
```

これは Flex によって SolidColor オブジェクトに変換されます。

チャートコントロールを使用したグラデーションの塗りの使用

Flex には、グラデーションの塗りを指定できる塗りのクラスが 2 つ用意されています。LinearGradient クラスまたは RadialGradient クラスを GradientEntry クラスと共に使用して、グラデーションの塗りを指定します。次の表で、これらのクラスについて説明します。

クラス	内容
LinearGradient	チャートエレメントの境界を起点として開始されるグラデーションの塗りを定義します。LinearGradient オブジェクトと共に GradientEntry オブジェクトの配列を指定して、グラデーションによる色の段階的な変化を制御します。
RadialGradient	チャートエレメントの中央から放射状に広がるグラデーションの塗りを定義します。RadialGradient オブジェクトと共に GradientEntry オブジェクトの配列を指定して、グラデーションによる色の段階的な変化を制御します。
GradientEntry	グラデーションの色の段階的な変化を制御するオブジェクトを定義します。それぞれの GradientEntry オブジェクトには、次のプロパティが含まれています。 • `color` 色の値を指定します。 • `alpha` 透明度を指定します。有効な値は 0 (透明) ～ 1 (不透明) です。デフォルト値は 1 です。 • `ratio` 色 (`color`) をチャート内のどこで変化させるかをパーセント値で指定します。たとえば、`ratio` プロパティを .33 に設定した場合、チャートの 33% の地点で色が変化します。`ratio` プロパティを設定しなかった場合、他の GradientEntry オブジェクトの `ratio` プロパティに基づいて、均等に値が適用されます。有効な値の範囲は 0 から 1 です。

次の例では、LinearGradient クラスで 3 色を使用して、チャートのデータ系列にグラデーションの塗りを表現しています。

```
<?xml version="1.0"?>
<!-- charts/GradientFills.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000},
        {Expense:"Rent", Amount:1000},
        {Expense:"Bills", Amount:100}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Background Fill">
     <mx:BarChart
        dataProvider="{expenses}"
        showDataTips="true"
```

```
    >
       <mx:verticalAxis>
          <mx:CategoryAxis
               dataProvider="{expenses}"
               categoryField="Expense"
          />
       </mx:verticalAxis>
       <mx:fill>
          <mx:LinearGradient>
           <mx:entries>
               <mx:GradientEntry
                   color="0xC5C551"
                   ratio="0"
                   alpha="1"
               />
               <mx:GradientEntry
                   color="0xFEFE24"
                   ratio=".33"
                   alpha="1"
               />
               <mx:GradientEntry
                   color="0xECEC21"
                   ratio=".66"
                   alpha="1"
               />
           </mx:entries>
          </mx:LinearGradient>
       </mx:fill>
       <mx:series>
          <mx:BarSeries xField="Amount"/>
       </mx:series>
    </mx:BarChart>
  </mx:Panel>
</mx:Application>
```

LinearGradient オブジェクトに指定できる属性は、angle の 1 つだけです。デフォルトでは、トランジションはチャートの左から右へ定義されます。トランジションの方向を制御するには、angle プロパティを使用します。たとえば値が 180 の場合、トランジションは左から右ではなく、右から左に行われます。

次の例では、angle プロパティを 90 に設定しているため、トランジションはチャートの上から下に行われます。

```
<?xml version="1.0"?>
<!-- charts/GradientFillsAngled.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
```

```
      public var expenses:ArrayCollection = new ArrayCollection([
         {Expense:"Taxes", Amount:2000},
         {Expense:"Rent", Amount:1000},
         {Expense:"Bills", Amount:100}
      ]);
   ]]></mx:Script>
   <mx:Panel title="Background Fill">
      <mx:BarChart dataProvider="{expenses}" showDataTips="true">
         <mx:verticalAxis>
            <mx:CategoryAxis
                 dataProvider="{expenses}"
                 categoryField="Expense"
            />
         </mx:verticalAxis>
         <mx:fill>
            <mx:LinearGradient angle="90">
             <mx:entries>
                 <mx:GradientEntry
                     color="0xC5C551"
                     ratio="0"
                     alpha="1"
                 />
                 <mx:GradientEntry
                     color="0xFEFE24"
                     ratio=".33"
                     alpha="1"
                 />
                 <mx:GradientEntry
                     color="0xECEC21"
                     ratio=".66"
                     alpha="1"
                 />
              </mx:entries>
            </mx:LinearGradient>
         </mx:fill>
         <mx:series>
            <mx:BarSeries xField="Amount"/>
         </mx:series>
      </mx:BarChart>
   </mx:Panel>
</mx:Application>
```

複数のアルファ値を使用した塗りの表示

複数のデータ系列のチャートを作成するとき、系列同士が重なるように定義することができます。たとえば、縦棒グラフでは、縦棒それぞれを隣り合うように表示することも、複数のデータ系列を重ねて表示することもできます。これは、面グラフの系列についても同じです。

複数のデータ系列を重ねて表示する場合、各系列の塗りの alpha 値を 100% よりも小さく指定して、あるレベルの透明度を系列に設定することができます。alpha プロパティの有効な値は、0 (透明) ～1 (不透明) です。

次の例では、チャートの各系列で同じ透明度レベルの単色の塗りを使用する面グラフを定義します。

```
<?xml version="1.0"?>
<!-- charts/AlphaFills.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Area Chart">
     <mx:AreaChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:AreaSeries
                yField="Profit"
                displayName="Profit"
           >
                <mx:areaStroke>
                    <mx:Stroke
                        color="0x9A9A00"
                        weight="2"
                    />
                </mx:areaStroke>
                <mx:areaFill>
                    <mx:SolidColor
                        color="0x7EAEFF"
                        alpha=".3"
```

```
                    />
                </mx:areaFill>
            </mx:AreaSeries>
            <mx:AreaSeries
                yField="Expenses"
                displayName="Expenses"
            >
                <mx:areaStroke>
                    <mx:Stroke
                        color="0x9A9A00"
                        weight="2"
                    />
                </mx:areaStroke>
                <mx:areaFill>
                    <mx:SolidColor
                        color="0xAA0000"
                        alpha=".3"
                    />
                </mx:areaFill>
            </mx:AreaSeries>
        </mx:series>
    </mx:AreaChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

グラデーションの塗りを定義した場合、<mx:GradientEntry> タグの配列の各エントリで alpha プロパティを設定できます。次にその例を示します。

```
<?xml version="1.0"?>
<!-- charts/GradientAlphaFills.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Area Chart">
     <mx:AreaChart id="myChart" dataProvider="{expenses}"
     showDataTips="true">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
```

```
<mx:AreaSeries yField="Profit" displayName="Profit">
 <mx:areaStroke>
     <mx:Stroke color="0x9A9A00" weight="2"/>
 </mx:areaStroke>
 <mx:areaFill>
     <mx:LinearGradient angle="90">
         <mx:entries>
             <mx:GradientEntry
                 color="0xC5C551"
                 ratio="0"
                 alpha="1"
             />
             <mx:GradientEntry
                 color="0xFEFE24"
                 ratio=".33"
                 alpha="1"
             />
             <mx:GradientEntry
                 color="0xECEC21"
                 ratio=".66"
                 alpha=".2"
             />
         </mx:entries>
     </mx:LinearGradient>
 </mx:areaFill>
</mx:AreaSeries>
<mx:AreaSeries
 yField="Expenses"
 displayName="Expenses"
>
 <mx:areaStroke>
     <mx:Stroke color="0x9A9A00" weight="2"/>
 </mx:areaStroke>
 <mx:areaFill>
     <mx:LinearGradient angle="90">
         <mx:entries>
             <mx:GradientEntry
                 color="0xAA0000"
                 ratio="0"
                 alpha="1"
             />
             <mx:GradientEntry
                 color="0xCC0000"
                 ratio=".33"
                 alpha="1"
             />
             <mx:GradientEntry
                 color="0xFF0000"
                 ratio=".66"
                 alpha=".2"
```

```
                        />
                    </mx:entries>
                </mx:LinearGradient>
            </mx:areaFill>
        </mx:AreaSeries>
    </mx:series>
</mx:AreaChart>
<mx:Legend dataProvider="{myChart}"/>
</mx:Panel>
</mx:Application>
```

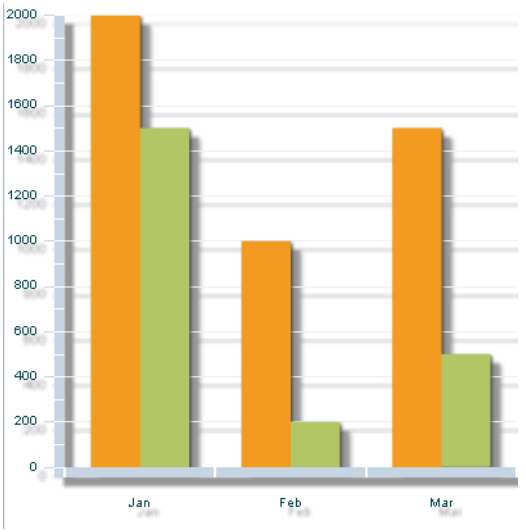
この例では、alpha プロパティを 0 に設定して、データ系列におけるグラデーションの最終色を完全な透明にしています。

フィルタの使用

flash.filters パッケージでクラスを使用して、ドロップシャドウなどのフィルタをチャートに追加できます。このような機能を持つフィルタは次のとおりです。

- BevelFilter
- BitmapFilter
- BlurFilter
- ColorMatrixFilter
- DisplacementMapFilter
- DropShadowFilter
- GlowFilter
- GradientBevelFilter
- GradientGlowFilter

フィルタは、チャートコントロール自体に適用することも、各チャート系列に適用することもできます。フィルタをチャートコントロールに適用すると、グリッド線、軸ラベル、系列内の各データポイントなど、そのチャートコントロールのすべての側面にフィルタが適用されます。次の図では、ColumnChart コントロールに適用されるドロップシャドウフィルタを示しています。



次の例では、ColumnChart コントロールにカスタムドロップシャドウフィルタを適用します。

```
<?xml version="1.0"?>
<!-- charts/ColumnWithDropShadow.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:flash="flash.filters.*">

  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month:"Jan", Profit:2000, Expenses:1500},
     {Month:"Feb", Profit:1000, Expenses:200},
     {Month:"Mar", Profit:1500, Expenses:500}
  ]);
  ]]></mx:Script>
  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
        <!-- Add a custom drop shadow filter to the
                ColumnChart control. -->
        <mx:filters>
           <flash:DropShadowFilter
                distance="10"
                color="0x666666"
```

```
                alpha=".8"
            />
        </mx:filters>
        <mx:horizontalAxis>
            <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
            />
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{column}"/>
  </mx:Panel>
</mx:Application>
```

Flex アプリケーションでフィルタを使用する場合は、次の例に示すように、MXML ファイルのトップレベルのタグに flash.filters 名前空間を追加してください。

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:flash="flash.filters.*">
```

フィルタの使用の詳細については、695 ページの「Flex でのフィルタの使用」を参照してください。

チャートコントロールによっては、ドロップシャドウフィルタを追加すると、予期しない結果が発生する可能性があります。たとえば、PieChart コントロールにドロップシャドウフィルタを追加すると、デフォルトで PieSeries に適用されるドロップシャドウフィルタに加えて、そのドロップシャドウフィルタがレンダリングされます。

フィルタは、次の例のように空の配列に filters 配列を設定すると削除できます。

```
<?xml version="1.0"?>
<!-- charts/ClearFilters.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
     {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
     {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
  ]);
```

```
  ]]></mx:Script>
  <mx:LineChart id="myChart" dataProvider="{expenses}">
     <mx:seriesFilters>
        <mx:Array/>
     </mx:seriesFilters>
     <mx:horizontalAxis>
        <mx:CategoryAxis
            dataProvider="{expenses}"
            categoryField="Month"
        />
     </mx:horizontalAxis>
     <mx:series>
        <mx:LineSeries
            yField="Profit"
            displayName="Profit"
        />
        <mx:LineSeries
            yField="Expenses"
            displayName="Expenses"
        />
        <mx:LineSeries
            yField="Amount"
            displayName="Amount"
        />
     </mx:series>
  </mx:LineChart>
</mx:Application>
```

次の例では、PieChart コントロールを作成してドロップシャドウフィルタを適用します。また、ドロップシャドウが1つになるように、PieSeries からデフォルトのドロップシャドウフィルタを削除します。

```
<?xml version="1.0"?>
<!-- charts/PieChartShadow.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:flash=
"flash.filters.*">
  <mx:Script>
    <![CDATA[
     [Bindable]
     public var expenses:Object = [
        {Expense:"Taxes", Amount:2000},
        {Expense:"Gas", Amount:100},
        {Expense:"Food", Amount:200}
     ];
    ]]>
  </mx:Script>
  <mx:Panel title="Pie Chart">
     <mx:PieChart id="pie"
        dataProvider="{expenses}"
        showDataTips="true"
     >
```

```
        <!-- Add a custom drop shadow to the PieChart control -->
        <mx:filters>
           <flash:DropShadowFilter
                distance="10"
                color="0x666666"
                alpha=".8"
           />
        </mx:filters>
        <mx:series>
           <mx:Array>
            <mx:PieSeries field="Amount" nameField="Expense"
            labelPosition="callout" explodeRadius=".2">
                <!-- Clear default shadow on the PieSeries -->
                <mx:filters>
                    <mx:Array/>
                </mx:filters>
            </mx:PieSeries>
           </mx:Array>
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{pie}"/>
  </mx:Panel>
</mx:Application>
```

グリッド線の使用方法の詳細については、1600 ページの「グリッド線の追加」を参照してください。

系列、グリッド線、凡例項目、軸など、表示オブジェクトである個別のチャートエレメントにフィルタを追加できます。次の例では、フィルタのセットを定義し、さまざまなチャートエレメントに適用しています。

```
<?xml version="1.0"?>
<!-- charts/MultipleFilters.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:flash=
"flash.filters.*" creationComplete="createFilters()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);

     private var myBlurFilter:BlurFilter;
     private var myGlowFilter:GlowFilter;
     private var myBevelFilter:BevelFilter;
     private var myDropShadowFilter:DropShadowFilter;

     private var color:Number = 0xFF33FF;

     public function applyFilters():void {
        // Apply filters to series, grid lines, legend, and axis.
        myGridlines.filters = [myBlurFilter];
```

```
      myLegend.filters = [myGlowFilter];
      myAxisRenderer.filters = [myBevelFilter];
      s1.filters = [myDropShadowFilter];
      s2.filters = [myDropShadowFilter];
   }

   public function createFilters():void {
      // Define filters.
      myBlurFilter = new BlurFilter(4,4,1);

      myGlowFilter = new GlowFilter(color, .8, 6, 6,
          2, 1, false, false);

      myDropShadowFilter = new DropShadowFilter(15, 45,
          color, 0.8, 8, 8, 0.65, 1, false, false);

      myBevelFilter = new BevelFilter(5, 45, color, 0.8,
          0x333333, 0.8, 5, 5, 1, BitmapFilterQuality.HIGH,
          BitmapFilterType.INNER, false);

      applyFilters();
   }
]]></mx:Script>

<mx:Panel>
   <mx:ColumnChart id="myChart" dataProvider="{expenses}">
      <mx:backgroundElements>
         <mx:GridLines id="myGridlines"
              horizontalChangeCount="1"
             verticalChangeCount="1"
             direction="both"
         >
         </mx:GridLines>
      </mx:backgroundElements>
      <mx:horizontalAxis>
         <mx:CategoryAxis
              dataProvider="{expenses}"
              categoryField="Month"
         />
      </mx:horizontalAxis>

      <mx:horizontalAxisRenderer>
         <mx:AxisRenderer id="myAxisRenderer"
              placement="bottom"
             canDropLabels="true"
         >
          <mx:axisStroke>
              <mx:Stroke color="#000080" weight="10"/>
          </mx:axisStroke>
          <mx:tickStroke>
              <mx:Stroke color="#000060" weight="5"/>
          </mx:tickStroke>
          <mx:minorTickStroke>
              <mx:Stroke color="#100040" weight="5"/>
```

```
            </mx:minorTickStroke>
         </mx:AxisRenderer>
      </mx:horizontalAxisRenderer>

      <mx:series>
         <mx:ColumnSeries id="s1"
              xField="Month"
              yField="Profit"
              displayName="Profit"
         />
         <mx:ColumnSeries id="s2"
              xField="Month"
              yField="Expenses"
              displayName="Expenses"
         />
      </mx:series>
   </mx:ColumnChart>
   <mx:Legend  id="myLegend" dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

グリッド線の追加

PieChart コントロール以外のチャートには、デフォルトでグリッド線が含まれています。これらのグリッド線は、CSS の gridLinesStyleName プロパティや、チャート系列の backgroundElements プロパティと annotationElements プロパティで制御できます。

GridLines オブジェクトを使用して、チャートに水平線、垂直線、または両方のグリッド線を表示できます。チャートの backgroundElements プロパティを使用すると、グリッド線をデータ系列の背後に設定できます。annotationElements プロパティを使用すると、グリッド線をデータ系列の手前に表示できます。

次の例では、両方向のグリッド線を有効にして、チャートに適用しています。

```
<?xml version="1.0"?>
<!-- charts/GridLinesBoth.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>

  <mx:Array id="bge">
     <mx:GridLines direction="both"/>
  </mx:Array>
```

```
  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart"
        dataProvider="{expenses}"
        backgroundElements="{bge}"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

メモ

annotationElements プロパティは、チャートの前面に表示される任意のチャートエレメントを示します。backgroundElements プロパティは、チャートのデータ系列の背後に表示される任意のチャートエレメントを示します。

次の例に示すように、各チャートコントロールの定義内でグリッド線を定義することもできます。

```
<?xml version="1.0"?>
<!-- charts/GridLinesBothInternal.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart" dataProvider="{expenses}">
        <mx:backgroundElements>
```

```
            <mx:GridLines direction="both"/>
        </mx:backgroundElements>

        <mx:horizontalAxis>
            <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
            />
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

グリッド線の塗りと線を定義するには、horizontalStroke、verticalStroke、horizontalFill、verticalFill の各プロパティを使用します。グリッド線の外観を定義する他のプロパティは、次のとおりです。

- horizontalAlternateFill
- horizontalChangeCount
- horizontalOriginCount
- horizontalShowOrigin
- horiztontalTickAligned
- verticalAlternateFill
- verticalChangeCount
- verticalOriginCount
- verticalShowOrigin
- verticalTickAligned

線の操作の詳細については、1577 ページの「線の使用」を参照してください。backgroundElements プロパティと annotationElements プロパティの使用の詳細については、1542 ページの「ChartElement オブジェクトの追加」を参照してください。

グリッド線の外観は、MXML の中で、または ActionScript か CSS を使用すると、直接操作できます。以降のセクションでは、Flex チャートオブジェクトのグリッド線を書式設定する方法について説明します。

MXML を使用したグリッド線の書式設定

グリッド線の外観を制御するには、次の例のように、GridLines オブジェクトの配列を MXML タグとして指定します。

```
<?xml version="1.0"?>
<!-- charts/GridLinesFormatMXML.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>

  <mx:Array id="bge">
     <mx:GridLines
        horizontalChangeCount="1"
        verticalChangeCount="1"
        direction="both"
     >
        <mx:horizontalStroke>
           <mx:Stroke weight="3"/>
        </mx:horizontalStroke>
        <mx:verticalStroke>
           <mx:Stroke weight="3"/>
        </mx:verticalStroke>
        <mx:horizontalFill>
           <mx:SolidColor color="0x99033" alpha=".66"/>
        </mx:horizontalFill>
     </mx:GridLines>
  </mx:Array>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart"
        dataProvider="{expenses}"
        backgroundElements="{bge}"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
```

```
                yField="Profit"
                displayName="Profit"
            />
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

この例では、changeCount プロパティを使用して、軸上の各目盛り位置にグリッド線が描画されるように指定し、さらに direction プロパティを both に設定しています。この設定では、縦横両方のグリッド線が描画されます。direction プロパティの値として、horizontal または vertical を指定することもできます。

ドロップシャドウ、グロー、ベベルなどのフィルタを使用して、グリッド線の外観を変更することもできます。詳細については、1594 ページの「フィルタの使用」を参照してください。

CSS を使用したグリッド線の書式設定

GridLines オブジェクトに CSS のスタイルを適用することで、グリッド線のスタイルを設定することができます。次の例では、グリッド線に myStyle スタイルを適用しています。

```
<?xml version="1.0"?>
<!-- charts/GridLinesFormatCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>

  <mx:Style>
     .myStyle {
        direction:"both";
        horizontalShowOrigin:true;
        horizontalTickAligned:false;
        horizontalChangeCount:1;
        verticalShowOrigin:false;
        verticalTickAligned:true;
        verticalChangeCount:1;
```

```
        horizontalFill:#990033;
        horizontalAlternateFill:#00CCFF;
     }
  </mx:Style>

  <mx:Array id="bge">
     <mx:GridLines styleName="myStyle">
        <mx:horizontalStroke>
           <mx:Stroke weight="3"/>
        </mx:horizontalStroke>
        <mx:verticalStroke>
           <mx:Stroke weight="3"/>
        </mx:verticalStroke>
     </mx:GridLines>
  </mx:Array>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart"
        dataProvider="{expenses}"
        backgroundElements="{bge}"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

ActionScript を使用したグリッド線の書式設定

ActionScript では、実行時に GridLines を操作できます。次の例では、チャート系列の手前と背後に塗りつぶされたグリッド線を追加しています。

```
<?xml version="1.0"?>
<!-- charts/GridLinesFormatActionScript.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.graphics.SolidColor;
     import mx.graphics.Stroke;
     import mx.charts.GridLines;
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);

     [Bindable]
     public var bge:GridLines;

     public function addGridLines():void {
        bge = new GridLines();

        var s:Stroke = new Stroke(0xff00ff, 2);
        bge.setStyle("horizontalStroke", s);

        var f:SolidColor = new SolidColor(0x990033, .3);
        bge.setStyle("horizontalFill",f);

        var f2:SolidColor = new SolidColor(0x336699, .3);
        bge.setStyle("horizontalAlternateFill",f2);

        myChart.backgroundElements = [bge];
     }
  ]]></mx:Script>

  <mx:Panel title="Column Chart">
     <mx:ColumnChart id="myChart"
        dataProvider="{expenses}"
        creationComplete="addGridLines()"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
```

```
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
           />
           <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:ColumnChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

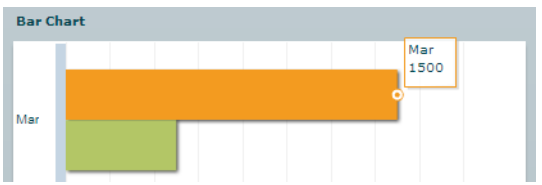
データヒントの使用

DataTip オブジェクトを有効にするには、チャートコントロールの showDataTips プロパティを使用します。データヒントは、マウスポインタの下に小さなポップアップウィンドウを表示して、データポイントに対応するデータ値を示すという点で、Flex のツールヒントに似ています。

メモ

DataTip コントロールと PieChart のラベルにはしばしば同じ情報が表示されますが、両者は別のものです。PieChart ラベルは、ユーザーのマウスポインタの位置に関係なく、常に表示されます。

次の図は単純なデータヒントです。



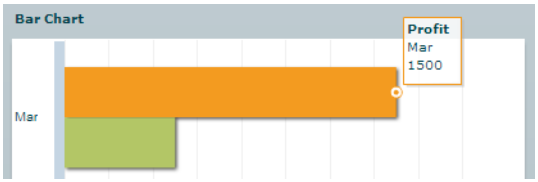
データヒントを有効にするには、次のように、チャートコントロールの showDataTips プロパティ値を true に設定します。

```
<?xml version="1.0"?>
<!-- charts/EnableDataTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
```

```
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
    ]);
  ]]></mx:Script>
  <mx:Panel title="Bar Chart">
     <mx:BarChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                yField="Month"
                xField="Profit"
           />
           <mx:BarSeries
                yField="Month"
                xField="Expenses"
           />
        </mx:series>
     </mx:BarChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

データヒントの形式はチャートの種類によって異なりますが、通常はデータプロバイダのフィールド名が表示されます。このフィールドによって、データプロバイダから渡されるフィールド名および対応するデータの値が、チャート内の選択された位置に取得されます。

チャートの系列に理解しやすい名前を付けることによって、データヒントの情報をユーザーにわかりやすく提供することができます。次の図に示すように、Adobe Flash Player では、この名前はデータヒントに表示されます。

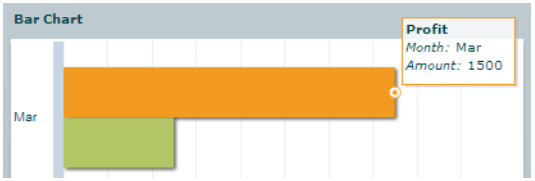


次の例では、データ系列の displayName プロパティを使用して、データ系列に名前を付けています。

```
<?xml version="1.0"?>
<!-- charts/DataTipsDisplayName.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Bar Chart">
     <mx:BarChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                yField="Month"
                xField="Profit"
                displayName="Profit"
           />
           <mx:BarSeries
                yField="Month"
                xField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:BarChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

また、displayName プロパティを使用して、軸に名前を付け、データヒント内にラベルを表示させることもできます。軸に名前を割り当てた場合、その名前がデータヒントに表示されます。軸名はラベルデータの前にイタリックフォントで表示されます。次の図にこの例を示します。



データヒントにラベルを追加するためだけに軸を追加する場合も考えられます。次の例では、縦横の軸に名前を付けて、両方のデータポイントがデータヒント内のラベルに表示されるようにしています。

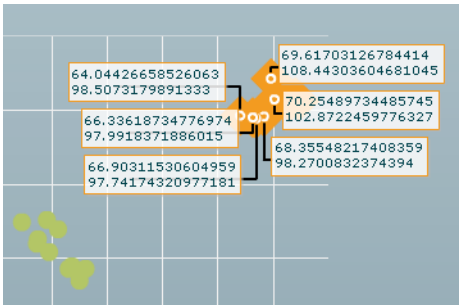
```
<?xml version="1.0"?>
<!-- charts/DataTipsAxisNames.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Bar Chart">
     <mx:BarChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
                displayName="Month"
           />
        </mx:verticalAxis>

        <mx:horizontalAxis>
           <mx:LinearAxis displayName="Amount"/>
        </mx:horizontalAxis>

        <mx:series>
           <mx:BarSeries
                yField="Month"
                xField="Profit"
                displayName="Profit"
           />
           <mx:BarSeries
                yField="Month"
                xField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:BarChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

複数のデータヒントの表示

チャートコントロールの dataTipMode プロパティを使用すると、複数のデータヒントを表示できます。表示オプションは、single と multiple です。dataTipMode を multiple に設定すると、チャートには、カーソルの範囲内にあるすべてのデータヒントが表示されます。次の例では、ColumnChart コントロールの dataTipMode プロパティの値を multiple に設定しています。

```
<?xml version="1.0"?>
<!-- charts/DataTipsMultiple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Bar Chart">
     <mx:BarChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
        mouseSensitivity="50"
        dataTipMode="multiple"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                yField="Month"
                xField="Profit"
           />
           <mx:BarSeries
                yField="Month"
                xField="Expenses"
           />
        </mx:series>
     </mx:BarChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

次の例は、dataTipMode プロパティを multiple に設定したデータヒントを示しています。



dataTipMode のデフォルト値は、チャートタイプによって異なります。設定値は、そのチャートタイプに重複するデータヒントがあるかどうかを基に決められています。次のチャートタイプの dataTipMode プロパティのデフォルト値は single です。

- BarChart
- CandlestickChart
- ColumnChart
- HLOCChart
- PieChart

その他のチャートタイプの dataTipMode プロパティのデフォルト値は multiple です。

データポイントの周りのインタラクティブな領域のサイズは、mouseSensitivity プロパティで設定します。データポイントの周りで、click や mouseOver などのマウスイベントに反応する領域の範囲は、mouseSensitivity プロパティを使用してピクセル単位で設定します。このプロパティを使用すると、ユーザーがマウスポインタをデータポイントの上ではなく近くに移動したときにデータヒントを表示させることができます。詳細については、1654 ページの「マウス感度の変更」を参照してください。

データヒントの値のカスタマイズ

データヒント内に表示されるテキストをカスタマイズするには、dataTipFunction コールバック関数を使用します。dataTipFunction コールバック関数を指定すると、Flex によってデータヒントのレンダリングとテキストのカスタマイズが行われる前に、データヒント内のデータにアクセスできます。

コールバック関数へのパラメータは HitData オブジェクトです。このため、データヒントのコールバック関数を使用する際には、mx.charts.HitData を読み込む必要があります。

コールバック関数で返された値が DataTip ボックスに表示されます。コールバック関数の戻り値の型として、String を指定する必要があります。

次の例では、データヒントの値を書式設定して返す新しいコールバック関数 dtFunc を定義しています。

```
<?xml version="1.0"?>
<!-- charts/CustomDataTips.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.charts.HitData;
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500},
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
     ]);

     public function dtFunc(hd:HitData):String {
        return hd.item.Month + ":<B>$" +
            hd.item.Profit + "</B>";
     }
  ]]></mx:Script>
  <mx:Panel title="Bar Chart">
     <mx:BarChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
        dataTipFunction="dtFunc"
     >
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
                displayName="Month"
           />
        </mx:verticalAxis>

        <mx:horizontalAxis>
           <mx:LinearAxis displayName="Amount"/>
        </mx:horizontalAxis>

        <mx:series>
           <mx:BarSeries
                yField="Month"
                xField="Profit"
                displayName="Profit"
           />
           <mx:BarSeries
                yField="Month"
                xField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
```

```
    </mx:BarChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

HitData オブジェクトを使用して、そのデータ項目が表示される系列に関する情報を取得することもできます。これは、次の例のように、HitData オブジェクトを Series クラスにキャストして行います。

```
<?xml version="1.0"?>
<!-- charts/HitDataCasting.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize=
"init()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.HitData;
     import mx.charts.series.ColumnSeries;

     [Bindable]
     public var dataSet:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Expenses:1500, Income:1590},
        {Month:"Feb", Expenses:1200, Income:1300},
        {Month:"Mar", Expenses:750, Income:900}
     ]);

     public var b:Boolean = true;

     public function myDataTipFunction(e:HitData):String {
        var s:String;
        s = ColumnSeries(e.element).displayName + "\n";
        s += "Profit: $" + (e.item.Income - e.item.Expenses);
        return s;
     }

  ]]></mx:Script>
  <mx:Panel title="Casting HitData Objects">
     <mx:ColumnChart id="myChart"
        dataProvider="{dataSet}"
        showDataTips="true"
        dataTipFunction="myDataTipFunction"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="Month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries
                yField="Expenses"
                displayName="Expenses '06"
           />
           <mx:ColumnSeries
                yField="Income"
                displayName="Income '06"
```

```
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

系列アイテムには、DataTip カスタム関数の HitData オブジェクトからアクセスすることもできます。chartItem プロパティは、ChartItem クラスのサブクラスのインスタンスを参照します。タイプは系列タイプによって異なります。たとえば、ColumnSeries の chartItem は ColumnSeriesItem クラスのインスタンスです。

次の例では、ColumnSeriesItem の yValue は、系列が 100% チャートで使用するパーセントを表しています。

```
<?xml version="1.0"?>
<!-- charts/HitDataCastingWithPercent.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize=
"init()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.HitData;
     import mx.charts.series.ColumnSeries;
     import mx.charts.series.items.ColumnSeriesItem;

     [Bindable]
     public var dataSet:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Income:1500, Profit:90},
        {Month:"Feb", Income:1200, Profit:100},
        {Month:"Mar", Income:750, Profit:150}
     ]);

     public var b:Boolean = true;

     public function myDataTipFunction(e:HitData):String {

        var s:String;
        s = "<B>" + ColumnSeries(e.element).displayName + "</B>\n";

        s += "<I>Income:</I> <FONT COLOR='#339966'>$" +
            e.item.Income + "</FONT>\n";
        s += "<I>Expenses:</I> <FONT COLOR='#FF0000'>$" +
            (e.item.Income - e.item.Profit) + "</FONT>\n";
        s += "------------------------\n";
        s += "<I>Profit:</I> $" + e.item.Profit + "\n";

        // The value of the Income will always be 100%,
        // so exclude adding that to the DataTip. Only
        // add percent when the user gets the Profit DataTip.
        var percentValue:Number =
```

```
            Number(ColumnSeriesItem(e.chartItem).yValue);
        if (percentValue < 100) {
            s += "Profit was equal to about <B>" +
                Math.round(percentValue) + "</B>% of the income.\n";
        }
        return s;

        //return e.item.Month + ":<B>$" + e.item.Profit + "</B>";
    }
  ]]></mx:Script>
  <mx:Panel title="Accessing ChartItems from HitData Objects">
    <mx:ColumnChart id="myChart"
      dataProvider="{dataSet}"
      type="100%"
      dataTipFunction="myDataTipFunction"
      showDataTips="true"
    >
      <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="Month"/>
      </mx:horizontalAxis>
      <mx:series>
        <mx:ColumnSeries
            yField="Profit"
            displayName="Profit '06"
        />
        <mx:ColumnSeries
            yField="Income"
            displayName="Income '06"
        />
      </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

チャートイベントの HitData オブジェクトの使用方法については、1645 ページの「HitData オブジェクトの使用」を参照してください。

単純な HTML を返す DataTip コールバック関数を作成すれば、データヒントに書式を設定することもできます。Flex では、<FONT>、<B>、<I>、
 などの HTML タグの小さなサブセットを使用できます。

ChartItem オブジェクトへのスキンの適用

ChartItem オブジェクトは、系列にデータポイントを示します。系列のデータプロバイダには、アイテムごとに 1 つの ChartItem インスタンスがあります。ChartItem オブジェクトには、データポイントのデータの詳細の他に、系列にそのデータポイントをレンダリングする際に使用するレンダラー (スキン) についての詳細が含まれています。ChartItem レンダラーでは、PlotChart コントロールにデータポイントを示すアイコンや、BarChart コントロールで横棒を構成する四角形などのオブジェクトを定義します。

各系列には、その系列の ChartItem オブジェクトを描画する際に Flex によって使用されるデフォルトレンダラーがありますが、系列の itemRenderer スタイルプロパティで、新しく使用するレンダラーを指定することができます。このプロパティで、ChartItem オブジェクトの外観を定義するクラスを指定します。

既存のクラスを使用して、チャートアイテムのデフォルトレンダラーを変更することができます。DiamondItemRenderer クラスは、PlotChart コントロールのデータ系列にある ChartItem オブジェクトに使用するデフォルトレンダラーです。次の例では、最初のデータ系列にはデフォルトの DiamondItemRenderer クラスを使用しています。2 番目のデータ系列では CircleItemRenderer クラスを使用して、そのデータ系列のデータポイントを円で示します。3 番目のデータ系列では CrossItemRenderer クラスを使用して、そのデータ系列のデータポイントを十字型で示します。

```
<?xml version="1.0"?>
<!-- charts/PlotRenderers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Plot Chart">
     <mx:PlotChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:series>
           <!-- First series uses default renderer. -->
           <mx:PlotSeries
                xField="Expenses"
                yField="Profit"
                displayName="Plot 1"
```

```
            />

            <!-- Second series uses DiamondItemRenderer. -->
            <mx:PlotSeries
                 xField="Amount"
                 yField="Expenses"
                 displayName="Plot 2"
                 itemRenderer="mx.charts.renderers.CircleItemRenderer"
            />

            <!-- Third series uses CrossItemRenderer. -->
            <mx:PlotSeries
                 xField="Profit"
                 yField="Amount"
                 displayName="Plot 3"
                 itemRenderer="mx.charts.renderers.CrossItemRenderer"
            />
        </mx:series>
    </mx:PlotChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

複数のレンダラークラスの使用

データ系列によっては、複数のレンダラーを選択できる場合があります。複数のレンダラーを使用すると、影やグラフィックを追加してチャートの外観を簡単に変更することができます。

次の表は、それぞれのチャートタイプの ChartItem オブジェクトで使用可能なレンダラークラスの一覧です。

チャートタイプ	使用可能なレンダラークラス
AreaChart	AreaRenderer
BarChart BubbleChart ColumnChart PlotChart	BoxItemRenderer CircleItemRenderer CrossItemRenderer DiamondItemRenderer ShadowBoxItemRenderer TriangleItemRenderer
CandlestickChart HLOCChart	CandlestickItemRenderer HLOCItemRenderer
LineChart	LineRenderer ShadowLineRenderer
PieChart	WedgeItemRenderer

ほとんどのレンダラーの外観は、その名前が示すとおりです。BoxItemRenderer クラスは、ChartItem オブジェクトを矩形で描画します。DiamondItemRenderer クラスは、ChartItem オブジェクトをダイヤモンド型で描画します。ShadowBoxItemRenderer クラスと ShadowLineRenderer クラスは、描画した ChartItem オブジェクトに影を付けます。

データ系列によっては、データを完全にレンダリングするためには複数のレンダラーが必要です。たとえば、LineSeries オブジェクトには itemRenderer スタイルプロパティと lineSegmentRenderer スタイルプロパティの両方があります。itemRenderer プロパティではデータアイテムのレンダラーを指定し、lineSegmentRenderer ではアイテムの間の線セグメントの外観を指定します。

2 つのレンダラーを必要とするもう 1 つの系列タイプは、AreaSeries です。areaRenderer プロパティで領域の外観を指定し、itemRenderer でデータアイテムの外観を指定します。

凡例マーカーに使用するレンダラーを指定することもできます。デフォルトは、データ系列の itemRenderer プロパティで指定するクラスです。詳細については、1624 ページの「Legend コントロールの使用」を参照してください。

1 つのチャートで複数の種類のデータ系列を使用できます。たとえば、ColumnSeries と LineSeries を使用すると、株価の上に移動平均のような数値を表示できます。この場合、系列でサポートされているレンダラーはすべて、1 つのチャート内で使用できます。複数のデータ系列の使用の詳細については、1524 ページの「複数のデータ系列の使用」を参照してください。

カスタムレンダラーの作成

カスタムレンダラーでチャート系列の itemRenderer プロパティを置き換えることができます。チャート系列の itemRenderer スタイルプロパティにカスタムレンダラーを定義します。このレンダラーには、グラフィカルレンダラーを使用することも、プログラムでレンダラーを定義するクラスを使用することもできます。

グラフィカルレンダラーの作成

GIF や JPEG などのグラフィックファイルをチャート系列のレンダラーとして使用することができます。これは、itemRenderer スタイルプロパティの値に埋め込みイメージを設定して行います。グラフィカルにチャートをレンダリングするこの方法は、他のコンポーネントに使用するグラフィカルスキンメソッドに似ています。詳細については、743 ページの「グラフィカルスキニング」を参照してください。

次の例では、グラフィックファイルを使用して PlotChart コントロールにデータポイントを示しています。

```
<?xml version="1.0"?>
<!-- charts/CustomPlotRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
```

```
    [Bindable]
    public var expenses:ArrayCollection = new ArrayCollection([
       {Month:"January", Profit:2000, Expenses:1500, Amount:450},
       {Month:"February", Profit:1000, Expenses:200, Amount:600},
       {Month:"March", Profit:1500, Expenses:500, Amount:300},
       {Month:"April", Profit:500, Expenses:300, Amount:500},
       {Month:"May", Profit:1000, Expenses:450, Amount:250},
       {Month:"June", Profit:2000, Expenses:500, Amount:700}
    ]);
 ]]></mx:Script>
 <mx:Panel title="Plot Chart">
    <mx:PlotChart id="myChart"
       dataProvider="{expenses}"
       showDataTips="true"
    >
       <mx:series>

          <!-- First series uses embedded image for renderer. -->
          <mx:PlotSeries
               xField="Expenses"
               yField="Profit"
               displayName="Plot 1"
               itemRenderer="@Embed('../assets/bird.gif')"
               radius="50"
               legendMarkerRenderer="@Embed('../assets/bird.gif')"
          />

          <!-- Second series uses DiamondItemRenderer. -->
          <mx:PlotSeries
               xField="Amount"
               yField="Expenses"
               displayName="Plot 2"
               itemRenderer="mx.charts.renderers.CircleItemRenderer"
          />

          <!-- Third series uses CrossItemRenderer. -->
          <mx:PlotSeries
               xField="Profit"
               yField="Amount"
               displayName="Plot 3"
               itemRenderer="mx.charts.renderers.CrossItemRenderer"
          />
       </mx:series>
    </mx:PlotChart>
    <mx:Legend dataProvider="{myChart}"/>
 </mx:Panel>
</mx:Application>
```

この例では、"bird.gif" グラフィックを使用して、プロットチャートに各データポイントを示しています。埋め込みイメージのサイズは、radius プロパティで制御します。

itemRenderer プロパティの値をインラインで設定する必要はありません。グラフィックファイルを Class として ActionScript に組み込んで ClassFactory にキャストし、インラインで参照することもできます。次に例を示します。

```
<?xml version="1.0"?>
<!-- charts/CustomPlotRendererAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ]);

     [Bindable]
     [Embed(source="../assets/bird.gif")]
     public var myBird:Class;

     [Bindable]
     public var myBirdFactory:ClassFactory =
        new ClassFactory(myBird);

  ]]></mx:Script>
  <mx:Panel title="Plot Chart">
     <mx:PlotChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:series>

           <!-- First series uses ActionScript class renderer. -->
           <mx:PlotSeries
                xField="Expenses"
                yField="Profit"
                displayName="Plot 1"
                itemRenderer="{myBirdFactory}"
                legendMarkerRenderer="{myBirdFactory}"
                radius="50"
           />

           <!-- Second series uses DiamondItemRenderer. -->
           <mx:PlotSeries
                xField="Amount"
                yField="Expenses"
                displayName="Plot 2"
                itemRenderer="mx.charts.renderers.CircleItemRenderer"
           />

           <!-- Third series uses CrossItemRenderer. -->
```

```
            <mx:PlotSeries
                 xField="Profit"
                 yField="Amount"
                 displayName="Plot 3"
                 itemRenderer="mx.charts.renderers.CrossItemRenderer"
            />
        </mx:series>
    </mx:PlotChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

プログラムレンダラーの作成

チャートアイテムのカスタムレンダラークラスを作成すると、単純なグラフィカルレンダラーよりも操作性が高くなります。クラスベースのレンダラーを使用することは、プログラムスキンを使用することに非常に似ています。詳細については、751 ページの「プログラムスキン」を参照してください。

その方法の 1 つは、ProgrammaticSkin クラスを拡張して、IDataRenderer インターフェイスを実装することです。この方法では、チャートアイテムを描画するすべてのロジックをカスタムクラスで使用でき、カスタムクラスの外観を自由に制御できます。たとえば、Graphics クラスでメソッドを使用して、BarChart コントロールの横棒の矩形を描画して塗りつぶします。

IDataRenderer インターフェイスを実装する場合は、setter メソッドと getter メソッドを定義して、data プロパティを実装する必要があります。この data プロパティは、系列アイテムのタイプと一致します。ColumnSeries の場合は ColumnSeriesItem です。その他のアイテムタイプには、BarSeriesItem、BubbleSeriesItem、LineSeriesItem、PlotSeriesItem などがあります。

クラスの中で、チャートアイテムを描画してカスタムプロパティを設定するロジックで、updateDisplayList() メソッドをオーバーライドします。super.updateDisplayList() メソッドも呼び出す必要があります。

次の例では、チャートアイテムをレンダリングし、色の配列を使用して ColumnChart コントロールの各列に異なる色を付けています。

```
// charts/CycleColorRenderer.as

package { // Empty package.

  import mx.charts.series.items.ColumnSeriesItem;
  import mx.skins.ProgrammaticSkin;
  import mx.core.IDataRenderer;
  import flash.display.Graphics;

  public class CycleColorRenderer extends mx.skins.ProgrammaticSkin
     implements IDataRenderer {

     private var colors:Array = [0xCCCC99,0x999933,0x999966];
     private var _chartItem:ColumnSeriesItem;
```

```
    public function CycleColorRenderer() {
       // Empty constructor.
    }

    public function get data():Object {
       return _chartItem;
    }

    public function set data(value:Object):void {
       _chartItem = value as ColumnSeriesItem;
       invalidateDisplayList();
    }

    override protected function
       updateDisplayList(unscaledWidth:Number,unscaledHeight:Number):void {
          super.updateDisplayList(unscaledWidth, unscaledHeight);
          var g:Graphics = graphics;
          g.clear();
          g.beginFill(colors[(_chartItem == null)? 0:_chartItem.index]);
          g.drawRect(0, 0, unscaledWidth, unscaledHeight);
          g.endFill();
    }
  } // Close class.
} // Close package.
```

Flex アプリケーションでは、次の例のように、ColumnSeries の itemRenderer プロパティを使用して、このクラスをレンダラーとして使用します。

```
<?xml version="1.0"?>
<!-- charts/ProgrammaticRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
        [Bindable]
        public var expenses:Object = [
           {Month:"Jan", Profit:2000, Expenses:1500},
           {Month:"Feb", Profit:1000, Expenses:200},
           {Month:"Mar", Profit:1500, Expenses:500}
        ];
    ]]>
  </mx:Script>

  <mx:Panel>
     <mx:ColumnChart id="column" dataProvider="{expenses}">
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
```

```
        <mx:series>
           <mx:Array>
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
                itemRenderer="CycleColorRenderer"
            />
           </mx:Array>
        </mx:series>
     </mx:ColumnChart>
  </mx:Panel>
</mx:Application>
```

updateDisplayList() メソッドのオーバーライドの詳細については、755 ページの「updateDisplayList() メソッドの実装」を参照してください。

Legend コントロールの使用

Legend コントロール (凡例) は、チャート上の塗りパターンを、それらの塗りパターンで示されるデータ系列について説明するラベルに対応させます。Legend コントロールの各項目を凡例項目と呼びます。凡例項目には、マーカーとラベルという 2 つの基本部分が含まれています。凡例項目は LegendItem タイプです。次の例は、凡例項目の 2 つの部分を示しています。

凡例マーカーは、塗りのパターンを対応させる以外にも、PlotChart コントロールのポイントなど、ChartItem オブジェクトのレンダラークラスも使用します。次の例は、3 つのデータ系列を含む PlotChart コントロールの Legend コントロールを示しています。それぞれのデータ系列では、独自のレンダラーを使用して特定の形を描画しており、Legend コントロールはそれらの形を示しています。

チャートへの Legend コントロールの追加

Legend クラスを使用すると、チャートに凡例を追加できます。Legend コントロールには、チャート内の各データ系列のラベルや、データ系列のチャートエレメントを示すキーが表示されます。

Legend コントロールを作成する場合、dataProvider プロパティを使用してチャートを Legend コントロールにバインドする方法が最も簡単です。次に例を示します。

```
<?xml version="1.0"?>
<!-- charts/LegendNamedSeries.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000, Cost:321, Discount:131},
        {Expense:"Rent", Amount:1000, Cost:95, Discount:313},
        {Expense:"Bills", Amount:100, Cost:478, Discount:841}
     ]);
  </mx:Script>

  <mx:Panel title="Bar Chart with Legend">
     <mx:BarChart id="myChart" dataProvider="{expenses}">
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Expense"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                xField="Amount"
                displayName="Amount (in $USD)"
           />
           <mx:BarSeries
                xField="Cost"
                displayName="Cost (in $USD)"
           />
           <mx:BarSeries
                xField="Discount"
                displayName="Discount (in $USD)"
           />
        </mx:series>
     </mx:BarChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>

</mx:Application>
```

次の例に示すように、Legend タイプの新しいオブジェクトをインスタンス化して、ActionScript でチャートに Legend を追加した後、コンテナの addChild() メソッドを呼び出して、コンテナに Legend を追加します。

```
<?xml version="1.0"?>
<!-- charts/LegendInAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="createLegend()">

  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.Legend;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000, Cost:321, Discount:131},
        {Expense:"Rent", Amount:1000, Cost:95, Discount:313},
        {Expense:"Bills", Amount:100, Cost:478, Discount:841}
     ]);

     private function createLegend():void {
        var myLegend:Legend = new Legend();
        myLegend.dataProvider = myChart;
        panel1.addChild(myLegend);
     }
  ]]></mx:Script>

  <mx:Panel id="panel1" title="Bar Chart with Legend (Created in
  ActionScript)">
     <mx:BarChart id="myChart" dataProvider="{expenses}">
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Expense"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                xField="Amount"
                displayName="Amount (in $USD)"
           />
           <mx:BarSeries
                xField="Cost"
                displayName="Cost (in $USD)"
           />
           <mx:BarSeries
                xField="Discount"
                displayName="Discount (in $USD)"
           />
        </mx:series>
```

```
    </mx:BarChart>
  </mx:Panel>

</mx:Application>
```

Legend コントロールにより、チャートコントロールの情報から凡例が作成されます。LegendItem マーカーの色と名前は、チャートのデータ系列の塗りとラベルと同じになります。上の例では、BarSeries コントロールの displayName プロパティを使用して、LegendItem ラベルを定義しています。

Legend コントロールでは、チャートのエレメントに名前が付いている必要があります。名前が付いていないと、Legend マーカーは表示されますが、ラベルは表示されません。

PieChart コントロールの Legend コントロールでは、データ系列の nameField プロパティを使用して凡例の値を検索します。nameField プロパティが参照する値には、ストリングを設定する必要があります。

次の例では、PieChart コントロールのデータ系列に対し、nameField プロパティを Expense に設定しています。Legend コントロールには、データプロバイダの Expense フィールドの値が使用されます。

```
<?xml version="1.0"?>
<!-- charts/LegendNameField.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>
  import mx.collections.ArrayCollection;
  [Bindable]
  public var expenses:ArrayCollection = new ArrayCollection([
     {Expense:"Taxes", Amount:2000},
     {Expense:"Rent", Amount:1000},
     {Expense:"Food", Amount:200}
  ]);
</mx:Script>

  <mx:Panel title="Pie Chart with Legend">
     <mx:PieChart id="myChart"
        dataProvider="{expenses}"
        showDataTips="true"
     >
        <mx:series>
           <mx:PieSeries
                field="Amount"
                nameField="Expense"
           />
        </mx:series>
     </mx:PieChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>

</mx:Application>
```

また、データヒントとラベルの系列も nameField プロパティによって定義されます。

Legend カスタムコントロールの作成

<mx:Legend> タグを定義して、<mx:LegendItem> タグを読み込むことで、MXML で Legend カスタムコントロールを作成できます。次の例では、複数の軸があるチャートのカスタムの凡例を作成しています。

```
<?xml version="1.0"?>
<!-- charts/MultipleAxesWithCustomLegend.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
      public var SMITH:ArrayCollection = new ArrayCollection([
        {date: "22-Aug-05", close: 41.87},
        {date: "23-Aug-05", close: 45.74},
        {date: "24-Aug-05", close: 42.77},
        {date: "25-Aug-05", close: 48.06},
     ]);

     [Bindable]
      public var DECKER:ArrayCollection = new ArrayCollection([
        {date: "22-Aug-05", close: 157.59},
        {date: "23-Aug-05", close: 160.3},
        {date: "24-Aug-05", close: 150.71},
        {date: "25-Aug-05", close: 156.88},
     ]);

  ]]></mx:Script>

  <mx:Panel title="Column Chart With Multiple Series">
     <mx:ColumnChart id="myChart"
        dataProvider="{SMITH}"
        secondDataProvider="{DECKER}"
        showDataTips="true"
     >

        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{SMITH}"
                categoryField="date"
           />
        </mx:horizontalAxis>

        <mx:verticalAxis>
           <mx:LinearAxis minimum="40" maximum="50"/>
        </mx:verticalAxis>

        <mx:series>
           <mx:ColumnSeries id="cs1"
```

```
                dataProvider="{SMITH}"
                xField="date"
                yField="close"
                displayName="SMITH"
           />
        </mx:series>

        <mx:secondVerticalAxis>
           <mx:LinearAxis minimum="150" maximum="170"/>
        </mx:secondVerticalAxis>

        <mx:secondSeries>
           <mx:LineSeries id="cs2"
                dataProvider="{DECKER}"
                xField="date"
                yField="close"
                displayName="DECKER"
           />
        </mx:secondSeries>
     </mx:ColumnChart>

     <mx:Legend>
        <mx:LegendItem label="SMITH" fontWeight="bold">
           <mx:fill>
            <mx:SolidColor color="0xFF9900"/>
           </mx:fill>
           <mx:stroke>
            <mx:Stroke color="0x000000" weight="1"/>
           </mx:stroke>
        </mx:LegendItem>
        <mx:LegendItem label="DECKER" fontWeight="bold">
           <mx:fill>
            <mx:SolidColor color="{0x999933}"/>
           </mx:fill>
           <mx:stroke>
            <mx:Stroke color="0x000000" weight="1"/>
           </mx:stroke>
         </mx:LegendItem>
     </mx:Legend>
  </mx:Panel>
</mx:Application>
```

Legend コントロールの書式設定

Legend コントロールは、Tile クラスのサブクラスです。Tile プロパティと Container クラスのいくつかのプロパティを使用すると、Legend コントロールを書式設定できます。さらに Legend コントロールには、labelPlacement、markerHeight、markerWidth などの、外観の書式設定に使用できるプロパティがあります。次の表で、Legend コントロールのプロパティについて説明します。

プロパティ	データ型	内容
labelPlacement	String	LegendItem オブジェクトのラベルの配置を指定します。有効な値は、right、left、top、および bottom です。
markerHeight	Number	LegendItem オブジェクトのマーカーの高さをピクセル単位で指定します。
markerWidth	Number	LegendItem オブジェクトのマーカーの幅をピクセル単位で指定します。
renderer	Object	LegendItem オブジェクトのマーカーのクラスを指定します。レンダラーには、IBoxRenderer インターフェイスを実装する必要があります。
stroke	Object	LegendItem オブジェクトのマーカーの線を指定します。線の定義の詳細については、1577 ページの「線の使用」を参照してください。

次の例では、Legend コントロールで、CSS を使用してスタイルを設定します。

```
<?xml version="1.0"?>
<!-- charts/FormattedLegend.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
     Legend {
        labelPlacement:left;
        markerHeight:30;
        markerWidth:30;
     }
  </mx:Style>

  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000, Cost:321, Discount:131},
        {Expense:"Rent", Amount:1000, Cost:95, Discount:313},
        {Expense:"Bills", Amount:100, Cost:478, Discount:841}
     ]);
  ]]></mx:Script>

  <mx:Panel title="Bar Chart with Legend">
```

```
    <mx:BarChart id="myChart" dataProvider="{expenses}">
       <mx:verticalAxis>
          <mx:CategoryAxis
               dataProvider="{expenses}"
               categoryField="Expense"
          />
       </mx:verticalAxis>
       <mx:series>
          <mx:BarSeries
               xField="Amount"
               displayName="Amount (in $USD)"
          />
          <mx:BarSeries
               xField="Cost"
               displayName="Cost (in $USD)"
          />
          <mx:BarSeries
               xField="Discount"
               displayName="Discount (in $USD)"
          />
       </mx:series>
    </mx:BarChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

LegendItem オブジェクトのマーカーの線の外観も変更することができます。これは、stroke プロパティまたは legendMarkerRenderer プロパティを使用して行います。詳細については、1577 ページの「線の使用」を参照してください。

Legend コントロールは、そのコントロールで定義する凡例がチャートのデータのスコープにアクセスできる限り、アプリケーションのどの場所にも配置できます。Legend コントロールは、コンテナなしでアプリケーションに直接配置することも、チャートと同じコンテナ内に配置することも、Panel コンテナなど独自のコンテナに配置することもできます。後者の方法では、Legend コントロールに境界線とタイトルバーが割り当てられます。次のように、Panel の title 属性を使用して、タイトルを作成することができます。

```
<?xml version="1.0"?>
<!-- charts/LegendInPanel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000, Cost:321, Discount:131},
        {Expense:"Rent", Amount:1000, Cost:95, Discount:313},
        {Expense:"Bills", Amount:100, Cost:478, Discount:841}
     ]);
  </mx:Script>
```

```
  <mx:Panel title="Bar Chart with Legend in Panel">
     <mx:BarChart id="myChart" dataProvider="{expenses}">
        <mx:verticalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Expense"
           />
        </mx:verticalAxis>
        <mx:series>
           <mx:BarSeries
                xField="Amount"
                displayName="Amount (in $USD)"
           />
           <mx:BarSeries
                xField="Cost"
                displayName="Cost (in $USD)"
           />
           <mx:BarSeries
                xField="Discount"
                displayName="Discount (in $USD)"
           />
        </mx:series>
     </mx:BarChart>
     <mx:Panel title="Legend">
        <mx:Legend dataProvider="{myChart}"/>
     </mx:Panel>
  </mx:Panel>

</mx:Application>
```

凡例の方向の設定

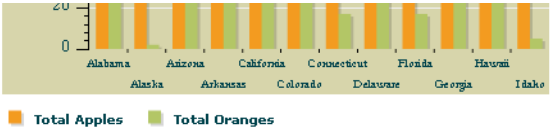
Tile コンテナから継承される、使用頻度の高いプロパティとして、direction プロパティがあります。<mx:Legend> タグの場合、このプロパティは LegendItem のオブジェクトを水平方向または垂直方向に整列させる目的で使用されます。direction のデフォルト値は vertical で、この場合は、LegendItem オブジェクトが縦に積み重なるようにして表示されます。

次の例では、direction プロパティを horizontal に設定しています。

```
<?xml version="1.0"?>
<!-- charts/HorizontalLegend.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
backgroundColor="white">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Expense:"Taxes", Amount:2000, Cost:321, Discount:131},
        {Expense:"Rent", Amount:1000, Cost:95, Discount:313},
```

```
        {Expense:"Bills", Amount:100, Cost:478, Discount:841}
    ]);
  ]]></mx:Script>

  <mx:Panel title="Bar Chart with Legend">
    <mx:BarChart id="myChart" dataProvider="{expenses}">
      <mx:verticalAxis>
        <mx:CategoryAxis
            dataProvider="{expenses}"
            categoryField="Expense"
        />
      </mx:verticalAxis>
      <mx:series>
        <mx:BarSeries
            xField="Amount"
            displayName="Amount (in $USD)"
        />
        <mx:BarSeries
            xField="Cost"
            displayName="Cost (in $USD)"
        />
        <mx:BarSeries
            xField="Discount"
            displayName="Discount (in $USD)"
        />
      </mx:series>
    </mx:BarChart>
    <mx:Legend
      dataProvider="{myChart}"
      direction="horizontal"
    />
  </mx:Panel>
</mx:Application>
```

次の例は、direction プロパティを horizontal に設定した凡例を示しています。



凡例マーカーの書式設定

凡例マーカーの外観は、プログラムレンダラークラスで定義できます。Flex では、凡例マーカーに使用できるデフォルトのレンダラークラスが複数用意されています。

LegendItem オブジェクトのレンダラーをデフォルトから ChartItem レンダラーのいずれかに変更するには、データ系列の legendMarkerRenderer スタイルプロパティを使用します。このプロパティでは、関連するすべての凡例にマーカーをレンダリングする場合に使用するクラスを指定します。

次の例では、3 つの系列すべての凡例マーカーを DiamondItemRenderer クラスに設定しています。

```
<?xml version="1.0"?>
<!-- charts/CustomLegendRenderer.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ]);

     [Bindable]
     [Embed(source="../assets/bird.gif")]
     public var myBird:Class;

     [Bindable]
     public var myBirdFactory:ClassFactory =
        new ClassFactory(myBird);

  ]]></mx:Script>
  <mx:Panel title="Plot Chart">
     <mx:PlotChart id="myChart" dataProvider="{expenses}"
     showDataTips="true">
        <mx:series>
           <!--
                Each series uses the default renderer for
                the ChartItems, but uses the same renderer
                for legend markers.
           -->

           <mx:PlotSeries
                xField="Expenses"
                yField="Profit"
                displayName="Plot 1"
                legendMarkerRenderer=
                "mx.charts.renderers.DiamondItemRenderer"
```

```
            />

            <mx:PlotSeries
                 xField="Amount"
                 yField="Expenses"
                 displayName="Plot 2"
                 legendMarkerRenderer=
                 "mx.charts.renderers.DiamondItemRenderer"
            />

            <mx:PlotSeries
                 xField="Profit"
                 yField="Amount"
                 displayName="Plot 3"
                 legendMarkerRenderer=
                 "mx.charts.renderers.DiamondItemRenderer"
            />
        </mx:series>
    </mx:PlotChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
</mx:Application>
```

legendMarkerRenderer プロパティを明示的に設定しない場合、データ系列の itemRenderer スタイルプロパティに指定されているデフォルトクラスが使用されます。それぞれの系列にはデフォルトレンダラーがあり、これらのスタイルプロパティのいずれも指定されていない場合に使用されます。

独自のカスタム汎用マーカークラスを作成できます。凡例マーカーとして使用するクラスには、IFlexDisplayObject インターフェイスを実装する必要があります。またオプションで ISimpleStyleClient インターフェイスと IDataRenderer インターフェイスを実装することもできます。

使用可能なレンダラークラスの詳細については、1617 ページの「ChartItem オブジェクトへのスキンの適用」を参照してください。

チャートの積み重ね

AreaChart、BarChart、ColumnChart の各コントロールを使用して複数のデータ系列をチャート化する場合、コントロールの type プロパティを使って系列の表示方法を制御できます。次の表で、type プロパティで使用できる値について説明します。

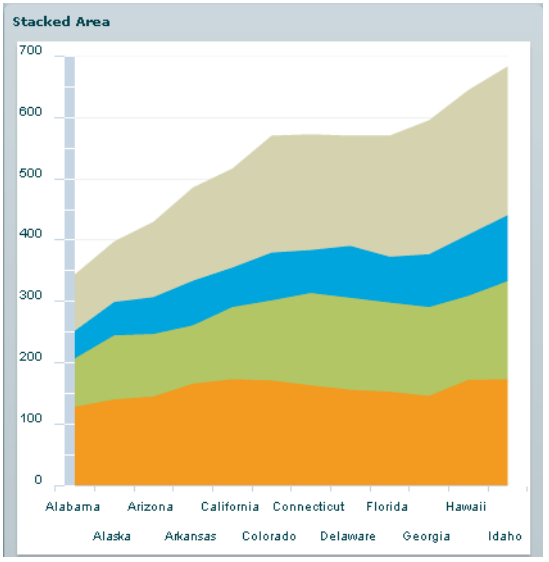
プロパティ	内容
clustered	各系列のチャートエレメントがカテゴリ別にグループ化されます。BarChart コントロールおよび ColumnChart コントロールでは、これがデフォルトの値です。
overlaid	各系列のチャートエレメントが縦に積み上げられ、最後の系列に対応するエレメントが一番上に表示されます。AreaChart コントロールでは、これがデフォルトの値です。
stacked	各系列のチャートエレメントが順次積み重ねられます。各エレメントには、積み重なったエレメントの累積値が表示されます。
100%	チャートエレメントは積み上げられ、合計が 100% になります。各チャートエレメントは、そのカテゴリの値の合計に対する割合 (%) で表示されます。

次の例では、4 つのデータ系列を積み重ねた AreaChart コントロールを作成しています。

```
<?xml version="1.0"?>
<!-- charts/AreaStacked.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
  ]]></mx:Script>
  <mx:Panel title="Line Chart">
     <mx:AreaChart
        dataProvider="{expenses}"
        showDataTips="true"
        type="stacked"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
           />
        </mx:horizontalAxis>
        <mx:series>
           <mx:AreaSeries
                yField="Profit"
```

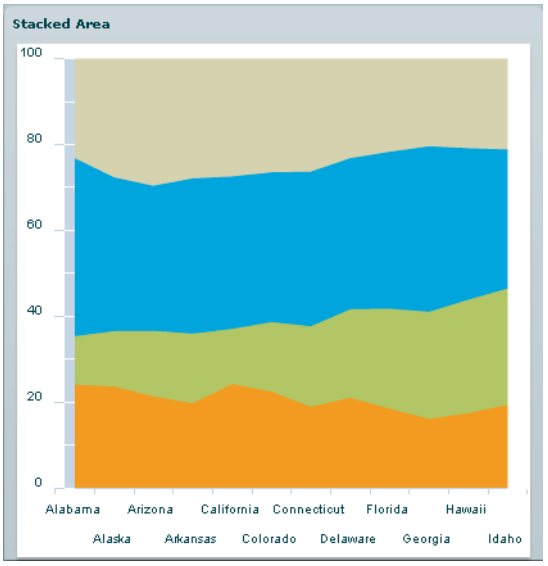
```
                displayName="Profit"
            />
            <mx:AreaSeries
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:AreaChart>
  </mx:Panel>
</mx:Application>
```

次の例は積み重ねた AreaChart です。



積み上げ (overlaid) の場合、最後の系列が手前に表示され、塗りの alpha プロパティを使って透明にしない限り、背後のデータ系列が見えにくくなる場合があります。詳細については、1583 ページの「塗りの使用」を参照してください。

type プロパティを 100% に設定すると、各系列が順次積み重ねられ、その合計を 100% とする領域が形成されます。各列は、そのカテゴリの値の合計に対する割合 (%) を示します。次に例を示します。



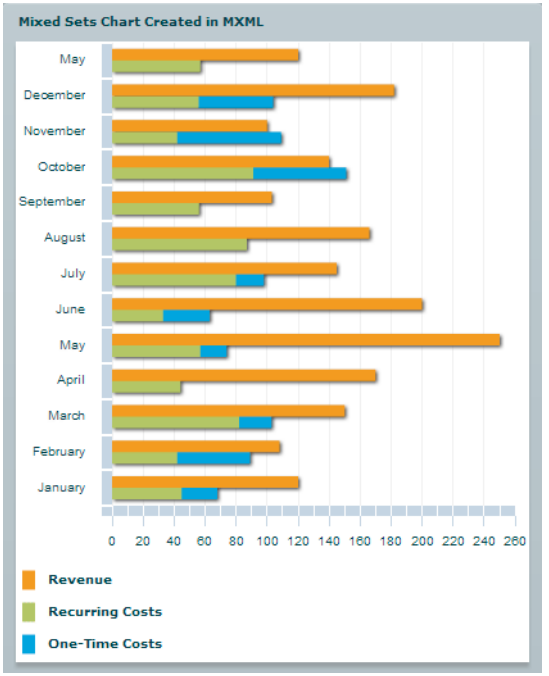
ColumnSet、BarSet、および AreaSet の各クラスを使用して、チャート系列のグループを結合することで、同じチャート内で異なるタイプの系列を使用できます。次の例では、BarSet クラスを使用して、1 つのチャートで集合と積み重ねの BarSeries を結合します。この例では、MXML と ActionScript の場合の方法を示しています。

```
<?xml version="1.0"?>
<!-- charts/UsingBarSets.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="initApp()">
  <mx:Script><![CDATA[
     import mx.charts.Legend;
     import mx.charts.BarChart;
     import mx.charts.series.BarSet;
     import mx.charts.series.BarSeries;
     import mx.collections.ArrayCollection;

     [Bindable]
     private var yearlyData:ArrayCollection = new ArrayCollection([
        {month:"January", revenue:120, costs:45,
            overhead:102, oneTime:23},
        {month:"February", revenue:108, costs:42,
            overhead:87, oneTime:47},
        {month:"March", revenue:150, costs:82,
            overhead:32, oneTime:21},
        {month:"April", revenue:170, costs:44,
            overhead:68},
        {month:"May", revenue:250, costs:57,
            overhead:77, oneTime:17},
```

```
    {month:"June", revenue:200, costs:33,
        overhead:51, oneTime:30},
    {month:"July", revenue:145, costs:80,
        overhead:62, oneTime:18},
    {month:"August", revenue:166, costs:87,
        overhead:48},
    {month:"September", revenue:103, costs:56,
        overhead:42},
    {month:"October", revenue:140, costs:91,
        overhead:45, oneTime:60},
    {month:"November", revenue:100, costs:42,
        overhead:33, oneTime:67},
    {month:"December", revenue:182, costs:56,
        overhead:25, oneTime:48},
    {month:"May", revenue:120, costs:57,
        overhead:30}
]);

private function initApp():void {
    var c:BarChart = new BarChart();
    c.dataProvider = yearlyData;

    var vAxis:CategoryAxis = new CategoryAxis();
    vAxis.categoryField = "month";
    c.verticalAxis = vAxis;

    var mySeries:Array = new Array();

    var outerSet:BarSet = new BarSet();
    outerSet.type = "clustered";
    var series1:BarSeries = new BarSeries();
    series1.xField = "revenue";
    series1.displayName = "Revenue";
    outerSet.series = [series1];

    var innerSet:BarSet = new BarSet();
    innerSet.type = "stacked";
    var series2:BarSeries = new BarSeries();
    var series3:BarSeries = new BarSeries();
    series2.xField = "costs";
    series2.displayName = "Recurring Costs";
    series3.xField = "oneTime";
    series3.displayName = "One-Time Costs";
    innerSet.series = [series2, series3];

    c.series = [outerSet, innerSet];

    var l:Legend = new Legend();
    l.dataProvider = c;
```

```
        panel2.addChild(c);
        panel2.addChild(l);
    }
  ]]></mx:Script>

  <mx:Panel title="Mixed Sets Chart Created in MXML" id="panel1">
     <mx:BarChart id="myChart" dataProvider="{yearlyData}">
        <mx:verticalAxis>
            <mx:CategoryAxis categoryField="month"/>
        </mx:verticalAxis>
        <mx:series>
            <mx:BarSet type="clustered">
                <mx:BarSeries xField="revenue"
                    displayName="Revenue"/>
                <mx:BarSet type="stacked">
                    <mx:BarSeries
                        xField="costs"
                        displayName="Recurring Costs"/>
                    <mx:BarSeries
                        xField="oneTime"
                        displayName="One-Time Costs"/>
                </mx:BarSet>
            </mx:BarSet>
        </mx:series>
     </mx:BarChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>
  <mx:Panel title="Mixed Sets Chart Created in ActionScript"
    id="panel2">
  </mx:Panel>
</mx:Application>
```

作成されるチャートには、2 つの集合系列が表示されます。次の例に示すように、一方はスタンドアローンの系列で、もう一方は積み重ね系列です。



第 56 章

チャートにおけるイベントとエフェクトの使用

56

Adobe Flex チャートを使用すると、人目を引く興味深いチャートを作成できます。考慮すべき重要なことは、ユーザーのアプリケーション操作でエフェクトやイベントをトリガする方法です。

目次

チャートでのユーザー操作の処理

チャートコントロールは、データポイントへのマウスの移動から、そのデータポイントのクリックまたはダブルクリックまで、多くの種類のユーザー操作を受け入れます。これらの各操作にイベントハンドラを作成し、そのハンドラの中で Event オブジェクトを使用して、操作に関連するデータにアクセスすることができます。たとえば、ColumnChart コントロールの縦棒をクリックすると、そのチャートの click イベントハンドラの中で縦棒のデータにアクセスできます。

チャートコントロールは、mouseMove、mouseOver、mouseUp、mouseDown、mouseOut などの、UIComponent クラスから継承されたマウスイベントをサポートします。これらのイベントのタイプは MouseEvent です。また、いくつかのチャートデータイベントは、すべてのチャートコントロールの基本クラスである ChartBase によって追加されています。次の表で、追加されているイベントについて説明します。

チャートデータイベント	説明
itemClick	ユーザーがデータポイント上でマウスボタンをクリックするとブロードキャストされます。
itemDoubleClick	ユーザーがデータポイント上でマウスボタンをダブルクリックするとブロードキャストされます。
itemMouseDown	ユーザーがデータポイント上でマウスボタンを押すとブロードキャストされます。
itemMouseMove	ユーザーがデータポイント上にマウスポインタを移動するとブロードキャストされます。
itemRollOut	マウスポインタから最も近いデータポイントが変わるとブロードキャストされます。
itemRollOver	ユーザーが新しいデータポイント上にマウスポインタを移動するとブロードキャストされます。
itemMouseUp	ユーザーがデータポイント上でマウスボタンを離すとブロードキャストされます。

チャートデータイベントは、ユーザーがマウスポインタをデータポイント上に移動した場合にのみトリガされます。一方、UIComponent イベントは、コントロールに何らかのマウス操作が行われた場合にトリガされます。

チャートデータイベントのタイプは ChartItemEvent です。ChartItemEvent イベントは、イベントパッケージではなくチャートパッケージに属しているため、ChartItemEvent を使用するには、mx.charts.events パッケージにある適切なクラスを読み込む必要があります。

次の例では、ユーザーがチャート内のデータポイント (縦棒) をクリックすると、メッセージが表示されます。

```
<?xml version="1.0"?>
<!-- charts/DataPointAlert.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.controls.Alert;
     import mx.charts.events.ChartItemEvent;
     import mx.collections.ArrayCollection;

     [Bindable]
     public var dataSet:ArrayCollection = new ArrayCollection([
```

```
        {Month:"Jan", Expenses:1500},
        {Month:"Feb", Expenses:200},
        {Month:"Mar", Expenses:500}
    ];

    public function myHandler(e:ChartItemEvent):void {
        Alert.show("Chart data was clicked");
    }

]]></mx:Script>
<mx:Panel title="Clickable Column Chart">
    <mx:ColumnChart id="myChart"
        itemClick="myHandler(event)"
        dataProvider="{dataSet}"
    >
        <mx:horizontalAxis>
            <mx:CategoryAxis
                dataProvider="{dataSet}"
                categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:ColumnSeries yField="Expenses"/>
        </mx:series>
    </mx:ColumnChart>
</mx:Panel>
</mx:Application>
```

チャートデータイベントにアクセスする際に、HitData オブジェクトを使用して、イベントがトリガされたときにマウスの下にあったデータを確認することができます。この機能により、特定のチャートデータにアクションを実行することができます。詳細については、1645 ページの「HitData オブジェクトの使用」を参照してください。

HitData オブジェクトの使用

Flex では、チャートデータイベントごとに ChartItemEvent オブジェクトが生成されます。すべての Event オブジェクトに含まれる標準の `target` プロパティと `type` プロパティの他に、ChartItemEvent オブジェクトには `hitData` プロパティが追加されています。このプロパティは、HitData クラスのオブジェクトインスタンスです。`hitData` プロパティには、マウスイベントの発生時にマウスポインタの最も近くにあったデータポイントの情報が格納されます。

次の例では、itemDoubleClick イベントハンドラを使用して、ユーザーが縦棒をクリックしたときの、縦棒グラフ内にあるデータポイントの HitData 情報を表示します。ColumnChart コントロールのそれぞれの縦棒には、単一のデータポイントの値が関連付けられているため、マウスで縦棒のどこをクリックしても同じ情報が表示されます。

```
<?xml version="1.0"?>
<!-- charts/HitDataOnClick.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize=
"init()">
  <mx:Script><![CDATA[
     import mx.charts.events.ChartItemEvent;
     import mx.collections.ArrayCollection;
     [Bindable]
     public var dataSet:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Expenses:1500},
        {Month:"Feb", Expenses:200},
        {Month:"Mar", Expenses:500}
     ]);
     // Define the event handler.
     public function myListener(e:ChartItemEvent):void {
        ti1.text = e.hitData.item.Expenses;
        ti2.text = e.hitData.x + "/" + e.hitData.y;
     }

     // Define event listeners when the application initializes.
     public function init():void {
        chart.addEventListener(ChartItemEvent.
            ITEM_DOUBLE_CLICK, myListener);
     }
  ]]></mx:Script>
  <mx:Panel title="Accessing HitData Object in Event Handlers">
     <mx:ColumnChart id="chart"
        dataProvider="{dataSet}" doubleClickEnabled="true"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="Month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:ColumnSeries yField="Expenses"/>
        </mx:series>
     </mx:ColumnChart>
     <mx:Form>
        <!--Define a form to display the event information.-->
        <mx:FormItem label="Expenses:">
           <mx:TextInput id="ti1"/>
        </mx:FormItem>
        <mx:FormItem label="x/y:">
           <mx:TextInput id="ti2"/>
        </mx:FormItem>
     </mx:Form>
  </mx:Panel>
</mx:Application>
```

HitData オブジェクトの item プロパティはデータポイントを参照します。前述の例のように、このプロパティを使用することにより、名前を指定してその値にアクセスすることができます。HitData の x プロパティと y プロパティは、データポイントの画面座標を参照します。

mouseSensitivity プロパティで指定した半径内にあるデータポイントによってのみ、そのデータポイント上にあるイベントがトリガされます。詳細については、1654 ページの「マウス感度の変更」を参照してください。

チャートエレメントの取得

HitData オブジェクトは、チャートの ChartItem オブジェクトにアクセスします。ChartItem オブジェクトは、画面上のデータポイントを示します。ChartItem オブジェクトは、データポイントのデータへのアクセスを提供する以外に、チャートを構成するグラフィカルエレメントのサイズと位置についての情報を提供します。たとえば、ColumnChart の縦棒の x 位置と y 位置を取得できます。

次の例では、半透明の Canvas コンテナを使用して、ユーザーがマウスでクリックしたデータポイントをハイライト表示します。また、ChartItem オブジェクトを調べて、その Canvas のツールヒントに表示する現在の値を取得します。

```
<?xml version="1.0"?>
<!-- charts/ChartItemObjectAccess.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">

  <mx:Style>
     ToolTip {
        fontSize:24;
     }
     ColumnChart {
        gutterLeft: 54;
     }
  </mx:Style>

  <mx:Script><![CDATA[
     import mx.core.IFlexDisplayObject;
     import mx.charts.events.ChartItemEvent;
     import mx.charts.series.items.ColumnSeriesItem;
     import mx.charts.series.ColumnSeries;
     import mx.effects.Move;
     import mx.charts.HitData;
     import mx.collections.ArrayCollection;

     public var m:mx.effects.Move;
     public var hitData:mx.charts.HitData;
     public var chartItem:ColumnSeriesItem;
     public var renderer:IFlexDisplayObject;
     public var series:ColumnSeries;
```

```
public var chartItemPoint:Point;
public var highlightBoxPoint:Point;
public var leftAdjust:Number;

private function init():void {
   m = new Move(highlightBox);

   // This is used to adjust the x location of
   // highlightBox to account for the left
   // gutter width.
   leftAdjust = myChart.getStyle("gutterLeft") - 14;
}

[Bindable]
private var dataSet:ArrayCollection = new ArrayCollection([
   {month:"Jan", income:12300, expense:3210},
   {month:"Feb", income:12450, expense:3100},
   {month:"Mar", income:13340, expense:3550},
   {month:"Apr", income:13489, expense:3560},
   {month:"May", income:11020, expense:4600},
   {month:"Jun", income:14030, expense:3410},
   {month:"Jul", income:15600, expense:4485},
   {month:"Aug", income:17230, expense:3892},
   {month:"Sep", income:15212, expense:3562},
   {month:"Oct", income:14980, expense:5603},
   {month:"Nov", income:15020, expense:4102},
   {month:"Dec", income:15923, expense:4789}]);

private function overData(
   event:mx.charts.events.ChartItemEvent):void
{

   hitData = event.hitData;
   chartItem = ColumnSeriesItem(hitData.chartItem);
   renderer = chartItem.itemRenderer;
   series = ColumnSeries(hitData.element);

   // Add 10 pixels to give it horizontal overlap.
   highlightBox.width = renderer.width * 2 + 10;

   // Add 20 pixels to give it vertical overlap
   highlightBox.height = renderer.height + 20;

   highlightBoxPoint = new Point(highlightBox.x,
       highlightBox.y);

   // Convert the ChartItem's pixel values from local
   // (relative to the component) to global (relative
   // to the stage). This enables you to place the Canvas
   // container using the x and y values of the stage.
```

```
        chartItemPoint = myChart.localToGlobal(new
            Point(chartItem.x, chartItem.y));

        // Define the parameters of the move effect and
        // play the effect.
        m.xTo = chartItemPoint.x + leftAdjust;
        m.yTo = chartItemPoint.y;
        m.duration = 500;
        m.play();

        highlightBox.toolTip = "$" + chartItem.yValue.toString();
    }

]]></mx:Script>

<mx:Panel id="p1">
    <mx:ColumnChart id="myChart"
        dataProvider="{dataSet}"
        itemClick="overData(event)"
        mouseSensitivity="0"
    >
        <mx:horizontalAxis>
            <mx:CategoryAxis categoryField="month"/>
        </mx:horizontalAxis>
        <mx:series>
            <mx:ColumnSeries
                 displayName="Expense"
                 yField="expense"
            />
            <mx:ColumnSeries
                 displayName="Income"
                 yField="income"
            />
        </mx:series>
    </mx:ColumnChart>
</mx:Panel>

<!-- Define the canvas control that will be used as a highlight -->
<mx:Canvas id="highlightBox"
       y="0"
       x="0"
       backgroundColor="0xFFFF00"
       alpha=".5"
/>
</mx:Application>
```

ChartItem オブジェクトの外観の変更の詳細については、1617 ページの「ChartItem オブジェクトへのスキンの適用」を参照してください。

座標を使用したデータの取得

チャートを作成する際、画面上の座標を使用して最も近いデータポイント (複数可) を取得できます。またその逆で、データポイントを渡して座標を取得することもできます。

`findDataPoints()` メソッドと `localToData()` メソッドは、座標を取得してデータを返します。`findDataPoints()` メソッドは HitData オブジェクトを返します。詳細については、1650 ページの「findDataPoints() メソッドの使用」を参照してください。`localToData()` メソッドは、データの配列を返します。詳細については、1651 ページの「localToData() メソッドの使用」を参照してください。

`dataToLocal()` メソッドを使用すると、データそのものを渡して座標を取得することもできます。詳細については、1653 ページの「dataToLocal() メソッドの使用」を参照してください。

findDataPoints() メソッドの使用

チャートコントロールの `findDataPoints()` メソッドに x 座標と y 座標を渡すことにより、HitData オブジェクトの配列を取得できます。指定した座標に対応する位置にデータポイントが存在しない場合、`findDataPoints()` メソッドは `null` を返します。データポイントが存在する場合は HitData オブジェクトの配列を返します。

`findDataPoints()` メソッドのシグネチャは次のとおりです。

```
findDataPoints(x:Number, y:Number):Array
```

次の例では、ユーザーがチャートでマウスポインタを移動すると、PlotChart コントロールを作成して、マウスポインタの場所を記録します。次に `findDataPoints()` メソッドを使用して HitData オブジェクトの配列を取得し、最初のオブジェクトのプロパティの一部を表示します。

```
<?xml version="1.0"?>
<!-- charts/FindDataPoints.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.charts.HitData;
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500},
        {Month:"February", Profit:1000, Expenses:200},
        {Month:"March", Profit:1500, Expenses:500},
        {Month:"April", Profit:500, Expenses:300},
        {Month:"May", Profit:1000, Expenses:450},
        {Month:"June", Profit:2000, Expenses:500}]);

     public function handleMouseMove(e:MouseEvent):void {
        // Use coordinates to get HitData object of
        // current data point.
        var hda:Array =
            chart.findDataPoints(e.currentTarget.mouseX,
```

```
            e.currentTarget.mouseY);
        if (hda[0]) {
           ta.text = "Found data point " +
                hda[0].chartItem.index + " (x/y):" +
                Math.round(hda[0].x) + "," +
                Math.round(hda[0].y) + "\n";
           ta.text += "Expenses:" + hda[0].item.Expenses;
        } else {
           ta.text = "No data point found  (x/y):" +
                Math.round(e.currentTarget.mouseX) +
                "/" + Math.round(e.currentTarget.mouseY);
        }
     }
  ]]></mx:Script>
  <mx:Panel title="Plot Chart">
     <mx:PlotChart id="chart"
        mouseMove="handleMouseMove(event)"
        dataProvider="{expenses}"
        showDataTips="true"
        mouseSensitivity="5"
     >
        <mx:series>
           <mx:PlotSeries
                xField="Profit"
                yField="Expenses"
           />
        </mx:series>
     </mx:PlotChart>
  </mx:Panel>
  <mx:TextArea id="ta" width="300" height="50"/>
</mx:Application>
```

localToData() メソッドの使用

localToData() メソッドは、そのポイントの上または近くにデータアイテムがあるかどうかに関係なく、必要なデータの x 座標と y 座標を表す Point オブジェクトを取得し、データの値の配列を返します。

次の例では、画面上のマウスポインタの位置を基に Point オブジェクトを作成し、そのポイントに関連するデータ値を表示します。

```
<?xml version="1.0"?>
<!-- charts/LocalToData.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     public var p:Point;
```

```
    [Bindable]
    public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500 },
        {Month:"Feb", Profit:1000, Expenses:200},
        {Month:"Mar", Profit:1500, Expenses:500}
    ]);

    private function updateDetails(e:MouseEvent):void {
        p = new Point(myChart.mouseX,myChart.mouseY);
        mpos.text = "(" + p.x + "," + p.y + ")";

        var d:Array = myChart.localToData(p);
        dval.text = "(" + d[0] + "," + Math.floor(d[1]) + ")";

        p = myChart.dataToLocal(d[0],d[1]);
        dpos.text ="(" + Math.floor(p.x) + "," +
            Math.floor(p.y) + ")";
    }


]]></mx:Script>
<mx:Panel title="Column Chart">
    <mx:ColumnChart id="myChart"
        dataProvider="{expenses}"
        mouseMove="updateDetails(event)"
    >
        <mx:horizontalAxis>
            <mx:CategoryAxis
                dataProvider="{expenses}"
                categoryField="Month"
            />
        </mx:horizontalAxis>
        <mx:series>
            <mx:ColumnSeries
                xField="Month"
                yField="Profit"
                displayName="Profit"
            />
            <mx:ColumnSeries
                xField="Month"
                yField="Expenses"
                displayName="Expenses"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
</mx:Panel>


<mx:Form width="300">
```

```
    <mx:FormItem label="Mouse Position:">
       <mx:Label id="mpos"/>
    </mx:FormItem>
    <mx:FormItem label="Data Position:">
       <mx:Label id="dpos"/>
    </mx:FormItem>
    <mx:FormItem label="DATA:">
       <mx:Label id="dval"/>
    </mx:FormItem>
  </mx:Form>

</mx:Application>
```

個々のチャートタイプによって、座標のマッピング方法と、配列に返される値の数が異なります。戻り値は通常、数値です。

BarChart コントロールや ColumnChart コントロールなど、CartesianChart クラスを基にしたチャートでは、返される配列の最初のアイテムは水平軸の x 座標の値で、2 番目のアイテムは垂直軸の y 座標の値です。

PieChart などの PolarChart クラスを基にしたチャートでは、返される配列が各座標を 1 組の極座標にマッピングします。極座標とは、チャートの中心周辺の角度と中心からの距離です。これらの値は、チャートの最初の (角度) 軸と 2 番目の (放射) 軸を使用するデータ値にマッピングされます。

dataToLocal() メソッドの使用

`dataToLocal()` メソッドは、1 組の値を画面上の x 座標と y 座標に変換します。dataToLocal() メソッドに与える値は、チャートの " データ領域 " の中にあります。このメソッドはこれらの値を座標に変換します。データ領域とは、チャートで表現できるデータ値のすべての組み合わせを集めたものです。

`dataToLocal()` メソッドに渡されるパラメータの数と意味は、チャートタイプによって異なります。BarChart コントロールや ColumnChart コントロールなどの CartesianChart コントロールでは、最初の値は x 軸へのマッピングに使用され、2 番目の値は y 軸へのマッピングに使用されます。

PieChart コントロールなどの PolarChart コントロールでは、最初の値はチャートの中心周辺の角度にマッピングされ、2 番目の値はチャートの中心から半径に沿った距離にマッピングされます。

返される座標は 0 を基準としています。チャートの左上隅が 0 です。たとえば ColumnChart コントロールでは、縦棒の高さは、返される x 座標に反比例します。

マウス感度の変更

マウスポインタがデータポイント上にあると見なす範囲は、次の例のように、チャートコントロールの mouseSensitivity プロパティで指定します。

```
<mx:ColumnChart id="chart" dataProvider="{dataSet}" mouseSensitivity="30">
```

マウスポインタからの距離が mouseSensitivity プロパティで指定したピクセル数以内のデータポイントのうち、マウスポインタに最も近いポイントが現在のデータポイントと見なされます。

mouseSensitivity プロパティのデフォルト値は 3 ピクセルです。マウスポインタがデータポイントから 4 ピクセル以上離れている場合、itemRollOver や itemClick などのチャートデータイベントはトリガされませんが、Event オブジェクトを生成することで、Flex は mouseOver や click などのイベントには反応します。

次の例では、mouseSensitivity プロパティの初期値を 20 に設定し、この値をユーザーが HSlider コントロールを使用して変更できるようにします。

```
<?xml version="1.0"?>
<!-- charts/MouseSensitivity.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
     ]);

  ]]></mx:Script>
  <mx:Panel title="Mouse Sensitivity">
     <mx:PlotChart id="chart"
        dataProvider="{expenses}"
        showDataTips="true"
        mouseSensitivity="{mySlider.value}"
    >
        <mx:series>
           <mx:PlotSeries
                xField="Expenses"
                yField="Profit"
                displayName="P 1"
           />
           <mx:PlotSeries
                xField="Amount"
                yField="Expenses"
                displayName="P 2"
```

```
            />
            <mx:PlotSeries
                 xField="Profit"
                 yField="Amount"
                 displayName="P 3"
            />
         </mx:series>
      </mx:PlotChart>

      <mx:HSlider id="mySlider"
         minimum="0"
         maximum="300"
         value="20"
         dataTipPlacement="top"
         tickColor="black"
         snapInterval="1"
         tickInterval="20"
         labels="['0','300']"
         allowTrackClick="true"
         liveDragging="true"
      />
   </mx:Panel>

</mx:Application>
```

mouseSensitivity プロパティを使用して、データヒントイベントの表示やチャート関連のイベントが発生する領域を拡大できます。複数のデータポイントの範囲にマウスポインタが入っている場合、最も近いデータポイントが自動的に選択されます。データヒントイベントの場合、複数の DataTip コントロールが有効になっていると、範囲内のすべての DataTip コントロールが表示されます。詳細については、1611 ページの「複数のデータヒントの表示」を参照してください。

ユーザー操作の無効化

チャート内の特定の系列ですべてのマウスイベントを無視するには、系列の interactive プロパティを false に設定します。デフォルト値は true です。false に設定すると、その系列に対するマウス入力が無効になります。その他の系列に対するマウス入力は有効です。

次の例では、1 番目と 3 番目の PlotSeries オブジェクトに対するイベントへのユーザー入力を無効にします。

```
<?xml version="1.0"?>
<!-- charts/DisableInteractivity.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
```

```
        {Month:"January", Profit:2000, Expenses:1500, Amount:450},
        {Month:"February", Profit:1000, Expenses:200, Amount:600},
        {Month:"March", Profit:1500, Expenses:500, Amount:300},
        {Month:"April", Profit:500, Expenses:300, Amount:500},
        {Month:"May", Profit:1000, Expenses:450, Amount:250},
        {Month:"June", Profit:2000, Expenses:500, Amount:700}
    ]);

  ]]></mx:Script>
  <mx:Panel title="Disable Interactivity">
    <mx:PlotChart id="chart"
      dataProvider="{expenses}"
      showDataTips="true"
    >
      <mx:series>
        <mx:PlotSeries
            xField="Expenses"
            yField="Profit"
            displayName="P 1"
            interactive="false"
        />
        <mx:PlotSeries
            xField="Amount"
            yField="Expenses"
            displayName="P 2"
        />
        <mx:PlotSeries
            xField="Profit"
            yField="Amount"
            displayName="P 3"
            interactive="false"
        />
      </mx:series>
    </mx:PlotChart>
  </mx:Panel>
</mx:Application>
```

系列に対する入力を無効にした場合の動作は、次のようになります。

- その系列の DataTip コントロールは表示されません。
- 系列に対してどのチャートデータイベントが発生しても、hitData 構造体は生成されません。
- チャートの findDataPoints() メソッドを呼び出しても、その系列に対する hitData 構造体は返されません。

チャートにおけるエフェクトの使用

チャートコントロールは、Zoom や Fade など、Flex の標準的なエフェクトをサポートしています。これらのエフェクトを使用すると、Flex アプリケーションのすべてのチャートコントロールをズームインまたはフェードアウトすることができます。また、チャートのデータ系列では、チャート内のデータに適用される次のようなエフェクトクラスをサポートしています。

- SeriesInterpolate
- SeriesSlide
- SeriesZoom

これらのエフェクトを使用すると、チャートコントロールの中でチャートアイテムをズームまたはスライドできます。これらのクラスは、mx.charts.effects.effects パッケージに入っています。チャートエフェクトの基本クラスは SeriesEffect です。

すべてのチャートコントロールと系列で、UIComponent から継承された Flex の標準的なトリガとエフェクトを使用できます。これらのトリガには、`focusInEffect`、`focusOutEffect`、`moveEffect`、`showEffect`、および `hideEffect` が含まれます。Flex チャート コントロールには、`showDataEffect` トリガおよび `hideDataEffect` トリガも含まれます。

複雑なエフェクトの作成方法については、603 ページ、第 17 章の「ビヘイビアの使用」を参照してください。

標準的なエフェクトトリガの使用

チャートコントロールでは、`showEffect` や `hideEffect` などの標準的なエフェクトトリガをサポートしています。

次の例では、一連のワイプエフェクトを定義して、ユーザーが Button コントロールを使用してチャートの表示と非表示を切り替えたときにエフェクトが実行されるようにします。また、チャートの作成後、Flex でチャートにフェードエフェクトをかけます。

```
<?xml version="1.0"?>
<!-- charts/StandardEffectTriggers.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.effects.Fade;

     [Bindable]
     public var expenses:ArrayCollection = new ArrayCollection([
        {Month:"Jan", Profit:2000, Expenses:1500, Amount:450},
        {Month:"Feb", Profit:1000, Expenses:200, Amount:600},
        {Month:"Mar", Profit:1500, Expenses:500, Amount:300}
     ]);
```

```
  ]]></mx:Script>

  <!-- Define the effects -->
  <mx:Parallel id="showEffects">
     <mx:WipeRight duration="2000"/>
     <mx:Fade alphaFrom="0" alphaTo="1" duration="4000"/>
  </mx:Parallel>

  <mx:Parallel id="hideEffects">
     <mx:Fade alphaFrom="1" alphaTo="0" duration="2500"/>
     <mx:WipeLeft duration="3000"/>
  </mx:Parallel>

  <mx:Panel title="Area Chart with Effects">
     <mx:AreaChart id="myChart"
        dataProvider="{expenses}"
        creationCompleteEffect="showEffects"
        hideEffect="hideEffects"
        showEffect="showEffects"
     >
        <mx:horizontalAxis>
           <mx:CategoryAxis categoryField="Month"/>
        </mx:horizontalAxis>
        <mx:series>
           <mx:AreaSeries
                yField="Profit"
                displayName="Profit"
           />
           <mx:AreaSeries
                yField="Expenses"
                displayName="Expenses"
           />
        </mx:series>
     </mx:AreaChart>
     <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>

  <mx:Button label="Toggle visibility"
        click="myChart.visible=!myChart.visible"
  />
</mx:Application>
```

チャートで標準的なエフェクトトリガを使用することの欠点は、エフェクトがチャートコントロールの対象データのみでなく、チャートコントロール全体に適用されることです。この結果、Fade などのエフェクトにより、エフェクトの動作中にチャートのデータ以外にチャートの軸、グリッド線、ラベル、およびその他のチャートエレメントもフェードインまたはフェードアウトされることになります。これを解決するには、特殊なチャートエフェクトトリガを使用します。1659 ページの「チャートエフェクトトリガの使用」を参照してください。

チャートエフェクトトリガの使用

チャートには、hideDataEffect および showDataEffect という 2 種類の固有のエフェクトトリガがあります。これらのトリガは、チャートのデータ系列で設定します。チャートのデータが変更されると、続けてこれらのトリガがチャートのデータに対して実行されます。グリッド線、軸線、ラベルなどのチャートコントロールの他のエレメントは、エフェクトに影響されません。

hideDataEffect トリガは、現在のデータを非表示にするときに使用されるエフェクトを定義します。showDataEffect トリガは、現在のデータを画面上の最終的な位置に移動するときに使用されるエフェクトを定義します。

hideDataEffect と showDataEffect に関連付けられるエフェクトはデータを変更するときにトリガされるので、変更前の " 古い " データと変更後の " 新しい " データがあります。hideDataEffect トリガに関連付けられるエフェクトは、新しいデータに置き換えられる " 古い " データです。

イベントの実行順序は次のとおりです。

1. まず、これから変更するチャートの各エレメントに対し、Flex から hideDataEffect トリガによってエフェクトセットが呼び出されます。それらのトリガは同時に発生します。
2. Flex で次に、新しいデータを使用してチャートを更新します。グリッド線や軸など、エフェクトが関連付けられていないすべてのエレメントは、新しい値によって即座に更新されます。
3. 次に、エフェクトが関連付けられているエレメントに対し、showDataEffect トリガによってエフェクトセットが呼び出されます。新しいデータがチャートで動きます。

データ系列でのチャートエフェクト

チャート固有のエフェクトには、SeriesInterpolate、SeriesSlide、SeriesZoom の 3 種類があります。それぞれのエフェクトは、データ系列のデータが変更されたときにエフェクトを実行するために、データ系列で使用されます。これらのエフェクトを使用することによって、絶大な視覚効果が得られます。データ系列では、これ以外の Flex エフェクトはサポートされません。

次の例では、データが変更されたときに、SeriesSlide エフェクトを使用して画面上でデータをスライドインまたはスライドアウトする方法を示します。チャート系列内のデータへの変更をトリガする方法は多数ありますが、チャートコントロールのデータプロバイダが変更されたときにエフェクトをトリガするのが、最も一般的です。次の例では、ユーザーがボタンをクリックしたときに、チャートコントロールのデータプロバイダが変更され、エフェクトがトリガされます。

```
<?xml version="1.0"?>
<!-- charts/BasicSeriesSlideEffect.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var expenses1:ArrayCollection = new ArrayCollection([
```

```
      {Month:"Jan", Income:2000, Expenses:1500},
      {Month:"Feb", Income:1000, Expenses:200},
      {Month:"Mar", Income:1500, Expenses:500}
   ]);

   [Bindable]
   public var expenses2:ArrayCollection = new ArrayCollection([
      {Month:"Jan", Income:1200, Expenses:800},
      {Month:"Feb", Income:2500, Expenses:300},
      {Month:"Mar", Income:575, Expenses:490}
   ]);

   public function changeProvider():void {
      myChart.dataProvider=expenses2;
   }

]]></mx:Script>

<!-- Define chart effects -->
<mx:SeriesSlide
  id="slideIn"
  duration="1000"
  direction="up"
/>
<mx:SeriesSlide
  id="slideOut"
  duration="1000"
  direction="down"
/>

<mx:Panel title="Column Chart with Basic Series Slide Effect">
   <mx:ColumnChart id="myChart" dataProvider="{expenses1}">
      <mx:horizontalAxis>
         <mx:CategoryAxis
              dataProvider="{expenses1}"
              categoryField="Month"
         />
      </mx:horizontalAxis>

      <mx:verticalAxis>
         <mx:LinearAxis minimum="0" maximum="3000"/>
      </mx:verticalAxis>

      <mx:series>
         <mx:ColumnSeries
          xField="Month"
          yField="Income"
          displayName="Income"
          showDataEffect="slideIn"
          hideDataEffect="slideOut"
```

```
            />
            <mx:ColumnSeries
             xField="Month"
             yField="Expenses"
             displayName="Expenses"
             showDataEffect="slideIn"
             hideDataEffect="slideOut"
            />
        </mx:series>
    </mx:ColumnChart>
    <mx:Legend dataProvider="{myChart}"/>
  </mx:Panel>

  <mx:Button id="b1" click="changeProvider()"
    label="Change Data Provider"
  />

</mx:Application>
```

この例では、垂直軸の最小値と最大値を明示的に定義します。これを行わないと、新しいデータプロバイダの適用時にこれらの値が再計算されます。再計算されると、エフェクト時に軸ラベルが変更されます。

データプロバイダが変更されると、まず元のデータプロバイダの hideDataEffect がトリガされ、元のデータプロバイダが " スライドアウト " します。次に新しいデータプロバイダの showDataEffect がトリガされ、新しいデータプロバイダが " スライドイン " します。

メモ	データ系列にデータプロバイダを設定して、チャートコントロールは設定しない場合、データ系列ではデータプロバイダを変更する必要があります。

データエフェクトは、新しいデータポイントがデータ系列に追加されたときにもトリガされます。次の例では、ユーザーがボタンをクリックしたときに showDataEffect をトリガし、新しいアイテムをデータ系列のデータプロバイダに追加します。

```
<?xml version="1.0"?>
<!-- charts/AddItemEffect.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import mx.collections.ArrayCollection;

        [Bindable]
        public var items:ArrayCollection = new ArrayCollection([
            {item: 2000},
            {item: 3300},
            {item: 3000},
            {item: 2100},
            {item: 3200}
        ]);

        public function addDataItem():void {
```

```
            // Add a randomly generated value to
            // the data provider.
            var n:Number = Math.random() * 3000;
            var o:Object = {item: n};
            items.addItem(o);
        }
    ]]></mx:Script>

    <!-- Define chart effect -->
    <mx:SeriesSlide id="slideIn"
        duration="1000"
        direction="up"
    />

    <mx:Panel title="Column Chart with Series Effect">
        <mx:ColumnChart id="myChart" dataProvider="{items}">
            <mx:series>
                <mx:ColumnSeries
                    yField="item"
                    displayName="Quantity"
                    showDataEffect="slideIn"
                />
            </mx:series>
        </mx:ColumnChart>
    </mx:Panel>

    <mx:Button id="b1"
        click="addDataItem()"
        label="Add Data Item"
    />

</mx:Application>
```

実行時のチャートデータ変更の詳細については、1475 ページの「実行時におけるチャートデータの変更」を参照してください。

チャートエフェクトには、従来のエフェクトにはないさまざまなプロパティが共通して含まれています。これらのプロパティはすべて省略可能です。以下の表は、チャートエフェクトの共通プロパティの一覧です。

プロパティ	説明
duration	エフェクト全体を完了するのに必要な時間をミリ秒単位で指定します。このプロパティでは、エフェクトを実行するのにかかる時間を定義します。 duration プロパティは、継続時間の最小値として機能します。他のプロパティの設定によっては、エフェクトの完了までの時間がこの値より長くなることがあります。 デフォルト値は 500 です。
elementOffset	系列に含まれる各エレメントのエフェクトを遅延させる時間をミリ秒単位で指定します。 elementOffset プロパティを 0 に設定すると、すべてのエレメントが同時に始まり、同時に終了します。 elementOffset プロパティに整数値 (30 など) を設定すると、各エレメントのエフェクトの開始が指定時間分ずれます。たとえば、スライドエフェクトの場合、最初のエレメントは即座にスライドします。2 番目以降のエレメントは 30 ミリ秒ずつ遅れて次々にスライドします。各エフェクトの実行にかかる時間はすべてのエレメントで同じですが、全体の継続時間はこれより長くなります。 elementOffset プロパティに負の値を設定すると、エフェクトにより系列の最後のエレメントが最初に表示されます。 デフォルト値は 20 です。

プロパティ	説明
minimumElementDuration	各エレメントでエフェクトを完了するのに必要な時間をミリ秒単位で指定します。 系列に含まれるデータポイントの数が可変のチャートの場合、duration プロパティだけではスムーズにエフェクトを生成できないことがあります。 たとえば、duration が 1000 で elementOffset が 100 のエフェクトがあるとします。ここで、系列内に 2 つのエレメントが存在する場合、各エレメントは 900 ミリ秒でエフェクトを完了する必要があります。各エフェクトは 100 ミリ秒遅れて開始され、また、全体として 1000 ミリ秒で完了する必要があるからです。 系列内に 4 つのエレメントが存在する場合、各エレメントは 700 ミリ秒でエフェクトを完了する必要があります。最後のエフェクトは 300 ミリ秒遅れて開始され、全体として 1000 ミリ秒で完了する必要があるからです。 仮に 10 個のエレメントが存在する場合、各エレメントを実行する時間は 100 ミリ秒しか残されていないことになります。 minimumElementDuration プロパティの値は、各エレメントの最小持続時間を設定します。系列に含まれるどのエレメントについても、この値 (ミリ秒) より短い時間でエフェクトが実行されることはありません。したがって、duration、offset、minimumElementDuration の 3 つのプロパティのうち、2 つ以上が指定されていても、全体としてエフェクトの実行時間が指定した持続時間より長くなる可能性があります。 デフォルト値は 0 です。
offset	エフェクト開始の遅延時間をミリ秒単位で指定します。複数の系列がある場合に時間差を付けてエフェクトを開始する場合は、このプロパティを使用します。 デフォルト値は 0 です。

SeriesSlide エフェクトの使用

SeriesSlide エフェクトは、データ系列をチャートの境界にスライドインまたはスライドアウトします。SeriesSlide エフェクトには、系列をスライドする方向を定義する direction プロパティがあります。direction プロパティの有効な値は、left、right、up、または down です。

SeriesSlide エフェクトを hideDataEffect トリガで使用すると、系列は画面上の現在位置から画面外へと、direction プロパティで指定した方向にスライドします。SeriesSlide を showDataEffect トリガで使用すると、系列は画面外から画面上の位置へと、指定された方向にスライドします。

SeriesSlide エフェクトを使用すると、エフェクトの開始時にデータ系列全体がチャートで非表示になります。その後、エフェクトの特性に基づいてデータが再表示されます。エフェクト中に常に画面にデータを表示したままにするには、SeriesInterpolate エフェクトを使用します。詳細については、1668 ページの「SeriesInterpolate エフェクトの使用」を参照してください。

次の例では、slideDown というエフェクトを作成します。各エレメントは前のエレメントの 30 ミリ秒後に開始し、それぞれのスライドを完了するのに 20 ミリ秒かかります。エフェクト全体では、データ系列がすべてスライドするまで少なくとも 1 秒 (1000 ミリ秒) かかります。Flex は、チャートから古いデータが消え、新しいデータが表示されたときにエフェクトを呼び出します。

```
<?xml version="1.0"?>
<!-- charts/CustomSeriesSlideEffect.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var items:ArrayCollection = new ArrayCollection([
        {item:2000},
        {item:3300},
        {item:3000},
        {item:2100},
        {item:3200}
     ]);

     public function addDataItem():void {
        // Add a randomly generated value to the data provider
        var n:Number = Math.random() * 3000;
        var o:Object = {item:n};
        items.addItem(o);
     }
  ]]></mx:Script>

  <!-- Define chart effect -->
  <mx:SeriesSlide duration="1000"
    direction="down"
    minimumElementDuration="20"
    elementOffset="30"
    id="slideDown"
  />

  <mx:Panel title="Column Chart with Custom Series Slide Effect">
     <mx:ColumnChart id="myChart" dataProvider="{items}">
        <mx:series>
           <mx:ColumnSeries
            yField="item"
            displayName="Quantity"
            showDataEffect="slideDown"
            hideDataEffect="slideDown"
           />
        </mx:series>
     </mx:ColumnChart>
  </mx:Panel>
  <mx:Button id="b1"
    click="addDataItem()"
```

```
    label="Add Data Item"
  />
</mx:Application>
```

SeriesZoom エフェクトの使用

SeriesZoom エフェクトは、指定した焦点にチャートデータを縮小 (ズームアウト)、または指定した焦点からチャートデータを拡大 (ズームイン) します。SeriesSlide エフェクトと同様、ズームインとズームアウトのどちらが実行されるかは、showDataEffect トリガと hideDataEffect トリガのどちらで使用されるかによって決まります。

SeriesZoom エフェクトには、エフェクトの動作を定義するプロパティがいくつかあります。これらのプロパティについて次の表で説明します。

プロパティ	説明
horizontalFocus verticalFocus	ズームの焦点の位置を定義します。 horizontalFocus プロパティと verticalFocus プロパティを組み合わせて、データ系列がズームインまたはズームアウトする際の起点を定義します。たとえば、horizontalFocus プロパティを left に設定し、verticalFocus プロパティを top に設定すると、系列データがチャートまたはエレメント (relativeTo プロパティの設定によって異なります) の左上からズームインまたはズームアウトします。 horizontalFocus プロパティでできる値は、left、center、right、または undefined のいずれかです。 verticalFocus プロパティで使用できる値は、top、center、bottom、または undefined のいずれかです。 これらのプロパティの一方のみを指定した場合は、ズームインまたはズームアウトする先が点ではなく水平線または垂直線になります。たとえば、horizontalFocus プロパティを left に設定すると、エレメントは境界線の左の垂直線にズームインまたはズームアウトします。verticalFocus プロパティを center に設定すると、エレメントは境界線の中心の水平線にズームインまたはズームアウトします。 これらのプロパティのデフォルト値は、いずれも center です。
relativeTo	ズームの焦点を計算するための境界線を制御します。relativeTo で使用できる値は、series と chart です。 relativeTo プロパティを series に設定すると、各エレメントはそれ自身を基準としてズームします。たとえば、ColumnChart のそれぞれの縦棒は、縦棒の左上を起点としてズームします。 relativeTo プロパティを chart に設定すると、各エレメントはチャートの領域を基準としてズームします。たとえば、それぞれの縦棒は軸の左上や軸の中央などにズームします。

次の例では、データ系列がチャートの右上隅からズームインします。elementOffset が負の値なので、ズームイン時には系列の最後のエレメントが最初に表示されます。

```
<?xml version="1.0"?>
<!-- charts/SeriesZoomEffect.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var items:ArrayCollection = new ArrayCollection([
        {item: 2000},
        {item: 3300},
        {item: 3000},
        {item: 2100},
        {item: 3200}
     ]);

     public function addDataItem():void {
        // Add a randomly generated value to the data provider
        var n:Number = Math.random() * 3000;
        var o:Object = {item: n};
        items.addItem(o);
     }
  ]]></mx:Script>

  <!-- Define chart effects -->
  <mx:SeriesZoom id="zoomOut"
     duration="2000"
     minimumElementDuration="50"
     elementOffset="50"
     verticalFocus="top"
     horizontalFocus="left"
     relativeTo="chart"
  />
  <mx:SeriesZoom id="zoomIn"
     duration="2000"
     minimumElementDuration="50"
     elementOffset="-50"
     verticalFocus="top"
     horizontalFocus="right"
     relativeTo="chart"
  />

  <mx:Panel title="Column Chart with Series Zoom Effect">
     <mx:ColumnChart id="myChart" dataProvider="{items}">
        <mx:series>
           <mx:ColumnSeries
            yField="item"
            displayName="Quantity"
```

```
            showDataEffect="zoomIn"
            hideDataEffect="zoomOut"
           />
        </mx:series>
    </mx:ColumnChart>
  </mx:Panel>
  <mx:Button id="b1" click="addDataItem()" label="Add Data Item"/>
</mx:Application>
```

SeriesZoom エフェクトを使用すると、エフェクトの開始時にデータ系列全体がチャートで非表示になります。その後、エフェクトの特性に基づいてデータが再表示されます。エフェクト中に常に画面にデータを表示したままにするには、SeriesInterpolate エフェクトを使用します。詳細については、1668 ページの「SeriesInterpolate エフェクトの使用」を参照してください。

SeriesInterpolate エフェクトの使用

SeriesInterpolate エフェクトは、系列に含まれる既存のデータを表すグラフィックを新しい場所に移動します。チャートを削除して再作成する SeriesZoom や SeriesSlide と異なり、このエフェクトではデータが画面から消えることはありません。

SeriesInterpolate エフェクトは、showDataEffect エフェクトトリガでのみ使用できます。hideDataEffect トリガでこのエフェクトを設定した場合、エフェクトは無効です。

次の例では、SeriesInterpolate の elementOffset プロパティを 0 に設定します。その結果、すべてのエレメントが画面で非表示にならずに新しい場所に同時に移動します。

```
<?xml version="1.0"?>
<!-- charts/SeriesInterpolateEffect.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var items:ArrayCollection = new ArrayCollection([
        {item: 2000},
        {item: 3300},
        {item: 3000},
        {item: 2100},
        {item: 3200}
     ]);

     public function addDataItem():void {
        // Add a randomly generated value to the data provider
        var n:Number = Math.random() * 3000;
        var o:Object = {item: n};
        items.addItem(o);
     }
  ]]></mx:Script>
```

```
  <!-- Define chart effect -->
  <mx:SeriesInterpolate id="rearrangeData"
     duration="1000"
     minimumElementDuration="200"
     elementOffset="0"
  />

  <mx:Panel title="Column Chart with Series Interpolate Effect">
     <mx:ColumnChart id="myChart" dataProvider="{items}">
        <mx:series>
           <mx:ColumnSeries
            yField="item"
            displayName="Quantity"
            showDataEffect="rearrangeData"
           />
        </mx:series>
     </mx:ColumnChart>
  </mx:Panel>
  <mx:Button id="b1" click="addDataItem()" label="Add Data Item"/>
</mx:Application>
```

ActionScript でのエフェクトの適用

ActionScript を使用してエフェクトを定義しチャート系列に適用できます。エフェクトを適用する 1 つの方法は、スタイルプロパティを適用する方法と同じです。setStyle() メソッドとカスケーディングスタイルシート (CSS) を使用してエフェクトを適用できます。スタイルの使用の詳細については、647 ページ、第 18 章の「スタイルとテーマの使用」を参照してください。

次の例では、ユーザーがデータアイテムをチャートコントロールの ColumnSeries に追加するたびに実行される slideIn エフェクトを定義しています。アプリケーションが最初にロードされるときに、setStyle() メソッドを使用して、このエフェクトが系列に適用されます。

```
<?xml version="1.0"?>
<!-- charts/ApplyEffectsAsStyles.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.effects.SeriesInterpolate;

     [Bindable]
     public var items:ArrayCollection = new ArrayCollection([
        {item: 2000},
        {item: 3300},
        {item: 3000},
        {item: 2100},
        {item: 3200}
     ]);
```

```
    public var rearrangeData:SeriesInterpolate;

    public function init():void {
       rearrangeData = new SeriesInterpolate();
       rearrangeData.duration = 1000;
       rearrangeData.minimumElementDuration = 200;
       rearrangeData.elementOffset = 0;

       // Apply the effect as a style.
       mySeries.setStyle("showDataEffect", "rearrangeData");
    }

    public function addDataItem():void {
       // Add a randomly generated value to the data provider.
       var n:Number = Math.random() * 3000;
       var o:Object = {item: n};
       items.addItem(o);
    }
 ]]></mx:Script>

 <mx:Panel title="Column Chart with Series Interpolate Effect">
    <mx:ColumnChart id="myChart" dataProvider="{items}">
       <mx:series>
          <mx:ColumnSeries
           id="mySeries"
           yField="item"
           displayName="Quantity"
           showDataEffect="rearrangeData"
          />
       </mx:series>
    </mx:ColumnChart>
 </mx:Panel>

 <mx:Button id="b1"
   click="addDataItem()"
   label="Add Data Item"
 />

</mx:Application>
```

ActionScript でエフェクトを定義する場合、適切なクラスを読み込む必要があります。MXML を使用してエフェクトを定義すると、コンパイラによってクラスが読み込まれます。

MXML アプリケーションでエフェクトを事前に定義しておくと、setStyle() メソッドでエフェクトを適用する代わりに、CSS でエフェクトを適用できます。以下に例を示します。

```
<?xml version="1.0"?>
<!-- charts/ApplyEffectsWithCSS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
```

```
  <mx:Style>
     ColumnSeries {
        showDataEffect:slideDown;
        hideDataEffect:slideDown;
     }
  </mx:Style>

  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;

     [Bindable]
     public var items:ArrayCollection = new ArrayCollection([
        {item:2000},
        {item:3300},
        {item:3000},
        {item:2100},
        {item:3200}
     ]);

     public function addDataItem():void {
        // Add a randomly generated value to the data provider
        var n:Number = Math.random() * 3000;
        var o:Object = {item:n};
        items.addItem(o);
     }
  ]]></mx:Script>

  <!-- Define chart effect -->
  <mx:SeriesSlide
    duration="1000"
    direction="down"
    minimumElementDuration="20"
    elementOffset="30"
    id="slideDown"
  />

  <mx:Panel title="Column Chart with Custom Series Slide Effect">
     <mx:ColumnChart id="myChart" dataProvider="{items}">
        <mx:series>
           <mx:ColumnSeries
            yField="item"
            displayName="Quantity"
           />
        </mx:series>
     </mx:ColumnChart>
  </mx:Panel>
  <mx:Button id="b1"
    click="addDataItem()"
    label="Add Data Item"
  />
</mx:Application>
```

また、スタイルを使用せずに ActionScript でエフェクトを適用することもできます。これを行うには、次の例に示すように、エフェクトのコンストラクタでエフェクトのターゲット (系列) を指定します。

```
<?xml version="1.0"?>
<!-- charts/ApplyEffectsWithActionScript.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.effects.SeriesInterpolate;

     [Bindable]
     public var items:ArrayCollection = new ArrayCollection([
        {item: 2000},
        {item: 3300},
        {item: 3000},
        {item: 2100},
        {item: 3200}
     ]);

     public var rearrangeData:SeriesInterpolate;

     public function init():void {
        // Specify the effect's target in the constructor.
        rearrangeData = new SeriesInterpolate(mySeries);
        rearrangeData.duration = 1000;
        rearrangeData.minimumElementDuration = 200;
        rearrangeData.elementOffset = 0;
     }

     public function addDataItem():void {
        // Add a randomly generated value to the data provider.
        var n:Number = Math.random() * 3000;
        var o:Object = {item: n};
        items.addItem(o);
     }
  ]]></mx:Script>

  <mx:Panel title="Column Chart with Series Interpolate Effect">
     <mx:ColumnChart id="myChart" dataProvider="{items}">
        <mx:series>
           <mx:ColumnSeries
            id="mySeries"
            yField="item"
            displayName="Quantity"
            showDataEffect="rearrangeData"
           />
        </mx:series>
```

```
    </mx:ColumnChart>
  </mx:Panel>

  <mx:Button id="b1"
    click="addDataItem()"
    label="Add Data Item"
  />

</mx:Application>
```

この例を拡張し、スライダコントロールを使用してエフェクトのプロパティを調整するようにできます。エフェクトなど、ActionScript オブジェクトのプロパティをコントロールにバインドするには、次の例に示すように、BindingUtils クラスのメソッドを使用します。

```
<?xml version="1.0"?>
<!-- charts/ApplyEffectsWithActionScriptSlider.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()">
  <mx:Script><![CDATA[
     import mx.collections.ArrayCollection;
     import mx.charts.effects.SeriesInterpolate;
     import mx.binding.utils.BindingUtils;

     public var rearrangeData:SeriesInterpolate;
     [Bindable]
     public var items:ArrayCollection = new ArrayCollection([
        {item: 2000},
        {item: 3300},
        {item: 3000},
        {item: 2100},
        {item: 3200}
     ]);

     public function init():void {
        // Specify the effect's target in the constructor.
        rearrangeData = new SeriesInterpolate(mySeries);

        // Bind effect properties to slider controls
        BindingUtils.bindProperty(rearrangeData,
            "duration", durationSlider, "value");
        BindingUtils.bindProperty(rearrangeData,
            "minimumElementDuration",
            minimumElementDurationSlider, "value");
        BindingUtils.bindProperty(rearrangeData,
            "elementOffset", elementOffsetSlider, "value");
     }

     public function addDataItem():void {
        // Add a randomly generated value to the data provider.
        var n:Number = Math.random() * 3000;
```

```
            var o:Object = {item: n};
            items.addItem(o);
        }

        public function resetSliders():void {
            durationSlider.value = 1000;
            minimumElementDurationSlider.value = 200;
            elementOffsetSlider.value = 0;
        }
    ]]></mx:Script>

    <mx:Panel title="Column Chart with Series Interpolate Effect">
        <mx:ColumnChart id="myChart" dataProvider="{items}">
            <mx:series>
                <mx:ColumnSeries
                 id="mySeries"
                 yField="item"
                 displayName="Quantity"
                 showDataEffect="rearrangeData"
                />
            </mx:series>
        </mx:ColumnChart>
    </mx:Panel>

    <mx:Button id="b1"
      click="addDataItem()"
      label="Add Data Item"
    />

    <mx:Panel>
        <mx:Form>
            <mx:FormItem label="Duration">
                <mx:HSlider id="durationSlider"
                 minimum="1"
                 maximum="10000"
                 value="1000"
                 dataTipPlacement="top"
                 tickColor="black"
                 snapInterval="500"
                 tickInterval="500"
                 labels="['0','10000']"
                 allowTrackClick="true"
                 liveDragging="true"
                />
            </mx:FormItem>
            <mx:FormItem label="Minimum Element Duration">
                <mx:HSlider id="minimumElementDurationSlider"
                 minimum="0"
                 maximum="1000"
                 value="200"
```

```
            dataTipPlacement="top"
            tickColor="black"
            snapInterval="50"
            tickInterval="50"
            labels="['0','1000']"
            allowTrackClick="true"
            liveDragging="true"
           />
        </mx:FormItem>
        <mx:FormItem label="Element Offset">
           <mx:HSlider id="elementOffsetSlider"
            minimum="0"
            maximum="1000"
            value="0"
            dataTipPlacement="top"
            tickColor="black"
            snapInterval="50"
            tickInterval="50"
            labels="['0','1000']"
            allowTrackClick="true"
            liveDragging="true"
           />
        </mx:FormItem>
     </mx:Form>
  </mx:Panel>

  <mx:Button id="b2"
    label="Reset Sliders"
    click="resetSliders()"
  />

</mx:Application>
```

ActionScript でのデータバインディングの詳細については、1147 ページの「ActionScript でのバインディングの定義」を参照してください。

索引

記号

数字

A

B

C

D

E

F

G

H

I

J

L

M

N

P

R

S

T

U

V

W

X

Y

Z

あ

い

う

え

お

か

き

さ

ち

つ

て

と

へ

ほ

ま

み

む

め

も

ゆ