Entwickeln von Mobilanwendungen mit ADOBE® FLEX® 4.6 und ADOBE® FLASH® BUILDER™ 4.6

Rechtliche Hinweise

Letzte Aktualisierung 6.12.2011

Inhalt

Kapitel 1: Erste Schritte

Erste Schritte mit Mobilanwendungen

Adobe Flex bringt Flex und Adobe Flash Builder auf Smartphones und Tablet-Computer. Mithilfe von Adobe AIR können Sie nun Mobilanwendungen in Flex so einfach und hochwertig wie auf Desktopplattformen entwickeln.

Viele vorhandene Flex-Komponenten wurden für die Verwendung auf Mobilgeräten erweitert, wobei nun auch der Bildlauf per Berührung unterstützt wird. Flex enthält außerdem eine Reihe neuer Komponenten, mit denen Sie problemlos Anwendungen erstellen können, die Standarddesignmustern für Telefone und Tablets entsprechen.

Darüber hinaus wurde Flash Builder aktualisiert und weist neue Funktionen zur Unterstützung der Anwendungsentwicklung für Mobilgeräte auf. Mit Flash Builder können Sie Anwendungen auf dem Desktop oder direkt auf dem Mobilgerät entwickeln, testen und debuggen.

Adobe-Guru Mark Doherty beschreibt in einem Video, wie Sie Anwendungen für Desktop, Mobiltelefone und Tablets erstellen.

Unter Building Mobile Apps with Flex finden Sie ein Video von Adobe-Entwickler James Ward zum Erstellen von Mobilanwendungen mit Flex.

Adobe Community-Experte Joseph Labrecque schrieb eine Anleitung zu Mobile Flex-Mobilprojekten.

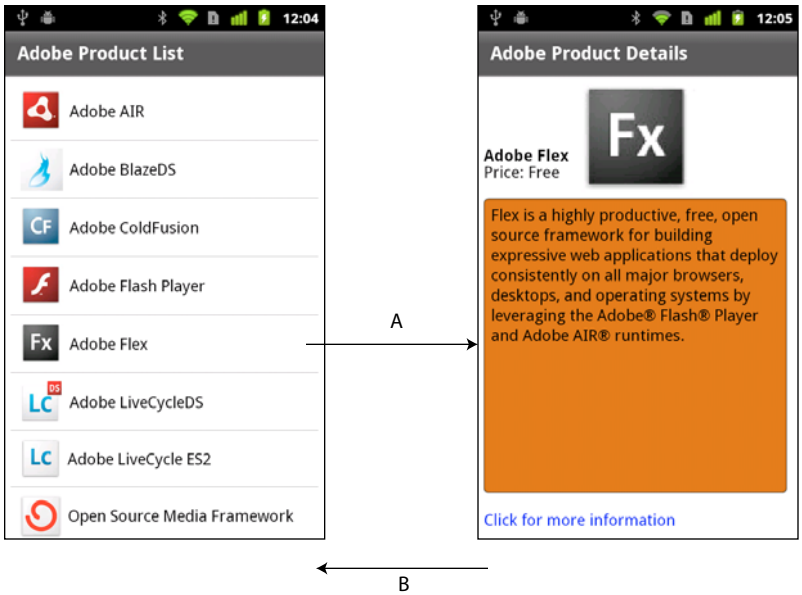
Flash-Entwickler Fabio Biondi hat einen AIR-basierten YouTube-Player für Android-Geräte mit Flash Builder erstellt.

Mobilanwendung planen

Wegen der geringeren Bildschirmgröße auf Mobilgeräten gelten für Mobilanwendungen in der Regel andere Designmuster als für browserbasierte Anwendungen. Bei der Entwicklung von Mobilanwendungen wird der Inhalt normalerweise in mehrere Ansichten aufgeteilt, die auf dem Mobilgerät dargestellt werden können.

Jede Ansicht enthält Komponenten für eine bestimmte Aufgaben oder für eine bestimmte Informationsgruppe. Der Benutzer führt in der Regel einen „Drilldown“ aus bzw. wechselt zwischen Ansichten, indem er in der Ansicht auf Komponenten tippt. Der Benutzer kann dann mit der Zurück-Taste des Geräts zu einer vorherigen Ansicht zurückkehren oder aber die Navigation in die Anwendung integrieren.

Im folgenden Beispiel enthält die Ausgangsansicht der Anwendung eine Liste von Produkten:



A. Wählen Sie ein Listenelement aus, um die Ansicht in der Anwendung zu wechseln. B. Kehren Sie mithilfe der Zurück-Taste des Geräts zur vorherigen Ansicht zurück.

Der Benutzer wählen ein Produkt in der Liste aus, um weitere Informationen zu erhalten. Durch die Auswahl wird eine detaillierte Beschreibung des Produkts angezeigt.

Wenn Sie eine Anwendung für die Mobilplattform, das Internet und den Computer entwickeln, entwerfen Sie normalerweise für jede Plattform eine eigene Benutzeroberfläche. Die Anwendungen können jedoch den zugrunde liegenden Modell- und Datenzugriffscode für alle Plattformen nutzen.

Anwendungen für Telefone und Tablet-Computer erstellen

Für eine Tablet-Anwendung stellt die eingeschränkte Bildschirmgröße kein solches Problem dar wie bei Telefonen. Sie müssen eine Tablet-Anwendung nicht für kleine Displays strukturieren. Stattdessen können Sie zum Erstellen der Anwendung den standardmäßigen Spark-Anwendungscontainer mit den unterstützten Mobilkomponenten und -skins verwenden.

Hinweis: *Eine Anwendung für ein Mobiltelefon können Sie basierend auf dem Spark-Anwendungscontainer erstellen. In der Regel verwenden Sie stattdessen den ViewNavigatorApplication-Container und den TabbedViewNavigatorApplication-Container.*

Mobilprojekte in Flash Builder erstellen Sie für Tablet-Computer genau wie für Telefone. Tablet- und Telefonanwendungen erfordern dasselbe Mobildesign, um die für Mobilanwendungen optimierten Komponenten und Skins nutzen zu können.

Mobilanwendungen in Flash Builder erstellen

Flash Builder führt produktive Entwurfs-, Erstellungs- und Debugarbeitsabläufe für die Entwicklung von Mobilanwendungen ein. Mit den mobilen Funktionen in Flash Builder soll die Entwicklung von ActionScript- oder Flex-basierten Mobilanwendungen so einfach werden wie die Entwicklung einer Desktop- oder Webanwendung.

Flash Builder bietet zwei Optionen zum Testen und Debuggen. Mit dem AIR Debug Launcher (ADL) können Sie eine Anwendung auf dem Desktop starten und debuggen. Wenn Sie mehr Kontrolle wünschen, starten und debuggen Sie die Anwendung direkt auf einem Mobilgerät. In beiden Fällen können Sie die Debugfunktionen von Flash Builder verwenden und unter anderem Haltepunkte festlegen und den Anwendungsstatus mithilfe der Bedienfelder „Variablen“ und „Ausdrücke“ analysieren.

Wenn die Anwendung für die Bereitstellung bereit ist, verwenden Sie die Option „Releasebuild exportieren“ so wie Sie das von Desktop- und Webanwendungen gewohnt sind. Der Hauptunterschied besteht darin, dass beim Exportieren des Releasebuilds eines Mobilprojekts Flash Builder den Build als natives Installationsprogramm und nicht als AIR-Datei verpackt. Beispielsweise erstellt Flash Builder für Android eine APK-Datei, die genauso aussieht wie ein natives Android-Anwendungspaket. Mit dem nativen Installationsprogramm können AIR-basierte Anwendungen auf dieselbe Weise wie native Anwendungen auf jeder Plattform verteilt werden.

Mobilanwendungen in AIR bereitstellen

In Flex erstellte Mobilanwendungen können Sie über Adobe AIR für Mobilgeräte bereitstellen. Jedes Gerät, auf dem Sie eine Mobilanwendung bereitstellen möchten, muss AIR unterstützen.

Ihre Anwendungen können die AIR-Integration in die Mobilplattform in vollem Umfang nutzen. Beispielsweise können Mobilanwendungen die Zurück-Taste und die Menütaste bedienen und auf den lokalen Speicher zugreifen. Außerdem können Sie sämtliche von AIR für Mobilgeräte angebotene Funktionen nutzen. Zu diesen Funktionen zählt die Integration von Ortung, Beschleunigungsmesser und Kamera.

Auf einem Mobilgerät muss AIR nicht installiert werden, bevor Sie eine in Flex erstellte Anwendung ausführen. Wenn ein Benutzer eine in Flex erstellte Anwendung zum ersten Mal ausführt, wird er zum Herunterladen von AIR aufgefordert.

Hier können Sie sich mit AIR vertraut machen und weitere Informationen zu den Funktionen von AIR abrufen:

- Informationen zu Adobe AIR
- Aufrufen und Beenden von AIR-Anwendungen
- Arbeiten mit AIR-Laufzeit- und Betriebssysteminformationen
- Arbeiten mit nativen AIR-Fenstern
- Arbeiten mit lokalen SQL-Datenbanken in AIR

Bei der Entwicklung von Mobilanwendungen können die folgenden Flex-Komponenten für AIR nicht verwendet werden: WindowedApplication und Window. Verwenden Sie stattdessen den ViewNavigatorApplication-Container und den TabbedViewNavigatorApplication-Container. Zur Entwicklung von Mobilanwendungen für Tablets können Sie auch den Spark-Anwendungscontainer verwenden.

Weitere Informationen finden Sie unter Using the Flex AIR components und „Mobilanwendungen und Begrüßungsbildschirme definieren“ auf Seite 34.

Mobildesign in der Anwendung verwenden

In einem *Design* ist das Layout der visuellen Komponenten einer Anwendung definiert. Ein Design kann beispielsweise das Farbschema oder die Schriftart einer Anwendung definieren oder aber auch ein komplett neues Skinning aller Komponenten definieren, die von der Anwendung verwendet werden.

CSS-Stile können Sie in Flex-Komponenten nur festlegen, wenn diese Stile im aktuellen Design enthalten sind. Um zu bestimmen, ob ein CSS-Stil vom aktuellen Design unterstützt wird, rufen Sie den Eintrag für den Stil im ActionScript 3.0 Referenzhandbuch für die Adobe Flash-Plattform auf.

Flex unterstützt drei Primärdesigns: Mobile, Spark und Halo. Im Mobildesign ist die Standarddarstellung von Flex-Komponenten beim Erstellen einer Mobilanwendung definiert. Um Kompatibilität einiger Flex-Komponenten mit dem Mobildesign zu schaffen, hat Adobe für die Komponenten neue Skins erstellt. Deshalb weisen einige Komponenten Skins speziell für ein Design auf.

Mit Flex können Anwendungen für verschiedene Mobilgeräte mit unterschiedlichen Bildschirmgrößen und -auflösungen erstellt werden. Flex vereinfacht die Entwicklung auflösungsunabhängiger Anwendungen durch Bereitstellung dpi-unabhängiger Skins für Mobilkomponenten. Weitere Informationen zu Mobilskins finden Sie unter „Grundlagen des Mobilskinnings“ auf Seite 173.

Weitere Informationen zu Stilen und Designs finden Sie unter Stile und Designs und „Mobilstile“ auf Seite 173.

Community-Ressourcen

Hier finden Sie Informationen zu den neuen Funktionen in Flex und Flash Builder:

- Einführung in das Adobe Flex 4.5 SDK der Adobe-Produktmanagerin Deepa Subramaniam
- Mobile development using Adobe Flex SDK and Flash Builder von Adobe-Produktdesigner Narciso Jaramillo
- What's new in Flex 4.6 SDK von Adobe-Produktmanager Jacob Surber und What's New in Flash Builder 4.6 von Adobe Produktmanager Adam Lehman.

Im Flex Developer Center finden Sie viele Ressourcen, die Ihnen bei den ersten Schritten beim Entwickeln von Mobilanwendungen mithilfe von Flex helfen:

- Artikel zu den ersten Schritten, Links und Übungen
- Beispiele von echten in Flex erstellten Anwendungen
- Das Flex Cookbook mit Antworten zu häufigen Kodierungsproblemen
- Links zur Flex-Community und zu anderen Websites, die sich mit Flex befassen

Adobe TV ist eine weitere Ressource mit Videos von Adobe-Ingenieuren, Produktgurus und Kunden zur Anwendungsentwicklung in Flex. Eines der verfügbaren Videos ist Erstellen der ersten Mobilanwendung in Flash Builder 4.5.

Unterschiede zwischen der Bereitstellung von Mobil-, Desktop- und Browseranwendungen

Verwenden Sie Flex zum Entwickeln von Anwendungen für die folgenden Bereitstellungsumgebungen:

Browser Sie stellen die Anwendung als SWF-Datei zur Verwendung in Flash Player in einem Browser bereit.

Desktop Sie stellen eine eigenständige AIR-Anwendung für einen Desktopcomputer bereit, etwa einen Windows-PC oder einen Macintosh.

Mobile Sie stellen eine eigenständige AIR-Anwendung für ein Mobilgerät (z. B. Telefon oder Tablet) bereit.

Die Flash Player- und die AIR-Laufzeit ähneln sich. Die meisten Vorgänge können in beiden Laufzeiten ausgeführt werden. AIR ermöglicht nicht nur das Bereitstellen eigenständiger Anwendungen außerhalb eines Browsers, sondern auch die enge Integration in die Hostplattform. Diese Integration wiederum ermöglicht Funktionen wie den Zugriff auf das Dateisystem des Geräts, das Erstellen von und Arbeiten mit lokalen SQL-Datenbanken und mehr.

Überlegungen beim Entwerfen und Entwickeln von Mobilanwendungen

Anwendungen für Mobilgeräte mit Touchscreen unterscheiden sich von Desktop- und Browseranwendungen wie folgt:

- Für die problemlose Fingereingabe weisen Mobilkomponenten im Allgemeinen größere Berührungsbereiche als Desktop- oder Browseranwendungen auf.
- Die Interaktionsmuster für Aktionen wie etwa einen Bildlauf sind bei Touchscreen-Geräten unterschiedlich.
- Aufgrund des begrenzten Bildschirmbereichs ist bei Mobilanwendungen in der Regel jeweils nur ein kleiner Ausschnitt der Benutzeroberfläche auf dem Bildschirm sichtbar.
- Beim Benutzeroberflächendesign müssen die unterschiedlichen Bildschirmauflösungen bei den Geräten berücksichtigt werden.
- Die CPU- und GPU-Leistung ist bei Telefonen und Tablets geringer als bei Desktopgeräten.
- Aufgrund des begrenzt verfügbaren Arbeitsspeichers auf Mobilgeräten müssen Anwendungen sorgfältig mit dem Arbeitsspeicher umgehen.
- Mobilanwendungen können jederzeit beendet und neu gestartet werden, beispielsweise wenn das Gerät einen Anruf oder eine Textnachricht empfängt.

Deshalb geht es beim Entwickeln einer Anwendung für ein Mobilgerät nicht nur um das Reduzieren einer Desktopanwendung auf eine andere Bildschirmgröße. Mit Flex können Sie separate Benutzeroberflächen erstellen, die für jedes Format geeignet sind, und das zugrunde liegende Modell und den Datenzugriffscode für Mobil-, Browser- und Desktopprojekte gemeinsam verwenden.

Einschränkungen bei der Verwendung von Spark- und MX-Komponenten in Mobilanwendungen

Beim Erstellen von Mobilanwendungen in Flex verwenden Sie den Spark-Komponentensatz. Die Spark-Komponenten sind in spark.components.*-Paketen. Aus Leistungsgründen, oder weil nicht alle Spark-Komponenten Skins für das Mobildesign bieten, unterstützen Mobilanwendungen nicht den gesamten Spark-Komponentensatz.

Mit Ausnahme der MX-Diagrammsteuerelemente und des Spacer-Steuerelements von MX unterstützen Mobilanwendungen den in mx.*-Paketen.

In der folgenden Tabelle sind die Komponenten aufgelistet, die Sie für Mobilanwendungen verwenden können, nicht verwenden können bzw. mit Umsicht verwenden sollten:

Komponente	Komponente	Verwendung im Mobilbereich	Hinweis
Spark ActionBar Spark BusyIndicator Spark Callout Spark CalloutButton Spark DateSpinner Spark SpinnerList Spark SpinnerListContainer	Spark TabbedViewNavigator Spark TabbedViewNavigatorApplication Spark ToggleSwitch Spark View Spark ViewMenu Spark ViewNavigator Spark ViewNavigatorApplication	Ja	Diese neuen Komponenten unterstützen Mobilanwendungen.
Spark Button Spark CheckBox Spark DataGroup Spark Group/HGroup/VGroup/TileGroup Spark Image/BitmapImage Spark Label	Spark List Spark RadioButton/RadioButtonGroup Spark SkinnableContainer Spark Scroller Spark TextArea Spark TextInput	Ja	Die meisten dieser Komponenten bieten Skins für das Mobildesign. Label, Image und BitmapImage können verwendet werden, obwohl sie keine Mobilskin bieten. Manche Spark-Layoutcontainer, z. B. Group und dessen Unterklassen, bieten keine Skins. Deshalb können sie in Mobilanwendungen verwendet werden.
Weitere Spark-Komponenten mit Skinfunktionen		Nicht empfohlen	Die Verwendung anderer skinfähiger Spark-Komponenten als der oben aufgelisteten wird nicht empfohlen, da diese keine Skins für das Mobildesign bieten. Wenn die Komponente keine Skin für das Mobildesign bietet, können Sie eine für Ihre Anwendung erstellen.
Spark DataGrid	Spark RichEditableText Spark RichText	Nicht empfohlen	Die Verwendung dieser Komponenten wird aus Leistungsgründen nicht empfohlen. Sie können sie zwar in Mobilanwendungen verwenden, müssen jedoch mit Leistungseinbußen rechnen. Beim DataGrid-Steuerelement ist die Leistung von der Datenmenge abhängig, die gerendert werden soll. Beim RichEditableText-Steuerelement und beim RichText-Steuerelement ist die Leistung von der Textmenge und der Anzahl der Steuerelemente in der Anwendung abhängig.

Komponente	Komponente	Verwendung im Mobilbereich	Hinweis
Andere MX-Komponenten als Spacer und Diagrammsteuerelemente		Nein	Mobilanwendungen unterstützen keine MX-Komponenten, z. B. MX Button, CheckBox, List und DataGrid. Diese Komponenten entsprechen den Flex 3-Komponenten in den mx.controls.*- und mx.containers.*-Paketen.
MX Spacer		Ja	Spacer verwendet keine Skin und kann daher in Mobilanwendungen eingesetzt werden.
MX-Diagrammkomponenten		Ja, mit Auswirkungen auf die Leistung	MX-Diagrammsteuerelemente wie AreaChart und BarChart können Sie in Mobilanwendungen verwenden. Die MX-Diagrammsteuerelemente befinden sich in den mx.charts.*-Paketen. Je nach Umfang und Typ der Diagrammdaten kann jedoch die Leistung auf Mobilgeräten beeinträchtigt werden. Standardmäßig enthält Flash Builder nicht die MX-Komponenten unter dem Bibliothekspfad von Mobilprojekten. Wenn Sie die MX-Diagrammkomponenten in einer Anwendung verwenden möchten, fügen Sie dem Bibliothekspfad „mx.swc“ und „charts.swc“ hinzu.

Die folgenden Flex-Funktionen werden in Mobilanwendungen nicht unterstützt:

- Keine Unterstützung für Drag-&-Drop-Vorgänge
- Keine Unterstützung für das ToolTip-Steuerelement
- Keine Unterstützung für RSLs

Leistungserwägungen bei Mobilanwendungen

Aufgrund der Leistungseinschränkungen von Mobilgeräten sind manche Aspekte der Entwicklung von Mobilanwendungen und der Entwicklung für Browser- und Desktopanwendungen unterschiedlich. Folgende Leistungsaspekte sind zu berücksichtigen:

- **Schreiben von Elementrenderern in ActionScript**

  In Mobilanwendungen soll die Bildlauffunktion für Listen so schnell wie möglich funktionieren. Schreiben Sie Elementrenderer in ActionScript, um die höchstmögliche Leistung zu erzielen. Zwar können Sie Elementrenderer auch in MXML schreiben, doch kann dies die Anwendungsleistung beeinträchtigen.

  Flex bietet zwei Elementrenderer, die für Mobilanwendungen konzipiert sind: spark.components.LabelItemRenderer und spark.components.IconItemRenderer. Weitere Informationen zu diesen Elementrenderern finden Sie unter Using a mobile item renderer with a Spark list-based control.

  Weitere Informationen zur Erstellung benutzerdefinierter Elementrenderer in ActionScript finden Sie unter Custom Spark item renderers. Mehr zu den Unterschieden zwischen Elementrenderern für Mobilgeräte und Computer finden Sie unter Unterschiede zwischen Elementrenderern für Mobilgeräte und Computer.

- **Entwickeln benutzerdefinierter Skins mit ActionScript und kompilierten FXG-Grafiken**

  Die mit Flex gelieferten Mobilskins wurden unter Verwendung kompilierter FXG-Grafiken in ActionScript geschrieben, um die bestmögliche Leistung zu erzielen. Sie können Skins auch in MXML schreiben, doch kann dies je nach Anzahl der Komponenten, die MXML-Skins verwenden, die Anwendungsleistung beeinträchtigen. Um die höchstmögliche Leistung zu erzielen, schreiben Sie Skins in ActionScript und verwenden kompilierte FXG-Grafiken. Weitere Informationen finden Sie unter Spark-Skins und FXG- und MXML-Grafiken.

- **Texteingabekomponenten verwenden, die StageText verwenden**

  Verwenden Sie beim Hinzufügen von Texteingabekomponenten wie TextInput und TextArea die Standardeinstellungen. Diese Steuerelemente verwenden StageText als zugrunde liegenden Mechanismus für die Texteingabe, wobei eine Verknüpfung zu den nativen Texteingabeklassen hergestellt wird. Damit erhalten Sie eine höhere Leistung sowie Zugriff auf native Funktionen wie Autokorrektur, automatische Großschreibung, Texteinschränkung und benutzerdefinierte virtuelle Tastaturen.

  Die Verwendung von StageText hat jedoch auch einige Nachteile. So kann beispielsweise kein Bildlauf in der Ansicht, in der sich die Steuerelemente befinden, ausgeführt werden. Darüber hinaus können Sie keine eingebetteten Schriftarten und keine benutzerdefinierte Größenänderung für die StageText-basierten Steuerelemente verwenden. Sind diese Funktionen notwendig, können Sie Texteingabesteuerelemente verwenden, die auf der TextField-Klasse basieren.

  Weitere Informationen finden Sie unter „Text in Mobilanwendungen“ auf Seite 147.

- **Umsicht bei der Verwendung von MX-Diagrammkomponenten in Mobilanwendungen**

  MX-Diagrammsteuerelemente wie AreaChart und BarChart können Sie in Mobilanwendungen verwenden. Je nach Umfang und Typ der Diagrammdaten kann jedoch die Leistung beeinträchtigt werden.


Blogger Nahuel Foronda schrieb einige Artikel zu Mobil-ItemRenderer in ActionScript.


Blogger Rich Tretola beschreibt, wie eine Liste mit einem ItemRenderer für eine Mobilanwendung erstellt wird.

Kapitel 2: Entwicklungsumgebung

Android-Anwendung in Flash Builder erstellen

Im Folgenden soll ein allgemeiner Arbeitsablauf zum Erstellen einer Flex-Mobilanwendung für Google Android vorgestellt werden. Dieser Arbeitsablauf setzt eine bereits entwickelte Mobilanwendung voraus. Weitere Informationen finden Sie unter „Mobilanwendung planen“ auf Seite 1.

Adobe-Evangelist Mike Jones gibt Tipps, die er sich beim Entwickeln des Spiels „Mode“, das auf mehreren Plattformen gespielt werden kann, angeeignet hat: 10 tips when developing for multiple devices.

AIR-Anforderungen

Für Flex- und ActionScript-Mobilprojekte ist AIR 2.6 oder höher erforderlich. Sie können Mobilprojekte auf Geräten ausführen, die AIR 2.6 oder höher unterstützen.

Sie können AIR 2.6 oder höher auf unterstützten Android-Geräten installieren, auf denen Android 2.2 oder höher ausgeführt wird. Eine vollständige Liste für unterstützte Android-Geräte finden Sie unter Certified Devices. Überprüfen Sie außerdem die Mindestsystemanforderungen zum Ausführen von Adobe AIR auf Android-Geräten unter Systemanforderungen für Mobilgeräte.

Hinweis: *Wenn Ihr Gerät nicht mit AIR 2.6 oder höher kompatibel ist, können Sie Mobilanwendungen mit Flash Builder auf dem Computer ausführen und debuggen.*

Jede Version des Flex SDK bietet die erforderliche Adobe AIR-Version. Wenn Sie Mobilanwendungen auf einem Gerät über eine ältere Version des Flex SDK installiert haben, deinstallieren Sie AIR auf dem Gerät. Flash Builder installiert die korrekte AIR-Version, wenn Sie eine Mobilanwendung auf einem Gerät ausführen oder debuggen.

Anwendung erstellen

1 Wählen Sie in Flash Builder „Datei“ > „Neu“ > „Flex-Mobilprojekt“.

Bei einem Flex-Mobilprojekt handelt es sich um ein spezielles AIR-Projekt. Folgen Sie den Anweisungen im Projekterstellungsassistenten wie bei der Einrichtung eines AIR-Projekts mit Flash Builder. Weitere Informationen finden Sie unter Flex-Mobilprojekte.

Informationen zum Festlegen Android-spezifischer Voreinstellungen für Mobilprojekte finden Sie unter „Voreinstellungen für Mobilprojekte festlegen“ auf Seite 15.

Bei der Erstellung eines Flex-Mobilprojekts erstellt Flash Builder die folgenden Dateien für das Projekt:

- *`Projektname`*`.mxml`

  Die standardmäßige Anwendungsdatei für das Projekt.

  Standardmäßig gibt Flash Builder dieser Datei den Namen des Projekts. Enthält der Projektname ungültige ActionScript-Zeichen, wird die Datei „Main.mxml“ benannt. Diese MXML-Datei enthält das Basis-Spark-Anwendungstag für das Projekt. Bei dem Basis-Spark-Anwendungstag kann es sich um ViewNavigatorApplication oder TabbedViewNavigatorApplication handeln.

  Mit Ausnahme des ActionBar-Inhalts, der für alle Ansichten angezeigt wird, wird normalerweise nicht direkt Inhalt in die Standardanwendungsdatei eingefügt. Um Inhalt zu ActionBar hinzuzufügen, setzen Sie die Eigenschaften `navigatorContent`, `titleContent` oder `actionContent`.

- *ProjectName*`HomeView.mxml`

  Die Datei mit der Startansicht für das Projekt. Flash Builder legt die Datei in einem Ansichtenpaket ab. Das Attribut `firstView` des ViewNavigatorApplication-Tags in *ProjectName*`.mxml` legt diese Datei als standardmäßige Startansicht der Anwendung fest.

Weitere Informationen zum Definieren von Ansichten finden Sie unter „Ansichten in einer Mobilanwendung definieren“ auf Seite 39.

Sie können auch ein reines ActionScript-Projekt erstellen. Siehe „ActionScript-Mobilprojekte erstellen“ auf Seite 12.

2 (Optional) Fügen Sie Inhalt in die ActionBar der Hauptanwendungsdatei ein.

Die ActionBar enthält Inhalte und Funktionen zur Anwendung oder zur aktuellen Ansicht. Fügen Sie hier Inhalte ein, die in allen Ansichten der Anwendung angezeigt werden sollen. Siehe „Navigation, Titel und Aktionssteuerelemente in einer Mobilanwendung definieren“ auf Seite 63.

3 Legen Sie das Layout der Startansicht Ihrer Anwendung fest.

In Flash Builder werden Komponenten im Designmodus oder Quellmodus hinzugefügt.

Verwenden Sie nur Komponenten, die Flex für die mobile Entwicklung unterstützt. Im Design- und im Quellmodus schlägt Flash Builder Ihnen die unterstützten Komponenten vor. Siehe „Benutzeroberfläche und Layout“ auf Seite 25.

Fügen Sie innerhalb einer Ansicht nur Inhalte in die ActionBar ein, die nur für diese Ansicht gelten.

4 (Optional) Fügen Sie der Anwendung ggf. weitere Ansichten hinzu.

Wählen Sie im Flash Builder Package Explorer aus dem Kontextmenü des Ansichtenpakets im Projekt „Neue MXML-Komponente“. Der Assistent „Neue MXML-Komponente“ begleitet Sie bei der Erstellung der Ansicht.

Weitere Informationen zu Ansichten finden Sie unter „Ansichten in einer Mobilanwendung definieren“ auf Seite 39.

5 (Optional) Fügen Sie auf Mobilanwendungen ausgelegte Elementrenderer für Listenkomponenten hinzu.

Adobe stellt für Mobilanwendungen den ActionScript-basierten Elementrenderer IconItemRenderer bereit. Siehe Using a mobile item renderer with a Spark list-based control.

6 Konfigurieren Sie Startkonfigurationen zum Ausführen und Debuggen der Anwendung.

Sie können die Anwendung auf dem Computer oder einem Mobilgerät ausführen und/oder debuggen.

Zum Ausführen und/oder Debuggen einer Anwendung über Flash Builder wird eine Startkonfiguration vorausgesetzt. Wenn Sie zum ersten Mal eine Mobilanwendung ausführen oder debuggen, fordert Flash Builder Sie zur Erstellung einer Startkonfiguration auf.

Beim Ausführen oder Debuggen einer Mobilanwendung auf einem Gerät installiert Flash Builder die Anwendung auf dem Gerät.

Siehe „Testen und Debuggen“ auf Seite 187.

7 Exportieren Sie die Anwendung als Installer-Paket.

Mit der Option „Releasebuild exportieren“ erstellen Sie Pakete, die auf Mobilgeräten installiert werden können. Flash Builder erstellt die Pakete für die Plattform, die Sie zum Export bestimmt haben. Siehe „Android APK-Pakete für die Veröffentlichung exportieren“ auf Seite 195.

Der zertifizierte Adobe-Experte für Flex, Brent Arnold, hat folgende Videotutorials erstellt:

•Create a Flex mobile application with multiple views

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

iOS-Anwendung in Flash Builder erstellen

Im Folgenden soll ein allgemeiner Arbeitsablauf zum Erstellen einer Mobilanwendung für Apple iOS vorgestellt werden.

1 Bevor Sie mit dem Erstellen der Anwendung beginnen, führen Sie folgende Schritte unter „Apple iOS-Entwicklungsprozess mit Flash Builder“ auf Seite 21 durch.

2 Wählen Sie in Flash Builder „Datei“ > „Neu“ > „Flex-Mobilprojekt“.

Wählen Sie als Zielplattform „Apple iOS“ aus und legen Sie die Mobilprojekteinstellungen fest.

Befolgen Sie die Anweisungen im Assistenten für neue Projekte wie bei der Einrichtung eines gewöhnlichen Projekts mit Flash Builder. Weitere Informationen hierzu finden Sie unter „Anwendung erstellen“ auf Seite 9.

Sie können auch ein reines ActionScript-Projekt erstellen. Weitere Informationen finden Sie unter ActionScript-Mobilprojekte erstellen.

3 Konfigurieren Sie Startkonfigurationen zum Ausführen und Debuggen der Anwendung. Sie können die Anwendung auf dem Computer oder einem angeschlossenen Mobilgerät ausführen und/oder debuggen.

Weitere Informationen finden Sie unter „Anwendung auf einem Apple iOS-Gerät debuggen“ auf Seite 191.

4 Exportieren Sie die Anwendung in den Apple App Store oder stellen Sie die iOS-Paketanwendung (IPA) auf einem Gerät bereit.

Weitere Informationen finden Sie unter „Apple iOS-Pakete für die Veröffentlichung exportieren“ auf Seite 196 und „Anwendung auf einem Apple iOS-Gerät installieren“ auf Seite 193.

Verwandte Themen

Beginning a Mobile Application (engl. Video)

BlackBerry Tablet OS-Anwendung in Flash Builder erstellen

Flash Builder enthält ein Plug-In von Research In Motion (RIM), mit dem Sie sowohl Flex- als auch ActionScript-Anwendungen für das BlackBerry® Tablet OS erstellen und verpacken können.

Anwendung erstellen

Im Folgenden wird ein allgemeiner Arbeitsablauf zum Erstellen von Anwendungen für das BlackBerry Tablet OS beschrieben.

1 Bevor Sie mit dem Erstellen der Mobilanwendung beginnen, installieren Sie das BlackBerry Tablet OS SDK for AIR von der Website für die BlackBerry Tablet OS-Anwendungsentwicklung.

Das BlackBerry Tablet OS SDK for AIR enthält APIs, mit deren Hilfe Sie AIR-basierte Flex- und ActionScript-Anwendungen erstellen können.

Weitere Informationen zum Installieren des BlackBerry Tablet OS SDK finden Sie im BlackBerry Tablet OS Getting Started Guide.

2 Wählen Sie zum Erstellen einer Flex-basierten AIR-Anwendung in Flash Builder „Datei > Neu > Flex-Mobilprojekt“.

Folgen Sie den Anweisungen im Projekterstellungsassistenten wie bei der Einrichtung eines AIR-Projekts mit Flash Builder. Wählen Sie als Zielplattform „BlackBerry Tablet OS“.

Weitere Informationen finden Sie unter Flex-Mobilprojekte.

3 Wählen Sie zum Erstellen einer ActionScript-basierten AIR-Anwendung in Flash Builder „Datei > Neu > ActionScript-Mobilprojekt“.

Folgen Sie den Anweisungen im Projekterstellungsassistenten wie bei der Einrichtung eines AIR-Projekts mit Flash Builder. Wählen Sie als Zielplattform „BlackBerry Tablet OS“.

Weitere Informationen finden Sie unter ActionScript-Mobilprojekte erstellen.

Anwendung signieren, verpacken und bereitstellen

Informationen zum Signieren, Verpacken und Bereitstellen der Anwendung finden Sie im BlackBerry Tablet OS SDK for Adobe AIR Development Guide von RIM.

Weitere Ressourcen zur BlackBerry Tablet OS-Entwicklung von Adobe und RIM finden Sie unter Adobe Developer Connection.

ActionScript-Mobilprojekte erstellen

Verwenden Sie Flash Builder zum Erstellen einer ActionScript-Mobilanwendung. Die erstellte Anwendung basiert auf der Adobe AIR-API.

1 Wählen Sie „Datei“ > „Neu“ > „ActionScript-Mobilprojekt“.

2 Geben Sie einen Projektnamen und Projektpfad ein. Der Standardspeicherort ist der aktuelle Arbeitsbereich.

3 Verwenden Sie das Standard-SDK von Flex 4.6, mit dem die Entwicklung von Mobilanwendungen möglich ist.

Klicken Sie auf „Weiter“.

4 Wählen Sie die Zielplattformen für Ihre Anwendung aus und geben Sie für jede Plattform Mobilprojekteinstellungen an.

Weitere Informationen zu Mobilprojekteinstellungen finden Sie unter „Voreinstellungen für Mobilprojekte festlegen“ auf Seite 15.

5 Klicken Sie auf „Fertig stellen“ oder auf „Weiter“, um zusätzliche Konfigurationsoptionen und Erstellungspfade anzugeben.

Weitere Informationen zu den Projektkonfigurationsoptionen und Erstellungspfaden finden Sie unter Erstellungspfade, native Erweiterungen und sonstige Projektkonfigurationsoptionen.

Verwenden nativer Erweiterungen

Mit nativen Erweiterungen können Sie Ihrer Mobilanwendung native Plattformfunktionen hinzufügen.

Eine native Erweiterung enthält ActionScript-Klassen und nativen Code. Mit der nativen Implementierung für Code können Sie auf gerätespezifische Funktionen zugreifen, auf die mithilfe von reinen ActionScript-Klassen nicht zugegriffen werden kann. Beispiel: Zugriff auf die Vibrationsfunktion des Geräts.

Native Implementierung für Code kann als Code definiert werden, der außerhalb der AIR-Laufzeit ausgeführt wird. Sie definieren plattformspezifische ActionSript-Klassen und native Implementierung für Code in der Erweiterung. Die ActionScript-Erweiterungsklassen greifen mit dem nativen Code auf Daten zu und tauschen sie aus. Dabei wird ActionScript-Klasse ExtensionContext verwendet.

Erweiterungen sind spezifisch für eine Hardwareplattform eines Geräts. Sie können plattformspezifische Erweiterungen oder eine einzelne Erweiterung, die auf mehrere Plattformen ausgerichtet ist, erstellen. Beispielsweise können Sie eine native Erweiterung erstellen, die sowohl auf Android- als auch auf iOS-Plattformen ausgerichtet ist. Native Erweiterungen werden auf folgenden Mobilgeräten unterstützt:

- Android-Geräte mit Android 2.2 oder höher
- iOS-Geräte mit iOS 4.0 oder höher

Ausführliche Informationen zum Erstellen von plattformübergreifenden nativen Erweiterungen finden Sie unter Entwickeln nativer Erweiterungen für Adobe AIR.

Von Adobe und der Community zusammengetragene Beispiele nativer Erweiterungen finden Sie unter Native extensions for Adobe AIR.

Native Erweiterungen verpacken

Um native Erweiterungen für Anwendungsentwickler bereitzustellen, können Sie alle erforderlichen Dateien in eine ActionScript Native Extensions(ANE)-Datei verpacken, indem Sie folgende Schritte ausführen:

1 Erstellen Sie die ActionScript-Bibliothek der Erweiterung in einer SWC-Datei.

2 Erstellen Sie die nativen Bibliotheken der Erweiterung. Wenn die Erweiterung mehrere Plattformen unterstützen muss, erstellen Sie für jede Zielplattform eine Bibliothek.

3 Erstellen Sie ein signiertes Zertifikat für Ihre Erweiterung. Wenn die Erweiterung nicht signiert ist, zeigt Flash Builder eine Warnmeldung an, wenn Sie Ihrem Projekt eine Erweiterung hinzufügen.

4 Erstellen Sie eine Erweiterungsdeskriptordatei.

5 Fügen Sie alle externen Ressourcen für die Erweiterung hinzu, z. B. Bilder.

6 Erstellen Sie mit dem AIR Developer Tool ein Erweiterungspaket. Weitere Informationen finden Sie in der AIR-Dokumentation.

Ausführliche Informationen zum Verpacken von plattformübergreifenden nativen Erweiterungen finden Sie unter Entwickeln nativer Erweiterungen für Adobe AIR.

Projekt native Erweiterungen hinzufügen

Sie fügen eine ActionScript Native Extension-Datei (ANE) in den Erstellungspfad eines Projekt genau wie beim Hinzufügen einer SWC-Datei ein.

1 Wenn Sie ein Flex-Mobilprojekt in Flash Builder erstellen, wählen Sie auf der Einstellungsseite „Erstellungspfade“ die Registerkarte „Native Erweiterungen“.

Sie können außerdem Erweiterungen vom Dialogfeld „Projekteigenschaften“ hinzufügen, indem Sie „Flex-Erstellungspfad“ auswählen.

2 Navigieren Sie zur ANE-Datei oder zu dem Ordner, der die ANE-Dateien enthält, um sie dem Projekt hinzuzufügen. Wenn Sie eine ANE-Datei hinzufügen, wird die Erweiterungs-ID zur Anwendungsdeskriptordatei des Projekts (*project name*-app.xml) standardmäßig hinzugefügt.

Flash Builder zeigt ein Fehlersymbol für die hinzugefügte Erweiterung in folgenden Szenarien an:

- Die AIR-Laufzeitversion der Erweiterung ist höher als die Laufzeitversion der Anwendung.
- Die Erweiterung beinhaltet nicht alle ausgewählten Plattformen, auf die die Anwendung ausgerichtet ist.

Hinweis: *Sie können eine native ActionScript-Erweiterung erstellen, die auf mehrere Plattformen ausgerichtet ist. Um eine Anwendung, die diese ANE-Datei beinhaltet, auf Ihrem Entwicklercomputer mithilfe von AIR-Simulator zu testen, stellen Sie sicher, dass die ANE-Datei die Computerplattform unterstützt. Um die Anwendung z. B. mithilfe des AIR-Simulators unter Windows zu testen, stellen Sie sicher, dass die ANE-Datei Windows unterstützt.*

Native ActionScript-Erweiterungen in einem Anwendungspaket einschließen

Wenn Sie die Option „Releasebuild exportieren“ zum Exportieren der Mobilanwendung verwenden, werden die im Projekt verwendeten Erweiterungen im Anwendungspaket standardmäßig eingeschlossen.

Um die standardmäßige Auswahl zu ändern, führen Sie folgende Schritte durch:

1 Wählen Sie im Dialogfeld „Releasebuild exportieren“ unter „Paketeinstellungen“ die Registerkarte „Native Erweiterungen“.

2 Die nativen ActionScript-Erweiterungsdateien, auf die in Ihrem Projekt hingewiesen wird, sind aufgelistet und zeigen an, ob die ANE-Datei im Projekt verwendet wird.

Wird die ANE-Datei im Projekt verwendet, wird sie standardmäßig im Anwendungspaket ausgewählt.

Wenn sich die ANE-Datei im Projekt befindet, aber nicht verwendet wird, erkennt der Compiler die ANE-Datei nicht. Dann wird sie dem Anwendungspaket nicht hinzugefügt. Um dem Anwendungspaket die ANE-Datei hinzuzufügen, führen Sie folgende Schritte durch:

a Wählen Sie im Dialogfeld „Projekteigenschaften“ die Option „Flex-Erstellungsverpackung“ und anschließend die erforderliche Plattform.

b Wählen Sie die nativen Erweiterungen aus, die dem Anwendungspaket hinzugefügt werden sollen.

Unterstützung für native iOS5-Erweiterungen

Zum Verpacken nativer Erweiterungen, die iOS5 SDK-Funktionen verwenden, benötigt das AIR Developer Tool (ADT) den Speicherort des iOS5 SDK.

Unter Mac OS können Sie in Flash Builder den Speicherort des iOS5 SDK über das Dialogfeld „Verpackungseinstellungen“ auswählen. Nachdem Sie den Speicherort des iOS SDK ausgewählt haben, wird dieser über den ADT-Befehl `-platformsdk` übergeben.

Hinweis: *Diese Funktion wird derzeit unter Windows nicht unterstützt.*

Weitere Informationen finden Sie unter Entwickeln nativer Erweiterungen für Adobe AIR.

Voreinstellungen für Mobilprojekte festlegen

Gerätekonfigurationen festlegen

Für die Vorschau von Gerätedisplaygrößen in der Designansicht und beim Starten von Anwendungen auf dem Desktop mit dem AIR Debug Launcher (ADL) verwendet Flash Builder Gerätekonfigurationen. Siehe „Geräteinformationen für die Desktopvorschau konfigurieren“ auf Seite 188.

Öffnen Sie zum Festlegen von Gerätekonfigurationen „Voreinstellungen“ und wählen Sie „Flash Builder“ > „Gerätekonfigurationen“ aus.

Flash Builder bietet diverse voreingestellte Gerätekonfigurationen. Sie können weitere Gerätekonfigurationen hinzufügen, bearbeiten oder entfernen. Die Standardkonfigurationen von Flash Builder können nicht geändert werden.

Durch Klicken auf die Schaltfläche „Standardwerte wiederherstellen“ können Sie die standardmäßigen Gerätekonfigurationen wiederherstellen, aber keine Gerätekonfigurationen entfernen. Ferner gilt: Wenn Sie eine Gerätekonfiguration mit einem Namen hinzugefügt haben, der einer der Standardkonfigurationen entspricht, überschreibt Flash Builder die hinzugefügte Konfiguration mit den Standardeinstellungen.

Gerätekonfigurationen sind mit folgenden Eigenschaften ausgestattet:

Eigenschaft	**Beschreibung**
Name des Geräts	Eine eindeutige Bezeichnung für das Gerät.
Plattform	Die Geräteplattform. Wählen Sie eine Plattform aus der Liste der unterstützten Plattformen aus.
Vollbildgröße	Breite und Höhe des Gerätebildschirms
Nutzbare Bildschirmgröße	Die Standardgröße einer Anwendung auf dem Gerät. Hierbei handelt es sich um die geschätzte Größe einer Anwendung, die im Nicht-Vollbildmodus gestartet wurde und unter Berücksichtigung der reservierten Bereiche (z. B. Statusleiste).
Pixel pro Zoll	Pixel pro Zoll auf dem Gerätebildschirm.

Zielplattformen auswählen

Flash Builder unterstützt je nach Anwendungstyp verschiedene Zielplattformen.

Öffnen Sie zum Auswählen einer Plattform „Voreinstellungen“ und wählen Sie „Flash Builder“ > „Zielplattformen“ aus.

Zu Plug-Ins von Drittanbietern schlagen Sie bitte im entsprechenden Begleitmaterial nach.

Anwendungsvorlage auswählen

Bei der Erstellung einer Mobilanwendung können Sie aus folgenden Anwendungsvorlagen wählen:

Leer Das Spark-Application-Tag wird als Basisanwendungselement verwendet.

Mithilfe dieser Option erstellen Sie eine benutzerdefinierte Anwendung, ohne die Standardansichtsnavigation zu verwenden.

Ansichtsbasierte Anwendung Das Spark-ViewNavigatorApplication-Tag wird als Basisanwendungselement verwendet, um eine Anwendung mit einer einzigen Ansicht zu erstellen.

Sie können für die Startansicht einen Namen eingeben.

Anwendung im Registerkartenformat Das Spark-TabbedViewNavigatorApplication-Tag wird als Basisanwendungselement verwendet, um eine Anwendung im Registerkartenformat zu erstellen.

Um eine Registerkarte zu erzeugen, geben Sie einen Namen ein und klicken auf „Hinzufügen“. Durch Klicken auf „Nach oben“ und „Nach unten“ können Sie die Reihenfolge der Registerkarten ändern. Um eine Registerkarte zu entfernen, wählen Sie sie aus und klicken auf „Entfernen“.

Der Name der Ansicht besteht aus dem Registerkartennamen, an den „View“ angefügt wird. Wenn Sie beispielsweise eine Registerkarte „FirstTab“ benennen, generiert Flash Builder eine Ansicht namens „FirstTabView“.

Für jede Registerkarte wird eine neue MXML-Datei im Paket „Ansichten“ erstellt.

Hinweis: *Der Paketname lässt sich nicht mit dem Flex-Mobilprojekt-Assistenten konfigurieren.*

Bei der Erzeugung von MXML-Dateien gelten folgende Regeln:

- Wenn der Name der Registerkarte einem gültigen ActionScript-Klassenamen entspricht, generiert Flash Builder die MXML-Datei mithilfe des Registerkartennamens, an den „View“ angefügt wird.
- Wenn es sich beim Registerkartennamen nicht um einen gültigen Klassennamen handelt, wird der Registerkartenname von Flash Builder geändert, indem ungültige Zeichen entfernt und durch gültige Anfangszeichen ersetzt werden. Wenn der geänderte Name inakzeptabel ist, wird der MXML-Dateiname von Flash Builder in „ViewN“ geändert, wobei „N“ für die Position der Ansicht, angefangen bei N=1, steht.


Der zertifizierte Adobe-Experte für Flex, Brent Arnold, hat ein Videotutorial zur Verwendung der Vorlage für die Anwendung im Registerkartenformat erstellt.

Mobilanwendungsberechtigungen auswählen

Beim Erstellen einer Mobilanwendung können Sie die Standardberechtigungen für eine Zielplattform festlegen oder ändern. Die Berechtigungen werden beim Kompilieren festgelegt und können nicht zur Laufzeit geändert werden.

Wählen Sie zunächst die Zielplattform aus und legen Sie dann die Berechtigungen für jede Plattform fest. Sie können die Berechtigungen später in der Anwendungsbeschreibungs-XML-Datei ändern.

Mit einigen Plug-Ins von Drittanbietern lassen sich Flex- und ActionScript-Projekte auch auf anderen Plattformen bearbeiten. Zu plattformspezifischen Berechtigungen konsultieren Sie bitte die jeweilige Dokumentation.

Berechtigungen für die Google Android-Plattform

Für Google Android können Sie folgende Berechtigungen festlegen:

INTERNET Erlaubt Netzwerkanforderungen und Remote-Debugging

Die INTERNET-Berechtigung ist standardmäßig ausgewählt. Wenn Sie diese Berechtigung deaktivieren, können Sie die Anwendung nicht auf einem Gerät debuggen.

WRITE_EXTERNAL_STORAGE Ermöglicht das Schreiben auf ein externes Gerät.

Wenn Sie diese Berechtigung auswählen, kann die Anwendung auf eine externe Speicherkarte auf dem Gerät schreiben.

READ_PHONE_STATE Schaltet den Ton während eines eingehenden Anrufs aus

Wenn Sie diese Berechtigung auswählen, kann die Anwendung den Ton während eines Telefongesprächs ausschalten. Beispiel: Sie können diese Berechtigung auswählen, wenn die Anwendung im Hintergrund Audio abspielt.

ACCESS_FINE_LOCATION Ermöglicht den Zugriff auf eine GPS-Position.

Wenn Sie diese Berechtigung auswählen, kann die Anwendung mithilfe der Geolocation-Klasse auf GPS-Daten zugreifen.

DISABLE_KEYGUARD und WAKE_LOCK Deaktiviert den Energiesparmodus auf dem Gerät

Wenn Sie diese Berechtigung auswählen, kann mithilfe der Einstellungen der SystemIdleMode-Klasse verhindert werden, dass der Schlaf-Modus auf dem Gerät eingeschaltet wird.

CAMERA Ermöglicht den Zugriff auf eine Kamera

Wenn Sie diese Berechtigung auswählen, kann die Anwendung auf eine Kamera zugreifen.

RECORD_AUDIO Ermöglicht den Zugriff auf ein Mikrofon

Wenn Sie diese Berechtigung auswählen, kann die Anwendung auf ein Mikrofon zugreifen.

ACCESS_NETWORK_STATE und ACCESS_WIFI_STATE Ermöglicht den Zugriff auf Informationen zu Netzwerkschnittstellen, die mit dem Gerät verbunden sind.

Wenn Sie diese Berechtigung auswählen, kann die Anwendung mithilfe der NetworkInfo-Klasse auf Netzwerkinformationen zugreifen.

Weitere Informationen zum Festlegen von Einstellungen für Mobilanwendungen finden Sie in der Adobe AIR-Dokumentation.

Berechtigungen für die Apple iOS-Plattform

Die Apple iOS-Plattform arbeitet nicht mit vordefinierten Berechtigungen, sondern prüft die Berechtigungen zur Laufzeit. Das heißt, wenn eine Anwendung eine bestimmte Funktion von Apple iOS aufrufen möchte, für die eine Benutzerberechtigung erforderlich ist, wird die Berechtigung in einem Popup-Fenster angefordert.

Plattformeinstellungen auswählen

In den Plattformeinstellungen könenn Sie eine Zielgerätefamilie auswählen. Je nach der ausgewählten Plattform können Sie das Zielgerät oder eine Zielgerätefamilie auswählen. Sie können ein bestimmtes Gerät oder alle Geräte auswählen, die von der Plattform erkannt werden.

Mit einigen Plug-Ins von Drittanbietern lassen sich Flex- und ActionScript-Projekte auch auf anderen Plattformen bearbeiten. Zu plattformspezifischen Berechtigungen konsultieren Sie bitte die jeweilige Produktdokumentation.

Plattformeinstellungen für die Google Android-Plattform

Es gibt keine plattformspezifischen Einstellungen für die Google Android-Plattform.

Plattformeinstellungen für die Apple iOS-Plattform

Für Flex- oder ActionScript-Mobilprojekte können Sie folgende Zielgeräte für Apple iOS auswählen:

iPhone/iPod Touch Anwendungen für diese Zielfamilie werden im Apple App Store als kompatibel mit iPhone- oder iPod Touch-Geräten aufgeführt.

iPad Anwendungen für diese Zielfamilie werden im Apple App Store als kompatibel mit iPad-Geräten aufgeführt.

Alle Anwendungen für diese Zielfamilie werden im Apple App Store als kompatibel mit iPhone- oder iPod Touch-Geräten sowie iPad-Geräten aufgeführt. Dies ist die Standardeinstellung.

Anwendungseinstellungen auswählen

Automatisch neu ausrichten Dreht die Anwendung, wenn der Benutzer das Gerät dreht. Wenn diese Einstellung nicht aktiviert ist, wird die Anwendung immer mit einer festen Ausrichtung angezeigt.

Vollbild Zeigt die Anwendung im Vollbildmodus auf dem Gerät an. Wenn diese Einstellung aktiviert ist, wird die Statusleiste des Geräts nicht über der Anwendung angezeigt. Ihre Anwendung erstreckt sich über das gesamte Bild.

Wenn Ihre Anwendung auf diverse Gerätetypen mit unterschiedlichen Bildschirmdichten ausrichten möchten, wählen Sie „Anwendung automatisch für verschiedene Bildschirmdichten skalieren“. Mit dieser Option wird die Anwendung automatisch an die Auflösung des Gertäs angepasst. Siehe „Anwendungsskalierung festlegen“ auf Seite 18.

Anwendungsskalierung festlegen

Mit der Mobilanwendungsskalierung erstellen Sie eine einzige Mobilanwendung, die mit unterschiedlichen Geräten mit diversen Displaygrößen und Bildschirmauflösungen kompatibel ist.

Mobilgerätebildschirme weisen unterschiedliche Bildschirmauflösungen oder DPI-Werte (Dots per Inch, Punkte pro Zoll) auf. Je nach Auflösung des Zielgeräts können Sie einen DPI-Wert von 160, 240 oder 320 festlegen. Wenn Sie die automatische Skalierung aktivieren, optimiert Flex die Darstellung der Anwendung für die Bildschirmauflösung des jeweiligen Geräts.

Angenommen, Sie geben als DPI-Zielwert „160“ an und aktivieren die automatische Skalierung. Wenn Sie die Anwendung auf einem Gerät mit einem DPI-Wert von 320 ausführen, wird die Anwendung von Flex automatisch um den Faktor 2 skaliert. Das heißt, die Darstellung wird um 200 % vergrößert.

Um den DPI-Wert für das Zielgerät anzugeben, legen Sie ihn in der Hauptanwendungsdatei als `applicationDPI`-Eigenschaft des `<s:ViewNavigatorApplication>`-Tags oder des `<s:TabbedViewNavigatorApplication>`-Tags fest:

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

Wenn Ihre Anwendung nicht automatisch skaliert werden soll, müssen Sie die Auflösungsänderungen für das Layout bei Bedarf manuell vornehmen. Flex passt die Skins jedoch an die Auflösung des jeweiligen Geräts an.

Weitere Informationen zum Erstellen auflösungsunabhängiger Mobilanwendungen finden Sie unter „Unterstützung mehrerer Bildschirmgrößen und DPI-Werte in Mobilanwendungen“ auf Seite 132.

Google Android-Geräte anschließen

Sie können ein Google Android-Gerät an Ihren Entwicklungscomputer anschließen, um die Anwendung auf dem Android-Gerät zu testen oder zu debuggen.

Unterstützte Android-Geräte

Für Flex- und ActionScript-Mobilprojekte ist AIR 2.6 oder höher erforderlich. Sie können Mobilprojekte nur auf Geräten ausführen und debuggen, die AIR 2.6 oder höher unterstützen.

Sie können AIR 2.6 auf unterstützten Android-Geräten installieren, auf denen Android 2.2 oder höher ausgeführt wird. Eine Liste der unterstützten Geräte finden Sie unter http://www.adobe.com/flashplatform/certified_devices/. Überprüfen Sie außerdem die Mindestsystemanforderungen zum Ausführen von Adobe AIR auf Android-Geräten unter Systemanforderungen für Mobilgeräte.

Android-Geräte konfigurieren

Zum Ausführen und Debuggen von Flex-Mobilanwendungen auf einem Android-Gerät muss USB-Debugging aktiviert sein:

1. Vergewissern Sie sich wie folgt, dass auf dem Gerät USB-Debugging aktiviert ist:
   a. Tippen Sie die Starttaste an, um den Startbildschirm aufzurufen.
   b. Wählen Sie „Einstellungen“ > „Anwendungen“ > „Entwicklung“.
   c. Aktivieren Sie „USB-Debugging“.
2. Schließen Sie das Gerät über ein USB-Kabel am Computer an.
3. Ziehen Sie den Benachrichtigungsbereich am oberen Bildschirmrand nach unten. Es wird „USB verbunden“ oder „USB-Verbindung“ angezeigt.
   a. Tippen Sie „USB verbunden“ oder „USB-Verbindung“ an.
   b. Wenn eine Reihe Optionen angezeigt wird, darunter „Reiner Lademodus“, wählen Sie „Reiner Lademodus“ und tippen Sie auf „OK“.
   c. Wird eine Schaltfläche zum Ausschalten des Massenspeichermodus angezeigt, klicken Sie auf die Schaltfläche, um die Massenspeicherung zu deaktivieren.
4. (Nur Windows) Installieren Sie den entsprechenden USB-Treiber für Ihr Gerät. Siehe „USB-Gerätetreiber für Android-Geräte (Windows) installieren“ auf Seite 19.
5. Ziehen Sie den Benachrichtigungsbereich am oberen Bildschirmrand nach unten.

   Wenn USB-Debugging nicht als Eintrag angezeigt wird, überprüfen Sie den USB-Modus (siehe Schritt 3). Stellen Sie sicher, dass der USB-Modus nicht auf PC-Modus gesetzt ist.

Hinweis: *Für das Debuggen sind noch weitere Konfigurationseinstellungen erforderlich. Siehe „Mobilanwendungen auf einem Gerät testen und debuggen“ auf Seite 188.*

USB-Gerätetreiber für Android-Geräte (Windows) installieren

Gerätetreiber und -konfigurationen

Unter Windows muss ein USB-Treiber installiert werden, damit ein Android-Gerät an den Computer angeschlossen werden kann. Flash Builder verfügt über Gerätetreiber und Konfigurationen für diverse Android-Geräte.

Diese Gerätetreiberkonfigurationen sind in der Datei `android_winusb.inf` aufgeführt. Bei der Installation des Gerätetreibers greift der Windows Geräte-Manager auf diese Datei zu. Flash Builder installiert `android_winusb.inf` an folgender Stelle:

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

Eine vollständige Liste der unterstützten Geräte finden Sie unter Certified devices. Bei Android-Geräten, die nicht aufgeführt sind, können Sie die Datei `android_winusb.inf` mit USB-Treibern aktualisieren. Siehe „Weitere Android USB-Gerätetreiberkonfigurationen hinzufügen“ auf Seite 20.

USB-Gerätetreiber installieren

1 Schließen Sie das Android-Gerät an den USB-Port des Computers an.

2 Wechseln Sie zum folgenden Speicherort:

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

Installieren Sie den USB-Treiber entweder mit dem Windows-Assistenten „Neue Hardware gefunden“ oder mit dem Geräte-Manager von Windows.

Wichtig: *Wenn Windows das Gerät weiterhin nicht erkennt, müssen Sie den entsprechenden USB-Treiber des Geräteherstellers installieren. Unter OEM USB Drivers finden Sie Links zu den Websites einiger Gerätehersteller, von denen Sie den entsprechenden USB-Treiber für Ihr Gerät herunterladen können.*

Weitere Android USB-Gerätetreiberkonfigurationen hinzufügen

Falls Ihr Android-Gerät nicht unter „USB-Gerätetreiber für Android-Geräte (Windows) installieren“ auf Seite 19 aufgeführt ist, können Sie die Datei `android_winusb.inf` um einen Eintrag für Ihr Gerät erweitern.

1 Schließen Sie das Gerät an einen USB-Port am Computer an. Windows teilt Ihnen mit, dass der Treiber nicht gefunden werden kann.

2 Öffnen Sie im Geräte-Manager von Windows in den Geräteeigenschaften die Registerkarte „Details“.

3 Wählen Sie die Eigenschaft „Hardwarekennungen“, um die Hardware-ID anzuzeigen.

4 Öffnen Sie `android_winusb.inf` in einem Text-Editor. Suchen Sie die Datei `android_winusb.inf` im folgenden Speicherort:

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

5 Suchen Sie in der Datei nach Auflistungen für die passende Architektur, entweder: `[Google.NTx86]` oder `[Google.NTamd64]`. Die Auflistungen enthalten wie im Folgenden dargestellt jeweils einen beschreibenden Kommentar und eine oder mehrere Zeilen mit Hardware-IDs:

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

6 Kopieren Sie einen Kommentar mit Hardware-Auflistung heraus. Bearbeiten Sie den gewünschten Gerätetreiber wie folgt:

a Geben Sie im Kommentar den Namen des Geräts ein.

b Ersetzen Sie die Hardware-ID durch die Hardware-ID, die Sie in Schritt 3 ermittelt haben.

Beispiel:

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

7 Installieren Sie das Gerät mithilfe des Windows Geräte-Managers. Dies wird weiter oben unter „USB-Gerätetreiber für Android-Geräte (Windows) installieren“ auf Seite 19 beschrieben.

Bei der Installation wird von Windows eine Warnung angezeigt, dass der Treiber von einem unbekannten Hersteller stammt. Der Treiber gestattet Flash Builder jedoch den Zugriff auf das Gerät.

Apple iOS-Entwicklungsprozess mit Flash Builder

Vor der Entwicklung einer iOS-Anwendung mit Flash Builder ist es wichtig, den iOS-Entwicklungsprozess zu verstehen und wie die erforderlichen Zertifikate von Apple erworben werden können.

Überblick über iOS-Entwicklungs- und -Bereitstellungsprozesse

Diese Tabelle stellt eine Liste von Schritten im iOS-Entwicklungsprozess bereit. Darin sind Schritte zum Erhalten der erforderlichen Zertifikate sowie Voraussetzungen für jeden Schritt enthalten.

Ausführliche Informationen zu den einzelnen Schritten finden Sie unter „Erstellen, Debuggen oder Bereitstellen einer iOS-Anwendung vorbereiten“ auf Seite 22.

Schrittnr.	Schritt	Pfad	Voraussetzungen
1.	Nehmen Sie am Apple-Entwicklungsprogramm teil.	Apple Developer-Website	none
2.	Registrieren Sie die Geräte-ID (UDID) Ihres iOS-Geräts.	iOS Provisioning Portal	Entwickler-ID von Apple (Schritt 1)
3.	Generieren Sie eine Datei für die Zertifikatsignaturanforderung (*.certSigningRequest, CSR).	• Verwenden Sie unter Mac OS das Programm „Schlüsselbund“. • Verwenden Sie unter Windows OpenSSL.	none
4.	Generieren Sie ein iOS-Entwickler-/Verteilungszertifikat (*.cer).	iOS Provisioning Portal	• Entwickler-ID von Apple (Schritt 1) • CSR-Datei (Schritt 3)
5.	Konvertieren Sie das iOS-Entwickler-/Verteilungszertifikat in ein P12-Format.	• Verwenden Sie unter Mac OS das Programm „Schlüsselbund“. • Verwenden Sie unter Windows OpenSSL.	• Entwickler-ID von Apple (Schritt 1) • iOS-Entwickler-/Verteilungszertifikat (Schritt 4)
6.	Generieren Sie die Anwendungs-ID.	iOS Provisioning Portal	Entwickler-ID von Apple (Schritt 1)
7.	Erstellen Sie ein Bereitstellungsprofil (*.mobileprovision).	iOS Provisioning Portal	• Entwickler-ID von Apple (Schritt 1) • UDID Ihres iOS-Geräts (Schritt 2) • Anwendungs-ID (Schritt 6)
8.	Erstellen Sie die Anwendung.	Flash Builder	• Entwickler-ID von Apple (Schritt 1) • P12-Entwickler-/Verteilungszertifikat (Schritt 5) • Anwendungs-ID (Schritt 6)
9.	Stellen Sie die Anwendung bereit.	iTunes	• Bereitstellungsprofil (Schritt 7) • Anwendungspaket (Schritt 8)

Erstellen, Debuggen oder Bereitstellen einer iOS-Anwendung vorbereiten

Bevor Sie eine iOS-Anwendung mit Flash Builder erstellen und sie auf einem iOS-Gerät bereitstellen oder an den Apple App Store senden, führen Sie folgende Schritte durch:

1 Schließen Sie sich dem Apple iOS-Entwicklerprogramm hier an: Apple iOS Developer Program.

Sie können sich mit Ihrer Apple-ID anmelden oder eine Apple-ID erstellen. Die Apple-Entwicklerregistrierung führt Sie durch die erforderlichen Schritte.

2 Registrieren Sie die Geräte-ID (UDID) des Geräts.

Dieser Schritt trifft nur zu, wenn Sie Ihre Anwendung auf einem iOS-Gerät und nicht im Apple App Store bereitstellen. Wenn Sie Ihre Anwendung auf verschiedenen iOS-Geräten bereitstellen möchten, registrieren Sie die Geräte-ID jedes Geräts.

Rufen Sie die Geräte-ID Ihres iOS-Geräts ab.

a Verbinden Sie das iOS-Gerät mit Ihrem Entwicklungscomputer und starten Sie iTunes. Das verbundene iOS-Gerät wird in iTunes im Abschnitt „Geräte“ angezeigt.

b Klicken Sie auf den Gerätenamen, um eine Zusammenfassung des iOS-Geräts anzuzeigen.

c Klicken Sie auf der Registerkarte „Übersicht“ auf die Seriennummer, um die 40 Zeichen lange Geräte-ID des iOS-Geräts anzuzeigen.

Sie können die Geräte-ID von iTunes mithilfe der Tastenkombination Strg+C (Windows) oder Cmd+C (Mac) kopieren.

Registrieren Sie die Geräte-ID Ihres Geräts

Melden Sie sich beim iOS Provisioning Portal mit Ihrer Apple-ID an und registrieren Sie die Geräte-ID.

3 Generieren Sie eine Datei für die Zertifikatsignaturanforderung (*.certSigningRequest, CSR).

Sie generieren eine CSR, um ein iOS-Entwickler-/Verteilungszertifikat zu erhalten. Sie können eine CSR mithilfe der Funktion „Schlüsselbund“ unter Mac oder OpenSSL unter Windows generieren. Beim Generieren einer CSR geben Sie nur Ihren Benutzernamen und Ihre E-Mail-Adresse an. Sie stellen keine Informationen zu Ihrer Anwendung oder Ihrem Gerät bereit.

Das Generieren einer CSR erstellt einen öffentlichen Schlüssel und einen privaten Schlüssel sowie eine Datei „*.certSigningRequest“. Der öffentliche Schlüssel ist in der CSR enthalten und der private Schlüssel wird zum Signieren der Anforderung verwendet.

Weitere Informationen zum Generieren einer CSR finden Sie unter Generating a certificate signing request.

4 Erstellen Sie nach Bedarf ein iOS-Entwicklerzertifikat oder ein iOS-Verteilungszertifikat (*.cer).

Hinweis: *Um eine Anwendung auf einem Gerät bereitzustellen, benötigen Sie ein Entwicklerzertifikat. Um die Anwendung im Apple App Store bereitzustellen, benötigen Sie ein Verteilungszertifikat.*

Generieren Sie ein iOS-Entwicklerzertifikat

a Melden Sie sich beim iOS Provisioning Portal mit Ihrer Apple-ID an und wählen Sie die Geräte-Entwickler-Registerkarte.

b Klicken Sie auf die Option zum Anfordern eines Zertifikats und durchsuchen Sie die von Ihnen generierte und auf Ihrem Computer gespeicherte CSR-Datei (Schritt 3).

c Wählen Sie die CSR-Datei und klicken Sie auf „Senden“.

d Klicken Sie auf der Seite „Zertifikate“ auf die Option zum Herunterladen.

e Speichern Sie die heruntergeladene Datei (*.developer_identity.cer).

Generieren Sie ein iOS-Verteilungszertifikat

f Melden Sie sich beim iOS Provisioning Portal mit Ihrer Apple-ID an und wählen Sie die Geräte-Verteilungsregisterkarte.

g Klicken Sie auf die Option zum Anfordern eines Zertifikats und durchsuchen Sie die von Ihnen generierte und auf Ihrem Computer gespeicherte CSR-Datei (Schritt 3).

h Wählen Sie die CSR-Datei und klicken Sie auf „Senden“.

i Klicken Sie auf der Seite „Zertifikate“ auf die Option zum Herunterladen.

j Speichern Sie die heruntergeladene Datei (*.distribution_identity.cer).

5 Konvertieren Sie das iOS-Entwicklerzertifikat oder das iOS-Verteilungszertifikat in ein P12-Dateiformat (*.p12).

Sie konvertieren das iOS-Entwickler- oder das iOS-Verteilungszertifikat in ein P12-Format, damit Flash Builder Ihre iOS-Anwendung digital signieren kann. Das Konvertieren in ein P12-Format kombiniert das iOS-Entwickler-/Verteilungszertifikat und den dazugehörigen privaten Schlüssel in eine Einzeldatei.

Hinweis: *Wenn Sie die Anwendung auf dem Desktop mithilfe des AIR Debug Launcher (ADL) testen, müssen Sie das iOS-Entwickler-/Verteilungszertifikat nicht in ein P12-Format konvertieren.*

Verwenden Sie die Option „Schlüsselbund“ unter Mac oder Open SSL unter Windows, um eine Personal Information Exchange-Datei (*.p12) zu erstellen. Weitere Informationen finden Sie unter Konvertieren eines Entwicklerzertifikats in eine P12-Datei.

6 Generieren Sie die Anwendungs-ID, indem Sie folgende Schritte durchführen:

a Melden Sie sich beim iOS Provisioning Portal mit Ihrer Apple-ID an.

b Navigieren Sie zur App-ID-Seite und klicken Sie auf die Option für eine neue App-ID.

c Geben Sie auf der Registerkarte „Verwalten“ eine Beschreibung für Ihre Anwendung ein, generieren Sie eine neue Bundle-Seed-ID und geben Sie einen Bundle-Bezeichner ein.

Jede Anwendung weist eine eindeutige Anwendungs-ID auf, die Sie in der Anwendungsbeschreibungs-XML-Datei angeben. Eine Anwendungs-ID besteht aus einer von Apple bereitgestellten Bundle-Seed-ID aus 10 Zeichen und einem Bundle-Bezeichner-Suffix, das Sie festlegen. Der von Ihnen festgelegte Bundle-Bezeichner muss mit der Anwendungs-ID in der Deskriptordatei für die Anwendung übereinstimmen. Beispiel: Wenn Ihre Anwendungs-ID com.meineDomain.* lautet, muss die ID in der Deskriptordatei für die Anwendung mit com.myDomain beginnen.

Wichtig: *Platzhalter-Bundle-Bezeichner sind für das Entwickeln und Testen von iOS-Anwendungen nützlich, können aber nicht für die Bereitstellung von Anwendungen im Apple App Store verwendet werden.*

7 Generieren Sie eine Entwickler-Bereitstellungsprofildatei oder eine Verteilungs-Bereitstellungsprofildatei (*.mobileprovision).

Hinweis: *Um eine Anwendung auf einem Gerät bereitzustellen, benötigen Sie ein Entwickler-Bereitstellungsprofil. Um die Anwendung im Apple App Store bereitzustellen, benötigen Sie ein Verteilungs-Bereitstellungsprofil. Sie verwenden ein Verteilungs-Bereitstellungsprofil, um Ihre Anwendung zu signieren.*

Entwickler-Bereitstellungsprofil erstellen

a Melden Sie sich beim iOS Provisioning Portal mit Ihrer Apple-ID an.

b Navigieren Sie zu „Zertifikat“ > „Provisioning“ und klicken Sie auf „Neues Profil“.

c Geben Sie einen Profilnamen ein, wählen Sie das iOS-Entwicklerzertifikat, die App-ID und die Geräte-IDs, auf denen Sie die Anwendung installieren möchten.

d Klicken Sie auf „Senden“.

e Laden Sie die generierte Entwickler-Bereitstellungsprofildatei (*.mobileprovision) herunter und speichern Sie sie auf Ihrem Computer.

Verteilungs-Bereitstellungsprofil erstellen

f Melden Sie sich beim iOS Provisioning Portal mit Ihrer Apple-ID an.

g Navigieren Sie zu „Zertifikat“ > „Provisioning“ und klicken Sie auf „Neues Profil“.

h Geben Sie einen Profilnamen ein, wählen Sie das iOS-Verteilungszertifikat und die App-ID. Wenn Sie die Anwendung vor dem Bereitstellen testen möchten, geben Sie die Geräte-IDs der Geräte ein, auf denen Sie testen möchten.

i Klicken Sie auf „Senden“.

j Laden Sie die generierte Bereitstellungsprofildatei (*.mobileprovision) herunter und speichern Sie sie auf Ihrem Computer.

Verwandte Themen

„iOS-Anwendung in Flash Builder erstellen“ auf Seite 11

Auszuwählende Dateien beim Testen, Debuggen oder Installieren einer iOS-Anwendung

Wählen Sie folgende Dateien im Konfigurationsdialogfeld „Ausführen/Debug“, um eine Anwendung zum Testen auf einem iOS-Gerät auszuführen, zu debuggen oder zu installieren:

- iOS-Entwicklerzertifikat in P12-Format (Schritt 5)
- XML-Deskriptordatei für die Anwendung, die die Anwendungs-ID enthält (Schritt 6)
- Entwickler-Bereitstellungsprofil (Schritt 7)

Weitere Informationen finden Sie unter „Anwendung auf einem Apple iOS-Gerät debuggen“ auf Seite 191 und „Anwendung auf einem Apple iOS-Gerät installieren“ auf Seite 193.

Auszuwählende Dateien beim Bereitstellen einer Anwendung im Apple App Store

Um eine Anwendung im Apple App Store bereitzustellen, wählen Sie den Pakettyp im Dialogfeld „Releasebuild exportieren“ als endgültiges Releasebuildpaket für den Apple App Store und wählen Sie folgende Dateien:

- iOS-Verteilungszertifikat in P12-Format (Schritt 5)
- XML-Deskriptordatei für die Anwendung, die die Anwendungs-ID enthält (Schritt 6).

  Hinweis: *Sie können beim Senden einer Anwendung an den Apple App Store keine Platzhalter-Anwendungs-ID verwenden.*

- Verteilungs-Bereitstellungsprofil (Schritt 7)

Weitere Informationen finden Sie unter „Apple iOS-Pakete für die Veröffentlichung exportieren“ auf Seite 196.

Kapitel 3: Benutzeroberfläche und Layout

Layout einer einfachen Mobilanwendung

Layout einer Mobilanwendung mithilfe von Ansichten und Abschnitten festlegen

Mobilanwendungen bestehen aus einem oder mehreren Bildschirmen, auch als *Ansichten* bezeichnet. Beispielsweise könnten Mobilanwendungen aus den folgenden drei Ansichten bestehen:

1. Eine Startansicht zum Hinzufügen von Kontaktinformationen
2. Eine Kontaktansicht mit einer Liste vorhandener Kontakte
3. Eine Suchansicht zum Durchsuchen der Kontaktliste

Einfache Mobilanwendung

Die folgende Abbildung zeigt den Hauptbildschirm einer in Flex erstellten einfachen Mobilanwendung:

***A.** ActionBar-Steuerelement **B.** Inhaltsbereich*

Diese Abbildung zeigt die Hauptbereiche einer Mobilanwendung:

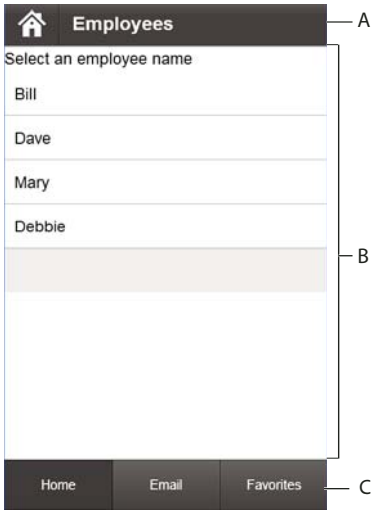
ActionBar-Steuerelement Mit dem ActionBar-Steuerelement können Sie Kontextinformationen zum aktuellen Status der Anwendung anzeigen. Zu diesen Informationen gehört ein Titelbereich, ein Bereich für Navigationssteuerelemente und ein Bereich für Aktionssteuerelemente. Im ActionBar-Steuerelement können Sie globale Inhalte hinzufügen, die für die gesamte Anwendung gelten, sowie Elemente, die nur für eine bestimmte Ansicht gelten.

Inhaltsbereich Im Inhaltsbereich werden die einzelnen Bildschirme oder *Ansichten* angezeigt, aus denen die Anwendung besteht. Benutzer navigieren in den Ansichten der Anwendung über die in die Anwendung integrierten Komponenten sowie die Eingabesteuerelemente des Mobilgeräts.

In Abschnitte gegliederte Mobilanwendung

In komplexeren Anwendungen können mehrere Bereiche oder *Abschnitte* der Anwendung definiert sein. Beispielsweise kann eine Anwendung Abschnitte für Kontakte, E-Mails, Favoriten und weitere enthalten. Jeder Abschnitt der Anwendung enthält eine oder mehrere Ansichten. Einzelne Ansichten können in mehreren Abschnitten verfügbar gemacht werden, sodass Sie die Ansichten nicht mehrmals definieren müssen.

Die nächste Abbildung zeigt eine Mobilanwendung mit einer Registerkartenleiste am unteren Rand des Anwendungsfensters:



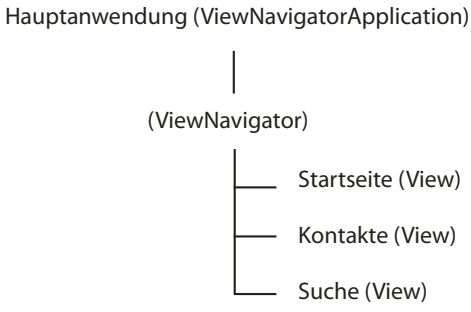
***A.** ActionBar-Steuerelement **B.** Inhaltsbereich **C.** Registerkartenleiste*

Flex verwendet das ButtonBarBase-Steuerelement zum Implementieren der Registerkartenleiste. Jede Schaltfläche der Registerkartenleiste entspricht einem anderen Abschnitt. Wählen Sie in der Registerkartenleiste eine Schaltfläche aus, um den aktuellen Abschnitt zu wechseln.

Für jeden Abschnitt der Anwendung wird eine eigene ActionBar definiert. Daher gilt die Registerkartenleiste global für die gesamte Anwendung, die ActionBar für jeweils einen Abschnitt.

Layout einer einfachen Mobilanwendung

Die folgende Abbildung zeigt die Architektur einer einfachen Mobilanwendung:



In der Abbildung ist eine Anwendung bestehend aus vier Dateien dargestellt. Eine Mobilanwendung enthält eine Anwendungs-Hauptdatei sowie jeweils eine Datei pro Ansicht. Es gibt keine separate Datei für ViewNavigator; sie wird vom ViewNavigatorApplication-Container erstellt.

Hinweis: *Zwar zeigt dieses Diagramm die Anwendungsarchitektur, doch stellt es nicht die Anwendung zur Laufzeit dar. Zur Laufzeit ist nur eine Ansicht aktiv und im Speicher vorhanden. Weitere Informationen finden Sie unter „In den Ansichten einer Mobilanwendung navigieren“ auf Seite 29.*

Klassen in einer Mobilanwendung

Sie definieren eine Mobilanwendung über die folgenden Klassen:

Klasse	Beschreibung
ViewNavigatorApplication	Definiert die Hauptanwendungsdatei. Der ViewNavigatorApplication-Container akzeptiert keine untergeordneten Elemente.
ViewNavigator	Steuert die Navigation in den Ansichten einer Anwendung. Der ViewNavigator erstellt zudem das ActionBar-Steuerelement. Der ViewNavigatorApplication-Container erstellt automatisch einen einzelnen ViewNavigator-Container für die gesamte Anwendung. Mit den Methoden des ViewNavigator-Containers können Sie zwischen den einzelnen Ansichten wechseln.
View	Definiert die Ansichten der Anwendung, wobei jede Ansicht in einer eigenen MXML- oder ActionScript-Datei definiert wird. Jede Ansicht der Anwendung wird von einer Instanz des View-Containers dargestellt. Definieren Sie jede Ansicht in einer eigenen MXML- oder ActionScript-Datei.

Im ViewNavigatorApplication-Container definieren Sie wie im folgenden Beispiel dargestellt die Hauptanwendungsdatei:

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

Der ViewNavigatorApplication-Container erstellt automatisch ein einzelnes ViewNavigator-Objekt, das die ActionBar definiert. Mit dem ViewNavigator navigieren Sie in den Ansichten der Anwendung.

View-Container zu einer Mobilanwendung hinzufügen

Jede Mobilanwendung besitzt mindestens eine Ansicht. Von der Hauptanwendungsdatei wird zwar der ViewNavigator erstellt, jedoch keine in der Anwendung verwendete Ansicht definiert.

Jede Ansicht in einer Anwendung entspricht einem View-Container in einer ActionScript- oder MXML-Datei. Jede Ansicht enthält eine `data`-Eigenschaft, die die Daten für die betreffende Ansicht angibt. In den Ansichten kann die `data`-Eigenschaft beim Navigieren des Benutzers in der Anwendung zum Übergeben von Informationen verwendet werden.

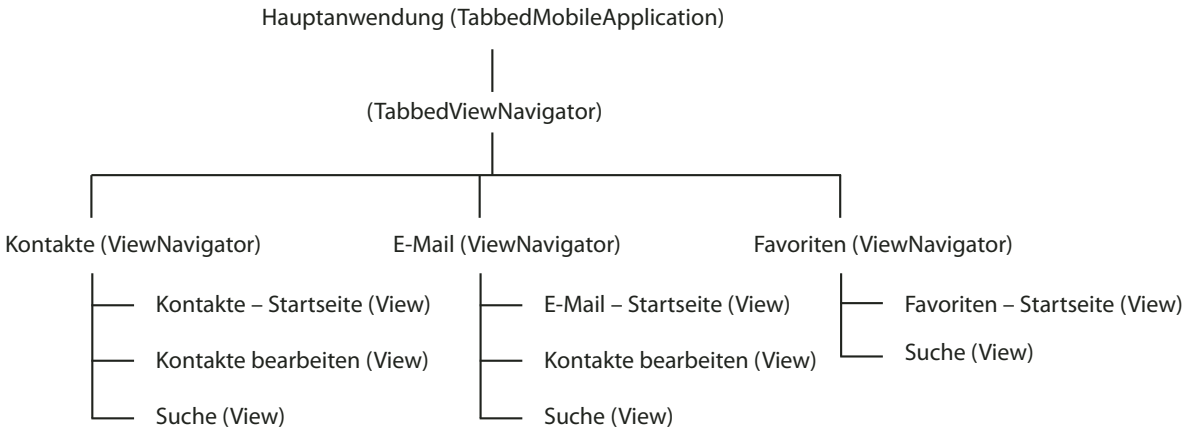
Über die `ViewNavigatorApplication.firstView`-Eigenschaft geben Sie die Datei an, die die erste Ansicht in der Anwendung definiert. In der vorherigen Anwendung gibt die `firstView`-Eigenschaft die Ansicht `views.HomeView` an. Das folgende Beispiel veranschaulicht die Datei „HomeView.mxml“, in der diese Ansicht definiert ist:

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

Blogger David Hassoun schrieb einen Artikel über ViewNavigator-Grundlagen.

Layout einer Mobilanwendung mit mehreren Abschnitten

In Mobilanwendungen können zusammengehörige Ansichten zu unterschiedlichen Anwendungsabschnitten zusammengefasst werden. Beispielsweise zeigt die folgende Abbildung die Anordnung einer Mobilanwendung mit drei Abschnitten.



Jede Ansicht kann in jedem Abschnitt verwendet werden. Anders gesagt, gehört keine Ansicht zu einem bestimmten Abschnitt. Der Abschnitt gibt lediglich eine Anordnungs- und Navigationsweise für eine Gruppe von Ansichten an. In der Abbildung ist die Suchansicht Teil jedes Anwendungsabschnitts.

Zur Laufzeit ist nur eine Ansicht aktiv und im Speicher vorhanden. Weitere Informationen finden Sie unter „In den Ansichten einer Mobilanwendung navigieren“ auf Seite 29.

Klassen in einer Mobilanwendung mit mehreren Bereichen

In der folgenden Tabelle finden Sie die Klassen, mit denen Sie eine Mobilanwendung mit mehreren Abschnitten erstellen können:

Klasse	Beschreibung
TabbedViewNavigatorApplication	Definiert die Hauptanwendungsdatei. Für den TabbedViewNavigatorApplication-Container ist nur ViewNavigator als untergeordnetes Element zulässig. Definieren Sie für jeden Abschnitt der Anwendung einen ViewNavigator.
TabbedViewNavigator	Steuert die Navigation zwischen den Abschnitten, aus denen die Anwendung besteht. Der TabbedViewNavigatorApplication-Container erstellt automatisch einen einzelnen TabbedViewNavigator-Container für die gesamte Anwendung. Der TabbedViewNavigator-Container erstellt die Registerkartenleiste, mit der Sie zwischen den Abschnitten navigieren.
ViewNavigator	Definieren Sie für jeden Abschnitt einen ViewNavigator-Container. Der ViewNavigator steuert die Navigation zwischen den Ansichten, aus denen der Abschnitt besteht. Außerdem wird von diesem das ActionBar-Steuerelement für den Abschnitt erstellt.
View	Definiert die Ansichten der Anwendung. Jede Ansicht der Anwendung wird von einer Instanz des View-Containers dargestellt. Definieren Sie jede Ansicht in einer eigenen MXML- oder ActionScript-Datei.

Eine Mobilanwendung mit Abschnitten enthält eine Hauptanwendungsdatei sowie jeweils eine Definitionsdatei pro Ansicht. Im TabbedViewNavigatorApplication-Container definieren Sie wie im folgenden Beispiel dargestellt die Hauptanwendungsdatei:

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

Verwenden von ViewNavigator in einer Anwendung mit mehreren Abschnitten

Für den TabbedViewNavigatorApplication-Container ist nur ViewNavigator als untergeordnete Komponente zulässig. Jeder Abschnitt der Anwendung entspricht einem anderen ViewNavigator-Container.

Mit dem ViewNavigator-Container navigieren Sie in den Ansichten jedes Abschnitts und definieren jeweils das ActionBar-Steuerelement für diesen. Über die `ViewNavigator.firstView`-Eigenschaft geben Sie die Datei an, die die erste Ansicht im Abschnitt definiert.

Verwenden von TabbedViewNavigator in einer Anwendung mit mehreren Abschnitten

Der TabbedViewNavigatorApplication-Container erstellt automatisch einen einzelnen Container vom Typ TabbedViewNavigator. Der TabbedViewNavigator-Container erstellt dann eine Registerkartenleiste am unteren Rand der Anwendung. Sie müssen der Anwendung keine Logik hinzufügen, um zwischen den Abschnitten navigieren zu können.

In den Ansichten einer Mobilanwendung navigieren

Ein Stapel von View-Objekten steuert die Navigation in Mobilanwendungen. Das oberste View-Objekt des Stapels definiert die derzeit sichtbare Ansicht.

Der Stapel wird vom ViewNavigator-Container verwaltet. Zum Wechseln der Ansicht legen Sie das neue View-Objekt auf dem Stapel ab (Push) oder entfernen das aktuelle View-Objekt aus dem Stapel (Pop). Wenn Sie das derzeit sichtbare View-Objekt aus dem Stapel entfernen, wird dieses View-Objekt zerstört, und der Benutzer kehrt zur vorherigen Ansicht auf dem Stapel zurück.

In Anwendungen mit Abschnitten navigieren Sie mithilfe der Registerkartenleiste zwischen den einzelnen Abschnitten. Da jeder Abschnitt von einem eigenen ViewNavigator definiert wird, entsprechen Abschnittswechsel einem Wechsel des aktuellen ViewNavigator-Elements und des Stapels. Das oberste View-Objekt des Stapels für den neuen ViewNavigator wird zur aktuellen Ansicht.

Um Arbeitsspeicher zu sparen, stellt der ViewNavigator standardmäßig sicher, dass sich stets nur eine Ansicht im Speicher befindet. Die Daten für vorherige Ansichten auf dem Stapel werden jedoch beibehalten. Wenn der Benutzer zurück zur vorherigen Ansicht navigiert, kann die Ansicht anhand der richtigen Daten erneut instanziiert werden.

Hinweis: *Der View-Container definiert die* `destructionPolicy`*-Eigenschaft. Bei Festlegung auf den Standardwert* `auto` *zerstört der ViewNavigator die Ansicht, wenn sie nicht aktiv ist. Bei Festlegung auf* `none` *wird die Ansicht im Arbeitsspeicher abgelegt.*

Blogger Mark Lochrie hat einen Artikel über ViewNavigator verfasst.

ViewNavigator-Navigationsmethoden

Die Navigation steuern Sie mit den folgenden Methoden der ViewNavigator-Klasse:

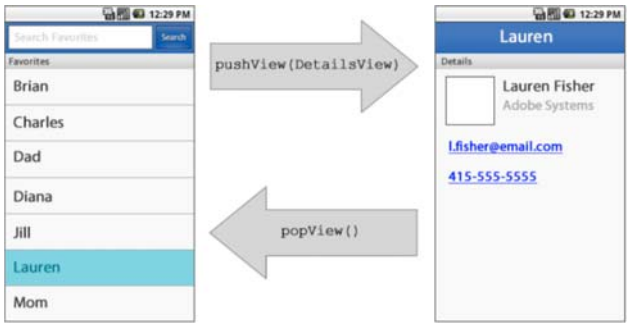
pushView() Legt ein View-Objekt auf dem Stapel ab. Das als Argument an die `pushView()`-Methode übergebene View-Objekt wird zur aktuellen Ansicht.

popView() Entfernt das aktuelle View-Objekt vom Navigationsstapel und zerstört das View-Objekt. Das vorherige View-Objekt auf dem Stapel wird zur aktuellen Ansicht.

popToFirstView() Entfernt alle View-Objekte aus dem Stapel und zerstört sie, mit Ausnahme des ersten View-Objekts auf dem Stapel. Das erste View-Objekt auf dem Stapel wird zur aktuellen Ansicht.

popAll() Leert den Stapel des ViewNavigator und zerstört alle View-Objekte. Die Ansicht der Anwendung ist leer.

Die folgende Abbildung zeigt zwei Ansichten. Um die aktuelle Ansicht zu wechseln, legen Sie mit der `ViewNavigator.pushView()`-Methode ein View-Objekt auf dem Stapel ab, das die neue Ansicht darstellt. Die `pushView()`-Methode führt dazu, dass der ViewNavigator für die Ansicht das neue View-Objekt verwendet.



Ablegen (Push) und Entfernen (Pop) von View-Objekten zum Wechseln der Ansicht.

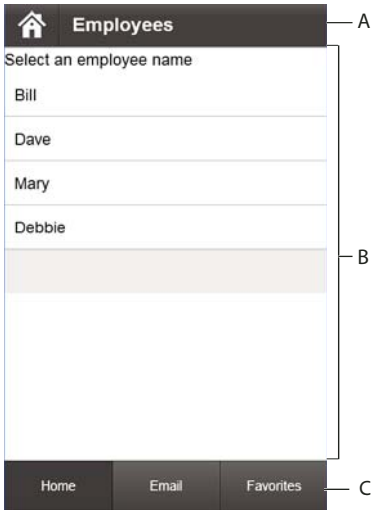
Mit der `ViewNavigator.popView()`-Methode entfernen Sie das aktuelle View-Objekt vom Stapel. Der ViewNavigator wechselt die Ansicht auf das vorherige View-Objekt auf dem Stapel zurück.

Hinweis: *Das Mobilgerät selbst steuert einen Großteil der Navigation in einer Mobilanwendung. Beispielsweise bedienen in Flex erstellte Mobilanwendungen automatisch die Zurück-Taste an Mobilgeräten. Daher müssen Sie der Anwendung keine Unterstützung für die Zurück-Taste hinzufügen. Wenn der Benutzer die Zurück-Taste des Mobilgeräts drückt, ruft Flex automatisch die* `popView()`*-Methode zum Wiederherstellen der vorherigen Ansicht auf.*

Blogger David Hassoun schrieb einen Artikel über Datenverwaltung in einer Ansicht.

Navigation für eine Anwendung mit mehreren Abschnitten erstellen

In der folgenden Abbildung sind die Ansichten in mehreren Abschnitten angeordnet. Jeder Abschnitt wird durch einen anderen ViewNavigator-Container definiert. Jeder Abschnitt enthält eine oder mehrere Ansichten:



***A.** ActionBar-Steuerelement **B.** Inhaltsbereich **C.** Registerkartenleiste*

Die Ansicht im aktuellen Abschnitt wechseln Sie mit der `pushView()`-Methode und der `popView()`-Methode des aktuellen ViewNavigator.

Den aktuellen Abschnitt wechseln Sie mithilfe der Registerkartenleiste. Wenn Sie Abschnitte wechseln, wechseln Sie in den ViewNavigator-Container des neuen Abschnitts. Auf dem Bildschirm wird nun das View-Objekt angezeigt, das sich derzeit oben auf Stapel des neuen ViewNavigator befindet.

Mithilfe der `TabbedViewNavigator.selectedIndex`-Eigenschaft können Sie den Abschnitt auch programmgesteuert wechseln. Diese Eigenschaft enthält den auf 0 basierenden Index des ausgewählten Ansichtsnavigators.

Benutzereingabe in einer Mobilanwendung verarbeiten

Die Benutzereingabe erfordert bei einer Mobilanwendung im Vergleich zu einer Desktop- oder Browseranwendung eine unterschiedliche Vorgehensweise. In für AIR erstellten Desktopanwendungen und in für Flash Player erstellten Browseranwendungen bestehen die hauptsächlichen Eingabegeräte aus einer Maus und einer Tastatur. Bei Mobilgeräten besteht das hauptsächliche Eingabegerät im Touchscreen. Häufig ist bei einem Mobilgerät eine Art Tastatur vorhanden, manche Geräte bieten zudem eine Fünf-Richtungs-Eingabemethode (nach links, nach rechts, nach oben, nach unten und auswählen).

Die mx.core.UIComponent-Klasse definiert die `interactionMode`-Stileigenschaft, über die Sie Komponenten für den in der Anwendung verwendeten Eingabetyp konfigurieren. Beim Halo- und beim Spark-Design ist der Standardwert `mouse`, d. h., die Maus ist als hauptsächliches Eingabegerät festgelegt. Beim Mobildesign ist der Standardwert `touch`, d. h., als primäres Eingabegerät ist der Touchscreen festgelegt.

Unterstützen von Hardwaretasten in einer Mobilanwendung

Anwendungen, die in den ViewNavigatorApplication- oder TabbedViewNavigatorApplication-Containern aktiviert sind, sprechen auf die Zurück-Taste und die Menü-Taste eines Mobilgeräts an. Wenn der Benutzer die Zurück-Taste drückt, navigiert die Anwendung zur vorherigen Ansicht. Wenn es keine vorherige Ansicht gibt, wird die Anwendung beendet und der Startbildschirm des Geräts angezeigt.

Wenn der Benutzer die Zurück-Taste drückt, empfängt die aktive Ansicht der Anwendung ein `backKeyPressed`-Ereignis. Diesen Vorgang können Sie abbrechen, indem Sie in der Ereignisprozedur für das `backKeyPressed`-Ereignis `preventDefault()` aufrufen.

Beim Drücken der Menü-Taste wird, sofern definiert, der ViewMenu-Container der aktuellen Ansicht angezeigt. Im ViewMenu-Container ist ein Menü definiert, das sich am unteren Rand eines View-Containers befindet. Jeder View-Container definiert ein eigenes, für die betreffende Ansicht spezifisches Menü.

Der aktuelle View-Container löst ein `menuKeyPressed`-Ereignis aus, wenn der Benutzer die Menütaste drückt. Um den Vorgang der Menü-Taste abzubrechen und ViewMenu zu unterbinden, rufen Sie die `preventDefault()`-Methode in der Ereignisprozedur des `menuKeyPressed`-Ereignisses auf.

Weitere Informationen hierzu finden Sie unter „Menüs in Mobilanwendungen definieren“ auf Seite 75.

Hardwaretastatur-Ereignisse in einer Mobilanwendung verarbeiten

In einer in Flex erstellten Mobilanwendung können Sie erkennen, wenn ein Benutzer eine Hardwaretaste auf einem Mobilgerät drückt. So können Sie beispielsweise auf einem Android-Gerät erkennen, wenn der Benutzer die Start-, Zurück- oder Menütaste drückt.

Damit das Drücken einer Hardwaretaste erkannt wird, erstellen Sie Ereignisprozeduren für das `KEY_UP`- oder `KEY_DOWN`-Ereignis. Für gewöhnlich hängen Sie die Ereignisprozeduren an das Anwendungsobjekt an, wie es von den Application-, ViewNavigatorApplication- oder TabbedViewNavigatorApplication-Containern definiert wird.

Das Stage-Objekt definiert den Zeichenbereich einer Anwendung. Jede Anwendung verfügt über ein Stage-Objekt. Daher ist ein Anwendungscontainer eigentlich ein untergeordneter Container des Stage-Ojbekts.

Die `Stage.focus`-Eigenschaft legt die Komponente fest, die zurzeit den Tastaturfokus besitzt, oder enthält `null`, wenn keine Komponente den Fokus hat. Die Komponente mit Tastaturfokus erhält Ereignisbenachrichtigungen, wenn der Benutzer mit der Tastatur interagiert. Daher werden die Ereignisprozeduren des Anwendungsobjekts aufgerufen, falls `Stage.focus` für das Anwendungsobjekt festgelegt ist.

Auf einem Mobilgerät kann Ihre Anwendung durch eine andere Anwendung gestört werden. Zum Beispiel wird, während die Anwendung ausgeführt wird, auf dem Mobilgerät ein Anruf empfangen oder der Benutzer wechselt zu einer anderen Anwendung. Wechselt der Benutzer zurück zu Ihrer Anwendung, ist die `Stage.focus`-Eigenschaft „Null“. Aus diesem Grund reagieren Ereignisprozeduren, die dem Anwendungsobjekt zugewiesen sind, nicht auf die Tastatur.

Da die `Stage.focus`-Eigenschaft in einer Mobilanwendung „null“ sein kann, sollten Tastaturereignisse im Stage-Objekt selbst abgehört werden, um sichzuerstellen, dass Ihre Anwendung das Ereignis erkennt. Das folgende Beispiel weist Tastaturereignisprozeduren dem Stage-Objekt zu:

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

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

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

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

Maus- und Berührungsereignisse in einer Mobilanwendung verarbeiten

AIR generiert für die unterschiedlichen Eingabetypen unterschiedliche Ereignisse. Zu diesen Ereignissen zählen folgende:

Mausereignisse Ereignisse, die durch die Benutzerinteraktion mit einer Maus oder einem Touchscreen generiert werden. Zu den Mausereignissen zählen `mouseOver`, `mouseDown` und `mouseUp`.

Berührungsereignisse Ereignisse, die auf Geräten generiert werden, die Benutzerkontakte mit dem Gerät erkennen, etwa eines Fingers auf einem Touchscreen. Zu den Berührungsereignissen zählen `touchTap`, `touchOver` und `touchMove`. Wenn ein Benutzer mit einem Gerät mit einem Touchscreen interagiert, berührt der Benutzer normalerweise den Bildschirm mit einem Finger oder einem Zeigegerät.

Gestenereignisse Ereignisse, die durch Multitouch-Interaktionen generiert werden, z. B. das gleichzeitige Berühren eines Touchscreens mit zwei Fingern. Zu den Gestenereignissen zählen `gesturePan`, `gestureRotate` und `gestureZoom`. Bei manchen Geräten können Sie beispielsweise zum Auszoomen (Verkleinern) eines Bildes die Finger zusammenführen.

Integrierte Unterstützung für Mausereignisse

Das Flex-Framework und der Flex-Komponentensatz bieten integrierte Unterstützung für Mausereignisse, nicht jedoch für Berührungs- und Gestenereignisse. Beispielsweise interagiert der Benutzer über den Touchscreen mit Flex-Komponenten in einer Mobilanwendung. Die Komponenten reagieren auf Mausereignisse wie `mouseDown` und `mouseOver`, nicht jedoch auf Berührungs- und Gestenereignisse.

Beispielsweise kann der Benutzer auf den Touchscreen drücken, um das Button-Steuerelement von Flex auszuwählen. Das Button-Steuerelement signalisiert mithilfe des `mouseUp`-Ereignisses und des `mouseDown`-Ereignisses, dass der Benutzer mit dem Steuerelement interagiert hat. Das -Steuerelement gibt mithilfe des `mouseMove`-Ereignisses und des `mouseUp`-Ereignisses an, dass der Benutzer auf dem Bildschirm einen Bildlauf durchführt.


Adobe-Entwickler Paul Trani über Berührungen und Gesten auf Mobilgeräten: Touch Events and Gesture on Mobile.

Von AIR erzeugte Steuerereignisse

Die von AIR und Flash Player generierten Ereignisse werden von der `flash.ui.Multitouch.inputMode`-Eigenschaft gesteuert. Die `flash.ui.Multitouch.inputMode`-Eigenschaft kann einen der folgenden Werte annehmen:

- `MultitouchInputMode.NONE` AIR löst Mausereignisse aus, nicht jedoch Berührungs- oder Gestenereignisse.
- `MultitouchInputMode.TOUCH_POINT` AIR löst Maus- und Berührungsereignisse aus, nicht jedoch Gestenereignisse. In diesem Modus empfängt das Flex-Framework die gleichen Mausereignisse wie in `MultitouchInputMode.NONE`.
- `MultitouchInputMode.GESTURE` AIR löst Maus- und Gestenereignisse aus, nicht jedoch Berührungsereignisse. In diesem Modus empfängt das Flex-Framework die gleichen Mausereignisse wie in `MultitouchInputMode.NONE`.

Wie Sie der Liste entnehmen können, löst AIR unabhängig von der Einstellung der `flash.ui.Multitouch.inputMode`-Eigenschaft Mausereignisse immer aus. Flex-Komponenten können daher immer auf Benutzerinteraktionen reagieren, die über einen Touchscreen vorgenommen wurden.

Mit Flex können Sie in Ihrer Anwendung jeden Wert der `flash.ui.Multitouch.inputMode`-Eigenschaft verwenden. Daher reagieren die Flex-Komponenten zwar nicht auf Berührungs- und Gestenereignisse, doch können Sie der Anwendung Funktionen hinzufügen, um auf alle Ereignisse zu reagieren. Beispielsweise können Sie dem Button-Steuerelement eine Ereignisprozedur zum Verarbeiten von Berührungsereignissen hinzufügen, z. B. der Ereignisse `touchTap`, `touchOver` und `touchMove`.

Das ActionScript 3.0-Entwicklerhandbuch bietet einen Überblick über die Verarbeitung von Benutzereingaben auf unterschiedlichen Geräten sowie über die Eingabe per Berührung, Multitouch und Gesten. Weitere Informationen finden Sie unter:

- Grundlagen der Benutzerinteraktion
- Eingabe per Berührung, Multitouch und Gesten

Mobilanwendungen und Begrüßungsbildschirme definieren

Container für Mobilanwendung erstellen

Für das erste Tag in einer Mobilanwendung gibt es in der Regel die folgenden Optionen:

- Das `<s:ViewNavigatorApplication>`-Tag definiert eine Mobilanwendung mit einem einzelnen Abschnitt.
- Das `<s:TabbedViewNavigatorApplication>`-Tag definiert eine Mobilanwendung mit mehreren Abschnitten.

Bei der Anwendungsentwicklung für Tablet-Computer stellt die eingeschränkte Bildschirmgröße kein solches Problem dar wie bei Telefonen. Dies bedeutet, dass Sie die Anwendungen für Tablet-Computer nicht für kleine Ansichten strukturieren müssen. Stattdessen können Sie zum Erstellen der Anwendung den standardmäßigen Spark-Anwendungscontainer mit den unterstützten Mobilkomponenten und -skins verwenden.

Hinweis: *Zur Entwicklung von Mobilanwendungen können Sie den Spark-Anwendungscontainer verwenden, und das sogar für Telefone. Vom Spark-Anwendungscontainer werden jedoch Ansichtsnavigation, Datenpersistenz und die Zurück- und Menü-Tasten des Geräts nicht unterstützt. Weitere Informationen finden Sie unter „Unterschiede zwischen den Mobilanwendungscontainern und dem Spark-Anwendungscontainer“ auf Seite 35About the Application container.*

Die Container von Mobilanwendungen besitzen folgende Merkmale:

Merkmal	**Die Spark-Container ViewNavigatorApplication und TabbedViewNavigatorApplication**
Standardgröße	100 % Höhe und 100 % Breite, um den gesamten verfügbaren Platz auf dem Bildschirm einzunehmen.
Layout des untergeordneten Elements	Wird durch die einzelnen View-Container definiert, die die Ansichten der Anwendung darstellen.
Standardabstand	0 Pixel.
Bildlaufleisten	Keine. Wenn Sie Bildlaufleisten zur Skin des Anwendungscontainers hinzufügen, kann der Benutzer durch die gesamte Anwendung scrollen. Dazu zählen auch der ActionBar-Bereich und der Registerkartenleistenbereich der Anwendung. In diesen Bereichen der Ansicht sollen normalerweise keine Bildläufe ausgeführt werden. Fügen Sie deshalb Bildlaufleisten den einzelnen View-Containern der Anwendung und nicht der Skin des Anwendungscontainers hinzu.

Unterschiede zwischen den Mobilanwendungscontainern und dem Spark-Anwendungscontainer

Die Spark-Mobilanwendungscontainer besitzen viele Funktionen, die auch der Spark-Anwendungscontainer besitzt. Beispielsweise können Sie den Mobilanwendungscontainern auf dieselbe Weise wie dem Spark-Anwendungscontainer Stile hinzufügen.

Die Spark-Mobilanwendungscontainer weisen mehrere vom Spark-Anwendungscontainer abweichende Merkmale auf:

- **Unterstützung der Persistenz**

  Unterstützt die Datenspeicherung auf und das Laden von einer Festplatte. Durch Persistenz können Benutzer eine Mobilanwendung unterbrechen, zum Beispiel, um einen Telefonanruf anzunehmen, und den Status der Anwendung nach Anrufende wiederherstellen.

- **Unterstützung der Ansichtsnavigation**

  Der ViewNavigatorApplication-Container erstellt automatisch einen einzelnen ViewNavigator-Container, mit dem die Navigation zwischen den Ansichten gesteuert wird.

  Der TabbedViewNavigatorApplication-Container erstellt automatisch einen einzelnen TabbedViewNavigator-Container, mit dem die Navigation zwischen den Abschnitten gesteuert wird.

- **Unterstützung der Zurück- und Menü-Taste des Geräts**

  Wenn der Benutzer die Zurück-Taste drückt, navigiert die Anwendung zurück zur vorherigen Ansicht auf dem Stapel. Beim Drücken der Menü-Taste wird, sofern definiert, der ViewMenu-Container der aktuellen Ansicht angezeigt.

Weitere Informationen zum Spark-Anwendungscontainer finden Sie unter About the Application container.

Ereignisse auf Anwendungsebene verarbeiten

Die NativeApplication-Klasse stellt eine AIR-Anwendung dar. Sie bietet Anwendungsinformationen sowie anwendungsweit gültige Funktionen und löst Ereignisse auf Anwendungsebene aus. Auf die Instanz der NativeApplication-Klasse, die Ihrer Mobilanwendung entspricht, greifen Sie über die statische `NativeApplication.nativeApplication`-Eigenschaft zu.

Beispielsweise definiert die NativeApplication-Klasse die Ereignisse `invoke` und `exiting`, die in Ihrer Mobilanwendung verarbeitet werden können. Das folgende Beispiel zeigt, wie mit der NativeApplication-Klasse eine Ereignisprozedur für das Ereignis `exiting` definiert wird:

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

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

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

</s:ViewNavigatorApplication>
```

Bedenken Sie, dass ViewNavigator über die Eigenschaft `ViewNavigatorApplication.navigator` aufgerufen wird.

Begrüßungsbildschirm zu einer Anwendung hinzufügen

Der Spark-Anwendungscontainer ist eine Basisklasse für die Container ViewNavigatorApplication und TabbedViewNavigatorApplication. Wenn er im Spark-Design eingesetzt wird, unterstützt er einen Anwendungs-Preloader, der beim Herunterladen und Initialisieren einer Anwendungs-SWF-Datei eine Download-Fortschrittsanzeige anzeigt.

In Mobildesigns können Sie mit diesem Container stattdessen einen Begrüßungsbildschirm anzeigen lassen. Der Begrüßungsbildschirm wird beim Starten der Anwendung angezeigt.

Hinweis: *Um einen Begrüßungsbildschirm für eine Desktop-Anwendung zu erzeugen, legen Sie die* `Application.preloader`*-Eigenschaft auf „spark.preloaders.SplashScreen“ fest. Fügen Sie dem Bibliothekspfad der Anwendung außerdem frameworks\libs\mobile\mobilecomponents.swc hinzu.*


Blogger Joseph Labrecque schrieb einen Artikel über AIR für Android-Startbildschirm mit Flex.


Blogger Brent Arnold hat ein Video über das Hinzufügen eines Startbildschirms zur Android-Anwendung erstellt.

Begrüßungsbildschirm aus Bilddatei hinzufügen

Sie können einen Begrüßungsbildschirm direkt aus einer Bilddatei hinzufügen. Zur Konfiguration des Begrüßungsbildschirms dienen die Eigenschaften `splashScreenImage`, `splashScreenScaleMode` und `splashScreenMinimumDisplayTime` der Anwendungsklasse.

Im folgenden Beispiel wird ein Begrüßungsbildschirm aus einer JPG-Datei geladen, wobei das Letterbox-Format verwendet wird:

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

</s:ViewNavigatorApplication>
```

Begrüßungsbildschirm aus benutzerdefinierter Komponente hinzufügen

Im Beispiel im vorherigen Abschnitt wurde der Begrüßungsbildschirm mithilfe einer JPG-Datei definiert. Dieses Verfahren hat den Nachteil, dass die Anwendung ohne Rücksicht auf die Möglichkeiten des verwendeten Mobilgeräts immer dieselbe Bilddatei verwendet.

Die Bildschirmauflösungen und -größen von Mobilgeräten sind unterschiedlich. Es ist möglich, eine benutzerdefinierte Komponente zu definieren, anstatt immer dasselbe Bild für den Begrüßungsbildschirm zu verwenden. Die Komponente ermittelt die Spezifikationen des Mobilgeräts und wählt das passende Bild für den Begrüßungsbildschirm.

Mithilfe der Klasse SplashScreenImage können Sie die benutzerdefinierte Komponente definieren, wie im folgenden Beispiel gezeigt:

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

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

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

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

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

Verwenden Sie in der Definition der Komponente die Klasse SplashScreenImageSource, um die Bilder zu definieren, die Sie für den Begrüßungsbildschirm verwenden möchten. In der Eigenschaft `SplashScreenImageSource.source` geben Sie die zu verwendende Bilddatei an. Die Eigenschaften `dpi`, `aspectRatio` und `minResolution` von SplashScreenImageSource geben die für die Anzeige des jeweiligen Bildes erforderlichen Spezifikationen des Mobilgeräts an.

So gibt beispielsweise die erste SplashScreenImageSource-Definition nur die `source`-Eigenschaft für das Bild an. Da keine Einstellungen für die Eigenschaften `dpi`, `aspectRatio` und `minResolution` vorhanden sind, kann dieses Bild für beliebige Geräte verwendet werden. Mit diesem Bild wird daher das Standardbild definiert, das angezeigt wird, wenn keine weiteren Bilder den Spezifikationen des Geräts entsprechen.

Die zweite und die dritte SplashScreenImageSource-Definition geben ein Bild für ein 240-dpi-Gerät im Hoch- oder Querformatmodus an.

Die letzte SplashScreenImageSource-Definition gibt ein Bild für ein 160-dpi-Gerät im Hochformatmodus mit einer Mindestauflösung von 960 Pixeln an. Der Wert der Eigenschaft `minResolution` wird mit dem größeren der Werte für die Eigenschaften `Stage.stageWidth` und `Stage.stageHeight` verglichen. Der größere dieser beiden Werte muss dem Wert der Eigenschaft `minResolution` gleich oder größer als dieser sein.

In der folgenden Mobilanwendung wird diese Komponente verwendet:

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

Die SplashScreenImage-Klasse ermittelt automatisch das Bild, das den Spezifikationen des Geräts am besten entspricht. Dabei werden die Eigenschaften `dpi`, `aspectRatio` und `minResolution` aus den einzelnen SplashScreenImageSource-Definitionen zugrunde gelegt.

Das am besten passende Bild wird wie folgt ermittelt:

1. Alle SplashScreenImageSource-Definitionen, die den Einstellungen des Mobilgeräts entsprechen, werden ermittelt. Voraussetzung für eine Übereinstimmung ist:
   a. In der SplashScreenImageSource-Definition wurde die betreffende Einstellung nicht explizit definiert. So könnte z. B. keine der Einstellungen für die Eigenschaft `dpi` einer dpi-Angabe für das Gerät entsprechen.
   b. Für `dpi` oder `aspectRatio` muss eine exakte Übereinstimmung mit der entsprechenden Einstellung des Mobilgeräts vorhanden sein.
   c. Für die Eigenschaft `minResolution` liegt eine Übereinstimmung mit der Geräteeinstellung vor, wenn der größere der Werte für `Stage.stageWidth` und `Stage.stageHeight` gleich dem Wert für `minResolution` oder größer als dieser ist.
2. Sind mehrere für das Gerät passende SplashScreenImageSource-Definitionen vorhanden, gilt:
   a. Verwenden Sie die Definition mit den meisten explizit festgelegten Einstellungen. So ist z. B. eine SplashScreenImageSource-Definition mit Angaben sowohl für `dpi` als auch für `aspectRatio` besser geeignet als eine Definition, in der nur `dpi` festgelegt wurde.
   b. Sind nach wie vor mehrere Möglichkeiten vorhanden, wählen Sie die mit dem höchsten Wert für `minResolution`.
   c. Wenn auch dann noch mehrere Übereinstimmungen vorhanden sind, wählen Sie die erste, die in der Komponente definiert wurde.

Bild für Begrüßungsbildschirm explizit auswählen

In der `SplashScreenImage.getImageClass()`-Methode wird die SplashScreenImageSource-Definition festgelegt, die den Spezifikationen des jeweiligen Mobilgeräts am besten entspricht. Sie können diese Methode außer Kraft setzen und eigene benutzerdefinierte Logik hinzufügen, wie im folgenden Beispiel gezeigt.

In diesem Beispiel wird eine SplashScreenImageSource-Definition für einen iOS-Begrüßungsbildschirm hinzugefügt. Im Hauptteil des Codes, mit dem Sie die Vorgabe für die Methode `getImageClass()` überschreiben, legen Sie zunächst fest, ob die Anwendung unter iOS ausgeführt wird. Ist dies der Fall, legen Sie eine iOS-spezifische Anzeige des Bildes fest.

Andernfalls soll die `super.getImageClass()`-Methode aufgerufen werden. Diese Methode ermittelt mithilfe der Standardimplementierung die anzuzeigende Instanz von SplashScreenImageSource:

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

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

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

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

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

Ansichten in einer Mobilanwendung definieren

Mobilanwendungen bestehen üblicherweise aus mehreren Bildschirmen oder Ansichten. Wenn Benutzer in der Anwendung navigieren, wechseln sie zwischen verschiedenen Ansichten.

Die Navigation sollte für den Benutzer der Anwendung intuitiv verständlich sein. Benutzer erwarten beim Wechseln zwischen Ansichten, dass sie auch zur vorherigen Ansicht zurückwechseln können. In der Anwendung können eine Schaltfläche für die Startseite oder andere Navigationshilfen der obersten Ebene definiert werden, mit deren Hilfe der Benutzer in der Anwendung navigieren kann.

Die Ansichten einer Mobilanwendung definieren Sie im View-Container. Zum Steuern der Navigation zwischen den Ansichten einer Mobilanwendung verwenden Sie den ViewNavigator-Container.

Ansichten mithilfe von pushView() wechseln

Mit der `ViewNavigator.pushView()`-Methode legen Sie eine neue Ansicht auf dem Stapel ab. Rufen Sie ViewNavigator über die Eigenschaft `ViewNavigatorApplication.navigator` auf. Durch Ablegen einer Ansicht auf dem Stapel wechselt die Anzeige der Anwendung zur neuen Ansicht.

Für die `pushView()`-Methode verwenden Sie die folgende Syntax:

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

Dabei gilt:

- `viewClass` bezeichnet den Klassennamen der Ansicht. Diese Klasse entspricht normalerweise der MXML-Datei, in der die Ansicht definiert ist.
- `data` bezeichnet sämtliche Daten, die an die Ansicht übergeben werden. Dieses Objekt wird in der `View.data`-Eigenschaft der neuen Ansicht geschrieben.
- `context` bezeichnet ein beliebiges Objekt in der `ViewNavigator.context`-Eigenschaft. Wenn die neue Ansicht erstellt wurde, kann sie auf diese Eigenschaft verweisen und eine Aktion ausführen, die auf diesem Wert beruht. In der Ansicht können Daten je nach Wert von `context` auf unterschiedliche Weise angezeigt werden.
- `transition` bezeichnet den abzuspielenden Übergang beim Wechsel der Ansicht. Weitere Informationen zu Ansichtenübergängen finden Sie unter „Übergänge in einer Mobilanwendung definieren“ auf Seite 96.

Einzelne Objekte mit dem data-Argument übergeben

Mit dem `data`-Argument können Sie ein einzelnes Objekt mit Daten übergeben, die von der neuen Ansicht benötigt werden. Dann kann die Ansicht, wie im folgenden Beispiel gezeigt, über die `View.data`-Eigenschaft auf das Objekt zugreifen:

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

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

In diesem Beispiel wird die EmployeeView in der Datei „EmployeeView.mxml“ definiert. In dieser Ansicht wird mithilfe der `data`-Eigenschaft auf den Vor- und Nachnamen eines Mitarbeiters sowie auf die Mitarbeiter-ID zugegriffen, die im übergebenen Objekt enthalten sind.

Für den Zeitpunkt des `add`-Ereignisses für das View-Objekt wird die Gültigkeit der `View.data`-Eigenschaft garantiert. Weitere Informationen zum Lebenszyklus eines View-Containers finden Sie unter „Lebenszyklus des ViewNavigator-Containers und des View-Containers von Spark“ auf Seite 49.

Daten an die erste Ansicht in einer Anwendung übergeben

Mit den Eigenschaften `ViewNavigatorApplication.firstView` und `ViewNavigator.firstView` wird die erste Ansicht in einer Anwendung definiert. Um an die erste Ansicht Daten zu übergeben, verwenden Sie die `ViewNavigatorApplication.firstViewData`-Eigenschaft oder die `ViewNavigator.firstViewData`-Eigenschaft.

Daten an eine Ansicht übergeben

Im folgenden Beispiel wird eine Mobilanwendung mithilfe des ViewNavigatorApplication-Containers definiert. Der ViewNavigatorApplication-Container erstellt automatisch eine einzelne Instanz der ViewNavigator-Klasse, mit deren Hilfe Sie in den von der Anwendung definierten Ansichten navigieren können.

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

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

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

In diesem Beispiel wird im Navigationsbereich des ActionBar-Steuerelements eine Schaltfläche für die Startseite definiert. Bei Auswahl der Schaltfläche für die Startseite werden alle Ansichten aus dem Stapel entfernt, sodass wieder die erste Ansicht angezeigt wird. In der folgenden Abbildung ist diese Anwendung dargestellt:

In der Datei „EmployeeMainView.mxml“ ist, wie im folgenden Beispiel dargestellt, die erste Ansicht der Anwendung definiert:

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

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;
            protected function myList_changeHandler(event:IndexChangeEvent):void {
                navigator.pushView(views.EmployeeView,myList.selectedItem);
            }
        ]]>
    </fx:Script>

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

Diese Ansicht definiert ein List-Steuerelement, mit dem Benutzer einen Mitarbeiternamen auswählen können. Bei Auswahl eines Namens legt die Ereignisprozedur für das `change`-Ereignis auf dem Stapel eine Instanz einer anderen Ansicht mit dem Namen EmployeeView ab. Durch Ablegen einer Instanz von EmployeeView wechselt die Anwendung zur EmployeeView-Ansicht.

Die `pushView()`-Methode in diesem Beispiel akzeptiert zwei Argumente: die neue Ansicht sowie ein Objekt, in dem die Daten definiert sind, die an die neue Ansicht übergeben werden sollen. In diesem Beispiel übergeben Sie das Datenobjekt, das dem derzeit im List-Steuerelement ausgewählten Element entspricht.

Das folgende Beispiel zeigt die Definition für EmployeeView:

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

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

In der EmployeeView werden drei Felder aus dem Datenprovider des List-Steuerelements angezeigt. EmployeeView greift auf die von der `View.data`-Eigenschaft übergebenen Daten zu.


Blogger Steve Mathews beschreibt die Übergabe von Daten zwischen Ansichten.

Daten aus einer Ansicht beziehen

Die `ViewNavigator.popView()`-Methode führt die Steuerung von der aktuellen Ansicht zurück zur vorherigen Ansicht auf dem Stapel. Bei Ausführung der `popView()`-Methode wird die aktuelle Ansicht zerstört und die vorherige Ansicht auf dem Stapel wird wiederhergestellt. Das Wiederherstellen der vorherigen Ansicht beinhaltet auch das Zurücksetzen der `data`-Eigenschaft aus dem Stapel.

Eine vollständige Beschreibung des Lebenszyklus einer Ansicht, einschließlich der beim Erstellen ausgelösten Ereignisse, finden Sie unter „Lebenszyklus des ViewNavigator-Containers und des View-Containers von Spark“ auf Seite 49.

Die neue Ansicht wird mit dem ursprünglichen `data`-Objekt zu dem Zeitpunkt, als es deaktiviert wurde, wiederhergestellt. Deshalb verwenden Sie in der Regel nicht das ursprüngliche `data`-Objekt, um Daten von der alten Ansicht an die neue Ansicht zu übergeben. Stattdessen überschreiben Sie die `createReturnObject()`-Methode der alten Ansicht. Die Methode `createReturnObject()` gibt ein einzelnes Objekt zurück.

Rückgabeobjekttyp

Das von der Methode `createReturnObject()` zurückgegebene Objekt wird in die Eigenschaft `ViewNavigator.poppedViewReturnedObject` geschrieben. Der Datentyp der Eigenschaft `poppedViewReturnedObject` ist „ViewReturnObject“.

In ViewReturnObject werden die zwei Eigenschaften `context` und `object` definiert. Die Eigenschaft `object` enthält das von der Methode `createReturnObject()` zurückgegebene Objekt. Die `context`-Eigenschaft enthält den Wert des `context`-Arguments, das an die Ansicht übergeben wurde, als die Ansicht mithilfe von `pushView()` an den Navigationsstapel übergeben wurde.

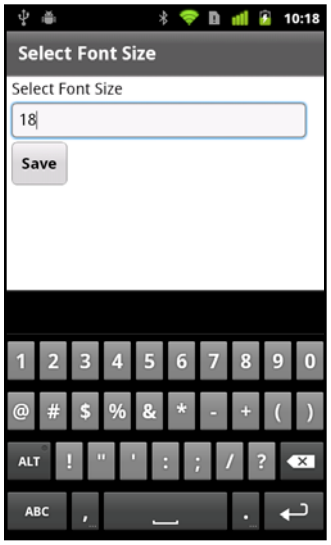
Die `poppedViewReturnedObject`-Eigenschaft wird garantiert in der neuen Ansicht festgelegt, bevor die Ansicht das `add`-Ereignis empfängt. Wenn die `poppedViewReturnedObject.object`-Eigenschaft null ist, wurden keine Daten zurückgegeben.

Beispiel: Übergeben von Daten an eine Ansicht

Das folgende Beispiel zu „SelectFont.mxml“ zeigt eine Ansicht, in der Sie eine Schriftgröße festlegen können. Beim Überschreiben der `createReturnObject()`-Methode wird der Wert als Zahl zurückgegeben. Mit dem `fontSize`-Feld der `data`-Eigenschaft, die von der vorherigen Ansicht übergeben wurde, wird der Ausgangswert des TextInput-Steuerelements festgelegt:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SelectFont.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Select Font Size"
    add="addHandler(event);">
    <s:layout>
        <s:VerticalLayout paddingTop="10"
            paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            // Define return Number object.
            protected var fontSize:Number;

            // Initialize the return object with the passed in font size.
            // If you do not set a value,
            // return this value for the font size.
            protected function addHandler(event:FlexEvent):void {
                fontSize = data.fontSize;
            }

            // Save the value of the specified font.
            protected function changeHandler(event:Event):void {
                fontSize=Number(ns.text);
                navigator.popView();
            }

            // Override createReturnObject() to return the new font size.
            override public function createReturnObject():Object {
                return fontSize;
            }
        ]]>
    </fx:Script>

    <s:Label text="Select Font Size"/>
    <!-- Set the initlial value of the TextInput to the passed fontSize -->
    <s:TextInput id="ns"
        text="{data.fontSize}"/>
    <s:Button label="Save" click="changeHandler(event);"/>
</s:View>
```

Die folgende Abbildung zeigt die durch „SelectFont.mxml“ definierte Ansicht:



In der Ansicht „MainFontView.mxml“ im folgenden Beispiel wird die in „SetFont.mxml“ definierte Ansicht verwendet. Die Ansicht „MainFontView.mxml“ definiert Folgendes:

- Ein Button-Steuerelement in der ActionBar zum Wechseln der von „SetFont.mxml“ definierten Ansicht.
- Eine Ereignisprozedur für das `add`-Ereignis, mit der zuerst bestimmt wird, ob die `View.data`-Eigenschaft null ist. Wenn sie null ist, wird das `data.fontSize`-Feld zur `View.data`-Eigenschaft hinzugefügt.

  Wenn die `data`-Eigenschaft nicht null ist, legt die Ereignisprozedur die Schriftgröße auf den Wert im `data.fontSize`-Feld fest.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\MainFontView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Font Size"
    add="addHandler(event);">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

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

            // Change to the SelectFont view, and pass the current data property.
            // The data property contains the fontSize field with the current font size.
            protected function clickHandler(event:MouseEvent):void {
                navigator.pushView(views.SelectFont, data);
            }
            // Set the font size in the event handler for the add event.
            protected function addHandler(event:FlexEvent):void {
                // If the data property is null,
                // initialize it and create the data.fontSize field.
                if (data == null) {
```

```
                data = new Object();
                data.fontSize = getStyle('fontSize');
                return;
            }

            // Otherwise, set data.fontSize to the retured value,
            // and set the font size.
            data.fontSize = navigator.poppedViewReturnedObject.object;
            setStyle('fontSize', data.fontSize);
        }
    ]]>
</fx:Script>

<s:actionContent>
    <s:Button label="Set Font&gt;"
        click="clickHandler(event);"/>
</s:actionContent>

<s:Label text="Text to size."/>
</s:View>
```

Eine Anwendung für Hoch- und Querformat konfigurieren

Ein Mobilgerät passt die Ausrichtung einer Anwendung automatisch an den Neigungswinkel des Geräts an. Zur Konfiguration der Anwendung für verschiedene Ausrichtungen definiert Flex zwei Ansichtsstatus für Hoch- und Querformat: `portrait` und `landscape`. Mithilfe dieser Ansichtszustände können Sie Merkmale der Anwendung anhand der Ausrichtung festlegen.

Im folgenden Beispiel wird über den Ansichtszustand anhand der aktuellen Ausrichtung die Eigenschaft `layout` eines Group-Containers gesteuert:

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

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

    <s:Group>
        <s:layout>
            <s:VerticalLayout/>
        </s:layout>
        <s:layout.landscape>
            <s:HorizontalLayout/>
        </s:layout.landscape>
        <s:TextInput text="Enter search text" textAlpha="0.5"/>
        <s:Button label="Search"/>
    </s:Group>
    <s:TextArea text="search results" textAlpha="0.5"/>
</s:View>
```

In diesem wird eine Suchansicht definiert. Der Group-Container steuert das Layout des Eingabesuchtexts und der Suchschaltfläche. Im Hochformat verwendet der Group-Container das vertikale Layout. Beim Wechsel des Layouts in das Querformat verwendet der Group-Container das horizontale Layout.

Benutzerdefinierte Skins für Layoutmodi vorsehen

Für Mobilanwendungen können Sie benutzerdefinierte Skinklassen definieren. Zur Unterstützung des Hoch- und Querformatlayouts muss die Skin den `portrait`-Ansichtszustand und den `landscape`-Ansichtszustand verarbeiten können.

Sie können eine Anwendung so konfigurieren, dass die Layoutausrichtung erhalten bleibt, wenn der Benutzer das Gerät dreht. Dazu bearbeiten Sie die XML-Datei der Anwendungsdatei (endet auf „-app.xml“), indem Sie sie auf die folgenden Eigenschaften festlegen:

- Zum Deaktivieren des Layoutwechsels der Anwendung legen Sie die `<autoOrients>`-Eigenschaft auf `false` fest.
- Zum Festlegen der Ausrichtung legen Sie die `<aspectRatio>`-Eigenschaft auf `portrait` oder `landscape` fest.

Overlaymodus eines Spark ViewNavigator-Containers festlegen

In der Standardeinstellung definieren die Registerkartenleiste und das ActionBar-Steuerelement einer Mobilanwendung einen Bereich, der von den Ansichten der Anwendung nicht verwendet werden kann. Dies bedeutet, dass der Inhalt nicht die gesamte Bildschirmgröße des Mobilgeräts einnehmen kann.

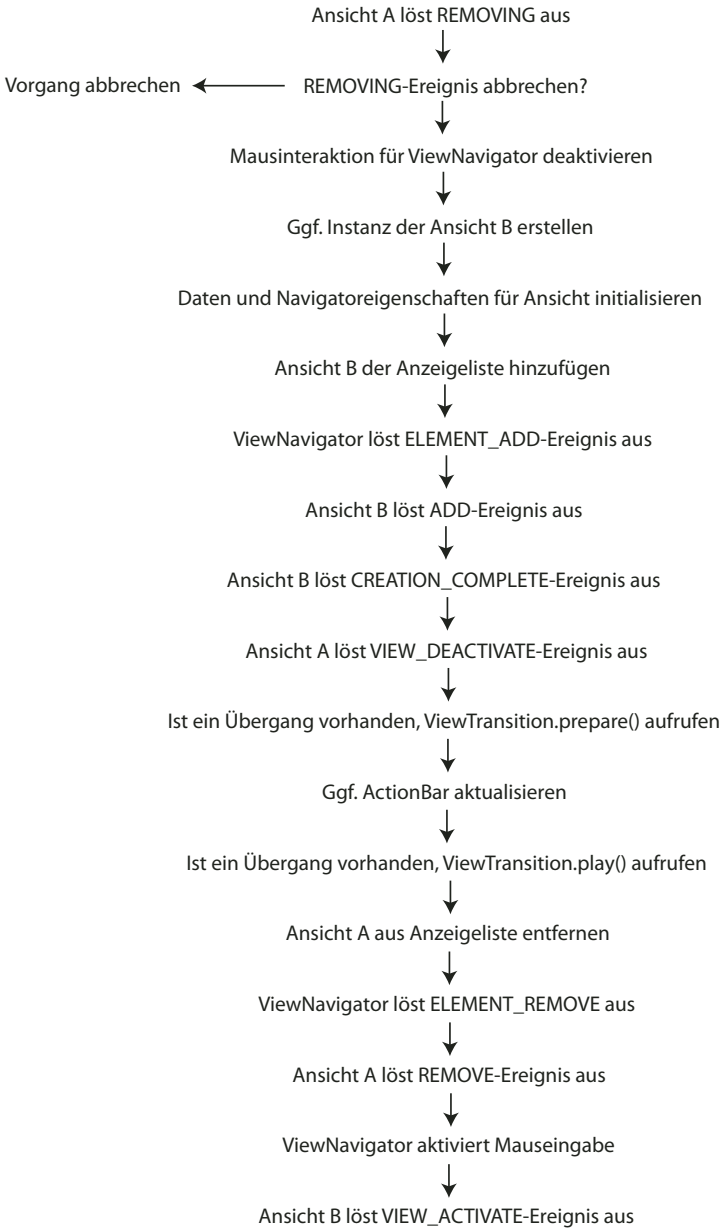
Über die `ViewNavigator.overlayControls`-Eigenschaft können Sie jedoch das Standardlayout dieser Komponenten ändern. Wenn Sie die `overlayControls`-Eigenschaft auf `true` festlegen, umfasst der Inhaltsbereich der Anwendung die gesamte Breite und Höhe des Bildschirms. Das ActionBar-Steuerelement und die Registerkartenleiste werden mit dem Alphawert über dem Inhaltsbereich angezeigt, d. h., sie sind halb transparent.

Die Skinklasse für den ViewNavigator-Container, spark.skins.mobile.ViewNavigatorSkin, definiert Ansichtszustände für die Verarbeitung unterschiedlicher Werte der `overlayControls`-Eigenschaft. Wenn die `overlayControls`-Eigenschaft auf `true` festgelegt ist, wird an den Namen des aktuellen Zustands „AndOverlay“ angehängt. Beispielsweise befindet sich die Skin ViewNavigator standardmäßig im Zustand „portrait“. Wenn die `overlayControls`-Eigenschaft auf `true` festgelegt ist, wechselt der Skinzustand des Navigators nach „portraitAndOverlay“.

Lebenszyklus des ViewNavigator-Containers und des View-Containers von Spark

Flex führt eine Reihe von Operationen aus, wenn Sie in einer Mobilanwendung von einer Ansicht zu einer anderen wechseln. An verschiedenen Punkten dieses Ansichtswechselvorgangs löst Flex Ereignisse aus. Diese Ereignisse können Sie überwachen, um während des Prozesses Aktionen auszuführen. Beispielsweise können Sie mit dem `removing`-Ereignis den Wechsel zwischen den Ansichten abbrechen.

Das folgende Diagramm beschreibt den Prozess des Wechsels von der aktuellen Ansicht A zu einer anderen Ansicht B:



Registerkarten in einer Mobilanwendung definieren

Anwendungsabschnitte definieren

Im TabbedViewNavigatorApplication-Container definieren Sie eine Mobilanwendung mit mehreren Abschnitten. Der TabbedViewNavigatorApplication-Container erstellt automatisch einen einzelnen TabbedViewNavigator-Container. Der TabbedViewNavigator-Container erstellt eine Registerkartenleiste für die Unterstützung der Navigation in den Abschnitten der Anwendung.

Jeder ViewNavigator-Container definiert einen anderen Anwendungsabschnitt. Mit der `navigators`-Eigenschaft des TabbedViewNavigatorApplication-Containers geben Sie ViewNavigator-Container an.

Im folgenden Beispiel definieren Sie drei Abschnitte, die den drei ViewNavigator-Tags entsprechen. Jeder ViewNavigator definiert die jeweils erste Ansicht, die beim Wechsel in den Abschnitt angezeigt wird:

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

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

</s:TabbedViewNavigatorApplication>
```

Hinweis: *Das untergeordnete `navigators`-Tag müssen Sie in MXML nicht angeben, da es sich um die Standardeigenschaft von TabbedViewNavigator handelt.*

Jeder ViewNavigator verwaltet einen eigenen Navigationsstapel. Daher beziehen sich die ViewNavigator-Methoden, z. B. `pushView()` und `popView()`, stets auf die derzeit aktive Auswahl. Mit der Zurück-Taste auf dem Mobilgerät wird wieder die vorherige Ansicht auf dem Stapel des aktuellen ViewNavigator-Containers angezeigt. Durch den Ansichtswechsel wird die aktuelle Auswahl nicht geändert.

Sie müssen der Anwendung keine besondere Logik für die Navigation zwischen den Abschnitten hinzufügen. Der TabbedViewNavigator-Container erstellt am unteren Rand der Anwendung automatisch eine Registerkartenleiste für die Navigation in den Abschnitten.

Obwohl nicht zwingend erforderlich, können Sie den aktuellen Abschnitt programmseitig steuern lassen. Um Abschnitte programmgesteuert zu ändern, legen Sie die `TabbedViewNavigator.selectedIndex`-Eigenschaft auf den Index des gewünschten Abschnitts fest. Abschnittsindizes basieren auf 0. Das heißt, der erste Abschnitt der Anwendung erhält einen Index von 0, der zweite Abschnitt einen Index von 1 usw.


Der zertifizierte Adobe-Experte für Flex, Brent Arnold, hat ein Video zur Verwendung des ViewNavigator-Navigationsstapels erstellt.


Unter Flex Mobile Development - Passing Data Between Tabs beschreibt Adobe-Guru Holly Schinsky, wie Daten zwischen Registerkarten in einer Mobilanwendung übertragen werden können.


Im Video zum TabbedViewNavigator-Container von video2brain wird beschrieben, wie Sie einen Ansichtsnavigator im Registerkartenformat erstellen.

Abschnittsänderungsereignisse abwickeln

Wenn sich der Abschnitt ändert, löst der TabbedViewNavigator-Container folgende Ereignisse aus:

- Das Ereignis `changing` wird vor der Abschnittsänderung ausgelöst. Sie können die Abschnittsänderung durch einen Aufruf der `preventDefault()`-Methode in der Ereignisprozedur des `changing`-Ereignisses verhindern.
- Das Ereignis `changing` wird direkt nach der Abschnittsänderung ausgelöst.

ActionBar mit mehreren Abschnitten konfigurieren

ActionBar-Steuerelemente sind mit ViewNavigator verknüpft. Daher können Sie bei der Definition von ViewNavigator für einen Abschnitt die ActionBar für jeden Abschnitt konfigurieren. Im folgenden Beispiel konfigurieren Sie die ActionBar separat für die verschiedenen ViewNavigator-Container, die die drei Anwendungsabschnitte definieren:

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

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

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

</s:TabbedViewNavigatorApplication>
```

In der folgenden Abbildung ist diese Anwendung mit ausgewählter Registerkarte „Contacts“ in der Registerkartenleiste dargestellt:

Sie können die ActionBar auch in jeder Ansicht der Anwendung definieren. Auf diese Weise verwendet jede Ansicht denselben ActionBar-Inhalt, unabhängig davon, wo sie in der Anwendung verwendet werden.

Registerkartenleiste steuern

Registerkarten-Steuerelement in einer Ansicht ausblenden

In jeder Ansicht kann die Registerkartenleiste ausgeblendet werden, indem die `View.tabBarVisible`-Eigenschaft auf `false` festgelegt wird. Standardmäßig ist die `tabBarVisible`-Eigenschaft auf `true` festgelegt, sodass die Registerkartenleiste angezeigt wird.

Die Sichtbarkeit kann auch über die Methoden `TabbedViewNavigator.hideTabBar()` und `TabbedViewNavigator.showTabBar()` gesteuert werden.

Der zertifizierte Adobe-Experte für Flex, Brent Arnold, hat ein Video über das Ausblenden der Registerkartenleiste erstellt.

Effekte auf die Registerkartenleiste des TabbedViewNavigator-Containers anwenden

Die Registerkartenleiste verwendet standardmäßig einen Übergangseffekt beim Ein- und Ausblenden. Die Registerkartenleiste verwendet keinen Effekt, wenn Sie die Registerkarte wechseln.

Sie können den Standardeffekt der Registerkartenleiste für das Ein- und Ausblenden ändern. Setzen Sie hierzu die Methoden `TabbedViewNavigator.createTabBarHideEffect()` und `TabbedViewNavigator.createTabBarShowEffect()` außer Kraft. Nach dem Ausblenden der Registerkartenleiste müssen Sie die `visible`-Eigenschaft und die `includeInLayout`-Eigenschaft der Registerkartenleiste auf `false` festlegen.

Mehrere Bereiche in einer Mobilanwendung erstellen

SplitViewNavigator ist ein skinfähiger Container, der die Anzeige von zwei oder mehr untergeordneten Navigationselementen für Ansichten auf demselben Bildschirm des Mobilgeräts ermöglicht. Die Navigationselemente werden in separaten Bereichen angezeigt, die mithilfe des SplitViewNavigator-Containers verwaltet werden.

Die untergeordneten Elemente des SplitViewNavigator-Containers können beliebige Komponenten sein, die ViewNavigatorBase erweitern. Sie können daher die Container ViewNavigator und TabbedViewNavigator als untergeordnete Elemente dieses Containers verwenden.

Hinweis: *Wegen des für die gleichzeitige Anzeige mehrerer Bereiche benötigten Platzes auf dem Bildschirm empfiehlt Adobe die Verwendung von SplitViewNavigator ausschließlich auf Tablets.*

SplitViewNavigator erstellt standardmäßig ein horizontales Layout der Bereiche entsprechend den untergeordneten Elementen: Sie können stattdessen auch ein vertikales Layout wählen oder ein benutzerdefiniertes erstellen.

SplitViewNavigator-Container erstellen

Eine häufig verwendete Struktur für die Benutzeroberfläche auf Tablets ist „Master/Detail“. Dabei wird der Bildschirm in zwei Hauptbereiche für den Inhalt aufgeteilt: den Master- und den Detailbereich. Der Benutzer verwendet in der Regel den Masterbereich, um die Anzeige des Inhalts im Detailbereich zu steuern.

Jeder Bereich entspricht einem untergeordneten Element von SplitViewNavigator, wobei diese untergeordneten Elemente entweder ViewNavigator- oder TabbedViewNavigator-Container sind. Da die untergeordneten Elemente Ansichtsnavigationselemente sind, enthalten sie jeweils eigene Ansichtsstapel und Aktionsleisten für den jeweiligen Bereich.

Die folgende Abbildung zeigt den SplitViewNavigator-Container mit Master- und Detailbereich in einer Anwendung:

In diesem Beispiel enthält der Masterbereich links ein List-Steuerelement aus Spark mit einer Reihe von Adobe-Produkten. Im Detailbereich rechts werden zusätzliche Informationen zum momentan im Masterbereich ausgewählten Produkt angezeigt.

Die Hauptanwendungsdatei für dieses Beispiel wird unten gezeigt:

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

Das SplitViewNavigator-Element kann den Container Application oder TabbedViewNavigatorApplication untergeordnet sein. In diesem Beispiel ist SplitViewNavigator das einzige dem Application-Container untergeordnete Element. Im SplitViewNavigator-Element ist sowohl für die Höhe als auch für die Breite jeweils 100 % angegeben, sodass der Bildschirm des Geräts vollständig ausgefüllt wird.

Die untergeordneten Elemente von SplitViewNavigator in diesem Beispiel sind ViewNavigator-Container. Das erste der ViewNavigator-Elemente definiert den Master-, das zweite den Detailbereich.

Hinweis: *Zu einem SplitViewNavigator können mehr als zwei untergeordnete Elemente gehören. Damit ist es möglich, SplitViewNavigator-Elemente mit drei, vier oder mehr Bereichen zu erstellen.*

Zugriff auf Bereiche und Ansichten eines SplitViewNavigator-Containers

Im SplitViewNavigator-Container werden Methoden und Eigenschaften für den Zugriff auf seine untergeordneten Elemente definiert. Sie können z. B. mithilfe der Eigenschaft `SplitViewNavigator.numViewNavigators` die Anzahl der untergeordneten Ansichtsnavigatorelemente in SplitViewNavigator festlegen.

Die Methode `SplitViewNavigator.getViewNavigatorAt()` ermöglicht den Zugriff auf die untergeordneten Elemente von SplitViewNavigator anhand ihrer Indizes. Im Beispiel oben lautet der Index für den ViewNavigator des Masterbereichs 0, der Index für den ViewNavigator des Detailbereichs 1.

Hinweis: *Der SplitViewNavigator-Container erbt die getElementAt()- und getElementIndex()-Methoden. Verwenden Sie diese Methoden nicht für SplitViewNavigator. Verwenden Sie stattdessen getViewNavigatorAt().*

Das SplitViewNavigator-Element kann über einen Verweis auf das ViewNavigator-Element für den betreffenden Bereich auf die einzelnen Ansichten des Bereichs zugreifen.

Der Zugriff auf den SplitViewNavigator-Container von einem untergeordneten Element aus ist über die Eigenschaft `parentNavigator` des untergeordneten Elements möglich. So enthält z. B. `ViewNavigator.parentNavigator` einen Verweis auf den übergeordneten SplitViewNavigator-Container.

Ein View-Container verwendet die Eigenschaft `View.navigator` für den Zugriff auf das ihm übergeordnete Ansichtsnavigatorselement. Damit ist der Zugriff auf SplitViewNavigator von einer Ansicht aus mithilfe der Eigenschaft `View.navigator.parentNavigator` möglich.

Im Beispiel oben wird im ViewNavigator für den Masterbereich die Ansicht „MasterCategory“ als erste Ansicht angegeben. Diese Ansicht wird, wie unten gezeigt, in der Datei „MasterCategory.mxml“ definiert:

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

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

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

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

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

In „MasterCategory.mxml“ wird ein einzelnes List-Steuerelement für die Anzeige von Informationen zu Adobe-Produkten definiert. In diesem List-Steuerelement werden mithilfe eines benutzerdefinierten Elementrenderers eine Beschriftung und ein Symbol für jedes Produkt angezeigt. Weitere Informationen zum Definieren von Elementrenderern finden Sie unter Using a mobile item renderer with a Spark list-based control.

Das List-Steuerelement im Masterbereich aktualisiert mithilfe des `change`-Ereignisses den Detailbereich entsprechend der Eingabe des Benutzers. Die Ereignisprozedur ruft zuerst einen Verweis auf den SplitViewNavigator-Container ab. Aus diesem Verweis ruft sie einen Verweis auf das ViewNavigator-Element des Detailbereichs ab.

Schließlich ruft die Ereignisprozedur die `push()`-Methode für das ViewNavigator-Element des Detailbereichs auf. Für die `push()`-Methode müssen zwei Argumente angegeben werden: die auf dem Stapel des ViewNavigator-Elements abzulegende Ansicht und ein Objekt mit Informationen zum ausgewählten Eintrag aus dem List-Element.

Detailbereich eines SplitViewNavigator-Containers aktualisieren

Im oben gezeigten Beispiel enthält der Detailbereich Informationen über den ausgewählten Eintrag aus dem List-Steuerelement im Masterbereich. Der Detailbereich hat den Namen „DetailView“ und ist wie unten gezeigt in der Datei „DetailView.mxml“ definiert:

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

    <fx:Script>
        <![CDATA[

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


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

Aus dem Masterbereich wird entsprechend der Auswahl im List-Steuerelement ein Objekt an die Datei „DetailView.mxml“ übergeben. Der Detailbereich greift mithilfe der Eigenschaft `View.data` auf diese Daten zu. Daraufhin werden im Detailbereich eine Abbildung des Produkts und Informationen dazu angezeigt und es wird ein Hyperlink auf eine Seite mit weiteren Informationen zum Produkt erstellt.

Weitere Informationen zum Übergeben von Daten an eine Ansichtscontainer finden Sie unter „Daten an eine Ansicht übergeben“ auf Seite 41.

Anzeigebereiche anhand von Geräteausrichtung

Bei der Entwicklung von Anwendungen für Tablets können Sie unterschiedliche Layouts entsprechend der Ausrichtung des Geräts verwenden. So können z. B. im Querformatmodus problemlos mehrere Bereiche nebeneinander auf dem Tablet angezeigt werden. Im Hochformatmodus ist der Bildschirm dagegen schmal. Sie haben daher die Möglichkeit, einzelne Bereiche auszublenden.

Im SplitViewNavigator-Container wird die Eigenschaft `autoHideFirstViewNavigator` definiert. Mithilfe dieser Eigenschaft können Sie die Sichtbarkeit des ersten Bereichs bei unterschiedlicher Ausrichtung festlegen. Die Standardeinstellung für `autoHideFirstViewNavigator` ist `false`, d. h., der Container zeigt den ersten Bereich ohne Berücksichtigung der Ausrichtung immer an.

Wenn Sie für `autoHideFirstViewNavigator` die Einstellung `true` festlegen, zeigt der Container den ersten Bereich im Querformat an, blendet ihn im Hochformat jedoch aus. Der SplitViewNavigator-Container blendet den ersten Bereich aus, indem für die Eigenschaft `visible` des dazugehörigen Ansichtsnavigationselements der Wert `false` festgelegt wird.

Wenn der erste Bereich im Hochformat ausgeblendet ist, können Sie ihn mithilfe der Methode `SplitViewNavigator.showFirstViewNavigatorInPopUp()` öffnen. Bei Aufruf dieser Methode wird der erste Bereich in einem Callout-Container geöffnet. Dies ist ein Popup-Container, der über der Anwendung im Vordergrund angezeigt wird wie in der folgenden Abbildung gezeigt:

In diesem Beispiel wird der Aktionsleiste im Detailbereich des SplitViewNavigator eine Schaltfläche mit der Beschriftung „Navigationsleiste anzeigen“ hinzugefügt. Wenn der erste Bereich des Containers ausgeblendet ist, kann der Benutzer mithilfe dieser Schaltfläche den Masterbereich öffnen.

Hinweis: *Erstellen Sie eine benutzerdefinierte Skin für den SplitViewNavigator, um den ersten Bereich in einem SkinnablePopUpContainer oder einer Unterklasse davon zu öffnen.*

Wenn der Callout bereits geöffnet wird, wird die Methode `showFirstViewNavigatorInPopUp()` ignoriert. Wenn die Ausrichtung des Geräts geändert und das Querformat verwendet wird, wird der Callout automatisch geschlossen und der erste Bereich wird wieder angezeigt.

Durch Klicken außerhalb des Callout-Containers schließen Sie diesen. Er kann auch durch Aufrufen der Methode `SplitViewNavigator.hideViewNavigatorPopUp()` geschlossen werden.

Weitere Informationen zum Callout-Container finden Sie unter „Callout-Container einer Mobilanwendung hinzufügen“ auf Seite 83.

Aktionsleiste zu einem Bereich innerhalb eines Callout-Containers hinzufügen

Die unten gezeigte Datei ist die Hauptanwendungsdatei, in der für die Eigenschaft `autoHideFirstViewNavigator` von SplitViewNavigator der Wert `true` festgelegt wird. In diesem Beispiel wird der Aktionsleiste des Detailbereichs im Hochformatmodus eine Schaltfläche hinzugefügt. Hierfür werden Anzeigestatus verwendet.

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

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

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

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

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

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

Die Anwendung fügt eine Ereignisprozedur für das `resize`-Ereignis des Anwendungscontainers hinzu. Flex löst das `resize`-Ereignis aus, wenn die Ausrichtung des Tablet geändert wird. In der Ereignisprozedur für das resize-Ereignis legen Sie den Anzeigestatus der Anwendung anhand der aktuellen Ausrichtung fest. Weitere Informationen zu Anzeigestatus finden Sie unter Anzeigestatus.

Im Ansichtsnavigator des Detailbereichs wird mithilfe des aktuellen Status das Aussehen des Button-Steuerelements in der Aktionsleiste gesteuert. Im Querformatmodus wird diese Schaltfläche nicht angezeigt, da der Masterbereich sichtbar ist.

Im Hochformat ist der Masterbereich ausgeblendet und die Schaltfläche wird in der Aktionsleiste des Detailbereichs angezeigt. Der Benutzer kann mithilfe dieser Schaltfläche den Masterbereich in einem Callout öffnen.

Übergeben Sie einen Verweis an das Button-Steuerelement als Argument an die Methode `showFirstViewNavigatorInPopUp()`. Dieses Argument gibt das host-Element des Callout-Containers an, d. h., der Callout wird relativ zu dem Button-Steuerelement positioniert.

SplitViewNavigator-Callout auf Benutzerbefehl hin schließen

Durch Klicken auf eine Stelle außerhalb des Callout-Containers wird dieser geschlossen. Bei einem Klick innerhalb des Callout-Containers bleibt dieser jedoch standardmäßig geöffnet.

In diesem Beispiel wird das Callout geschlossen, wenn der Benutzer einen Eintrag in der Liste auswählt. Dies wird durch Aufrufen der Methode `SplitViewnavigator.hideViewNavigatorPopUp()` erreicht. Sie rufen diese Methode in der Ereignisprozedur des `change`-Ereignisses im List-Steuerelement auf, wie im folgenden Beispiel gezeigt:

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

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

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

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

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

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

Persistenz für den SplitViewNavigator-Container implementieren

Anwendungen für Mobilgeräte werden häufig durch andere Aktionen unterbrochen, z. B. Textnachrichten, Telefonanrufe oder andere Mobilanwendungen. Beim erneuten Start der unterbrochenen Anwendung erwartet der Benutzer normalerweise die Wiederherstellung des zuletzt angezeigten Status. Der Persistenzmechanismus ermöglicht es dem Gerät, den vorherigen Status der Anwendung wiederherzustellen. Weitere Informationen finden Sie unter „Persistenz in Mobilanwendungen aktivieren“ auf Seite 128.

SplitViewNavigator implementiert die aus der Basisklasse ViewNavigatorBase übernommenen Methoden `loadViewData()` und `saveViewData()`. SplitViewNavigator kann daher den Navigationsstapel serialisieren und seine Serialisierung aufheben sowie Daten für alle untergeordneten Navigationselemente anzeigen.

Bei einer Unterbrechung der Anwendung müssen diese Methoden jedoch manuell aufgerufen werden, wie im folgenden Beispiel gezeigt:

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

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

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

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

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

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

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

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

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

Navigation, Titel und Aktionssteuerelemente in einer Mobilanwendung definieren

ActionBar-Steuerelement konfigurieren

Der ViewNavigator-Container definiert das ActionBar-Steuerelement. Das ActionBar-Steuerelement bietet einen Standardbereich für einen Titel und für die Navigations- und Aktionssteuerelemente. Damit können Sie globale Steuerelemente definieren, auf die Benutzer an beliebiger Stelle in der Anwendung oder in einer bestimmten Ansicht zugreifen können. Beispielsweise können Sie mithilfe des ActionBar-Steuerelements eine Schaltfläche für die Startseite, eine Suchschaltfläche oder andere Optionen hinzufügen.

Bei einer Mobilanwendung mit nur einem Abschnitt, d. h. einem einzelnen ViewNavigator-Container, wird für alle Ansichten dieselbe Aktionsleiste verwendet. Bei einer Mobilanwendung mit mehreren Abschnitten, d. h. mehreren ViewNavigator-Containern, ist für jeden Abschnitt eine eigene Aktionsleiste definiert.

Den Aktionsleistenbereich definieren Sie über das ActionBar-Steuerelement. Das ActionBar-Steuerelement definiert, wie die folgende Abbildung zeigt, drei unterschiedliche Bereiche:

***A**. Navigationsbereich **B**. Titelbereich **C**. Aktionsbereich*

ActionBar-Bereiche

- **Navigationsbereich**

  Enthält Komponenten, mit denen der Benutzer im Abschnitt navigieren kann. Beispielsweise können Sie im Navigationsbereich eine Schaltfläche für die Startseite definieren.

  Mit der `navigationContent`-Eigenschaft definieren Sie die Komponenten, die im Navigationsbereich angezeigt werden. Mit der `navigationLayout`-Eigenschaft definieren Sie das Layout des Navigationsbereichs.

- **Titelbereich**

  Enthält entweder einen String mit Titeltext oder Komponenten. Wenn Sie Komponenten angeben, können Sie nicht auch einen Titelstring angeben.

  Mit der `title`-Eigenschaft geben Sie den String an, der im Titelbereich angezeigt werden soll. Mit der `titleContent`-Eigenschaft definieren Sie die Komponenten, die im Titelbereich angezeigt werden. Mit der `titleLayout`-Eigenschaft definieren Sie das Layout des Titelbereichs. Wenn Sie einen Wert für die `titleContent`-Eigenschaft angeben, ignoriert die ActionBar-Skin die `title`-Eigenschaft.

- **Aktionsbereich**

  Enthält Komponenten zur Definition von Aktionen, die Benutzer in einer Ansicht ausführen können. Beispielsweise können Sie im Aktionsbereich eine Such- oder Aktualisierungsschaltfläche definieren.

  Mit der `actionContent`-Eigenschaft definieren Sie die Komponenten, die im Aktionsbereich angezeigt werden. Mit der `actionLayout`-Eigenschaft definieren Sie das Layout des Aktionsbereichs.

Obwohl Adobe empfiehlt, die beschriebenen Navigations-, Titel- und Aktionsbereiche zu verwenden, unterliegen die in diesen Bereichen platzierten Komponenten keinen Einschränkungen.

ActionBar-Eigenschaften im ViewNavigatorApplication-, ViewNavigator- oder View-Container festlegen

Die Eigenschaften, die den Inhalt des ActionBar-Steuerelements definieren, können Sie im ViewNavigatorApplication-Container, im ViewNavigator-Container oder in den einzelnen View-Containern festlegen. Der View-Container besitzt die höchste Priorität, gefolgt vom ViewNavigator- und ViewNavigatorApplication-Container. Die im ViewNavigatorApplication-Container festgelegten Eigenschaften werden daher auf die gesamte Anwendung angewendet, können jedoch im ViewNavigator- oder View-Container überschrieben werden.

Hinweis: *ActionBar-Steuerelemente sind jeweils einem ViewNavigator zugeordnet und gelten daher nur für einen einzelnen Abschnitt einer Mobilanwendung. Daher können Sie eine ActionBar nicht im TabbedViewNavigatorApplication- oder im TabbedViewNavigator-Container konfigurieren.*

Beispiel: Anpassen eines Spark ActionBar-Steuerelements auf der Anwendungsebene

Das folgende Beispiel zeigt die Hauptanwendungsdatei einer Mobilanwendung:

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

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Perform a refresh
            }
        ]]>
    </fx:Script>

    <s:navigationContent>
        <s:Button label="Home" click="navigator.popToFirstView();"/>
    </s:navigationContent>

    <s:actionContent>
        <s:Button label="Refresh" click="button1_clickHandler(event);"/>
    </s:actionContent>
</s:ViewNavigatorApplication>
```

In diesem Beispiel werden im Navigationsinhaltsbereich des ActionBar-Steuerelements eine Schaltfläche für die Startseite und im Aktionsinhaltsbereich eine Aktualisierungsschaltfläche definiert.

Im folgenden Beispiel wird der View-Container MobileViewHome definiert, in dem die erste Ansicht der Anwendung definiert wird. Der View-Container definiert einen Titelstring, „Home View“, überschreibt jedoch nicht den Navigationsinhalts- oder Aktionsinhaltsbereich des ActionBar-Steuerelements:

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

    <s:Label text="Home View"/>
    <s:Button label="Submit"/>
</s:View>
```

Beispiel: Anpassen eines ActionBar-Steuerelements in einem View-Container

In diesem Beispiel wird eine Hauptanwendungsdatei mit einem einzelnen Abschnitt verwendet, wobei im Navigationsbereich des ViewNavigatorApplication-Containers eine Schaltfläche für die Startseite (Home) definiert wird. Darüber hinaus wird im Aktionsbereich eine Suchschaltfläche definiert:

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

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                navigator.popToFirstView();
            }
            protected function button2_clickHandler(event:MouseEvent):void {
                // Handle search
            }
        ]]>
    </fx:Script>

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

    <s:actionContent>
        <s:Button icon="@Embed(source='assets/Search.png')"
            click="button2_clickHandler(event);"/>
    </s:actionContent>
</s:ViewNavigatorApplication>
```

Die erste Ansicht dieser Anwendung ist die Ansicht MobileViewHomeOverride. Die Ansicht MobileViewHomeOverride definiert ein Button-Steuerelement für die Navigation zu einem zweiten View-Container, in dem eine Suchseite definiert ist:

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

    <fx:Script>
        <![CDATA[

            // Navigate to the Search view.
            protected function button1_clickHandler(event:MouseEvent):void {
                navigator.pushView(SearchViewOverride);
            }
        ]]>
    </fx:Script>

    <s:Label text="Home View"/>
    <s:Button label="Search" click="button1_clickHandler(event)"/>
</s:View>
```

Der View-Container, in dem die Suchseite definiert ist, überschreibt wie unten dargestellt den Titelbereich und Aktionsbereich des ActionBar-Steuerelements:

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

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Perform a search.
            }
        ]]>
    </fx:Script>

    <!-- Override the title to insert a TextInput control. -->
    <s:titleContent>
        <s:TextInput text="Enter search text ..." textAlpha="0.5"
            width="250"/>
    </s:titleContent>

    <!-- Override the action area to insert a Search button. -->
    <s:actionContent>
        <s:Button label="Search" click="button1_clickHandler(event);"/>
    </s:actionContent>

    <s:Label text="Search View"/>
    <s:TextArea text="Search results appear here ..."
        height="75%"/>
</s:View>
```

Die folgende Abbildung zeigt das ActionBar-Steuerelement für diese Ansicht:

Da die Suchansicht den Navigationsbereich des ActionBar-Steuerelements nicht überschreibt, wird im Navigationsbereich weiterhin die Schaltfläche für die Startseite angezeigt.

ActionBar-Steuerelement ausblenden

In jeder Ansicht kann das ActionBar-Steuerelement ausgeblendet werden, indem die `View.actionBarVisible`-Eigenschaft auf `false` festgelegt wird. Standardmäßig ist die `actionBarVisible`-Eigenschaft auf `true` festgelegt, sodass das ActionBar-Steuerelement angezeigt wird.

Mit der `ViewNavigator.hideActionbar()`-Methode können Sie das ActionBar-Steuerelement, wie im folgenden Beispiel gezeigt, für alle vom ViewNavigator gesteuerten Ansichten ausblenden:

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

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

</s:ViewNavigatorApplication>
```

Sie können einen benutzerdefinierten Effekt für die ActionBar definieren, wenn die ActionBar ausgeblendet ist oder wenn sie eingeblendet wird. Standardmäßig nutzt die ActionBar den Animationseffekt zum Ein- und Ausblenden. Den Standardeffekt können Sie durch Überschreiben der Methoden `ViewNavigator.createActionBarHideEffect()` und `ViewNavigator.createActionBarShowEffect()` ändern. Nachdem das ActionBar-Steuerelement ausgeblendet wurde, legen Sie dessen Eigenschaften `visible` und `includeInLayout` auf `false` fest, damit es nicht mehr im Ansichtslayout angezeigt wird.

Bildlaufleisten in Mobilanwendungen

Überlegungen bei der Verwendung der Bildlaufleisten in Mobilanwendungen

Wenn der Inhalt nicht mehr auf den Bildschirm passt, werden in der Regel Bildlaufleisten angezeigt. Fügen Sie der Anwendung mithilfe des Scroller-Steuerelements Bildlaufleisten hinzu. Einige Komponenten wie das Spark List-Steuerelement unterstützen die Bildlauffunktion, ohne das dafür die Scroller-Komponente erforderlich ist. Weitere Informationen finden Sie unter Scrolling Spark containers.

Der Berührungsbereich einer Bildlaufleiste ist der Bildschirmbereich, in dem Sie die Maus zum Ausführen eines Bildlaufs positionieren. Bei einer Desktop- oder Browser-basierten Anwendung ist der Berührungsbereich der sichtbare Bereich der Bildlaufleiste. In einer Mobilanwendung werden Bildlaufleisten selbst dann ausgeblendet, wenn der Inhalt über den sichtbaren Bereich hinausgeht. Durch Ausblenden der Bildlaufleisten kann die Anwendung die volle Höhe und Breite des Bildschirms ausnutzen.

Eine Mobilanwendung muss unterscheiden können, ob der Benutzer ein Steuerelement (z. B. eine Schaltfläche) oder den Bildlauf bedient. Bei Bildlaufleisten in Mobilanwendungen gilt es zu beachten, dass Flex-Komponenten häufig ihr Aussehen in Abhängigkeit von einer Benutzerinteraktion ändern.

Wenn der Benutzer zum Beispiel auf ein Button-Steuerelement drückt, wird die Schaltfläche anders dargestellt. So wird kenntlich gemacht, dass sie gerade gedrückt wird. Lässt der Benutzer die Schaltfläche wieder los, nimmt sie ihre ursprüngliche Darstellung an. Wenn der Benutzer beim Ausführen eines Bildlaufs jedoch den Bildschirm über der Schaltfläche berührt, sollte die Darstellung der Schaltfläche nicht geändert werden.

Der Adobe-Ingenieur Steven Shongrunden zeigt unter Saving scroll position between views in a mobile Flex Application ein Beispiel für die Verwendung der Bildlaufleisten.

Begriffe zum Bildlauf

Mit den folgenden Begriffen wird der Bildlauf in einer Mobilanwendung beschrieben:

Inhalt Bei einer bildlauffähigen Komponente wie einem Group-Container oder List-Steuerelement der gesamte Bereich der Komponente. Je nach Bildschirmgröße und Anwendungslayout ist möglicherweise nur eine Untergruppe des Inhalts sichtbar.

Viewport Die Untergruppe des Inhaltsbereichs einer Komponente, die zurzeit sichtbar ist.

Ziehen Eine Berührungsgeste, die erfolgt, wenn der Benutzer einen bildlauffähigen Bereich berührt und dann den Finger so bewegt, dass sich der Inhalt entsprechend der Geste bewegt.

Geschwindigkeit Die Geschwindigkeit und Richtung der Bewegung einer Zieh-Geste. Wird in Pixel pro Sekunde entlang der X- und Y-Achse gemessen.

Schubsen Eine Zieh-Geste, bei der der Benutzer den Finger hochhebt, sobald eine bestimmte Geschwindigkeit erreicht wurde und sich der bildlauffähige Inhalt weiter bewegt.

Springen Bei einer Zieh- oder Schubs-Geste kann der Viewport einer bildlauffähigen Komponente außerhalb deren Inhalts verschoben werden. Der Viewport zeigt dann einen leeren Bereich an. Wenn Sie den Finger wieder hochheben oder die Geschwindigkeit einer Schubs-Geste 0 erreicht, springt der Viewport zurück zum Ruhepunkt und wird mit Inhalt ausgefüllt. Die Bewegung wird langsamer, wenn der Viewport den Ruhepunkt erreicht, sodass sie sanft aufhört.

Bildlaufmodi in einer Mobilanwendung

Bildlauffähige Komponenten wie List und Scroller unterstützen verschiedene Bildlauftypen basierend auf der Einstellung der `pageScrollingEnabled`- und `scrollSnappingMode`-Eigenschaften der Komponente. Diese Eigenschaften sind nur gültig, wenn für den `interactionMode`-Stil `touch` festgelegt ist.

In der folgenden Tabelle werden die Bildlauftypen beschrieben:

pageScrollingEnabled	scrollSnappingMode	Modus
false (Standardwert)	none (Standardwert)	Standardmäßig ist der Bildlauf pixelbasiert. Die endgültige Bildlaufposition ist eine beliebige Pixelposition, die auf der Zieh- oder Schubs-Geste basiert. Wenn Sie beispielsweise in einem List-Steuerelement einen Bildlauf durchführen, endet der Bildlauf, sobald Sie den Finger hochheben, auch wenn ein List-Element teilweise sichbar ist.
false	leadingEdge, center, trailingEdge	Bildlauf ist pixelbasiert; der Inhalt wird jedoch basierend auf dem Wert von `scrollSnappingMode` an einer endgültigen Position ausgerichtet. Wenn Sie beispielsweise in einer Liste bei `scrollSnappingMode = leadingEdge` einen vertikalen Bildlauf durchführen, springt das List-Steuerelement zu einer endgültigen Bildlaufposition, wobei das obere Listenelement oben an der Liste ausgerichtet wird.
true	none	Bildlauf ist seitenbasiert. Die Größe des Viewports der bildlauffähigen Komponente bestimmt die Größe der Seite. Sie können nur für jeweils eine Seite einen Bildlauf durchführen, unabhängig von der Geste. Führen Sie einen Bildlauf durch mindestens 50 % des sichtbaren Bereichs der Komponente durch, damit er auf der nächsten Seite weitergeführt wird. Wenn Sie einen kürzeren Bildlauf durchführen, bleibt die Komponente auf der aktuellen Seite. Wenn die Geschwindigkeit des Bildlaufs hoch genug ist, kann alternativ auch die nächste Seite angezeigt werden. Ist die Geschwindigkeit nicht hoch genug, bleibt die Komponente auf der aktuellen Seite. Wenn die Inhaltsgröße kein genaues Vielfaches der Viewport-Größe ist, wird der letzten Seite eine zusätzliche Auffüllung hinzugefügt, sodass sie vollständig in den Viewport passt.
true	leadingEdge, center, trailingEdge	Bildlauf ist seitenbasiert; die Komponente wird jedoch basierend auf dem Wert von `scrollSnappingMode` an einer endgültigen Position ausgerichtet. Um sicherzustellen, dass der Ausrichtungsmodus berücksichtigt wird, entspricht der Bildlaufabstand nicht immer exakt der Größe der Seite.

Bildlaufbeispiele in einer Mobilanwendung

Im folgenden Beispiel wird eine Scroller-Komponente verwendet, um einen Group-Container in einer Mobilanwendung einzuschließen. Der Group-Container besitzt als untergeordnetes Element ein Image-Steuerelement, das ein großes Bild enthält. Indem Sie den Group-Container im Scroller einschließen, können Sie im Bild einen Bildlauf durchführen:

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

Beachten Sie, dass in diesem Beispiel Einstellungen für die `pageScrollingEnabled`- und `scrollSnappingMode`-Eigenschaften weggelassen werden. Daher wird hier der standardmäßige pixelbasierte Bildlaufmodus verwendet und Sie können einen Bildlauf zu einer beliebigen Pixelposition im Bild durchführen.

Im nächsten Beispiel ist ein List-Steuerelement dargestellt, das die `pageScrollingEnabled`- und `scrollSnappingMode`-Eigenschaften festlegt:

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

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

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

        ]]>
    </fx:Script>

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

In diesem Beispiel wird ein seitenbasierter Bildlauf mit einer Ausrichtungseinstellung von `leadingEdge` verwendet. Daher kann beim Durchführen eines Bildlaufs in der Liste jeweils eine Seite durchlaufen werden. Beim Wechseln der Seite springt das Steuerelement zu einer endgültigen Bildlaufposition, wobei das obere Listenelement oben an der Liste ausgerichtet wird.

Überlegungen zum Bildlauf mit StageText

Mit StageText können Sie in einer Mobilanwendung native Texteingaben anstelle der standardmäßigen Textfeldsteuerelemente verwenden. Allerdings kann ein bildlauffähiger Container kein Texteingabesteuerelement wie TextInput oder TextArea, das StageText verwendet, beinhalten. Weisen Sie daher zum Verwenden eines Texteingabesteuerelements in einem bildlauffähigen Container dem Steuerelement eine neue Skin zu, sodass dieses nicht StageText verwendet.

Flex wird mit Skins für die TextInput- und TextArea-Steuerelemente bereitgestellt, die nicht auf StageText basieren. Verwenden Sie die folgenden Skins mit diesen Steuerelementen in einem bildlauffähigen Container:

- spark.skins.mobile.TextInputSkin Skin für TextInput, die nicht StageText verwendet.
- spark.skins.mobile.TextAreaSkin Skin für TextArea, die nicht StageText verwendet.

Das folgende Beispiel zeigt einen View-Container, der ein TextInput- und ein TextArea-Steuerelement in einem bildlauffähigen Container verwendet:

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

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

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

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

Ereignisse und Bildlaufleisten

Flex-Komponenten erkennen anhand von Ereignissen, dass es eine Benutzerinteraktion gab. Die Komponente kann daraufhin ihr Aussehen der Interaktion entsprechend ändern oder eine andere Aktion ausführen.

Anwendungsentwickler bedienen sich Ereignissen, um Benutzerinteraktionen verarbeiten zu können. So verwenden sie zum Beispiel das `click`-Ereignis des Button-Steuerelements, um als Reaktion auf die Auswahl der Schaltfläche durch den Benutzer eine Ereignisprozedur auszuführen. Mit dem `change`-Ereignis des List-Steuerelements wird bei Auswahl eines Listenelements eine Ereignisprozedur ausgeführt.

Der Bildlaufmechanismus von Flex wird vom Ereignis `mouseDown` bestimmt. Das heißt, der Bildlaufmechanismus achtet auf `mouseDown`-Ereignisse, um zu bestimmen, ob ein Bildlaufvorgang eingeleitet werden soll.

Gesten als Bildlaufvorgang interpretieren

Eine Anwendung besteht aus mehreren Button-Steuerelementen in einem bildlauffähigen Container:

1. Mit dem Finger wird eine Schaltfläche gedrückt. Die Schaltfläche löst ein `mouseDown`-Ereignis aus.
2. Daraufhin folgt eine zeitliche Verzögerung, die von Flex erzeugt wird. Durch die Verzögerung wird sichergestellt, dass der Benutzer die Schaltfläche gedrückt hat und nicht den Bildlauf aktivieren wollte.

   Wenn in dieser Zeit der Finger um mehr als einen bestimmten Wert verschoben wird, versteht Flex diese Bewegung als Scrollen. Dieser Abstand beträgt etwa 2,03 cm (0,08 Zoll). Dieser Abstand entspricht bei einem Gerät mit 252 DPI etwa 20 Pixel.

   Da der Finger noch vor Ablauf der Verzögerung bewegt wurde, hat das Steuerelement die Interaktion nicht als Tastendruck erkannt. Daher meldet das Steuerelement kein Ereignis oder ändert das Aussehen.
3. Nach Ablauf der Verzögerung erkennt das Steuerelement die Benutzerinteraktion. Die Darstellung der Schaltfläche ändert sich, um darauf hinzuweisen, dass sie betätigt wurde.

   Mit der Eigenschaft `touchDelay` des Steuerelements legen Sie die Dauer der Verzögerung fest. Der Standardwert ist 100 ms. Wird die Eigenschaft `touchDelay` auf 0 gesetzt, gibt es keine Verzögerung und der Bildlauf wird sofort ausgelöst.

4 Nach Ablauf der Verzögerung löst Flex die Maus-Ereignisse aus und der Benutzer kann den Finger über die 20 Pixel hinaus bewegen. Die Schaltfläche nimmt ihren ursprünglichen Zustand wieder an, und der Bildlauf wird eingeleitet.

   In diesem Fall hat die Schaltfläche ihr Aussehen geändert, weil die Verzögerungszeit abgelaufen ist. Wird der Finger jedoch nach Ablauf der Verzögerung über die 20 Pixel hinaus bewegt, interpretiert Flex diese Bewegung als Scrollen.

Hinweis: *Flex-Komponenten unterstützen neben Mausereignissen noch viele weitere Arten von Ereignissen. Bei der Arbeit mit Komponenten entscheiden Sie, wie Ihre Anwendung auf Ereignisse reagieren soll. Zur Zeit des Ereignisses* `mouseDown` *ist das intendierte Benutzerverhalten zweideutig. Der Benutzer könnte mit der Komponente interagieren wollen, er könnte aber auch einen Bildlauf ausführen wollen. Wegen dieser Zweideutigkeit empfiehlt es sich, auf die Ereignisse* `click` *oder* `mouseUp` *zu achten anstatt auf* `mouseDown`*.*

Bildlaufereignisse in einer Mobilanwendung

Um den Beginn eines Bildlaufvorgangs zu signalisieren, löst die Komponente, die das `mouseDown`-Ereignis auslöst, eine brodelndes `touchInteractionStarting`-Ereignis aus. Wenn das Ereignis nicht abgebrochen wird, löst die Komponente ein brodelndes `touchInteractionStart`-Ereignis aus.

Wenn eine Komponente ein `touchInteractionStart`-Ereignis erkennt, darf es nicht auf die Benutzergeste reagieren. Wenn zum Beispiel ein Button-Steuerelement ein `touchInteractionStart`-Ereignis erkennt, werden visuelle Anzeigen, die es basierend auf dem auslösenden `mouseDown`-Ereignis setzt, deaktiviert.

Wenn eine Komponenten den Bildlauf nicht einleitet, kann sie die Methode `preventDefault()` in der Ereignisprozedur für das `touchInteractionStarting`-Ereignis aufrufen.

Wenn der Scrollvorgang abgeschlossen ist, löst die Komponente, die das `mouseDown`-Ereignis auslöst, ein brodelndes `touchInteractionEnd`-Ereignis aus.

Scrollverhalten basierend auf dem Berührungspunkt

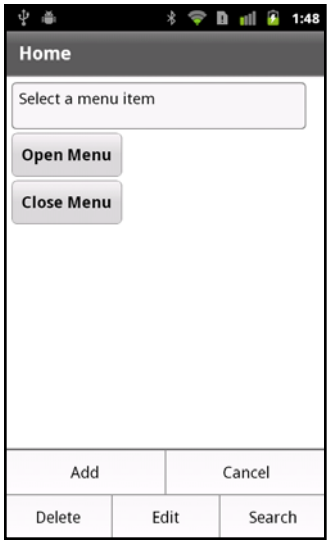
Die folgende Tabelle beschreibt das Bildlaufverhalten in Abhängigkeit von der Position des Berührungspunkts:

Ausgewähltes Objekt	**Verhalten**
Leerer Bereich Nicht editierbarer Text Nicht auswählbarer Text	Keine Komponente erkennt diese Bewegung. Der Scroller wartet darauf, dass der Benutzer den Finger um mehr als 20 Pixel bewegt, bevor der Bildlauf ausgeführt wird.
Element in einer Liste	Nach Ablauf der Verzögerung ändert der Elementrenderer das Aussehen des ausgewählten Elements in den ausgewählten Status. Sollte sich der Finger jedoch über die 20 Pixel hinaus bewegen, nimmt das Element wieder seine ursprüngliche Erscheinung an und das Scrolling setzt ein.
Button, CheckBox, RadioButton, DropDownList	Nach Ablauf der Verzögerung liegt der Status `mouseDown` vor. Sollte sich der Finger jedoch über die 20 Pixel hinaus bewegen, nimmt das Steuerelement wieder seine ursprüngliche Erscheinung an, und der Bildlauf wird gestartet.
Schaltflächenkomponente innerhalb eines Listenelementrenderers	Der Elementrenderer wird nie hervorgehoben. Die Bewegung wird vom Button oder vom Scroller verarbeitet, genau wie bei einer normalen Schaltfläche.

Menüs in Mobilanwendungen definieren

Im ViewMenu-Container wird ein Menü definiert, das sich in einer Mobilanwendung unten in einem View-Container befindet. Jeder View-Container definiert ein eigenes, für die betreffende Ansicht spezifisches Menü.

Die folgende Abbildung zeigt den ViewMenu-Container in einer Anwendung:



Im ViewMenu-Container wird ein Menü mit einer einzigen Hierarchie von Menüschaltflächen definiert. Das heißt, es können keine Untermenüs erstellt werden.

Die untergeordneten Elemente des ViewMenu-Containers werden als ViewMenuItem-Steuerelemente definiert. Jedes ViewMenuItem-Steuerelement steht für eine Schaltfläche im Menü.

Benutzerinteraktion mit dem ViewMenu-Container

Das Menü wird über die Menütaste am Mobilgerät geöffnet. Sie können es auch programmgesteuert öffnen.

Mit Auswahl einer Menüschaltfläche wird das Menü geschlossen. Das Steuerelement ViewMenuItem sendet ein `click`-Ereignis, wenn die Menüschaltfläche betätigt wird.

Wenn das Menü geöffnet ist, kann es durch Drücken der Zurück- oder Menütaste geschlossen werden. Das Menü schließt sich auch, wenn der Bildschirm außerhalb des Menüs berührt wird.

Das Winkelzeichen kennzeichnet jeweils die hervorgehobene Menüschaltfläche. Mit den Navigations- oder Pfeiltasten des Geräts ändern Sie das Winkelzeichen. Drücken Sie die Eingabetaste des Geräts oder die Navigationstaste, um das Winkelzeichenelement auszuwählen und das Menü zu schließen.

Menü in einer Mobilanwendung erstellen

Mit der Eigenschaft `View.viewMenuItems` wird das Menü für eine Ansicht definiert. Die Eigenschaft `View.viewMenuItems` übernimmt einen Vektor der ViewMenuItem-Steuerelemente, wie im folgenden Beispiel zu sehen:

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

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

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

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

In diesem Beispiel werden über die Eigenschaft `View.viewMenuItems` fünf Menüelemente hinzugefügt, wobei jedes Menüelement von einem ViewMenuItem-Steuerelement dargestellt wird. In jedem ViewMenuItem-Steuerelement ist in der Eigenschaft `label` der Text enthalten, der für das jeweilige Element im Menü angezeigt wird.

Der ViewMenu-Container wird nicht explizit definiert. Der View-Container erstellt automatisch eine Instanz des ViewMenu-Containers, in dem sich die ViewMenuItem-Steuerelemente befinden.

Den Stil `icon` des ViewMenuItem-Steuerelements übernehmen

Im Steuerelement ViewMenuItem wird die Stileigenschaft `icon` definiert, mit der Sie ein Bild einfügen können. Sie können den Stil `icon` mit oder ohne `label`-Eigenschaft verwenden.

Das `click`-Ereignis des Steuerelements ViewMenuItem verarbeiten

In jedem ViewMenuItem-Steuerelement ist auch eine Ereignisprozedur für das `click`-Ereignis festgelegt. Wenn ein Element ausgewählt wird, sendet ViewMenuItem das `click`-Ereignis. In diesem Beispiel verwenden alle Menüelemente dieselbe Ereignisprozedur. Es kann jedoch auch für jedes `click`-Ereignis eine eigene Ereignisprozedur verwendet werden.

Das ViewMenuItem-Steuerelement softwareseitig öffnen

Das Menü wird in der Regel mit der Menütaste des Mobilgeräts geöffnet. Diese Anwendung enthält jedoch auch zwei Button-Steuerelemente zum softwareseitigen Öffnen des Menüs.

Um das Menü softwareseitig zu öffnen, setzen Sie die Eigenschaft `viewMenuOpen` des Anwendungscontainers auf `true`. Zum Schließen des Menüs setzen Sie die Eigenschaft auf `false`. Die Eigenschaft `viewMenuOpen` wird in der Klasse ViewNavigatorApplicationBase definiert. Dies ist die Basisklasse der Container ViewNavigatorApplication und TabbedViewNavigatorApplication.

Skins auf die ViewMenu- und ViewMenuItem-Komponenten anwenden

Skins bestimmten das Aussehen der Komponenten ViewMenu und ViewMenuItem. Die standardmäßige Skinklasse für ViewMenu lautet spark.skins.mobile.ViewMenuSkin. Die standardmäßige Skinklasse für ViewMenuItem lautet spark.skins.mobile.ViewMenuItemSkin.

Blogger Daniel Demmel zeigt, wie man eine Skin für die ViewMenu-Steuerung erstellt, die dann wie Gingerbread Black aussieht.

Über Skinstatus wie „normal“, „geschlossen“ und „deaktiviert“ wird die Darstellung in den verschiedenen Zuständen simuliert. In Skins ist auch die Übergangsdarstellung zwischen zwei Anzeigezuständen des Menüs festgelegt.

Weitere Informationen finden Sie unter „Grundlagen des Mobilskinnings“ auf Seite 173.

Layout eines ViewMenu-Containers festlegen

Die Klasse ViewMenuLayout definiert das Layout des Ansichtmenüs. Ein Menü kann je nach Anzahl der Menüelemente mehrere Zeilen enthalten.

Regeln für das ViewMenuItem-Layout

Die Eigenschaft `requestedMaxColumnCount` der Klasse ViewMenuLayout bestimmt die maximale Anzahl an Menüelementen in einer Zeile. Standardmäßig ist die Eigenschaft `requestedMaxColumnCount` auf den Wert 3 gesetzt.

Die folgenden Regeln bestimmten die Gestaltung des Layouts der Klasse ViewMenuLayout:

- Werden drei oder weniger Menüelemente festgelegt, wobei die Eigenschaft `requestedMaxColumnCount` standardmäßig auf 3 gesetzt ist, werden die Menüelemente in einer einzigen Zeile angezeigt. Jedes Menüelement hat die gleiche Größe.

  Werden vier oder mehr Menüelemente festgelegt, also mehr Elemente als in der Eigenschaft `requestedMaxColumnCount` angegeben, erstellt der ViewMenu-Container mehrere Zeilen.

- Ist die Anzahl der Menüelemente durch den in `requestedMaxColumnCount` festgelegten Wert sauber teilbar, erhält jede Zeile die gleiche Anzahl an Menüelementen. Jedes Menüelement hat die gleiche Größe.

Beispiel: Für die `requestedMaxColumnCount`-Eigenschaft ist der Standardwert „3“ festgelegt, und Sie definieren sechs Menüelemente. Dadurch wird ein Menü mit zwei Zeilen und je drei Menüelementen erzeugt.

- Wenn die Anzahl der Menüelemente sich nicht durch den Wert in `requestedMaxColumnCount` teilen lässt, erhalten die Zeilen eine unterschiedliche Anzahl an Menüelementen. Die Größe des Menüelements wird von der Anzahl an Menüelementen pro Zeile bestimmt.

  Beispiel: Für die `requestedMaxColumnCount`-Eigenschaft ist der Standardwert „3“ festgelegt, und Sie definieren acht Menüelemente. Dadurch wird ein Menü mit drei Zeilen erzeugt. Die erste Zeile erhält zwei Menüelemente. Die zweite und die dritte Zeile enthält je drei Elemente.

Eigenes ViewMenuItem-Layout erstellen

Die Klasse ViewMenuLayout enthält Eigenschaften, mit denen die Lücken zwischen den Menüelementen und die Standardanzahl an Menüelementen pro Zeile geändert werden können. Sie können auch ein eigenes Layout für das Menü erstellen. Hierzu müssen Sie Ihre eigene Layout-Klasse erstellen.

Standardmäßig ist in der Klasse spark.skins.mobile.ViewMenuSkin der Skin für den ViewMenu-Container definiert. Um eine selbstdefinierte ViewMenuLayout-Klasse auf den ViewMenu-Container zu übertragen, muss eine neue Skinklasse für den ViewMenu-Container definiert werden.

Die standardmäßige ViewMenuSkin-Klasse enthält wie im folgenden Beispiel veranschaulicht eine Definition für einen Group-Container namens `contentGroup`:

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

In Ihrer Skinklasse muss auch ein Container namens `contentGroup` definiert werden. Die benutzerdefinierte Layout-Klasse wird im Container durch die Eigenschaft `layout` angegeben.

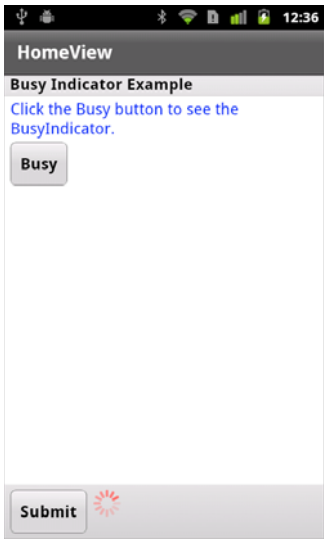
Dann können Sie Ihre benutzerdefinierte Skinklasse wie im folgenden Beispiel veranschaulicht in der Anwendung übernehmen:

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

BusyIndicator-Steuerelement für lange dauernde Aktivitäten in einer Mobilanwendung anzeigen

Das Spark-Steuerelement BusyIndicator zeigt ein sich drehendes Rad mit 12 Speichen an. Mit BusyIndicator wird sichtbar gemacht, dass ein länger dauernder Vorgang ausgeführt wird.

Die folgende Abbildung zeigt das BusyIndicator-Steuerelement im Steuerelementbereich eines Spark Panel-Containers, neben der Schaltfläche „Submit“:



Machen Sie das BusyIndicator-Steuerelement bei längeren Vorgängen sichtbar. Nach Abschluss des Vorgangs soll das Steuerelement wieder ausgeblendet werden.

Beispielsweise können Sie eine Instanz des BusyIndicator-Steuerelements in einer Ereignisprozedur erstellen, idealerweise in der Ereignisprozedur, mit der der lange dauernde Vorgang gestartet wird. Rufen Sie in der Ereignisprozedur die `addElement()`-Methode auf, um das Steuerelement in einen Container aufzunehmen. Wenn der Prozess abgeschlossen ist, rufen Sie `removeElement()` auf, um das BusyIndicator-Steuerelement aus dem Container zu entfernen.

Oder nutzen Sie die `visible`-Eigenschaft zum Ein- und Ausblenden des Elements. Im folgenden Beispiel wird das BusyIndicator-Steuerelement in die Steuerzeile eines Spark Panel-Containers in einem View-Container eingefügt:

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

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

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

In diesem Beispiel wird die `visible`-Eigenschaft zuerst auf `false` gesetzt, um sie auszublenden. Soll das Steuerelement wieder angezeigt werden, klicken Sie auf die Busy-Schaltfläche, um die `visible`-Eigenschaft auf `true` zu setzen.

Das BusyIndicator-Steuerelement dreht sich nur, wenn es sichtbar ist. Wird die `visible`-Eigenschaft auf `false` festgelegt, benötigt das Steuerelement keine Verarbeitungszyklen.

Hinweis: *Wird die `visible`-Eigenschaft auf `false` gesetzt, so wird das Steuerelement zwar ausgeblendet, befindet sich jedoch weiterhin im Layout des übergeordneten Containers. Um das Steuerelement aus dem Layout auszuschließen, setzen Sie die Eigenschaften `visible` und `includeInLayout` auf `false`.*

Das Spark-Steuerelement BusyIndicator unterstützt keine Skins. Sie können Stile verwenden, um die Farbe und das Drehintervall des Rads zu ändern. Im vorangegangenen Beispiel wird die Farbe der Anzeige über die `symbolColor`-Eigenschaft festgelegt.

Wechselschalter einer Mobilanwendung hinzufügen

Das ToggleSwitch-Spark-Steuerelement definiert einen einfachen binären Schalter. Das Steuerelement besteht aus einem Schieberegler und einer Spur, über die Sie den Schieberegler schieben.

Das ToggleSwitch-Steuerelement ähnelt den ToggleButton- und CheckBox-Steuerelementen. Mit all diesen Steuerelementen können Sie zwischen den Werten „ausgewählt“ und „nicht ausgewählt“ wechseln.

Im folgenden Bild ist das ToggleSwitch-Steuerelement in einer Anwendung dargestellt:

Das ToggleSwitch-Steuerelement weist zwei Positionen auf: ausgewählt und nicht ausgewählt. Das Steuerelement befindet sich in der Position „nicht ausgewählt“, wenn sich der Schieberegler links befindet. Die Position „ausgewählt“ liegt vor, wenn sich der Schieberegler rechts befindet. In der Darstellung befindet sich der Schieberegler in der Position „nicht ausgewählt“.

Wird auf eine beliebige Stelle im Steuerelement geklickt, wird die Position geändert. Sie können den Schieberegler auch entlang der Spur schieben, um die Position zu ändern. Wenn Sie den Schieberegler loslassen, wird dieser an die Position (ausgewählt oder nicht ausgewählt) geschoben, der er sich am nächsten befindet.

Standardmäßig entspricht die Beschriftung OFF der Position „nicht ausgewählt“ und ON der Position „ausgewählt“.

ToggleSwitch-Steuerelement erstellen

Im Folgenden ist ein View-Container dargestellt, der das ToggleSwitch-Steuerelement der vorigen Abbildung definiert:

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

In diesem Beispiel wird ON oder OFF im ersten Label-Steuerelement basierend auf der Schiebereglerposition angezeigt. Das zweite Label-Steuerelement zeigt die aktuelle Position des Schiebereglers als Wert zwischen 0.0 (nicht ausgewählt) und 1.0 (ausgewählt) an.

In diesem Beispiel ist außerdem der `slideDuration`-Stil auf 1000 festgelegt. Dieser Stil bestimmt die Dauer der Animation des Schiebereglers (in Millisekunden), wenn dieser zwischen den zwei Positionen verschoben wird.

Im Folgenden ist die Hauptanwendungsdatei dargestellt:

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

Callout-Standardkomponente eines ToggleSwitch-Steuerelements ändern

Im vorigen Beispiel verwendet das ToggleSwitch-Steuerelement die Standardwerte für die Beschriftungen für „nicht ausgewählt“ und „ausgewählt“: OFF (nicht ausgewählt) und ON (ausgewählt). Um die Beschriftungen oder andere visuelle Merkmale des Steuerelements anzupassen, definieren Sie eine Skinklasse als Unterklasse von spark.skins.mobile.ToggleSwitchSkin oder erstellen Sie Ihre eigene Skinklasse.

Die folgende Skinklasse ändert die Beschriftungen in „Yes“ und „No“:

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

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

Der folgende View-Container verwendet diese Skinklasse:

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

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

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

Callout-Container einer Mobilanwendung hinzufügen

Eine Callout-Komponente in einer Mobilanwendung ist ein Container, der in einer Anwendung eingeblendet wird. Der Container kann mindestens eine Komponente beinhalten und verschiedene Layouttypen unterstützen.

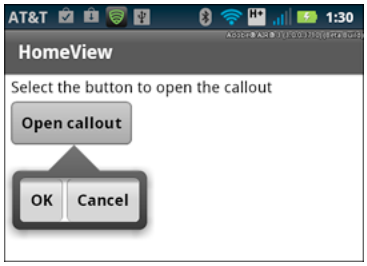
Ein Callout-Container kann modal oder nichtmodal sein. Ein modaler Container übernimmt alle Tastatur- und Mauseingaben, bis er geschlossen wird. Ein nichtmodaler Container ermöglicht es anderen Komponenten, Eingaben zu übernehmen, während der Container geöffnet ist.

Flex bietet zwei Komponenten, die Sie zum Hinzufügen von Callout-Containern zu einer Mobilanwendung verwenden können: CalloutButton und Callout.

CalloutButton-Steuerelement zum Erstellen eines Callout-Containers verwenden

Das CalloutButton-Steuerelement stellt eine einfache Methode zur Erstellung eines Callout-Containers bereit. Mit der Komponente können Sie die Komponenten, die im Callout angezeigt werden, definieren und das Container-Layout einstellen.

Wenn Sie das CalloutButton-Steuerelement auswählen, öffnet das Steuerelement den Callout-Container. Flex zeichnet automatisch einen Pfeil vom Callout-Container zurück zum CalloutButton-Steuerelement, wie die folgende Abbildung zeigt:



Das nachstehende Beispiel zeigt die Mobilanwendung, die das in der vorherigen Abbildung dargestellte CalloutButton-Steuerelement erstellt:

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

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

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

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

Das CalloutButton-Steuerelement definiert zwei Button-Steuerelemente, die im Callout-Container angezeigt werden. Das CalloutButton-Steuerelement gibt außerdem an, dass das HorizontalLayout als Layout des Callout-Containers zu verwenden ist. Standardmäßig verwendet der Container BasicLayout.

Callout-Container mit dem CalloutButton-Steuerelement öffnen und schließen

Der Callout-Container öffnet sich, wenn das CalloutButton-Steuerelement ausgewählt oder die `CalloutButton.openDropDown()`-Methode aufgerufen wird. Die `horizontalPosition`- und `verticalPosition`-Eigenschaften bestimmen die Position des Callout-Containers relativ zum CalloutButton-Steuerelement. Ein Beispiel finden Sie unter „Einen Callout-Container anpassen und positionieren“ auf Seite 94.

Der vom CalloutButton-Steuerelement geöffnete Callout-Container ist immer nichtmodal. Das heißt, dass andere Komponenten in der Anwendung Eingabe aufnehmen können, während die Callout-Komponente geöffnet ist. Verwenden Sie den Callout-Container zum Erstellen einer modalen Callout-Komponente.

Der Callout-Container bleibt geöffnet, bis Sie auf eine Stelle außerhalb des Callout-Containers klicken oder die `CalloutButton.closeDropDown()`-Methode aufrufen. In diesem Beispiel wird die `closeDropDown()`-Methode in der Ereignisprozedur für das `click`-Ereignis für die zwei Button-Steuerelemente im Callout-Container aufgerufen.

Callout-Container zum Erstellen eines Callouts verwenden

Das CalloutButton-Steuerelement enthält in einem einzigen Steuerelement den Callout-Container und die notwendige Logik zum Öffnen und Schließen der Callout-Komponente. Das CalloutButton-Steuerelement wird dann als *host* des Callout-Containers bezeichnet.

Sie können den Callout-Container außerdem in einer Mobilanwendung verwenden. Der Vorteil eines Callout-Containers ist, dass er nicht mit einem einzelnen Host verknüpft ist und daher an einer beliebigen Stelle in der Anwendung wiederverwendbar ist.

Verwenden Sie die `Callout.open()`- und `Callout.close()`-Methoden zum Öffnen eines Callout-Containers, normalerweise als Reaktion auf ein Ereignis. Wenn Sie die `open()`-Methode aufrufen, können Sie ein optionales Argument übergeben, um anzugeben, dass der Callout-Container modal ist. Standardmäßig ist der Callout-Container nichtmodal.

Die Position des Callout-Containers ist relativ zur Host-Komponente. Die `horizontalPosition`- und `verticalPosition`-Eigenschaften bestimmen die Position des Containers relativ zum Host. Ein Beispiel finden Sie unter „Einen Callout-Container anpassen und positionieren“ auf Seite 94.

Da es sich um ein Popup handelt, müssen Sie keinen Callout-Container als Teil des normalen MXML-Layoutcodes Ihrer Anwendung erstellen. Stattdessen definieren Sie den Callout-Container als eine benutzerdefinierte MXML-Komponente in einer MXML-Datei.

Definieren Sie im folgenden Beispiel einen Callout-Container in der Datei „MyCallout.mxml“ im Kompositionenordner Ihrer Anwendung:

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

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

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

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

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

MyCallout.mxml definiert ein einfaches Popup, damit der Benutzer Vor- und Nachnamen eingeben kann. Beachten Sie, dass die Schaltflächen die `close()`-Methode aufrufen, um den Callout als Reaktion auf ein `click`-Ereignis zu schließen.

Das folgende Beispiel zeigt einen View-Container, der „MyCallout.mxml“ als Reaktion auf ein `click`-Ereignis öffnet:

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

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

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

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

Importieren Sie zuerst die Komponente „MyCallout.mxml“ in die Anwendung. Die Schaltfläche „calloutB“ erstellt als Reaktion auf ein `click`-Ereignis eine Instanz von „MyCallout.mxml“ und ruft dann die `open()`-Methode auf.

Die `open()`-Methode gibt zwei Argumente an. Das erste Argument gibt an, dass „calloutB“ die Host-Komponente der Callout-Komponente ist. Daher positioniert sich die Callout-Komponente in der Anwendung relativ zur Position von „calloutB“. Das zweite Argument ist `true`, um eine modale Callout-Komponente zu erstellen.

Inneren Callout-Container definieren

Sie müssen den Callout-Container nicht in einer separaten Datei definieren. Das folgende Beispiel verwendet das `<fx:Declaration>`-Tag um ihn als innere Komponente eines View-Containers zu definieren:

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

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

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

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

Daten zurück an den Callout-Container übergeben

Verwenden Sie die `close()`-Methode des Callout-Containers, um Daten zurück an die Hauptanwendung zu übergeben. Die `close()`-Methode hat folgende Signatur:

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

Dabei gilt:

- `commit` enthält `true`, wenn die Anwendung die zurückgegebenen Daten übernehmen soll.
- `data` gibt die zurückgegebenen Daten an.

Das Aufrufen der `close()`-Methode löst ein `close`-Ereignis aus. Das Ereignisobjekt, das mit dem `close`-Ereignis verknüpft ist, ist ein Objekt des Typs spark.events.PopUpEvent. Die PopUpEvent-Klasse definiert zwei Eigenschaften, `commit` und `data`, die die Werte der entsprechenden Argumente der `close()`-Methode enthalten. Verwenden Sie diese Eigenschaften in der Ereignisprozedur des `close`-Ereignisses, um beliebige Daten, die von der Callout-Komponente zurückgegeben wurden, zu inspizieren.

Der Callout-Container ist eine Unterklasse der SkinnablePopUpContainer-Klasse, die denselben Mechanismus verwendet, um Daten an die Hauptanwendung zurückzugeben. Ein Beispiel zum Zurückgeben von Daten vom SkinnablePopUpContainer-Container finden Sie unter Zurückgeben von Daten vom Spark SkinnablePopUpContainer-Container.

Das folgende Beispiel ändert die oben gezeigte Callout-Komponente, um die Werte für den Vor- und Nachnamen zurückzugeben:

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

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

            public var retData:String = new String();

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

        ]]>
```

```
    </fx:Script>

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

In diesem Beispiel erstellen Sie eine Zeichenfolge, um den Vor- und Nachnamen als Reaktion auf den Benutzer zurückzugeben, der die Schaltfläche „OK“ auswählt.

Der View-Container verwendet dann das `close`-Ereignis beim Callout, um die zurückgegebenen Daten anzuzeigen:

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

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

            public var myCallout:MyCalloutPassBack = new MyCalloutPassBack();

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

            // Handle the close event from the Callout.
            protected function closeHandler(event:PopUpEvent):void {
```

```
                // If commit is false, no data is returned.
                if (!event.commit)
                    return;

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

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

        ]]>
    </fx:Script>

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

ViewNavigator zur Callout-Komponente hinzufügen

Sie können einen ViewNavigator in einem Callout-Container verwenden. Mit dem ViewNavigator können Sie der Callout-Komponente eine Aktionsleiste und mehrere Ansichten hinzufügen.

Beispielsweise öffnet die folgende Ansicht einen Callout-Container, der in der Datei „MyCalloutPassBackVN“definiert ist:

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

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

            public var myCallout:MyCalloutPassBackVN = new MyCalloutPassBackVN();

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

```
            }

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

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

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

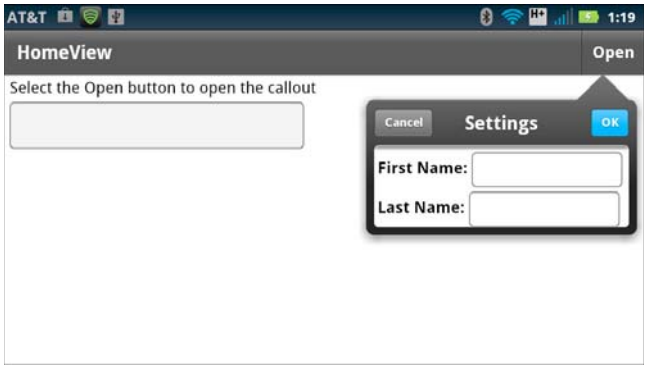
Die Datei „MyCalloutPassBackVN.mxml“ definiert den Callout-Container, der einen ViewNavigator-Container enthält:

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

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

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

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

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

In der Datei „MyCalloutPassBackVN.mxml“ geben Sie an, dass die erste Ansicht des ViewNavigators SettingsView ist. SettingsView definiert TextInput-Steuerelemente für den Vor- und Nachnamen eines Benutzers. Wenn der Benutzer die Schaltfläche „OK“ auswählt, können Sie die Callout-Komponente schließen und zurückgegebene Daten an die Datei „MyCalloutPassBackVN“ übergeben.

Hinweis: *Wenn in einem Callout-Container ein ViewNavigator angezeigt wird, hat die ActionBar einen transparenten Hintergrund. In diesem Beispiel setzen Sie für den Callout-Container* `contentBackgroundAppearance` *auf* `none`*. Diese Einstellung verhindert, dass die standardmäßig weiße* `contentBackgroundColor` *der Callout-Komponente im Bereich der transparenten ActionBar angezeigt wird.*

Die folgende Abbildung zeigt die Anwendung mit geöffneter Callout-Komponente:



Nachfolgend wird SettingsView.mxml angezeigt:

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

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

Hinweis: *Die ActionBar, die von einem ViewNavigator in einem Callout-Container definiert wird, hat einen transparenten Hintergrund. Standardmäßig wird ein Übergang von einer View zu einer anderen View in der Callout-Komponente ordnungsgemäß angezeigt. Wenn Sie jedoch einen nicht standardmäßigen Übergang angeben, wie einen CrossFadeViewTransition oder einen ZoomViewTransition, kann der ActionBar-Bereich der zwei Views überlappen. Um dieses Problem zu umgehen, erstellen Sie eine benutzerdefinierte Skinklasse für die ActionBar und die Callout-Komponente, die einen nichttransparenten Hintergrund verwendet.*

Einen Callout-Container anpassen und positionieren

Das CalloutButton-Steuerelement und der Callout-Container verwenden zwei Eigenschaften, um die Position des Callout-Containers relativ zu seinem Host zu definieren: `horizontalPosition` und `verticalPosition`. Diese Eigenschaften können folgende Werte haben: "`before`", "`start`", "`middle`", "`end`", "`after`" und "`auto`" (Standard).

Beispielsweise können Sie diese Eigenschaften wie nachfolgend beschrieben einstellen:

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

Der Callout-Container wird links und unterhalb der Host-Komponente geöffnet. Stellen Sie ihn wie nachfolgend ein:

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

Der Callout-Container wird oberhalb der Host-Komponente geöffnet, wobei die Mitte der Callout-Komponente an der Mitte der Host-Komponente ausgerichtet wird.

Pfeil von der Callout-Komponente zum Host ziehen

Außer bei fünf Kombinationen der Eigenschaften `horizontalPosition` und `verticalPosition` zeichnet die Callout-Komponente einen Pfeil, der auf den Host zeigt. Die Positionen, bei denen kein Pfeil angezeigt wird, treten auf, wenn die Callout-Komponente sich zentriert über der Mitte des Hosts und in einer Ecke befindet. Die folgenden Kombinationen zeigen keinen Pfeil:

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

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

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

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

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

Beim Callout-Container bestimmen die `horizontalPosition`- und `verticalPosition`-Eigenschaften außerdem den Wert der schreibgeschützten `Callout.arrowDirection`-Eigenschaft. Die Position des Callout-Containers relativ zum Host bestimmt den Wert der `arrowDirection`-Eigenschaft. Mögliche Werte sind u. a. `"up"` und `"left"`.

Der `Callout.arrow`-Skinteil verwendet den Wert der `arrowDirection`-Eigenschaft, um den Pfeil basierend auf der Position der Callout-Komponente zu zeichnen.

Speicherplatz für einen Callout-Container verwalten

Beim Verwenden eines Callout-Containers sollte beachtet werden, wie der von der Callout-Komponente verwendete Speicherplatz verwaltet wird. Wenn Sie beispielsweise den für die Anwendung verwendeten Speicherplatz reduzieren möchten, erstellen Sie jedes Mal eine Instanz der Callout-Komponente, wenn diese geöffnet wird. Die Callout-Komponente wird dann beim Schließen gelöscht. Stellen Sie aber sicher, dass Sie alle Verweise auf die Callout-Komponente, besonders Ereignisprozeduren, entfernen. Ansonsten wird die Callout-Komponente nicht gelöscht.

Alternativ können Sie dieselbe Callout-Komponente mehrere Male in der Anwendung verwenden, wenn es sich um einen relativ kleinen Callout-Container handelt. In dieser Konfiguration erstellt die Anwendung eine einzelne Instanz der Callout-Komponente. Anschließend wird diese Instanz wiederverwendet und die Callout-Komponente bleibt im Arbeitsspeicher. Diese Konfiguration reduziert die Ausführungszeit in der Anwendung, da die Anwendung die Callout-Komponente nicht bei jedem Öffnen neu erstellen muss.

Arbeitsspeicher mit dem CalloutButton-Steuerelement verwalten

Um die vom CalloutButton-Steuerelement verwendete Callout-Komponente zu konfigurieren, legen Sie die `CalloutButton.calloutDestructionPolicy`-Eigenschaft fest. Ein Wert `"auto"` konfiguriert das Steuerelement, um die Callout-Komponente nach dem Schließen zu löschen. Ein Wert `"never"` konfiguriert das Steuerelement, um die Callout-Komponente im Arbeitsspeicher zwischenzuspeichern.

Arbeitsspeicher mit dem Callout-Container verwalten

Der Callout-Container definiert nicht die `calloutDestructionPolicy`-Eigenschaft. Steuern Sie stattdessen den Arbeitsspeicher dadurch, wie Sie eine Instanz des Callout-Containers in Ihrer Anwendung erstellen. Im folgenden Beispiel erstellen Sie eine Instanz des Callout-Containers bei jedem Öffnen des Callout-Containers:

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

Alternativ können Sie eine einzelne Instanz des Callout-Containers definieren, den Sie bei jedem Öffnen wiederverwenden:

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

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

Übergänge in einer Mobilanwendung definieren

Durch Ansichtsübergänge in Spark wird definiert, wie der Wechsel zwischen View-Containern auf dem Bildschirm dargestellt wird. Bei Übergängen wird während des Ansichtswechsels eine Animation angewendet. Mithilfe von Übergängen können Sie ansprechende Schnittstellen für Ihre Mobilanwendungen erstellen.

Standardmäßig gleitet der vorhandene View-Container vom Bildschirm, während die neue Ansicht auf den Bildschirm gleitet. Sie können den Wechsel jedoch auch anpassen. Angenommen, in Ihrer Anwendung ist in einem View-Container ein Formular mit nur wenigen Feldern definiert, in einem nachfolgenden View-Container werden hingegen weitere Felder angezeigt. Statt einfach zwischen Ansichten zu wechseln, können Sie einen Kipp- oder Ausblendübergang verwenden.

Flex bietet die folgenden Ansichtsübergangsklassen für den Wechsel des View-Containers:

Übergang	Beschreibung
CrossFadeViewTransition	Führt eine Überblendung zwischen der vorhandenen und der neuen Ansicht aus. Die vorhandene Ansicht wird aus-, während die neue Ansicht eingeblendet wird.
FlipViewTransition	Führt einen Kippübergang zwischen der vorhandenen und der neuen Ansicht aus. Die Richtung und die Art des Kippens können Sie definieren.
SlideViewTransition	Führt einen Gleitübergang zwischen der vorhandenen und der neuen Ansicht aus. Die vorhandene Ansicht gleitet aus dem Bildschirm, während die neue Ansicht auf den Bildschirm gleitet. Die Richtung und die Art des Gleitens können Sie bestimmen. Dies ist der Standardübergang für Ansichten in Flex.
ZoomViewTransition	Führt einen Zoomübergang zwischen der vorhandenen und der neuen Ansicht aus. Sie können die vorhandene Ansicht aus- oder die neue Ansicht einzoomen.

Hinweis: *Ansichtsübergänge in Mobilanwendungen unterscheiden sich von Flex-Standardübergängen. Flex-Standardübergänge definieren die Effekte, die während einer Statusänderung wiedergegeben werden. Ansichtsübergänge werden durch Navigationsaktionen des ViewNavigator-Containers ausgelöst. Ansichtsübergänge können in MXML nicht definiert werden, und sie interagieren nicht mit Status.*

Übergänge anwenden

Einen Übergang wenden Sie an, wenn Sie den aktiven View-Container wechseln. Da Ansichtsübergänge beim Wechseln des View-Containers stattfinden, steuern Sie sie über den ViewNavigator-Container.

Beispielsweise können Sie zum Wechseln der aktuellen Ansicht die folgenden Methoden der ViewNavigator-Klasse verwenden:

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

Diese Methoden akzeptieren jeweils ein optionales Argument, das den beim Wechsel der Ansicht abzuspielenden Übergang definiert.

Sie können die aktuelle Ansicht auch mithilfe der Hardwarenavigationstasten des Geräts wechseln, etwa mit der Zurück-Taste. Beim Wechseln der Ansicht mithilfe einer Hardwaretaste verwendet der ViewNavigator die in der `ViewNavigator.defaultPopTransition`-Eigenschaft und der `ViewNavigator.defaultPushTransition`-Eigenschaft definierten Standardübergänge. Standardmäßig geben diese Eigenschaften die Verwendung der SlideViewTransition-Klasse an.

Im folgenden Beispiel wird eine Hauptanwendungsdatei gezeigt, die die Eigenschaften `defaultPopTransition` und `defaultPushTransition` für einen FlipViewTransition-Übergang initialisiert:

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

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

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

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

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

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

Die erste Ansicht von „EmployeeMainViewTrans.mxml“ definiert einen CrossFadeViewTransition-Übergang. Bei einem Wechsel zu EmployeeView wird CrossFadeViewTransition als Argument an die `pushView()`-Methode übergeben:

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

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

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

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

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

Die EmployeeView ist, wie im folgenden Beispiel dargestellt, in der Datei „EmployeeView.mxml“ definiert:

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

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

Einen Übergang auf das ActionBar-Steuerelement anwenden

Standardmäßig ist die ActionBar nicht in Übergängen enthalten. Unabhängig davon, ob ein bestimmter Übergang festgelegt wurde, erzeugt das ActionBar-Steuerelement beim Ansichtswechsel einen gleitenden Übergang. Um ActionBar beim Ansichtswechsel in den Übergang einzubeziehen, legen Sie die `transitionControlsWithContent`-Eigenschaft der Übergangsklasse auf `true` fest.

easing-Klasse für Übergange verwenden

Übergänge werden in zwei Phasen abgespielt: erst einer *Beschleunigungsphase* und dann einer *Abbremsphase*. Die Beschleunigungs- und Abbremseigenschaften eines Übergangs können Sie mithilfe einer easing-Klasse ändern. Durch Beschleunigung (easing) erzielen Sie eine realistischere Beschleunigungs- und Abbremsgeschwindigkeit. Zudem können Sie mit einer easing-Klasse einen Sprungeffekt erzielen oder andere Bewegungstypen steuern.

Flex stellt Spark easing-Klassen im spark.effects.easing-Paket bereit. Dieses Paket umfasst Klassen für die häufigsten Beschleunigungstypen (easing), darunter Bounce, Linear und Sine. Weitere Informationen zur Verwendung dieser Klassen finden Sie unter Using Spark easing classes.

Das folgende Beispiel zeigt eine Änderung an der Anwendung, die im vorherigen Abschnitt definiert wurde. In dieser Version wird FlipViewTransition eine Bounce-Easing-Klasse hinzugefügt:

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

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

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

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

```
            }

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

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

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

Drücken Sie auf dem Mobilgerät die Zurück-Taste, um den Sprungeffekt zu sehen.

Übergangslebenszyklus anzeigen

Ein Ansichtsübergang durchläuft während der Ausführung zwei Phasen: die *Vorbereitungsphase* und die *Ausführungsphase*.

Die Vorbereitungsphase wird von drei Methoden der Übergangsklasse definiert. Diese Methoden werden in der folgenden Reihenfolge aufgerufen:

1 `captureStartValues()`

   Beim Aufruf dieser Methode hat der ViewNavigator die neue Ansicht erstellt, die neue Ansicht jedoch nicht überprüft und den Inhalt des ActionBar-Steuerelements und der Registerkartenleiste nicht aktualisiert. Mit dieser Methode erfassen Sie die Ausgangswerte für die Komponenten, die bei dem Übergang eine Rolle spielen.

2 `captureEndValues()`

   Beim Aufruf dieser Methode wurde die neue Ansicht vollständig überprüft, und das ActionBar-Steuerelement und die Registerkartenleiste geben den Status der neuen Ansicht wieder. Beim Übergang können mit dieser Methode alle erforderlichen Werte aus der neuen Ansicht erfasst werden.

3 `prepareForPlay()`

   Mit dieser Methode initialisiert der Übergang die Effektinstanz, die zum Animieren der Übergangskomponenten verwendet wird.

Die Ausführungsphase beginnt, wenn der ViewNavigator die `play()`-Methode des Übergangs aufruft. Zu diesem Zeitpunkt wurde die neue Ansicht erstellt und überprüft, und das ActionBar-Steuerelement und die Registerkartenleiste wurden initialisiert. Beim Übergang wird ein `start`-Ereignis ausgelöst und alle in der Vorbereitungsphase erstellten Effektinstanzen werden nun über die `play()`-Methode des Effekts aufgerufen.

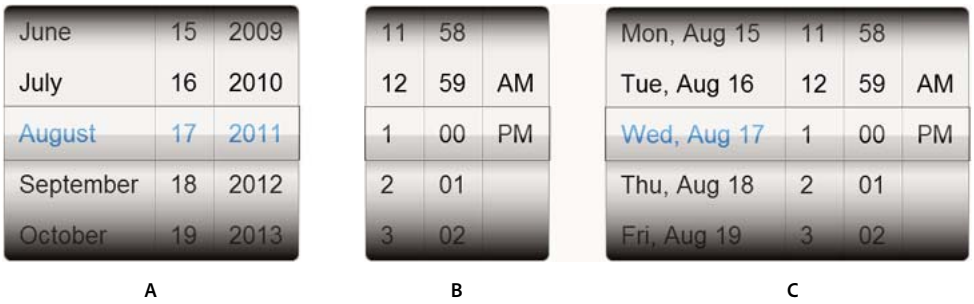
Bei Abschluss des Ansichtsübergangs wird ein `end`-Ereignis ausgelöst. Die Übergangsbasisklasse ViewTransitionBase definiert die `transitionComplete()`-Methode, mit der Sie das `end`-Ereignis auslösen können. Beim Übergang müssen unbedingt sämtliche temporären Objekte bereinigt und erstellten Listener entfernt werden, bevor das Abschlussereignis ausgelöst wird.

Nach dem Aufruf der `transitionComplete()`-Methode schließt der ViewNavigator den Änderungsvorgang ab und setzt den Übergang auf den nicht initialisierten Status zurück.

Datums- und Uhrzeitangaben in einer Mobilanwendung auswählen

Mithilfe des DateSpinner-Steuerelements können Benutzer Datums- und Uhrzeitangaben in einer Mobilanwendung wählen. Es verwendet die vertraute Mobile-Benutzeroberfläche mit einer Reihe von nebeneinander liegenden Bildlaufrädern. Jedes Rad zeigt einen anderen Teil des Datums und/oder der Uhrzeit an.

Ihnen stehen drei grundlegende Typen von DateSpinner-Steuerelementen zur Verfügung. Die folgende Abbildung zeigt die drei Typen von DateSpinner-Steuerelementen:



A. Datum. B. Uhrzeit. C. Datum und Uhrzeit

In der folgenden Tabelle werden die DateSpinner-Typen beschrieben:

Typ	Konstante (String-Äquivalent)	Beschreibung
Datum	`DateSelectorDisplayMode.DATE` („date“)	Zeigt den Monat, den Tag des Monats und das Jahr an. Beispiel: `|| June || 11 || 2011 ||` Datum ist der Standardtyp. Falls Sie die Eigenschaft `displayMode` eines DateSpinner-Steuerelements nicht festlegen, setzt Flex diese auf „Datum“. Das aktuelle Datum wird in der Farbe hervorgehoben, die in der `accentColor`-Eigenschaft festgelegt ist. Es werden Datumsangaben vom 1. Januar 1601 bis zum 31. Dezember 9999 unterstützt.
Uhrzeit	`DateSelectorDisplayMode.TIME` („time“)	Zeigt die Stunden und Minuten an. Bei Gebietsschemas, die die 12-Stunden-Uhr verwenden, wird außerdem AM/PM angezeigt. Beispiel: `|| 2 || 57 || PM ||` Die aktuelle Uhrzeit wird nicht hervorgehoben. Sie können im DateSpinner-Steuerelement keine Sekunden anzeigen. Es ist nicht möglich, zwischen dem 12- und dem 24-Stunden-Zeitformat umzuschalten. DateSpinner verwendet das für das Gebietsschema übliche Format.

Typ	Konstante (String-Äquivalent)	Beschreibung
Datum und Uhrzeit	`DateSelectorDisplayMode.DATE_AND_TIME` („dateAndTime“)	Zeigt die Tage, Stunden und Minuten an. Bei Gebietsschemas, die die 12-Stunden-Uhr verwenden, wird außerdem AM/PM angezeigt. Beispiel: `|| Mon Jun 13 || 2 || 57 || PM ||` Das aktuelle Datum wird in der Farbe hervorgehoben, die in der `accentColor`-Eigenschaft festgelegt ist. Die aktuelle Uhrzeit wird nicht hervorgehoben. Sie können im DateSpinner-Steuerelement keine Sekunden anzeigen. Der Monatsname wird in verkürzter Form angezeigt. Das Jahr wird nicht angezeigt.

Das DateSpinner-Steuerelement besteht aus mehreren SpinnerList-Steuerelementen. Jede SpinnerList zeigt eine Liste gültiger Werte für einen bestimmten Ort im DateSpinner-Steuerelement an. Ein DateSpinner-Steuerelement, das das Datum anzeigt, verfügt beispielsweise über drei SpinnerLists: eine für das Datum, eine für den Monat und eine für das Jahr. Ein DateSpinner, das nur die Uhrzeit anzeigt, verfügt über zwei oder drei SpinnerLists: eine für die Stunden, eine für die Minuten und optional eine für AM/PM (wenn die Uhrzeit in 12 Stunden angegeben wird).

Typ des DateSpinner-Steuerelements ändern

Sie wählen den DateSpinner-Typ aus, indem Sie den Wert der `displayMode`-Eigenschaft im Steuerelement festlegen. Sie können die `displayMode`-Eigenschaft auf die durch die DateSelectionDisplayMode-Klasse definierten Konstanten oder ihre String-Äquivalenten setzen.

Im folgenden Beispiel wird zwischen den verschiedenen DateSpinner-Typen umgeschaltet:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerTypes.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Types">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <s:ComboBox id="modeList" selectedIndex="0"
change="ds1.displayMode=modeList.selectedItem.value">
        <s:ArrayList>
            <fx:Object value="date" label="Date"/>
            <fx:Object value="time" label="Time"/>
            <fx:Object value="dateAndTime" label="Date and Time"/>
        </s:ArrayList>
    </s:ComboBox>
    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>

    <s:Label text="{ds1.selectedDate}"/>

</s:View>
```

Wenn Benutzer die DateSpinner-Steuerung verwenden, springen die Spinners zu dem nächsten Element in der Liste. Wenn sie nicht verwendet werden, befinden sich Spinners nie zwischen Auswahlen.

DateSpinner-Steuerungsauswahl an andere Steuerelemente binden

Sie können die Eigenschaft `selectedDate` eines DateSpinner-Steuerelements an andere Steuerelemente in einer Mobileanwendung binden. Die Eigenschaft `selectedDate` ist ein Zeiger auf ein Date-Objekt. Alle Methoden oder Eigenschaften eines Date-Objekts sind auf diese Weise verfügbar.

Im folgenden Beispiel wird der Tag, der Monat und das Jahr an die Steuerelemente „Label“ gebunden:

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

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var dayArrayC:ArrayCollection = new ArrayCollection();

            [Bindable]
            private var selectedDateProperty:Date;

            private function initAC():void {
                dayArrayC.addItem("Sunday");
                dayArrayC.addItem("Monday");
                dayArrayC.addItem("Tuesday");
                dayArrayC.addItem("Wednesday");
                dayArrayC.addItem("Thursday");
                dayArrayC.addItem("Friday");
                dayArrayC.addItem("Saturday");
            }

        ]]>
    </fx:Script>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>
    <fx:Binding source="ds1.selectedDate" destination="selectedDateProperty"/>

    <s:Label id="label1" text="Day: {dayArrayC.getItemAt(ds1.selectedDate.day)}"/>
    <s:Label id="label2" text="Day of month: {selectedDateProperty.getDate()}"/>
    <s:Label id="label3" text="Month: {ds1.selectedDate.getMonth() + 1}"/>
    <s:Label id="label4" text="Year: {selectedDateProperty.getFullYear()}"/>

</s:View>
```

Daten in einem DateSpinner-Steuerelement programmgesteuert auswählen

Sie können Daten in einem DateSpinner-Steuerelement programmgesteuert ändern, indem Sie dem Wert der Eigenschaft `selectedDate` ein neues Date-Objekt zuordnen.

Im folgenden Beispiel werden Sie aufgefordert, einen Tag, einen Monat und ein Jahr einzugeben. Wenn Sie auf die Schaltfläche klicken, wechselt DateSpinner zum neuen Datum:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerProgrammaticSelection.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Programmatic Selection"
        creationComplete="init()">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
          import spark.components.calendarClasses.DateSelectorDisplayMode;

            private function init():void {
                // change event is dispatched when DateSpinner changes from user interaction
                ds1.addEventListener("change", spinnerEventHandler);
                // valueCommit event is dispatched when DateSpinner programmatically changes
                ds1.addEventListener("valueCommit", spinnerEventHandler);
            }

            private function b1_clickHandler(e:Event):void {
                ds1.selectedDate = new Date(ti3.text,ti1.text,ti2.text);
            }

            protected function spinnerEventHandler(event:Event):void {
                eventLabel.text = event.type;
            }
        ]]>
    </fx:Script>

    <s:TextInput id="ti1" prompt="Enter a Month"/>
    <s:TextInput id="ti2" prompt="Enter a Day"/>
    <s:TextInput id="ti3" prompt="Enter a Year"/>
    <s:Button id="b1" label="Go!" click="b1_clickHandler(event)"/>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>

    <s:Label id="eventLabel"/>

</s:View>
```

Wird das Datum programmgesteuert geändert, löst das DateSpinner-Steuerelement sowohl ein `change`- als auch ein `valueCommit`-Ereignis aus. Falls das Datum durch den Benutzer geändert wird, löst das DateSpinner-Steuerelement ein `change`-Ereignis aus.

Wird das gewählte Datum programmgesteuert geändert, werden die Werte ohne Animierung durch die Zwischenwerte angezeigt.

Datumsbereiche in einem DateSpinner-Steuerelement einschränken

Sie können Daten, die Benutzer in einem DateSpinner-Steuerelement auswählen können, mithilfe der Eigenschaften `minDate` und `maxDate` beschränken. Diese Eigenschaften verwenden Date-Objekte. Auf Daten, die vor der `minDate`- oder nach der `maxDate`-Eigenschaft liegen, kann nicht im DateSpinner-Steuerelement zugegriffen werden. Zusätzlich werden ungültige Jahre nicht im Modus „date“ angezeigt und ungültige Datumswerte werden nicht im Modus „dateAndTime“ angezeigt.

Im folgenden Beispiel werden zwei DateSpinner-Steuerelemente erstellt, die unterschiedliche Bereiche verfügbarer Datumswerte aufweisen:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/MinMaxDates.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Min/Max Dates">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Min date today, max date October 31, 2012. -->
    <s:Label text="{dateSpinner1.selectedDate}"/>
    <s:DateSpinner id="dateSpinner1"
        displayMode="{DateSelectorDisplayMode.DATE}"
        minDate="{new Date()}"
        maxDate="{new Date(2012,9,31)}"/>
    <!-- Min date 3 days ago, max date 7 days from now. -->
    <s:Label text="{dateSpinner2.selectedDate}"/>
    <s:DateSpinner id="dateSpinner2"
        displayMode="{DateSelectorDisplayMode.DATE}"
        minDate="{new Date(new Date().getTime() - 1000*60*60*24*3)}"
        maxDate="{new Date(new Date().getTime() + 1000*60*60*24*7)}"/>
</s:View>
```

Sie können nur ein minimales und ein maximales Datum festlegen. Sie können ein Array von Datumswerten oder mehrere Auswahlbereiche festlegen.

Sie können außerdem die Eigenschaften `minDate` und `maxDate` verwenden, um einen DateSpinner im time-Modus zu beschränken. Im folgenden Beispiel wird die Zeitauswahl auf 8 Uhr bis 14 Uhr beschränkt:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/MinMaxTime.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Min/Max Time">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Limit time selection to between 8am and 2pm -->
    <s:DateSpinner id="dateSpinner1"
                   displayMode="{DateSelectorDisplayMode.TIME}"
                   minDate="{new Date(0,0,0,8,0)}"
                   maxDate="{new Date(0,0,0,14,0)}"/>

</s:View>
```

Auf ein Ereignis eines DateSpinner-Steuerelements reagieren

Das DateSpinner-Steuerelement löst ein `Change`-Ereignis aus, wenn der Benutzer das Datum ändert. Die `target`-Eigenschaft dieses `Change`-Ereignisses enthält einen Verweis auf DateSpinner, mit dem auf das ausgewählte Datum zugegriffen werden kann, wie im folgenden Beispiel gezeigt wird:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerChangeEvent.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Change Event">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;

            private var dayArray:Array = new Array(
                "Sunday","Monday","Tuesday",
                "Wednesday","Thursday","Friday","Saturday");

            private function ds1_changeHandler(e:Event):void {
                // Optionally cast the DateSpinner's selectedDate as a Date
                var d:Date = new Date(e.currentTarget.selectedDate);
                ta1.text = "You selected:";
                ta1.text += "\n Day of Week: " + dayArray[d.day];
                ta1.text += "\n Year: " + d.fullYear;
                // Month is 0-based in ActionScript, so add 1:
                ta1.text += "\n Month: " + int(d.month + 1);
                ta1.text += "\n Day: " + d.date;
            }
        ]]>
    </fx:Script>

    <s:DateSpinner id="ds1"
        displayMode="{DateSelectorDisplayMode.DATE}"
        change="ds1_changeHandler(event)"/>

    <s:TextArea id="ta1" height="200" width="350"/>
</s:View>
```

Das `Change`-Ereignis wird ausgelöst (und der Wert der Eigenschaft `selectedDate` wird aktualisiert), nur wenn sich kein Spinner in der Benutzeroberfläche mehr bewegt.

Zum Erfassen eines Datums, das programmgesteuert erfolgte, warten Sie auf das `value_commit`-Ereignis.

Minutenintervall eines DateSpinner-Steuerelements ändern

Sie können das Minutenintervall ändern, das ein DateSpinner-Steuerelement mithilfe der `minuteStepSize`-Eigenschaft anzeigt. Diese Eigenschaft wird nur auf ein DateSpinner-Steuerelement angewendet, wenn `displayMode` auf „time“ oder „dateAndTime“ gesetzt ist. Setzen Sie beispielsweise die `minuteStepSize`-Eigenschaft auf 10, zeigt das DateSpinner-Steuerelement die Werte 0, 10, 20, 30, 40, und 50 im Minutenspinner an.

Im folgenden Beispiel kann der Wert der `minuteStepSize`-Eigenschaft festgelegt werden. Der Minutenspinner wird entsprechend aktualisiert.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerMinuteInterval.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Minute Interval">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>
    <s:Label text="Select an interval:"/>

    <s:ComboBox id="intervalList" selectedIndex="0"
        change="ds1.minuteStepSize=intervalList.selectedItem.value">
        <s:ArrayList>
            <fx:Object value="1" label="1"/>
            <fx:Object value="2" label="2"/>
            <fx:Object value="3" label="3"/>
            <fx:Object value="4" label="4"/>
            <fx:Object value="5" label="5"/>
            <fx:Object value="6" label="6"/>
            <fx:Object value="10" label="10"/>
            <fx:Object value="12" label="12"/>
            <fx:Object value="15" label="15"/>
            <fx:Object value="20" label="20"/>
            <fx:Object value="30" label="30"/>
        </s:ArrayList>
    </s:ComboBox>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.TIME}"/>
</s:View>
```

Gültige Werte für die `minuteStepSize`-Eigenschaft müssen sich durch 60 teilen lassen. Legen Sie einen Wert fest, der sich nicht durch 60 teilen lässt (etwa 25), ist der Wert in der `minuteStepSize`-Eigenschaft standardmäßig 1.

Falls Sie ein Minutenintervall festlegen und die aktuelle Zeit keinem Wert im Minutenspinner entspricht, rundet das DateSpinner-Steuerelement die derzeitige Auswahl auf das nächstliegende Intervall. Wenn die Uhrzeit beispielsweise 10:29 beträgt und die `minuteStepSize`-Eigenschaft auf 15 festgelegt ist, rundet das DateSpinner-Steuerelement auf 10:15, vorausgesetzt der Wert 10:15 verstößt nicht gegen die `minDate`-Einstellung.

Darstellung eines DateSpinner-Steuerelements anpassen

Das DateSpinner-Steuerelement unterstützt die meisten Textstile wie `fontSize`, `color` und `letterSpacing`. Zusätzlich fügt es die neue Stileigenschaft `accentColor` hinzu. Dieser Stil ändert die Farbe des aktuellen Datums oder der Zeit in den Spinnerlisten. Im folgenden Beispiel wird die Farbe auf rot gesetzt:

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

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Acceptable style formats are color_name (e.g., 'red') or
        hex colors (e.g., '0xFF0000') -->
    <s:DateSpinner id="dateSpinner1" accentColor="0xFF0000"
        displayMode="{DateSelectorDisplayMode.DATE}"/>
</s:View>
```

Das DateSpinner-Steuerelement unterstützt die `textAlign`-Eigenschaft nicht. Textausrichtung wird vom Steuerelement festgelegt.

Um andere Aspekte der Darstellung eines DateSpinner-Steuerelements anzupassen, können Sie eine benutzerdefinierte Skin für das Steuerelement erstellen oder einige der Unterkomponenten mit CSS ändern.

Die DateSpinnerSkin-Klasse steuert die Größenanpassung des DateSpinner-Steuerelements. Bei jedem Spinner innerhalb eines DateSpinner-Steuerelements handelt es sich um ein SpinnerList-Objekt mit eigener SpinnerListSkin. Alle Spinner in einem DateSpinner-Steuerelement sind einem SpinnerListContainer untergeordnet, der über eine eigene Skin verfügt, der SpinnerListContainerSkin.

Sie können die `height`-Eigenschaft eines DateSpinner-Steuerelements explizit festlegen. Legen Sie die `width`-Eigenschaft fest, zentriert sich das Steuerelement in einem Bereich, der an die angeforderte Weite angepasst ist.

Um die Einstellungen der Spinner in einem DateSpinner-Steuerelement zu ändern, können Sie außerdem die SpinnerList-, SpinnerListContainer- und SpinnerListItemRenderer-CSS-Typselektoren verwenden. Die SpinnerList-Typselektoren steuern die Auffüllungseigenschaften in den Spinners.

Im folgenden Beispiel wird die Auffüllung in den Spinners geändert:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/DateSpinnerExamples2.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.CustomSpinnerListSkinExample">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        s|SpinnerListItemRenderer {
            paddingTop: 7;
            paddingBottom: 7;
            paddingLeft: 5;
            paddingRight: 5;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

In Mobilanwendungen müssen sich Typselektoren in der Anwendungsdatei auf oberster Ebene befinden und nicht in einer untergeordneten Ansicht der Anwendung. Wenn Sie versuchen, den SpinnerListItemRenderer-Typselektor in einem Stilblock in einer Ansicht festzulegen, gibt Flex eine Compiler-Warnung aus.

Sie können die SpinnerListContainerSkin-Klasse erweitern, um die Darstellung der Spinner in einem DateSpinner-Steuerelement weiter anzupassen. Im folgenden Beispiel wird eine benutzerdefinierte Skin auf SpinnerListContainer angewendet:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/DateSpinnerExamples3.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                            xmlns:s="library://ns.adobe.com/flex/spark"
                            firstView="views.CustomSpinnerListSkinExample">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        /* Change SpinnerListContainer for all DateSpinner controls */
        s|SpinnerListContainer {
            skinClass: ClassReference("customSkins.CustomSpinnerListContainerSkin");
        }

        /* Change padding for all DateSpinner controls */
        s|SpinnerListItemRenderer {
            paddingTop: 7;
            paddingBottom: 7;
            paddingLeft: 5;
            paddingRight: 5;
            fontSize: 12;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

Die folgende CustomSpinnerListContainerSkin-Klasse verringert die Größe des „Auswahlindikators“, sodass die neue Größe der Schriftarten und Auffüllung in den Zeilen des Spinners genauer angegeben werden:

```
// datespinner/customSkins/CustomSpinnerListContainerSkin.as
package customSkins {
    import mx.core.DPIClassification;

    import spark.skins.mobile.SpinnerListContainerSkin;
    import spark.skins.mobile.supportClasses.MobileSkin;
    import spark.skins.mobile160.assets.SpinnerListContainerBackground;
    import spark.skins.mobile160.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile160.assets.SpinnerListContainerShadow;
    import spark.skins.mobile240.assets.SpinnerListContainerBackground;
    import spark.skins.mobile240.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile240.assets.SpinnerListContainerShadow;
    import spark.skins.mobile320.assets.SpinnerListContainerBackground;
    import spark.skins.mobile320.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile320.assets.SpinnerListContainerShadow;

    public class CustomSpinnerListContainerSkin extends SpinnerListContainerSkin
    {
        public function CustomSpinnerListContainerSkin() {
            super();

            switch (applicationDPI)
            {
                case DPIClassification.DPI_320:
                {
                   borderClass = spark.skins.mobile320.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile320.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile320.assets.SpinnerListContainerShadow;

                    cornerRadius = 10;
                    borderThickness = 2;
                    selectionIndicatorHeight = 80; // was 120
                    break;
                }
                case DPIClassification.DPI_240:
                {
                   borderClass = spark.skins.mobile240.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile240.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile240.assets.SpinnerListContainerShadow;

                    cornerRadius = 8;
                    borderThickness = 1;
```

```
                    selectionIndicatorHeight = 60; // was 90
                    break;
                }
                default: // default DPI_160
                {
                   borderClass = spark.skins.mobile160.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile160.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile160.assets.SpinnerListContainerShadow;

                    cornerRadius = 5;
                    borderThickness = 1;
                    selectionIndicatorHeight = 40; // was 60

                    break;
                }
            }
        }
    }
}
```

Weitere Informationen zur Skingestaltung in Mobilkomponenten finden Sie unter „Grundlagen des Mobilskinnings“ auf Seite 173.

Lokalisierte Datums- und Uhrzeitangaben mit einem DateSpinner-Steuerelement verwenden

Das DateSpinner-Steuerelement unterstützt alle von dem Gerät unterstützen Gebietsschemas, auf dem die Anwendung ausgeführt wird. Setzen Sie das Gebietsschema auf ja-JP, ändert sich DateSpinner, um Datumsangaben im japanischen Standard anzuzeigen.

Sie können die `locale`-Eigenschaft direkt im DateSpinner-Steuerelement oder in einem Container, wie etwa einer Anwendung, festlegen. Das DateSpinner-Steuerelement übernimmt den Wert dieser Eigenschaft. Das Standardgebietsschema ist das Gebietsschema des Geräts, auf dem die Anwendung ausgeführt wird, es sei denn Sie überschreiben es mit der `locale`-Eigenschaft.

Im folgenden Beispiel wird das Standardgebietsschema auf „ja-JP“ gesetzt. Sie können ein Gebietsschema auswählen, um das Format des DateSpinners zu ändern:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/LocalizedDateSpinner.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Localized DateSpinner" locale="ja_JP">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            private function localeChangeHandler():void {
                ds1.setStyle('locale',localeSelector.selectedItem);
            }
        ]]>
    </fx:Script>

    <s:ComboBox id="localeSelector" change="localeChangeHandler()">
        <s:ArrayList>
            <fx:String>en-US</fx:String>
            <fx:String>en-UK</fx:String>
            <fx:String>es-AR</fx:String>
            <fx:String>he-IL</fx:String>
            <fx:String>ko-KR</fx:String>
            <fx:String>ja-JP</fx:String>
            <fx:String>vi-VN</fx:String>
            <fx:String>zh-CN</fx:String>
            <fx:String>zh-TW</fx:String>
        </s:ArrayList>
    </s:ComboBox>
    <s:DateSpinner id="ds1" displayMode="dateAndTime"/>

</s:View>
```

Spinnerliste in einer Mobilanwendung verwenden

Bei der SpinnerList-Komponente handelt es sich um eine spezialisierte Liste, mit der normalerweise Daten in Mobilanwendungen ausgewählt werden. Führt der Benutzer einen Bildlauf in den Listenelementen durch, werden die Elemente umgebrochen, nachdem der Benutzer das Ende der Liste erreicht hat. Das SpinnerList-Steuerelement wird häufig als NumericStepper-Komponente in Mobilanwendungen verwendet.

Die folgende Abbildung zeigt ein typisches SpinnerList-Steuerelement in einer Mobilanwendung:

SpinnerList-Steuerelement

Die SpinnerList verhält sich wie eine sich drehende zylindrische Trommel. Benutzer können die Liste bewegen, indem Sie sie nach oben oder unten schubsen oder ziehen und auf ein Element in der Liste klicken.

Sie schließen normalerweise ein SpinnerList-Steuerelement in ein SpinnerListContainer-Steuerelement ein. Diese Klasse stellt den Großteil des Chroms für die SpinnerList zur Verfügung und legt das Layout fest. Das Chrom enthält die Rahmen, Schatten und die Darstellung der Auswahlindikatoren.

Die Daten für eine SpinnerList werden in Form einer Liste gespeichert. Sie werden im Spinner mit einem SpinnerListItemRenderer dargestellt. Sie können den Elementrenderer überschreiben, um die Darstellung oder Inhalte der Listenelemente anzupassen.

Das DateSpinner-Steuerelement ist ein Beispiel für einen Satz von SpinnerList-Steuerelementen mit einem benutzerdefinierten Elementrenderer.

Sie können derzeit keine Elemente in einem SpinnerList-Steuerelement deaktivieren, ohne das gesamte Steuerelement zu deaktivieren. Die Einschränkung gilt nicht für die DateSpinner-Steuerung, die eine zusätzliche Logik zum Festlegen von Bereichen deaktivierter Daten bereitstellt.

Daten für eine Spinnerliste definieren

Führen Sie einen der folgenden Schritte aus, um Daten für ein SpinnerList-Steuerelement zu definieren:

- Definieren Sie die Daten inline in der `dataProvider`-Eigenschaft des SpinnerList-Steuerelements.
- Legen Sie Daten als untergeordnete Tags des `<s:SpinnerList>`-Tags fest.
- Definieren Sie Daten in ActionScript oder MXML und binden Sie sie an das SpinnerList-Steuerelement. Diese Daten können aus einem externen Dienst, einer eingebetteten Ressource wie etwa einer XML-Datei oder einer anderen Datenquelle stammen.
- Binden Sie das SpinnerList-Steuerelement mithilfe des Flash Builder-Dienstassistenten an einen Datendienstvorgang. Weitere Informationen zum Erstellen datenorientierter Anwendungen mit Flash Builder finden Sie unter Verbindung zu Datendiensten herstellen.

Das SpinnerList-Steuerelement kann jede beliebige Klasse als Datenprovider annehmen, die die IList-Schnittstelle implementiert. Diese Klassen umfassen die ArrayCollection-, ArrayList-, NumericDataProvider- und XMLListCollection-Klassen.

Definieren Sie keinen Datenprovider für das SpinnerList-Steuerelement, wenn es instanziiert wird, wird die SpinnerList mit einer leeren Zeile angezeigt. Nach dem Hinzufügen eines Datenproviders wird die Größe der SpinnerList geändert, um standardmäßig fünf Elemente in der Liste anzuzeigen.

Im folgenden Beispiel werden Daten für das SpinnerList-Steuerelement in untergeordneten Tags des `<s:SpinnerList>`-Tags definiert:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListComplexDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Complex Data Provider">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <s:Label text="Select a person:"/>

    <s:SpinnerListContainer>
        <s:SpinnerList id="peopleList" width="200" labelField="name">
            <s:ArrayList>
                <fx:Object name="Friedeman Friese" companyID="14266"/>
                <fx:Object name="Stephen Glenn" companyID="14266"/>
                <fx:Object name="Reiner Knizia" companyID="11233"/>
                <fx:Object name="Alan Moon" companyID="11543"/>
                <fx:Object name="Klaus Teuber" companyID="13455"/>
                <fx:Object name="Dale Yu" companyID="14266"/>
            </s:ArrayList>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:Label text="Selected ID: {peopleList.selectedItem.companyID}"/>

</s:View>
```

Im folgenden Beispiel werden SpinnerList-Daten im `<s:SpinnerList>`-Tag definiert:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListInlineDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Inline Data Provider">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <!-- Create data provider inline. -->
        <s:SpinnerList id="smallList" dataProvider="{new ArrayList([1,5,10,15,30])}"
                       wrapElements="false" typicalItem="44"/>
    </s:SpinnerListContainer>

    <s:Label text="Selected Item: {smallList.selectedItem}"/>

</s:View>
```

Im folgenden Beispiel werden SpinnerList-Daten in ActionScript definiert:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListBasicDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Basic Data Provider"
        creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            public var daysOfWeek:ArrayList;

            private function initApp():void {
                daysOfWeek = new ArrayList(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]);
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="daysList" width="100" dataProvider="{daysOfWeek}"/>
    </s:SpinnerListContainer>

    <s:Label text="Selected Day: {daysList.selectedItem}"/>

</s:View>
```

Handelt es sich bei den Daten in ActionScript um komplexe Objekte, geben Sie die `labelField`-Eigenschaft an, sodass die SpinnerList die richtigen Beschriftungen anzeigt, wie das folgende Beispiel veranschaulicht:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListComplexASDP.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Complex Data Provider in AS" creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            private var myAC:ArrayList;

            private function initApp():void {
                myAC = new ArrayList([
                    {name:"Alan Moon",id:42},
                    {name:"Friedeman Friese",id:44},
                    {name:"Dale Yu",id:45},
                    {name:"Stephen Glenn",id:47},
                    {name:"Reiner Knizia",id:48},
                    {name:"Klaus Teuber",id:49}
                ]);
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="peopleList" dataProvider="{myAC}"
                       width="200"
                       labelField="name"/>
    </s:SpinnerListContainer>
    <s:Label text="Selected ID: {peopleList.selectedItem.id}"/>
</s:View>
```

Sie können außerdem mithilfe einer Vereinfachungsklasse, NumericDataProvider, numerische Daten für ein SpinnerList-Steuerelement bereitstellen. Mit dieser Klasse können Sie einen Satz numerischer Daten mit einem Mindestwert, einem Höchstwert und einer Schrittgröße leicht definieren .

Im folgenden Beispiel wird die NumericDataProvider-Klasse als Datenquelle für das SpinnerList-Steuerelement verwendet:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/MinMaxSpinnerList.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Min/Max SpinnerLists"
        backgroundColor="0x000000">

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer top="10" left="10">
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="23" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="59" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="59" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100"
                       dataProvider="{new ArrayList(['AM','PM'])}"
                       wrapElements="false"/>
    </s:SpinnerListContainer>
</s:View>
```

Der Wert der `stepSize`-Eigenschaft kann eine negative Zahl sein. In diesem Fall wird der Höchstwert zuerst angezeigt. Der Spinner beginnt mit dem Höchstwert und bewegt sich zum Mindestwert.

Elemente in einer Spinnerliste auswählen

Das SpinnerList-Steuerelement unterstützt die Auswahl von nur jeweils einem Element. Das ausgewählte Element befindet sich immer in der Mitte der Komponente und wird standardmäßig unter dem Auswahlindikator angezeigt. Wenn sich die SpinnerList nicht bewegt, muss immer ein Element ausgewählt sein. Sie können weder ein deaktiviertes Element noch eine Zeile ohne Element auswählen.

Um das aktuell ausgewählte Element in einem SpinnerList-Steuerelement abzurufen, greifen Sie auf die `selectedIndex`- oder `selectedItem`-Eigenschaft des Steuerelements zu.

Um das aktuell ausgewählte Element in einem SpinnerList-Steuerelement festzulegen, legen Sie die `selectedIndex`- oder `selectedItem`-Eigenschaft des Steuerelements fest. Sie legen diese Eigenschaften normalerweise für das `<s:SpinnerList>`-Tag fest, sodass das Element ausgewählt wird, wenn die SpinnerList erstellt wird.

Legen Sie den Wert der `selectedIndex`- oder `selectedItem`-Eigenschaft der SpinnerList nicht explizit fest, wird standardmäßig das erste Element in der Liste ausgewählt.

Sie können mithilfe der `selectedIndex`- oder `selectedItem`-Eigenschaft das ausgewählte Element im Spinner programmgesteuert ändern. Legen Sie eine dieser Eigenschaften fest, springt das Steuerelement zu dem Element. Es animiert oder dreht den Spinner nicht zu dem Element.

Im folgenden Beispiel wird das SpinnerList-Steuerelement als Countdown-Timer verwendet. Das Timer-Objekt ändert das ausgewählte Element im Spinner, indem sich der Wert der `selectedIndex`-Eigenschaft jede Sekunde ändert:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListCountdownTimer.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Countdown Timer"
        creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            private var myTimer:Timer;

            private function initApp():void {
                myTimer = new Timer(1000, 0); // 1 second
                myTimer.addEventListener(TimerEvent.TIMER, changeSpinner);
                myTimer.start();
            }
            private function changeSpinner(e:Event):void {
                secList.selectedIndex = secList.selectedIndex - 1;
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer left="50" top="50">
        <s:SpinnerList id="secList" width="100" selectedIndex="60">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="60" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
    </s:SpinnerListContainer>
</s:View>
```

Benutzerinteraktionen und Ereignisse in einer Spinnerliste

Ändert sich das ausgewählte Element in einem SpinnerList-Steuerelement, löst das Steuerelement `change`- und `valueCommit`-Ereignisse aus. Dies geschieht als Folge einer Benutzerinteraktion wie etwa einer Fingerbewegung. Falls ein Benutzer ein Element auswählt, indem er es berührt, löst das Steuerelement sowohl ein `click`-Ereignis als auch `change`- und `valueCommit`-Ereignisse aus.

Wird das ausgewählte Element programmgesteuert geändert, löst das SpinnerList-Steuerelement nur ein `valueCommit`-Ereignis aus.

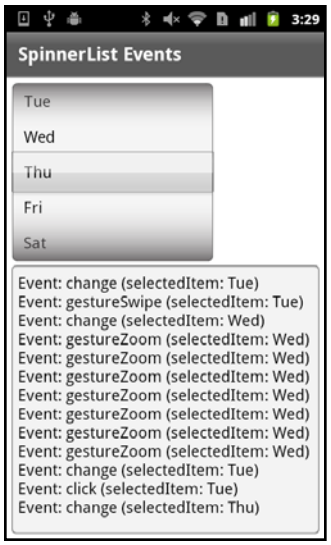
Bewegt sich das SpinnerList-Steuerelement, löst es nicht für jedes Element ein Ereignis aus, das es durchläuft. Es löst nur `change` oder `valueCommit`-Ereignisse aus, wenn es bei einem neuen Element anhält.

Nach erstmaliger Instanziierung des SpinnerList-Steuerelements mit einem Datenprovider löst es sowohl ein `change`- als auch ein `valueCommit`-Ereignis aus.

Das folgende Beispiel zeigt die üblichen bei Verwendung des SpinnerList-Steuerelements ausgelösten Ereignisse:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListEvents.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="SpinnerList Events"
        creationComplete="initApp()">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"
                          paddingRight="10" paddingBottom="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            public var daysOfWeek:ArrayList;

            private function initApp():void {
                daysOfWeek = new ArrayList(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]);
            }

            private function eventHandler(e:Event):void {
            ta1.text += "Event: " + e.type + " (selectedItem: " + e.currentTarget.selectedItem
+ ")\n";
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="daysList" width="300"
            dataProvider="{daysOfWeek}"
            change="eventHandler(event)"
            gestureSwipe="eventHandler(event)"
            click="eventHandler(event)"
            gestureZoom="eventHandler(event)"
        />
    </s:SpinnerListContainer>
    <s:TextArea id="ta1" width="100%" height="100%"/>
</s:View>
```

Die folgende Abbildung zeigt die Ausgabe nach Interaktion mit dem SpinnerList-Steuerelement:



Ereignisse des SpinnerList-Steuerelements

Umbruch in einer Spinnerliste festlegen

Wenn der Datenprovider des SpinnerList-Steuerelements weniger Elemente enthält als im Spinner angezeigt werden, wird der Spinner nicht umgebrochen, sondern hält beim letzten Element in der Liste an. Andernfalls springt der Spinner nach dem letzten Element automatisch wieder an den Anfang der Liste.

Standardmäßig werden in der Liste fünf Elemente angezeigt. Möchten Sie diese Anzahl ändern, erstellen Sie eine benutzerdefinierte Skin. Weitere Informationen finden Sie unter „Benutzerdefinierte Skin für Spinnerliste erstellen“ auf Seite 125.

Der Wert der `wrapElements`-Eigenschaft bestimmt, ob ein SpinnerList-Steuerelement erneut beim ersten Element beginnt, nachdem das letzte Element in der Liste erreicht wurde. Ist `wrapElements` auf `false` gesetzt, hält der Spinner am Ende der Liste an, unabhängig von der Anzahl der Elemente in der Liste und der angezeigten Elemente.

Ist die `wrapElements`-Eigenschaft auf `true` gesetzt, beginnt der Spinner erneut beim ersten Element, jedoch nur, wenn die Liste mindestens ein Element mehr enthält als Elemente angezeigt werden können. Falls die SpinnerList beispielsweise fünf Elemente anzeigt, die Liste jedoch nur vier Elemente enthält, wird die Liste unabhängig von der Einstellung der `wrapElements`-Eigenschaft nicht umgebrochen.

Sie können das standardmäßige Umbrechen der SpinnerList überschreiben, indem Sie die `wrapElements`-Eigenschaft auf `true` oder `false` setzen.

Im folgenden Beispiel kann der Wert der `wrapElements`-Eigenschaft geändert werden:

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

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="smallList" typicalItem="45"
                       dataProvider="{new ArrayList([1,5,6,10,15,30])}"
                       wrapElements="{cb1.selected}"/>
    </s:SpinnerListContainer>

    <!-- By default, cause the elements to be wrapped by setting this to true -->
    <s:CheckBox id="cb1" label="Wrap Elements" selected="true"/>

</s:View>
```

Im Allgemeinen erwarten Benutzer, dass die Liste umgebrochen wird, wenn diese mehr Elemente enthält als gleichzeitig angezeigt werden können. Enthält die Liste weniger Elemente als der Spinner anzeigen kann, erwarten Benutzer gewöhnlicherweise, dass die Liste nicht umgebrochen wird.

Stile in der Spinnerliste festlegen

Das SpinnerList-Steuerelement unterstützt alle für das Spark-Mobildesign üblichen Textstile. Zu diesen Stilen gehören `fontSize`, `fontWeight`, `color`, `textDecoration` und Ausrichtungseigenschaften. Sie können diese Stileigenschaften direkt in der Steuerung von MXML oder in CSS festlegen. Die SpinnerList übernimmt diese Eigenschaften ebenfalls, falls diese in einem übergeordneten Container festgelegt sind.

Sie können außerdem die Auffüllungseigenschaften einer SpinnerList definieren, indem Sie die SpinnerListItemRenderer-Stileigenschaften ändern.

Im folgenden Beispiel werden die textbezogenen Stileigenschaften im SpinnerList-Typselektor und die Auffüllungseigenschaften im SpinnerListItemRenderer-Typselektor festgelegt:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/SpinnerListExamples2.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                            xmlns:s="library://ns.adobe.com/flex/spark"
                            firstView="views.SpinnerListStyles">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|SpinnerList {
            textAlign: right;
            fontSize: 13;
            fontWeight: bold;
            color: red;
        }
        s|SpinnerListItemRenderer {
            paddingTop: 5;
            paddingBottom: 5;
            paddingRight: 5;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

Definieren Sie in einer Mobilanwendung den `<fx:Style>`-Block in der Anwendungsdatei auf oberster Ebene, falls Sie Typselektoren verwenden. Andernfalls gibt der Compiler eine Warnung aus und die Stile werden nicht angewendet.

Das SpinnerList-Steuerelement unterstützt keine `accentColor`-, `backgroundAlpha`-, `backgroundColor`-, oder `chromeColor`-Stileigenschaften.

Benutzerdefinierte Skin für Spinnerliste erstellen

Sie können eine benutzerdefinierte Skin für ein SpinnerList- oder das SpinnerListContainer-Steuerelement erstellen. Dazu müssen Sie normalerweise die Quelle von SpinnerListSkin oder SpinnerListContainerSkin als Grundlage Ihrer benutzerdefinierten Skinklasse kopieren.

Sie erstellen gewöhnlicherweise benutzerdefinierte SpinnerList-Skins, um folgende Aspekte eines SpinnerList-Steuerelements oder seines Containers zu ändern:

- Größe oder Form des Rahmens um das aktuell ausgewählte Element ändern (`selectionIndicator`). Dazu müssen Sie eine benutzerdefinierte SpinnerListContainerSkin-Klasse erstellen.
- Höhe jeder Zeile definieren (`rowHeight`). Dazu müssen Sie eine benutzerdefinierte SpinnerListSkin-Klasse erstellen.
- Anzahl der angezeigten Zeilen festlegen (`requestedRowCount`). Dazu müssen Sie eine benutzerdefinierte SpinnerListSkin-Klasse erstellen.
- Darstellung des Containers definieren (wie etwa den Eckradius und die Rahmenstärke). Dazu müssen Sie eine benutzerdefinierte SpinnerListContainerSkin-Klasse erstellen.

Ein Beispiel für eine benutzerdefinierte SpinnerListSkin und SpinnerListContainerSkin finden Sie unter „Darstellung eines DateSpinner-Steuerelements anpassen“ auf Seite 109.

Bilder in einer Spinnerliste verwenden

Sie können Bilder in einem SpinnerList-Steuerelement anstelle von Textbeschriftungen verwenden, indem Sie IconItemRenderer als Elementrenderer für die SpinnerList festlegen.

Sie können Bilder in einem IconItemRenderer-Objekt entweder einbetten oder sie während der Laufzeit laden. Mobilbenutzer sollten sie möglicherweise einbetten, um das Datennetzwerk zu entlasten.

Im folgenden Beispiel werden eingebettete Bilder in einem SpinnerList-Steuerelement verwendet:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListEmbeddedImage.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Embedded Images">

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
            [Embed(source="../../assets/product_icons/flex_50x50.gif")]
            [Bindable]
            public var icon0:Class;
            [Embed(source="../../assets/product_icons/acrobat_reader_50x50.gif")]
            [Bindable]
            public var icon1:Class;
            [Embed(source="../../assets/product_icons/coldfusion_50x50.gif")]
            [Bindable]
            public var icon2:Class;
            [Embed(source="../../assets/product_icons/flash_50x50.gif")]
            [Bindable]
            public var icon3:Class;
            [Embed(source="../../assets/product_icons/flash_player_50x50.gif")]
            [Bindable]
            public var icon4:Class;
            [Embed(source="../../assets/product_icons/photoshop_50x50.gif")]
            [Bindable]
            public var icon5:Class;

            // Return an ArrayList of icons for each spinner
            private function getIconList():ArrayList {
                var a:ArrayList = new ArrayList();
                a.addItem({icon:icon0});
                a.addItem({icon:icon1});
                a.addItem({icon:icon2});
                a.addItem({icon:icon3});
                a.addItem({icon:icon4});
                a.addItem({icon:icon5});
                return a;
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="productList1" width="90" dataProvider="{getIconList()}"
selectedIndex="0">
```

```
            <s:itemRenderer>
                <fx:Component>
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
        <s:SpinnerList id="productList2" width="90" dataProvider="{getIconList()}"
selectedIndex="2">
            <s:itemRenderer>
                <fx:Component>
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
        <s:SpinnerList id="productList3" width="90" dataProvider="{getIconList()}"
selectedIndex="1">
            <s:itemRenderer>
                <fx:Component>
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
    </s:SpinnerListContainer>
</s:View>
```

Das folgende Bild zeigt, wie die Anwendung auf einem Mobilgerät angezeigt wird:

SpinnerList-Steuerelement mit eingebetteten Bildern

Kapitel 4: Anwendungsdesign und Arbeitsablauf

Persistenz in Mobilanwendungen aktivieren

Anwendungen für Mobilgeräte werden häufig durch andere Aktionen unterbrochen, z. B. Textnachrichten, Telefonanrufe oder andere Mobilanwendungen. Beim erneuten Start der unterbrochenen Anwendung erwartet der Benutzer normalerweise die Wiederherstellung des zuletzt angezeigten Status. Der Persistenzmechanismus ermöglicht es dem Gerät, den vorherigen Status der Anwendung wiederherzustellen.

Das Flex-Framework bietet zwei Arten der Persistenz für Mobilanwendungen. Durch *speicherinterne Persistenz* werden Anzeigedaten gespeichert, während der Benutzer in der Anwendung navigiert. Durch *Sitzungspersistenz* werden Daten wiederhergestellt, wenn der Benutzer die Anwendung beendet und neu gestartet hat. Die Sitzungspersistenz ist in Mobilanwendungen wichtig, weil ein Mobilbetriebssystem Anwendungen jederzeit schließen kann (beispielsweise bei wenig verfügbarem Arbeitsspeicher).


Blogger Steve Mathews schrieb einen Artikel zur einfachen Datenpersistenz in Flex-Mobilanwendungen.


Bloggerin Holly Schinsky beschreibt in Ihrem Artikel die Persistenz und Verarbeitung von Daten in Flex.

Speicherinterne Persistenz

Ansicht-Container unterstützen speicherinterne Persistenz durch Einsatz der Eigenaschaft `View.data`. Die `data`-Eigenschaft einer vorhandenen Ansicht wird automatisch gespeichert, wenn der ausgewählte Abschnitt wechselt oder auf dem ViewNavigator-Stapel eine neue Ansicht abgelegt wird, wodurch die vorhandene Ansicht zerstört wird. Die `data`-Eigenschaft der Ansicht wird wiederhergestellt, wenn das Steuerelement zu der Ansicht zurückkehrt und die Ansicht erneut instanziiert und aktiviert wird. Daher werden durch speicherinterne Persistenz Zustandsinformationen einer Ansicht zur Laufzeit beibehalten.

Sitzungspersistenz

Sitzungspersistenz behält die Anwendungszustandsinformationen zwischen Anwendungsausführungen bei. Die Container ViewNavigatorApplication und TabbedViewNavigatorApplication definieren die Eigenschaft `persistNavigatorState`, um die Sitzungspersistenz zu implementieren. Setzen Sie `persistNavigatorState` auf `true`, um die Sitzungspersistenz zu aktivieren. Standardmäßig ist `persistNavigatorState` auf `false` gesetzt.

Wenn die Sitzungspersistenz aktiviert ist, schreibt sie den Zustand einer Anwendung über ein lokal freigegebenes Objekt namens `FxAppCache` auf Festplatte. Ihre Anwendung kann auch Methoden von spark.managers.PersistenceManager verwenden, um die zusätzlichen Informationen in das lokal freigegebene Objekt zu schreiben.

ViewNavigator-Sitzungspersistenz

Der ViewNavigator-Container unterstützt Sitzungspersistenz durch Speichern des Status seines Ansichtenstapels auf Festplatte beim Beenden der Anwendung. Dieser Speichervorgang beinhaltet die `data`-Eigenschaft der aktuellen Ansicht.

Wenn die Anwendung neu gestartet wird, wird der Stapel des ViewNavigator neu initialisiert und der Anwender sieht die gleiche Ansicht mit dem gleichen Inhalt wie zuvor. Da der Stapel für jede Ansicht eine Kopie der `data`-Eigenschaft enthält, können die bisherigen Ansichten im Stapel beim Aktivieren neu erstellt werden.

TabbedViewNavigator-Sitzungspersistenz

Für den TabbedViewNavigator-Container wird durch die Sitzungspersistenz die derzeit ausgewählte Registerkarte der Registerkartenleiste beim Beenden der Anwendung gespeichert. Die Registerkarte entspricht dem ViewNavigator- und Ansichten-Stapel, der die Registerkarte definiert. Dieser Speichervorgang beinhaltet die `data`-Eigenschaft der aktuellen Ansicht. Wenn die Anwendung neu startet, werden die aktive Registerkarte und der zugehörige ViewNavigator auf den Status vor dem Beenden der Anwendung zurückgesetzt.

Hinweis: *Bei Anwendungen, die vom TabbedViewNavigatorApplication-Container definiert sind, wird nur der Stapel des aktuellen ViewNavigator gespeichert. Wenn die Anwendung also neu gestartet wird, wird nur der Status des aktuellen ViewNavigator wiederhergestellt.*

Sitzungspersistenzdatendarstellung

Der von Flex verwendete Persistenzmechanismus ist nicht verschlüsselt oder geschützt. Beibehaltene Daten werden daher in einem Format gespeichert, das von einem anderen Programm oder Benutzer interpretiert werden kann. Vertrauliche Informationen wie Benutzerinformationen sollten mit diesem Mechanismus nicht beibehalten werden. Sie haben jedoch die Möglichkeit, einen eigenen Persistenzmanager zu programmieren, der einen besseren Schutz bietet. Weitere Informationen finden Sie im Abschnitt „Persistenzmechanismus anpassen“ auf Seite 131.

Sitzungspersistenz

Das folgende Beispiel setzt die `persistNavigatorState`-Eigenschaft für eine Anwendung auf `true`, um die Sitzungspersistenz zu aktivieren:

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

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

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

Diese Anwendung setzt EmployeeMainView.mxml als Startansicht ein. In EmployeeMainView.mxml ist ein Listen-Steuerelement definiert, über das Sie einen Benutzernamen auswählen können:

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

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;
            protected function myList_changeHandler(event:IndexChangeEvent):void {
                navigator.pushView(views.EmployeeView,myList.selectedItem);
            }
        ]]>
    </fx:Script>

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

Um die Sitzungspersistenz anzuzeigen, öffnen Sie die Anwendung und wählen dann im Listen-Steuerelement „Dave“, um zur EmployeeView.mxml-Ansicht zu navigieren:

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

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

In der EmployeeView.mxml-Ansicht werden Angaben zu „Dave“ angezeigt. Dann beenden Sie die Anwendung. Nach dem Neustart der Anwendung wird wieder die EmployeeView.mxml-Ansicht mit den gleichen Daten wie beim Beenden der Anwendung angezeigt.

Daten aus einem lokal freigegebenen Objekt abrufen

Informationen in einem lokal freigegebenen Objekt werden als Schlüssel/Wert-Paar gespeichert. Die Methoden des PeristenceManager, z. B. `setPropery()` und `getProperty()` hängen davon ab, dass der Schlüssel den zugehörigen Wert aus dem lokal freigegebenen Objekt abruft.

Sie können mit der Methode `setProperty()` Ihre eigenen Schlüssel/Wert-Paare in das lokal freigegebene Objekt schreiben. Die Methode `setProperty()` hat folgende Signatur:

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

Mit der Methode `getProperty()` rufen Sie den Wert eines bestimmten Schlüssels ab. Die Methode `setProperty()` hat folgende Signatur:

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

Wenn die Eigenschaft `persistNavigatorState` auf `true` gesetzt ist, speichert der Persistenzmanager automatisch zwei Schlüssel/Wert-Paare beim Beenden der Anwendung im lokal freigegebenen Objekt:

- `applicationVersion`

  Die Datei application.xml angegebene Anwendungsversion.

- `navigatorState`

  Der Anzeigestatus des Navigators, der dem Stapel des aktuellen ViewNavigators entspricht.

Persistenz manuell durchführen

Wenn die Eigenschaft `persistNavigatorState` auf `true` gesetzt ist, führt Flex automatisch eine Sitzungspersistenz durch. Sie können Anwendungsdaten immer noch beibehalten, indem die Eigenschaft `persistNavigatorState` auf `false` gesetzt wird. In diesem Fall implementieren Sie durch Übernahme von Methoden des Persistenzmanagers Ihren eigenen Persistenzmechanismus.

Mit den Methoden `setProperty()` und `getProperty()` schreiben und lesen Sie Informationen im lokal freigegebenen Objekt. Der Persistenzmanager wird durch Aufrufen der Methode `load()` initialisiert. Rufen Sie die Methode `save()` auf, um Daten auf Festplatte zu schreiben.

Hinweis: *Wenn die Eigenschaft `persistNavigatorState` auf `false` gesetzt ist, speichert Flex nicht automatisch den Ansichtenstapel des aktuellen ViewNavigators beim Beenden der Anwendung oder stellt ihn wieder her, wenn die Anwendung neu gestartet wird.*

Persistenzereignisse abwickeln

Mithilfe der folgenden Ereignisse der Mobilanwendungscontainer können Sie einen benutzerdefinierten Persistenzmechanismus entwickeln:

- `navigatorStateSaving`
- `navigatorStateLoading`

Sie können das Speichern der Anwendungszustände unterbinden, indem Sie die Methode `preventDefault()` in der Prozedur für das Ereignis `navigatorStateSaving` aufrufen. Sie brechen Sie das Laden beim Neustarten ab, indem Sie die Methode `preventDefault()` in der Prozedur für das Ereignis `navigatorStateLoading` aufrufen.

Persistenzmechanismus anpassen

Bei aktivierter Sitzungspersistenz wird die Anwendung mit der Ansicht geöffnet, die beim Schließen der Anwendung angezeigt wurde. In der `data`-Eigenschaft der Ansicht oder an einem anderen Ort, etwa in einem freigegebenen Objekt, müssen Sie genügend Informationen speichern, um den Anwendungszustand vollständig wiederherstellen zu können.

Zum Beispiel könnte die wiederhergestellte Ansicht basierend auf der der Eigenschaft `data` der Ansicht Berechnungen durchführen müssen. Ihre Anwendung muss dann einen Neustart der Anwendung erkennen und die notwendigen Berechnungen durchführen. Eine Option ist das Überschreiben der Methoden `serializeData()` und `deserializePersistedData()` der Ansicht, um Ihre eigenen Aktionen auszuführen, wenn die Anwendung beendet oder neu gestartet wird.

Eingebaute Datentyp-Unterstützung für die Sitzungspersistenz

Der Persistenzmechanismus unterstützt automatisch alle eingebauten Datentypen, u. a.: Nummer, String (Zeichenkette), Array, Vektor, Objekt, uint, int und Boolean. Diese Datentypen werden automatisch vom Persistenzmechanismus gespeichert.

Unterstützung benutzerdefinierter Klassen für Sitzungspersistenz

Viele Anwendungen definieren ihre Daten über benutzerdefinierte Klassen. Wenn eine benutzerdefinierte Klasse Eigenschaften enthält, die von den eingebauten Datentypen definiert werden, kann der Persistenzmechanismus die Klasse automatisch laden und speichern. Zuerst muss die Klasse jedoch beim Persistenzmechanismus angemeldet werden. Dazu dient die Methode flash.net.registerClassAlias(). In der Regel rufen Sie diese Methode im Ereignis `preinitialize` der Anwendung auf, bevor die Persistenzspeicherung initialisiert wird oder Daten darin gespeichert werden.

Wenn Sie eine komplexe Klasse definieren, die andere Datentypen als die integrierten Datentypen verwendet, müssen die Daten in einen bekannten Datentyp umgewandelt werden, z. B. in einen String. Wenn in der Klasse private Variablen definiert werden, bleiben sie nicht automatisch erhalten. Zur Unterstützung der komplexen Klasse im Persistenzmechanismus muss in die Klasse die flash.utils.IExternalizable-Schnittstelle implementiert werden. Diese Schnittstelle erfordert eine Implementierung der Methoden `writeExternal()` und `readExternal()` in die Klasse, damit eine Instanz der Klasse gespeichert und wiederhergestellt werden kann.

Unterstützung mehrerer Bildschirmgrößen und DPI-Werte in Mobilanwendungen

Richtlinien für die Unterstützung mehrerer Bildschirmgrößen und DPI-Werte

Zur Entwicklung einer plattformunabhängigen Anwendung sollten Sie die unterschiedlichen Ausgabegeräte beachten. Geräte können unterschiedliche Bildschirmgrößen oder Auflösungen und unterschiedliche DPI-Werte oder Pixeldichten aufweisen.

Flex-Ingenieur Jason SJ beschreibt zwei Herangehensweisen zum Erstellen auflösungsunabhängiger Mobilanwendungen in seinem Blog.

Terminologie

Auflösung ist die Anzahl von Pixeln für die Höhe und die Breite, d. h., die von einem Gerät insgesamt unterstützte Pixelanzahl.

DPI (Dots per Inch) ist die Anzahl von Punkten pro Zoll, d. h., die Pixeldichte eines Gerätebildschirms. Der Begriff DPI ist äquivalent mit PPI (Pixels per Inch, Pixel pro Zoll).

Flex-Unterstützung für DPIs

Die folgenden Flex-Funktionen vereinfachen die Entwicklung auflösungs- und DPI-unabhängiger Anwendungen:

Skins DPI-kompatible Skins für Mobilkomponenten. Die Standard-Mobilskins benötigen keine zusätzliche Codierung, um bei den Auflösungen der meisten Geräte gut skaliert zu werden.

applicationDPI Eine Eigenschaft für die Definition der Größe, für die die benutzerdefinierten Skins entworfen werden. Angenommen, Sie legen für diese Eigenschaft einen bestimmten DPI-Wert fest und ein Benutzer führt die Anwendung auf einem Gerät mit einem anderen DPI-Wert aus. Flex skaliert alle Anwendungselemente auf den DPI-Wert des verwendeten Geräts.

Die Standard-Mobilskins sind DPI-unabhängig, sowohl mit als auch ohne DPI-Skalierung. Wenn Sie keine Komponenten mit statischen Größen oder benutzerdefinierte Skins verwenden, müssen Sie die `applicationDPI`-Eigenschaft daher meist nicht festlegen.

Dynamische Layouts

Dynamische Layouts unterstützten das Ausgleichen von Unterschieden in der Auflösung. Wenn Sie beispielsweise die Breite eines Steuerelements auf 100 % festlegen, füllt dieses die Breite des Bildschirms immer aus, unabhängig davon, ob dessen Auflösung 480 x 854 oder 480 x 800 beträgt.

applicationDPI-Eigenschaft festlegen

Wenn Sie von der Pixeldichte unabhängige Anwendungen erstellen, können Sie den DPI-Zielwert im Stammanwendungs-Tag festlegen. (Für Mobilanwendungen lautet das Stamm-Tag `<s:ViewNavigatorApplication>`, `<s:TabbedViewNavigatorApplication>` oder `<s:Application>`.)

Sie legen den Wert der `applicationDPI`-Eigenschaft je nach etwaiger Auflösung des Zielgeräts auf 160, 240 oder 320 fest. Beispiel:

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

Wenn Sie die `applicationDPI`-Eigenschaft festlegen, definieren Sie letztlich eine Skalierung für die Anwendung nach Vergleich mit der eigentlichen Auflösung des Zielgeräts (der `runtimeDPI`) zur Laufzeit. Wenn Sie zum Beispiel die `applicationDPI`-Eigenschaft auf 160 festlegen und das Zielgerät eine `runtimeDPI` von 160 besitzt, ist der Skalierungsfaktor 1 (keine Skalierung). Wenn Sie die `applicationDPI`-Eigenschaft auf 240 festlegen, ist der Skalierungsfaktor 1,5 (alles wird auf 150 % vergrößert). Bei 320 beträgt der Skalierungsfaktor 2, sodass alles auf 200 % vergrößert wird.

In manchen Fällen kann die nicht-ganzzahlige Skalierung zu unerwünschten Artefakten aufgrund von Interpolation führen, wie z. B. verschwommene Zeilen.

DPI-Skalierung deaktivieren

Wenn Sie die DPI-Skalierung für die Anwendung deaktivieren möchten, legen Sie den Wert der `applicationDPI`-Eigenschaft nicht fest.

applicationDPI und runtimeDPI

In der folgenden Tabelle werden zwei Eigenschaften der Application-Klasse beschrieben, die für die Arbeit mit Anwendungen für unterschiedliche Auflösungen von wesentlicher Bedeutung sind:

Eigenschaft	Beschreibung
`applicationDPI`	Die Zieldichte oder DPI der Anwendung. Wenn Sie einen Wert für diese Eigenschaft angeben, wendet Flex einen Skalierungsfaktor auf die Stammanwendung an. Dadurch wird eine für einen bestimmten DPI-Wert entwickelte Anwendung so skaliert, dass sie auf einem Gerät mit einem anderen DPI-Wert gut dargestellt wird. Der Skalierungsfaktor wird durch Vergleich des Werts dieser Eigenschaft mit der `runtimeDPI`-Eigenschaft berechnet. Dieser Skalierungsfaktor wird auf die gesamte Anwendung angewendet, einschließlich des Preloaders, der Popups und aller Komponenten auf der Bühne. Ohne Angabe gibt diese Eigenschaft den gleichen Wert wie die `runtimeDPI`-Eigenschaft zurück. Diese Eigenschaft kann nicht in ActionScript, sondern nur in MXML festgelegt werden. Den Wert dieser Eigenschaft können Sie zur Laufzeit nicht ändern.
`runtimeDPI`	Die Dichte oder der DPI-Wert des Geräts, auf dem die Anwendung derzeit ausgeführt wird. Gibt den Wert der `Capabilities.screenDPI`-Eigenschaft zurück, gerundet auf eine der von der DPIClassification-Klasse definierten Konstanten. Diese Eigenschaft ist schreibgeschützt.

Auflösungs- und DPI-unabhängige Anwendungen erstellen

Auflösungs- und DPI-unabhängige Anwendungen weisen die folgenden Merkmale auf:

Bilder Vektorbilder werden problemlos auf die eigentliche Auflösung des Zielgeräts skaliert. Bitmaps werden andererseits nicht immer so optimal skaliert. In solchen Fällen können Sie mithilfe der MultiDPIBitmapSource-Klasse je nach Geräteauflösung Bitmaps mit unterschiedlichen Auflösungen laden.

Text Die Schriftgröße des Texts (nicht der Text selbst) wird auf die Auflösung skaliert.

Layouts Verwenden Sie dynamische Layouts, um sicherzustellen, dass die skalierte Anwendung einwandfrei dargestellt wird. Vermeiden Sie im Allgemeinen beschränkungsbasierte Layouts, in denen Sie Pixelgrenzwerte mit absoluten Werten angeben. Wenn Sie Beschränkungen verwenden, rechnen Sie im Wert der `applicationDPI`-Eigenschaft die Skalierung ein.

Skalieren Verwenden Sie für das Application-Objekt nicht die `scaleX`-Eigenschaft und die `scaleY`-Eigenschaft. Wenn Sie die `applicationDPI`-Eigenschaft festlegen, führt Flex die Skalierung automatisch aus.

Stile Stileigenschaften können mithilfe von Stylesheets an das Betriebssystem des Zielgeräts und die DPI-Einstellungen der Anwendung angepasst werden.

Skins Die Flex-Skins im Mobildesign bestimmen die zur Laufzeit zu verwendenden Elemente anhand des DPI-Werts der Anwendung. Alle in FXG-Dateien definierten visuellen Skinelemente werden an das Zielgerät angepasst.

Anwendungsgröße Legen Sie die Breite und Höhe der Anwendung nicht explizit fest. Verwenden Sie beim Berechnen der Größe von benutzerdefinierten Komponenten oder Popups nicht die `stageWidth`-Eigenschaft und die `stageHeight`-Eigenschaft. Verwenden Sie stattdessen die `SystemManager.screen`-Eigenschaft.

Laufzeit-DPI bestimmen

Beim Anwendungsstart ruft die Anwendung den Wert der `runtimeDPI`-Eigenschaft aus der `Capabilities.screenDPI`-Eigenschaft von Flash Player ab. Diese Eigenschaft ist einer der von der DPIClassification-Klasse definierten Konstanten zugeordnet. Beispielsweise ist ein Droid mit 232 DPI dem DPI-Laufzeitwert 240 zugeordnet. Die DPI-Werte von Geräten stimmen nicht immer genau mit den DPIClassification-Konstanten (160, 240 oder 320) überein. Stattdessen sind sie diesen Klassifikationen auf Grundlage von Zielwerten zugeordnet.

Dabei gelten die folgenden Zuordnungen:

DPIClassification-Konstante	160 DPI	240 DPI	320 DPI
Tatsächliche Geräte-DPI	<200	>=200 und <280	>=280

Sie können diese Zuordnungen anpassen, um das Standardverhalten zu überschreiben oder Geräte anzupassen, die ihren eigenen DPI-Wert falsch angeben. Weitere Informationen finden Sie unter „DPI-Standardwert überschreiben“ auf Seite 144.

Automatische Skalierung aktivieren bzw. deaktivieren

Wenn Sie die automatische Skalierung auswählen (durch Festlegen des Werts für die `applicationDPI`-Eigenschaft), müssen Sie zwischen Bequemlichkeit und der pixelgenauen grafischen Darstellung abwägen. Wenn Sie für die `applicationDPI`-Eigenschaft die automatische Skalierung der Anwendung festlegen, verwendet Flex Skins, die für die `applicationDPI`-Eigenschaft ausgelegt sind. Flex skaliert die Skins nach oben oder unten, um sie an die tatsächliche Pixeldichte des Geräts anzupassen. Andere Elemente der Anwendung und Layoutpositionen werden ebenfalls skaliert.

Wenn Sie die automatische Skalierung verwenden möchten und eigene Skins oder Elemente erstellen, die für einen bestimmten DPI-Wert ausgelegt sind, führen Sie in der Regel die folgenden Schritte aus:

- Sie erstellen einen einzelnen Satz von Skins und Ansicht-/Komponenten-Layouts, die auf die angegebene `applicationDPI` ausgerichtet sind.
- Sie erstellen mehrere Versionen eines Bitmap-Elements, das in Ihren Skins oder an anderer Stelle in Ihrer Anwendung verwendet wird, und geben Sie über die MultiDPIBitmapSource-Klasse an. Vektor-Elemente und Text in Ihren Skins brauchen bei der automatischen Skalierung nicht berücksichtigt zu werden.
- Verwenden Sie die `@media`-Regel nicht in Stylesheets, da für Ihre Anwendung nur ein einziger DPI-Zielwert berücksichtigt wird.
- Testen Sie Ihre Anwendung auf Geräten mit unterschiedlichen Auflösungen, um das Aussehen der skalierten Anwendung zu überprüfen. Überprüfen Sie insbesondere Geräte, die eine Skalierung mit einem nicht-ganzzahligen Faktor verursachen. Wenn beispielsweise `applicationDPI` den Wert 160 hat, testen Sie Ihre Anwendung auf Geräten mit einem DPI-Wert von 240.

Wenn Sie die automatische Skalierung nicht verwenden möchten (indem Sie die `applicationDPI`-Eigenschaft nicht festgelegt haben), rufen Sie den Wert für `applicationDPI` ab. Bestimmen Sie anhand dieser Eigenschaft die tatsächliche DPI-Klassifizierung des Geräts und passen Sie Ihre Anwendung zur Laufzeit wie folgt an:

- Sie erstellen mehrere Sätze von Skins und Layouts, die bei jeder Laufzeit-DPI-Klassifizierung anvisiert werden, oder erstellen einen einzigen Satz von Skins und Layouts, der an die verschiedenen Auflösungen dynamisch angepasst wird. (Die integrierten Flex-Skins arbeiten nach letzterem Ansatz – jede Skin-Klasse überprüft die `applicationDPI`-Eigenschaft und richtet sich selbst entsprechend ein.)
- Mit den `@media`-Regeln in Ihren Stylesheets filtern Sie die CSS-Regeln basierend auf der DPI-Klassifizierung des Geräts. Normalerweise passen Sie die Schriftgrößen und Auffüllungswerte für jeden DPI-Wert an.
- Testen Sie Ihre Anwendung auf Geräten mit unterschiedlichen Auflösungen, um sicherzustellen, dass Ihre Skins und Layouts entsprechend skaliert werden.

Stile anhand der DPI auswählen

Flex bietet Unterstützung für die Anwendung von Stilen anhand des Zielbetriebssystems und des Anwendungs-DPI-Werts in CSS. Dazu verwenden Sie die `@media`-Regel des Stylesheets. Die `@media`-Regel ist Teil der CSS-Spezifikation und wird von Flex um zusätzliche Eigenschaften erweitert: `application-dpi` und `os-platform`. Mit diesen Eigenschaften können Sie anhand des Anwendungs-DPI-Werts und der Plattform, auf der die Anwendung ausgeführt wird, Stile selektiv anwenden.

Im folgenden Beispiel wird die standardmäßige fontSize-Stileigenschaft des Spark `Button`-Steuerelements auf „12“ festgelegt. Wenn das Gerät 240 DPI verwendet und unter dem Betriebssystem Android läuft, beträgt die `fontSize`-Eigenschaft 10.

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

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

</s:ViewNavigatorApplication>
```

Werte für die application-dpi-Eigenschaft

Die `application-dpi`-Eigenschaft im CSS wird mit dem Wert der `applicationDPI`-Stileigenschaft verglichen, die für die Stammanwendung festgelegt ist. Folgende Werte sind für die `application-dpi`-Eigenschaft des CSS gültig:

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

Jedem unterstützten Wert für `application-dpi` entspricht eine Konstante in der DPIClassification-Klasse.

Werte für die os-platform-Eigenschaft

Die `os-platform`-Eigenschaft des CSS stimmt mit dem Wert der `flash.system.Capabilities.version`-Eigenschaft von Flash Player überein. Folgende Werte sind für die `os-platform`-Eigenschaft des CSS gültig:

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

Bei der Suche nach Übereinstimmungen wird die Groß- und Kleinschreibung nicht berücksichtigt.

Wenn keiner der Einträge übereinstimmt, sucht Flex nach einer sekundären Übereinstimmung, indem die ersten drei Buchstaben mit der Liste unterstützter Plattformen verglichen werden.

Standardwerte für die Eigenschaften application-dpi und os-platform

Wenn Sie nicht explizit einen Ausdruck definieren, der die `application-dpi`-Eigenschaft oder die `os-platform`-Eigenschaft enthält, wird davon ausgegangen, dass alle Ausdrücke übereinstimmen.

Operatoren für die @media-Regel

Die `@media`-Regel unterstützt die allgemeinen Operatoren „and“ und „not”. Außerdem unterstützt sie kommagetrennte Listen. Das Trennen von Ausdrücken durch ein Komma impliziert eine „or“-Bedingung.

Bei Verwendung des Operators „not“ muss „not“ das erste Schlüsselwort im Ausdruck sein. Mit diesem Operator wird der gesamte Ausdruck negiert, nicht nur die Eigenschaft, die auf „not“ folgt. Aufgrund von Bug SDK-29191 muss auf den Operator „not“ vor Ausdrücken ein Medientyp wie z. B. „all“ folgen.

Im folgenden Beispiel wird veranschaulicht, wie einige dieser häufigen Operatoren verwendet werden:

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

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

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

</s:ViewNavigatorApplication>
```

Bitmapelemente anhand der DPI auswählen

Bitmap-Bildelemente werden meist nur optimal in der Auflösung gerendert, für die sie entworfen wurden. Dies kann Schwierigkeiten beim Entwerfen von Anwendungen für mehrere Auflösungen bedeuten. Die Lösung besteht darin, mehrere Bitmaps für die einzelnen Auflösungen zu erstellen und je nach Wert der `runtimeDPI`-Eigenschaft der Anwendung die richtige zu laden.

Die BitmapImage-Komponente und die Image-Komponente von Spark besitzen eine `source`-Eigenschaft vom Typ Object. Aufgrund dieser Eigenschaft können Sie eine Klasse übergeben, die die zu verwendenden Elemente definiert. In diesem Fall übergeben Sie die MultiDPIBitmapSource-Klasse, um je nach Wert der `runtimeDPI`-Eigenschaft unterschiedliche Quellen zuzuordnen.

Im folgenden Beispiel wird je nach DPI ein jeweils anderes Bild geladen:

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

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

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


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

Wenn Sie die BitmapImage-Klasse und die Image-Klasse mit MultiDPIBitmapSource in einer *Desktop*anwendung verwenden, wird für die Quelle die `source160dpi`-Eigenschaft verwendet.

Die `icon`-Eigenschaft des Button-Steuerelements akzeptiert auch eine Klasse als Argument. Daher können Sie als Quelle für das Symbol des Button-Steuerelements auch ein MultiDPIBitmapSource-Objekt verwenden. Die Quelle des Symbols können Sie inline definieren, wie das folgende Beispiel veranschaulicht:

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

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

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

Symbole können Sie auch in einem `<fx:Declarations>`-Block deklarieren und die Quelle per Datenbindung zuweisen, wie das folgende Beispiel zeigt:

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

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

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

Wenn die `runtimeDPI`-Eigenschaft einer `source`*`XXX`*`dpi`-Eigenschaft zugeordnet wird, die `null` oder eine leere Zeichenfolge ("") ist, verwendet Flash Player die Eigenschaft mit der nächsthöheren Dichte als Quelle. Wenn dieser Wert ebenfalls `null` oder leer ist, wird die nächstniedrige Dichte verwendet. Wenn dieser Wert *ebenfalls*`null` oder leer ist, weist Flex `null` als Quelle zu, und es wird kein Bild angezeigt. Das heißt, Sie können nicht explizit angeben, dass für einen bestimmten DPI-Wert kein Bild angezeigt werden soll.

Skin-Elemente anhand der DPI auswählen

Die Standardskins im Mobildesign enthalten in ihren Konstruktoren eine Logik zur Elementauswahl anhand des Werts der `applicationDPI`-Eigenschaft. Mit diesen Klassen werden Elemente ausgewählt, die dem DPI-Zielwert am ehesten entsprechen. Wenn Sie benutzerdefinierte Skins entwerfen, die mit oder ohne DPI-Skalierung funktionieren, verwenden Sie die `applicationDPI`-Eigenschaft statt der `runtimeDPI`-Eigenschaft.

Beispielsweise verwendet die spark.skins.mobile.ButtonSkin-Klasse eine der folgenden ähnelnde switch-/case-Anweisung, mit der für bestimmte DPI-Werte entwickelte FXG-Elemente ausgewählt werden:

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

Neben der bedingten Auswahl von FXG-Elementen legen die Mobilskinklassen auch die Werte anderer Stileigenschaften fest, z. B. für Lücken und Abstände im Layout. Diese Einstellungen basieren auf dem DPI-Wert des Zielgeräts.

applicationDPI nicht festlegen

Wenn Sie die `applicationDPI`-Eigenschaft nicht festlegen, verwenden Skins die standardmäßige `runtimeDPI`-Eigenschaft. Dieser Mechanismus garantiert, dass eine Skin, deren Werte auf der `applicationDPI`-Eigenschaft und nicht auf der `runtimeDPI`-Eigenschaft beruhen, mit oder ohne DPI-Skalierung die richtige Ressource verwendet.

Beim Erstellen benutzerdefinierter Skins können Sie festlegen, dass die `applicationDPI`-Einstellung ignoriert wird. In diesem Fall werden Skins weiterhin dem DPI-Wert des Zielgeräts entsprechend skaliert, möglicherweise jedoch nicht optimal angezeigt, wenn deren Elemente nicht speziell für den betreffenden DPI-Wert entwickelt wurden.

applicationDPI in CSS verwenden

Über den Wert der `applicationDPI`-Eigenschaft in der `@media`-Auswahl des CSS können Sie die von der Mobil- oder Tablet-Anwendung verwendeten Stile anpassen, ohne benutzerdefinierte Skins erstellen zu müssen. Weitere Informationen finden Sie unter „Stile anhand der DPI auswählen“ auf Seite 136.

Skalierungsfaktor und aktuelle DPI manuell bestimmen

Um eine Mobil- oder Tablet-Anwendung manuell anzuweisen, Elemente anhand des DPI-Werts des Zielgeräts auszuwählen, können Sie den Skalierungsfaktor zur Laufzeit berechnen. Dazu teilen Sie den Wert der `runtimeDPI`-Eigenschaft durch den der `applicationDPI`-Stileigenschaft:

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

Den berechneten Skalierungsfaktor können Sie für die Elementauswahl verwenden. Im folgenden Beispiel werden für jeden DPI-Wert benutzerdefinierte Speicherorte von Bitmapelementen definiert. Anschließend wird ein Bild von diesem benutzerdefinierten Speicherort geladen:

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

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

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

</s:ViewNavigatorApplication>
```

Dies ist die Ansicht, die den Skalierungsfaktor verwendet:

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

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

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

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

DPI-Standardwert überschreiben

Wenn Sie den DPI-Wert der Anwendung festgelegt haben, wird die Anwendung anhand des DPI-Werts skaliert, der von dem Gerät gemeldet wird, auf dem die Anwendung ausgeführt wird. Es kann vorkommen, dass Geräte falsche DPI-Werte melden oder Sie die standardmäßige DPI-Auswahlmethode durch eine benutzerdefinierte Skalierungsmethode ersetzen möchten.

Das Standardskalierungsverhalten einer Anwendung überschreiben Sie in den DPI-Standardzuordnungen. Wenn ein Gerät zum Beispiel den falschen Wert von 240 DPI statt des richtigen von 160 DPI meldet, können Sie eine benutzerdefinierte Zuordnung erstellen, mit der dieses Gerät gesucht und als 160 DPI klassifiziert wird.

Zum Überschreiben des DPI-Werts eines speziellen Geräts verweisen Sie die `runtimeDPIProvider`-Eigenschaft der Application-Klasse auf eine Unterklasse der RuntimeDPIProvider-Klasse. In der Unterklasse überschreiben Sie den `runtimeDPI`-Getter und fügen eine Logik hinzu, die eine benutzerdefinierte DPI-Zuordnung bereitstellt. Fügen Sie im Framework keine Abhängigkeiten zu anderen Klassen hinzu, wie z. B. UIComponent. Mit dieser Unterklasse können nur Player-APIs aufgerufen werden.

Im folgenden Beispiel wird eine benutzerdefinierte DPI-Zuordnung für ein Gerät festgelegt, dessen `Capabilities.os`-Eigenschaft mit „Mac 10.6.5“ übereinstimmt:

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

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

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

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

        return DPIClassification.DPI_320;
    }
}
}
```

Die folgende Anwendung bestimmt mithilfe der DPITestClass den für die Skalierung zu verwendenden DPI-Laufzeitwert. Es wird auf die runtimeDPIProvider-Eigenschaft der `ViewNavigatorApplication`-Klasse verwiesen:

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

</s:ViewNavigatorApplication>
```

Im Folgenden finden Sie ein weiteres Beispiel für eine Unterklasse der RuntimeDPIProvider-Klasse. In diesem Fall überprüft die benutzerdefinierte Klasse die x- und y-Auflösung des Geräts, um zu ermitteln, ob der DPI-Wert vom Gerät nicht ordnungsgemäß gemeldet wird:

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

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

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

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

        return DPIClassification.DPI_320;
    }
}
}
```

Kapitel 5: Text

Text in Mobilanwendungen

Richtlinien für Text in Mobilanwendungen

Einige Spark-Textsteuerelemente wurden für Mobilanwendungen optimiert. Verwenden Sie nach Möglichkeit die folgenden Textsteuerelemente:

- Spark TextArea
- Spark TextInput
- Spark Label

Die Textsteuerelemente, die Benutzerinteraktion zulassen (TextArea und TextInput), verwenden die StageText-Klasse als zugrunde liegenden Eingabemechanismus. StageText stellt eine Verknüpfung mit den nativen Textsteuerelementen des zugrunde liegenden Betriebssystems her. Daher verhalten sich diese Textsteuerelemente wie native Steuerelemente und nicht wie typische Flex-Steuerelemente.

Das Verwenden von StageText auf kompatiblen Geräten hat die folgenden Vorteile:

- Native Leistung und Layout der virtuellen Tastatur
- Automatische Vervollständigung
- Autokorrektur
- Textauswahl durch Berührung
- Anpassbare virtuelle Tastaturen
- Tasteneinschränkungen

Adobe-Guru Christian Cantrell beschreibt die Vor- und Nachteile StageText-basierter Steuerelemente.

TextField-basierte Versionen des TextArea- und des TextInput-Steuerelements sind auch verfügbar. Sie können mithilfe dieser Versionen Schriftarten einbetten oder andere Funktionen verwenden, die in den StageText-basierten Versionen nicht verfügbar sind.

Skins für Textsteuerelemente von Mobilanwendungen

Beim Erstellen einer Mobilanwendung wird von Flex automatisch das Mobildesign angewendet. Folglich verwenden die Spark-Steuerelemente TextInput und TextArea standardmäßig die folgenden StageText-basierten Mobilskins:

- StageTextAreaSkin
- StageTextInputSkin

Die StageTextAreaSkin- und StageTextInputSkin-Klassen sind für Mobilanwendungen optimiert und basieren auf der StageTextSkinBase-Klasse. Sie dienen als Wrapper um die nativen Texteingabeklassen. Sie unterstützen jedoch nicht die folgenden Funktionen der Skins, die nicht auf TextField basieren:

Von TextField-basierten Steuerelementen unterstützt	Nicht von TextField-basierten Steuerelementen unterstützt
Bildlauf in Formularen	Text Layout Framework (TLF)
Textabmessung	Bidirektionalität und Spiegelung
Beschneidung	Compact Font Format (CFF)
Eingebettete Schriftarten	RichEditableText für die Textdarstellung
Alpha-Bruchwerte	HTML-Text
Flash Text Engine (FTE)	
Zugriff auf grundlegende Tastaturereignisse wie `keyUp` und `keyDown`	

Einige dieser Einschränkungen können mithilfe der TextField-basierten Versionen umgangen werden. Um die TextField-basierten Versionen der Texteingabesteuerelemente zu verwenden, müssen ihre Skinklassen auf die TextField-Versionen TextInputSkin und TextAreaSkin verweisen. Beispiel:

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin" text="TextField-based Skin"/>
```

Das Spark Label-Steuerelement verwendet keine Skin, aber auch kein TLF.

TLF in einer Mobilanwendung

In Mobilanwendungen, die mit Text Layout Framework (TLF) arbeiten, sollten Textsteuerelemente vermieden werden. Die Mobilskins der TextArea- und TextInput-Steuerelemente sind für Mobilanwendungen optimiert und verwenden nicht TLF wie ihre Entsprechungen in Desktop- und webbasierten Anwendungen. TLF stattet Anwendungen mit einer breiten Palette an Steuerelementen zur Textdarstellung aus.

Vermeiden Sie die folgenden Textsteuerelemente in einer Mobilanwendung, da sie TLF verwenden und die Skins nicht für Mobilanwendungen optimiert sind:

- Spark RichText
- Spark RichEditableText

Eingabe mit virtuellen Tastaturen

Wenn ein Benutzer einem Textsteuerelement zur Eingabe den Fokus erteilt, zeigen Mobilgeräte ohne Tastatur eine virtuelle Tastatur an. Sie können die verfügbaren Tasten sowie andere Eigenschaften der virtuellen Tastatur begrenzt steuern. Beispielsweise können Sie die Autokorrektur und die automatische Großschreibung aktivieren und aus verschiedenen vordefinierten Tastaturlayouts wählen.

Weitere Informationen finden Sie unter „Virtuelle Tastatur in Mobilanwendungen verwenden“ auf Seite 157.

Bildlauf mit Texteingabesteuerelementen

Die Standardmobilskins für die TextInput- und TextArea-Steuerelemente unterstützen keinen Bildlauf in Formularen. Das heißt, dass Sie diese Steuerelemente in Formularen oder Ansichten anzeigen können, in denen das Steuerelement Bildläufe durchführen muss. Fügen Sie sie dort dennoch ein, werden visuelle Artefakte infolge der Implementierung der Steuerelemente angezeigt.

Verwenden Sie für Texteingabesteuerelemente in einem Bildlaufcontainer die TextField-basierten Skins anstelle der StageText-basierten Skins. Weitere Informationen finden Sie unter „Überlegungen zum Bildlauf mit StageText“ auf Seite 72.

Übergänge mit Texteingabesteuerelementen

Zum nahtlosen Animieren ersetzt die Laufzeit beim Abspielen einer Animation StageText-Steuerelemente durch Bitmaps, die aus ihnen erfasst wurden. Dadurch kann es am Anfang von Übergangsanimationen zu einer kurzen Verzögerung kommen und es können einige Grafikartefakte am Anfang und Ende der Animationen auftreten.

Die kurze Verzögerung ist die Zeit, die zum Erfassen von Bitmap-Darstellungen des Texts in den Komponenten erforderlich ist. Diese Verzögerung nimmt zu, wenn der Bereich und die Anzahl von StageText-basierten Steuerelementen zunimmt. Um die Verzögerung zu reduzieren, vermeiden Sie die Animation großer oder zahlreicher StageText-basierter Komponenten.

Popups mit Texteingabesteuerelementen

StageText-basierte Texteingaben außerhalb des obersten Popups werden durch Bitmap-Darstellungen ersetzt, wenn ein Popup angezeigt wird. Ergebnis:

- Verwenden Sie modale Popups anstatt nicht-modaler Popups. Wenn ein modales Popup angezeigt wird, wird von Komponenten außerhalb des Popups erwartet, dass sie ihre Interaktivität verlieren. In diesen Fällen ist das Ersetzen von StageText-Komponenten durch Bitmaps weniger erkennbar.
- StageText-basierte Komponenten dürfen nur in Popups verwendet werden, die die IFocusManagerContainer-Schnittstelle implementieren. Verwenden Sie beispielsweise SkinnableContainer oder einen seiner Derivate als Basis für Popups.
- Verwenden Sie einen Callout-Container, wenn Textkomponenten in unteren Ebenen aktiv bleiben sollen. Wenn eine Textkomponente über eine Callout-Komponente verfügt, bleibt die Textkomponente aktiv und legt den Pfeil der Callout-Komponente so fest, dass er auf diese Textkomponente zeigt. Das ist ein Signal für den Benutzer, dass die Textkomponente noch immer verwendet werden kann.
- Verwenden Sie nicht mehrere Popups gleichzeitig. Wenn mehrere Popups gleichzeitig sichtbar sind, ist nur das oberste interaktiv. Wenn sich Popups jedoch nicht überlappen, wissen Benutzer nicht, welches das oberste Popup ist.
- Wenn nicht-modale Popups Textsteuerelemente überlappen, positionieren Sie das Popup so, dass sie sich fast vollständig überlappen. Wenn Benutzer den Text nicht sehen können, nehmen sie nicht an, dass das Textsteuerelement noch immer interaktiv ist.

Schriftarten in eine Mobilanwendung einbetten

Sie können keine Schriftarten in ein Texteingabesteuerelement einbetten, das StageText verwendet. Verwenden Sie stattdessen für die Texteingabesteuerelemente die TextField-basierten Skins. Außerdem können Sie nicht das Label-Steuerelement mit eingebetteten Schriftarten verwenden, da es FTE verwendet, was CFF-basierte Schriftarten erfordert. CFF-Schriftarten sind für Mobilanwendungen nicht gut geeignet.

Weitere Informationen finden Sie unter „Schriftarten in eine Mobilanwendung einbetten“ auf Seite 171.

Hierarchie der StageText-Steuerelementklasse

Die StageText-Klassen (TextInput und TextArea) verfügen über eine komplexe Klassenhierarchie. Die Basisklassen selbst weisen die folgende Hierarchie auf:

```
    UIComponent
        |
SkinnableComponent
        |
SkinnableTextBase
        |
TextInput/TextArea
```

Wie alle Spark-Klassen weisen die Skinklassen ihre eigene Hierarchie auf:

```
    UIComponent
        |
    MobileSkin    StyleableStageText
        |              |
StageTextSkinBase: textDisplay
        |
StageTextInputSkin/StageTextAreaSkin
```

In der Basisskinklasse stellt die Eigenschaft `textDisplay` die Verknüpfung zur nativen Texteingabe als StyleableStageText-Objekt bereit. Diese Klasse definiert außerdem die Stile, die in den StageText-basierten Texteingabesteuerelementen verfügbar sind.

Label-Steuerelement in Mobilanwendungen verwenden

Das Steuerelement Spark Label eignet sich ideal für einzelne Zeilen nicht editierbaren, nicht auswählbaren Texts.

Im folgenden Beispiel wird ein einfaches Label-Steuerelement in einer Mobilanwendung verwendet:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleLabel.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Simple Label">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="This is a simple Label control."/>

</s:View>
```

Das Steuerelement Label arbeitet mit FTE, was nicht so leistungsfähig ist wie Steuerelemente, die extra für Mobilanwendungen entwickelt wurden (z. B. TextInput und TextArea). Das Label-Steuerelement verwendet jedoch kein TLF, daher sollte es im Allgemeinen besser ausgeführt werden als Steuerelemente wie RichText und RichEditableText, die kein TLF unterstützen.

Allgemein gilt, dass Spark Label-Steuerelemente nur sparsam in Mobilanwendungen eingesetzt werden sollten. Verwenden Sie das Steuerelement Spark Label nicht für Skins oder Elementrenderer. Verwenden Sie beim Erstellen eines ActionScript-basierten Elementrenderers die StyleableTextField-Klasse zur Textdarstellung. Für eine MXML-basierte Komponente können Sie noch immer Label verwenden.

Das Label-Steuerelement sollten Sie nicht verwenden, wenn Sie Schriftarten in einer Mobilanwendung einbetten, da das Label-Steuerelement CFF verwendet. Verwenden Sie stattdessen die TextField-basierte Version des TextArea-Steuerelements. Weitere Informationen finden Sie unter „Schriftarten in eine Mobilanwendung einbetten“ auf Seite 171.

TextArea-Steuerelement in Mobilanwendungen verwenden

Das TextArea-Spark-Steuerelement ermöglicht die Eingabe und Bearbeitung mehrerer Textzeilen. Diese Klasse ist für Mobilanwendungen konzipiert.

Standardmäßig verwendet das TextArea-Steuerelement die virtuelle Tastatur, die Verknüpfungen zu nativen Methoden des zugrunde liegenden Betriebssystems bereitstellt. Demzufolge werden Funktionen wie Autokorrektur, automatische Vervollständigung und Anpassen der virtuellen Tastatur unterstützt.

Im folgenden Beispiel wird ein TextArea-Steuerelement in einer Mobilanwendung verwendet:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleTextArea.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Simple TextArea">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            // Note the use of \n to add line feeds/carriage returns
            // and \" to add quotation marks.
            [Bindable]
           public var myText:String ="\"Lorem ipsum dolor sit amet, consectetuer adipiscing
elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat
volutpat.\"\n\n\"Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit
lobortis nisl ut aliquip ex ea commodo consequat.\"";
        ]]>
    </fx:Script>

    <!-- Basic TextArea control with multiple lines of text. -->
    <s:TextArea id="myTA" height="75%" text="{myText}"
                paddingLeft="20" paddingTop="20"
                paddingRight="20" paddingBottom="20"/>

</s:View>
```

In Mobilanwendungen verwendet das TextArea-Steuerelement standardmäßig die StageTextAreaSkin-Klasse als Skin. Diese Skin verwendet zur Textdarstellung die StyleableStageText-Klasse anstelle der RichEditableText-Klasse. Demzufolge wird TLF vom TextArea-Steuerelement nicht unterstützt. Nur eine Untergruppe von Stilen, die im TextArea-Steuerelement mit der Skin, die keine Mobilskin ist, verfügbar sind, wird unterstützt.

Für einen nicht interaktiven, mehrzeiligen Textblock legen Sie für die Eigenschaft `editable` des TextArea-Steuerelements `false` fest. (Die Laufzeitumgebung berücksichtigt nicht die Eigenschaft `selectable`.) Außerdem können Sie den Rahmen entfernen, indem Sie die Eigenschaft `borderVisible` auf `false` setzen. Sie können die Hintergrundfarbe ändern, indem Sie die Eigenschaften `contentBackgroundColor` und `contentBackgroundAlpha` festlegen.

Im folgenden Beispiel wird ein nicht interaktiver Textblock erstellt, der sich dem Hintergrund der Anwendung anpasst:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/BlockOfText.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Block of Text">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <s:HGroup>
        <s:Image source="@Embed(source='../../assets/myImage.jpg')" width="30%"/>
        <!-- Create a multi-line block of text. -->
        <s:TextArea width="65%"
                editable="false"
                borderVisible="false"
                contentBackgroundColor="0xFFFFFF"
                contentBackgroundAlpha="0"
                height="400"
               text="Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat."/>
    </s:HGroup>

</s:View>
```

Da das Steuerelement TextArea kein TLF unterstützt, können Sie nicht die Eigenschaften `textFlow`, `content` oder `selectionHighlighting` verwenden. Zudem können folgende Methoden nicht verwendet werden:

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

TextInput-Steuerelement in Mobilanwendungen verwenden

Das TextInput-Spark-Steuerelement ermöglicht die Eingabe und Bearbeitung einer einzigen Textzeile. Diese Klasse ist für Mobilanwendungen konzipiert.

Standardmäßig verwendet das TextInput-Steuerelement die virtuelle Tastatur, die Verknüpfungen zu nativen Methoden des zugrunde liegenden Betriebssystems bereitstellt. Demzufolge werden Funktionen wie Autokorrektur, automatische Vervollständigung und Anpassen der virtuellen Tastatur unterstützt.

Das folgende Beispiel zeigt TextInput-Steuerelemente mit Aufforderungstext und benutzerdefinierten Fokusringen in einer Mobilanwendung:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleTextInput.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Simple TextInput">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout
            paddingTop="20"
            paddingLeft="20"
            paddingRight="20"/>
    </s:layout>

    <s:TextInput
        prompt="Enter text here"
        focusColor="green"
        focusThickness="5"
        focusAlpha=".1"/>
    <s:TextInput
        prompt="Enter text here, too"
        focusColor="red"
        focusThickness="5"
        focusAlpha=".1"/>
</s:View>
```

In Mobilanwendungen verwendet das TextInput-Steuerelement standardmäßig die StageTextInputSkin-Klasse als Skin. Diese Skin verwendet zur Textdarstellung die StyleableStageText-Klasse anstelle der RichEditableText-Klasse. Demzufolge wird TLF vom TextInput-Steuerelement nicht unterstützt. Nur eine Untergruppe von Stilen, die im TextInput-Steuerelement mit der Skin, die keine Mobilskin ist, verfügbar sind, wird unterstützt.

RichText- und RichEditableText-Steuerelemente in Mobilanwendungen verwenden

In Mobilanwendungen sollten die Steuerelemente RichText und RichEditableText vermieden werden. Diese Steuerelemente bieten keine Mobilskins und sind auch nicht für Mobilanwendungen vorgesehen. Wenn Sie diese Elemente dennoch verwenden, arbeiten sie mit dem rechenintensiven TLF.

MX-Textsteuerelemente

MX-Textsteuerelemente wie MX Text und MX Label können nicht in Mobilanwendungen verwendet werden. Verwenden Sie stattdessen die entsprechenden Spark-Steuerelemente.

Stile in Texteingabesteuerelemente in Mobilanwendungen festlegen

Die TextInput- und TextArea-Steuerelemente unterstützen lediglich eine Untergruppen von Stilen im Mobildesign. Die StyleableStageText-Klasse definiert diese Stile.

Im Folgenden werden die von TextInput und TextArea in Mobilanwendungen unterstützten Stile aufgeführt:

- `color`
- `contentBackgroundAlpha`
- `contentBackgroundColor`

- Fokusringstile: `focusAlpha`, `focusBlendMode`, `focusColor` und `focusThickness`
- `fontFamily`
- `fontStyle`
- `fontSize`
- `fontWeight`
- `locale`
- Auffüllstile: `paddingBottom`, `paddingLeft`, `paddingRight` und `paddingTop`
- `showPromptWhenFocused`
- `textAlign`

Das Label-Steuerelement unterstützt in einer Mobilanwendung neben diesen Stilen auch den `textDecoration`-Stil.

fontFamily-Stil verwenden

In Standardmobilskins unterstützt die Eigenschaft `fontFamily` keine kommagetrennte Schriftartenliste. Stattdessen können Sie lediglich eine Schriftart festlegen und die Laufzeitumgebung versucht dann, diese Schriftart einer Schriftart auf dem Gerät zuzuordnen.

Wenn Sie beispielsweise „Arial“ festlegen, stellt das Gerät den Text in Arial dar, sofern die Schriftart verfügbar ist. Ist die Schriftart nicht verfügbar, bestimmt die Laufzeitumgebung die Schriftart, die ihr am nächsten kommt. Wenn Sie einen Schriftartennamen angeben, den die Laufzeitumgebung nicht erkennt, verwendet das Gerät die Standardschriftart zur Textdarstellung. Die Standardschriftart auf einem Mobilgerät ist für gewöhnlich eine Sans Serif-Schrift.

Sie können `_sans`, `_serif`, bzw. `_typewriter` festlegen, um eine Sans Serif-, Serif- bzw. Code-Schriftart auf einem Mobilgerät zu erhalten.

Das folgende Beispiel zeigt, wie der Text basierend auf dem Wert für den `fontFamily`-Stil dargestellt wird:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/FontFamilyExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="The fontFamily style">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <s:TextInput prompt="This is _sans" fontSize="14" fontFamily="_sans"/>
    <s:TextInput prompt="This is _serif" fontSize="14" fontFamily="_serif"/>
    <s:TextInput prompt="This is _typewriter" fontSize="14" fontFamily="_typewriter"/>
    <s:TextInput prompt="This is Arial" fontSize="14" fontFamily="arial"/>
    <s:TextInput prompt="This is Times" fontSize="14" fontFamily="times"/>
    <s:TextInput prompt="This is Times New Roman" fontSize="14" fontFamily="Times New Roman"/>
    <!-- Try a gibberish font name to see what the device's default font is: -->
    <s:TextInput prompt="This is bugblatter" fontSize="14" fontFamily="bugblatter"/>
</s:View>
```

Benutzerdefinierte Mobilskin in einem Texteingabesteuerelement erstellen

Mithilfe von MXML und ActionScript können Sie das Aussehen und das Verhalten von Texteingabesteuerelementen in einer Mobilanwendung teilweise steuern. Beispielsweise können Sie in den TextArea- und TextInput-Steuerelementen die Rahmenfarbe ändern oder das Aussehen der Rahmen wechseln. In einigen Fällen müssen Sie eine benutzerdefinierte Skin erstellen, um das Aussehen bestimmter Teile der Textsteuerelemente zu ändern.

Die StageTextAreaSkin- und StageTextInputSkin-Klassen definieren die TextInput- und TextArea-Standardskins im Mobildesign. Das Layout und die Chromlogik dieser Skins stammen vor allem von der StageTextSkinBase-Klasse.

Erstellen Sie zum Erstellen einer benutzerdefinierten Skin eine benutzerdefinierte StageTextSkinBase-Klasse, die das neue Aussehen definiert. Erstellen Sie anschließend eine benutzerdefinierte StageTextAreaSkin- oder StageTextInputSkin-Klasse, die diese benutzerdefinierte Klasse erweitert.

Weitere Informationen zum Erstellen benutzerdefinierter Skins für das Mobildesign finden Sie unter „Grundlagen des Mobilskinnings“ auf Seite 173.

Tasten in einem Texteingabesteuerelement einschränken

Wenn Sie StageText-basierte Skins für Texteingabesteuerelemente verwenden, können Sie die zulässigen Zeichen mithilfe der Eigenschaft `restrict` des TextInput- oder TextArea-Steuerelements einschränken.

Der Standardwert der Eigenschaft `restrict` lautet `null`; das heißt, dass der Benutzer standardmäßig jedes Zeichen eingeben kann.

Die Eigenschaft `restrict` akzeptiert eine Folge zulässiger Zeichen. Verwenden Sie für Bereiche einen Bindestrich (-). Trennen Sie Bereiche nicht durch Leerzeichen, Komma oder ein anderes Zeichen, es sei denn, Sie möchten dieses Zeichen in der Definition mit einschließen. Im Folgenden finden Sie mehrere Beispiele für die Verwendung der Einschränkungssyntax.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/RestrictStrings.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Examples of restrict">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="20" paddingLeft="20" paddingRight="20"/>
    </s:layout>

    <s:TextInput prompt="Alpha-numeric only" restrict="a-zA-Z0-9"/>
    <s:TextInput prompt="Numbers only" restrict="0-9"/>
    <s:TextInput prompt="All chars, only uppercase alpha" restrict="^a-z"/>
    <!-- ASCII chars 32 (space) through 126 (tilde) only: -->
    <s:TextInput prompt="" restrict="\u0020-\u007E"/>
    <s:TextInput prompt="All chars but not the caret or hyphen" restrict="^\^\-"/>
</s:View>
```

Der Zeichenfolgenwert der Eigenschaft `restrict` wird von links nach rechts gelesen; alle Zeichen hinter einem Caret (^) sind nicht zulässig. Beispiel:

```
ta1.restrict = "A-Z^Q"; // All uppercase alpha characters, but exclude Q
```

Wenn das erste Zeichen in einer Zeichenfolge ein Caret (^) ist, sind alle Zeichen bis auf die hinter dem Caret zulässig. Beispiel:

```
ta1.restrict = "^a-z"; // All characters, but exclude lowercase alpha
```

Mit dem umgekehrten Schrägstrich können Sie Sonderzeichen ausschließen. So schließen Sie beispielsweise das Caret aus:

```
ta1.restrict = "^\^"; // All characters, but exclude the caret
```

Mit „\u“ geben Sie ASCII-Zeichencode ein. Beispiel:

```
ta1.restrict = "\u0020-\u007E"; // ASCII chars 32 through 126 only
```

Benutzerinteraktionen bei Text in Mobilanwendungen

Mit Textsteuerelementen können Sie Fingerbewegungen interpretieren. Das folgende Beispiel wartet auf eine Fingerbewegung und gibt anschließend die Richtung aus, in die der Finger bewegt wurde:

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

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

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

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

</s:View>
```

Durch Berühren und Ziehen wird stets Text ausgewählt. Dies funktioniert jedoch nur mit auswählbaren oder bearbeitbaren Textsteuerelementen. In einigen Fällen möchten Sie vielleicht nicht, dass der Benutzer per Berühren und Ziehen Text auswählen kann. Zu diesem Zweck können Sie die Eigenschaften `selectable` und `editable` auf `false` setzen oder die Auswahl durch einen Aufruf der Methode `selectRange(0,0)` im Swipe Event Handler rückgängig machen.

Wenn der Text sich innerhalb einer Bildlaufleiste befindet, so scrollt die Leiste nur dann, wenn die Bewegung außerhalb der Textkomponente stattfindet.

Virtuelle Tastatur in Mobilanwendungen verwenden

Zahlreiche Geräte besitzen keine Hardwaretastatur. Stattdessen wird bei diesen Geräten eine Tastatur verwendet, die bei Bedarf auf dem Bildschirm geöffnet wird. Die virtuelle Tastatur, auch als Bildschirmtastatur bezeichnet, wird nach der Informationseingabe oder bei Abbruch des Vorgangs durch den Benutzer geschlossen.

Die folgende Abbildung zeigt eine Anwendung, in der die virtuelle Tastatur verwendet wird:

Die virtuelle Tastatur hat je nach Komponente, von der sie eingeblendet wird, einen anderen Funktionssatz:

- Native Funktionen: Die Tastatur, die mit den Standardsteuerelementen TextArea und TextInput verwendet wird, stellt eine Verknüpfung mit der nativen Schnittstelle für Funktionen wie Autokorrektur, automatische Vervollständigung und benutzerdefinierte Tastaturlayouts her. Unterstützung für den gesamten Funktionssatz ist in die StageText-basierten Standardskinklassen der Texteingabesteuerelemente integriert. Nicht alle Geräte unterstützen sämtliche native Funktionen.
- Eingeschränkt: Die Tastatur, die mit allen Steuerelementen außer TextArea und TextInput verwendet wird oder mit TextArea und TextInput, wenn für diese TextField-basierte Skins verwendet werden. Der eingeschränkte Funktionssatz unterstützt keine nativen Betriebssystemfunktionen wie Autokorrektur, automatische Vervollständigung und benutzerdefinierte Tastaturlayouts.

Da die Tastatur einen Teil des Bildschirms einnimmt, muss Flex sicherstellen, dass eine Anwendung auch im reduzierten Bildschirmbereich funktioniert. Angenommen, der Benutzer wählt ein TextInput-Steuerelement, durch das die virtuelle Tastatur geöffnet wird. Nach dem Öffnen der Tastatur passt Flex die Größe der Anwendung automatisch an den verfügbaren Bildschirmbereich an. Flex kann dann das ausgewählte TextInput-Steuerelement so positionieren, dass es über der Tastatur angezeigt wird.

Blogger Peter Elst schrieb einen Artikel über die Steuerung der Bildschirmtastatur in Flex-Mobilanwendungen.

Virtuelle Tastatur in Flex-Mobilanwendungen öffnen

Es gibt drei Möglichkeiten, eine virtuelle Tastatur in einer Mobilanwendung zu öffnen:

- Setzen Sie den Fokus auf ein Texteingabesteuerelement wie TextInput oder TextArea.
- Legen Sie für die Eigenschaft `needsSoftKeyboard` des Steuerelements den Wert `true` fest und setzen Sie den Fokus auf dieses Steuerelement.
- Rufen Sie die `requestSoftKeyboard()`-Methode in einem Steuerelement auf (nicht unter iOS).

Die Tastatur bleibt geöffnet, bis eine der folgenden Aktionen ausgeführt wird:

- Der Benutzer verschiebt den Fokus auf ein Steuerelement, das keine Texteingabe empfängt. Dies geschieht, wenn der Benutzer manuell auf ein anderes Texteingabesteuerelement zeigt oder auf der Tastatur die Eingabetaste drückt und die Anwendung den Fokus auf ein anderes Steuerelement verschiebt.

  Wenn der Fokus auf ein anderes Texteingabesteuerelement oder auf ein Steuerelement mit auf `true` festgelegter `needsSoftKeyboard`-Eigenschaft wechselt, bleibt die Tastatur geöffnet.
- Der Benutzer bricht die Eingabe durch Drücken der Zurück-Taste des Geräts ab.
- Sie verschieben programmgesteuert den Fokus auf ein nicht interaktives Steuerelement oder setzen `stage.focus` auf `null`.

Benutzer ein Texteingabesteuerelement bereitstellen

Wenn Sie einem Benutzer ein Texteingabesteuerelement bereitstellen, wird eine virtuelle Tastatur angezeigt, sobald der Benutzer den Fokus auf dieses verschiebt, es sei denn, für die Eigenschaft `editable` ist `false` festgelegt.

Standardmäßig verwenden die TextInput- und TextArea-Steuerelemente für die Textdarstellung die StageText-Klasse. Damit unterstützt die Tastatur, die für diese Steuerelemente angezeigt wird, native Funktionen wie Autokorrektur, automatische Großschreibung und Tastaturtypen. Nicht alle Funktionen werden auf allen Geräten unterstützt.

Ändern Sie die Skinklassen so, dass diese für die Texteingabesteuerelemente die TextField-basierten Skins verwenden, sind die Funktionen begrenzt. Die Tastatur ist dieselbe; sie unterstützt jedoch keine nativen Funktionen.

Eigenschaft `needsSoftKeyboard` festlegen

Sie können Steuerelemente, die keinen Text eingeben, so konfigurieren, dass die virtuelle Tastatur geöffnet wird, z. B. das Button-Steuerelement oder das ButtonBar-Steuerelement. Soll die Tastatur geöffnet werden, wenn ein anderes Steuerelement als ein Texteingabesteuerelement den Fokus erhält, legen Sie für die Eigenschaft `needsSoftKeyboard` des betreffenden Steuerelements `true` fest. Alle Flex-Komponenten erben diese Eigenschaft von der InteractiveObject-Klasse.

Die Tastatur, die für andere Steuerelemente als TextInput und TextArea geöffnet wird, unterstützt keine nativen Funktionen wie automatische Großschreibung, Autokorrektur und anpassbare Tastaturtypen.

Hinweis: *Die Texteingabesteuerelemente öffnen immer die Tastatur, wenn sie den Fokus erhalten. Sie ignorieren die* `needsSoftKeyboard`*-Eigenschaft, sodass deren Festlegung keine Auswirkungen auf die Texteingabesteuerelemente besitzt.*

`requestSoftKeyboard()`-Methode aufrufen

Um eine virtuelle Tastatur programmgesteuert einzublenden, können Sie die `requestSoftKeyboard()`-Methode aufrufen. Für das Objekt, das diese Methode aufruft, muss außerdem die Eigenschaft `needsSoftKeyboard` auf `true` gesetzt sein. Bei dieser Methode wird der Fokus auf das Objekt, das die Methode aufgerufen hat, verschoben sowie eine virtuelle Tastatur eingeblendet, wenn das Gerät über keine Hardwaretastatur verfügt.

Die InteractiveObject-Klasse definiert die `requestSoftKeyboard()`-Methode. Damit können Sie diese Methode für jede Komponente aufrufen, die eine Unterklasse von InteractiveObject ist.

Wenn Sie die `requestSoftKeyboard()`-Methode im TextArea- oder TextInput-Steuerelement aufrufen, werden die nativen Tastaturfunktionen wie Autokorrektur und automatische Großschreibung unterstützt (sofern das Gerät diese unterstützt).

Die `requestSoftKeyboard()`-Methode funktioniert nicht auf iOS-Geräten.

Native Funktionen mit einer virtuellen Tastatur verwenden

Die StageTextInputSkin- und StageTextAreaSkin-Klassen definieren die Skins für die TextInput- und TextArea-Steuerelemente in einer Mobilanwendung. Die Skins verwenden die StageText-Klasse für die Textdarstellung und stellen eine Verknüpfung zu den nativen Funktionen der virtuellen Tastatur her. Zu diesen Funktionen zählen die folgenden:

- Autokorrektur
- Automatische Großschreibung
- Benutzerdefinierte Eingabetastenbeschriftungen
- Benutzerdefinierte Tastaturtypen

Texteingabesteuerelemente können auch die TextField-basierten Skins für die Textdarstellung verwenden. Diese Skins unterstützen nicht die nativen Funktionen. Sie stellen jedoch zusätzliche Funktionen für das zugrunde liegende Textsteuerelement bereit, z. B. Unterstützung für den Bildlauf in Formularen, eingebettete Schriftarten, Zugriff auf die `keyUp`- und `keyDown`-Ereignisse, Beschneidung, Textabmessung und Alpha-Bruchwerte.

Um die TextField-basierten Skins zu verwenden, legen Sie die Eigenschaft `skinClass` des Texteingabesteuerelements so fest, dass es auf die TextInputSkin- und TextAreaSkin-Klassen verweist. Beispiel:

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin"/>
<s:TextArea skinClass="spark.skins.mobile.TextAreaSkin"/>
```

Autokorrektur für virtuelle Tastaturen in Flex-Mobilanwendungen verwenden

Bei der Autokorrektur werden Rechtschreibfehler korrigiert und die Eingabe eines Benutzers mithilfe der Eingabevorhersage ergänzt. Je nach Gerät wird dieses Verhalten als Sprechblase über dem Test, Erweiterung der virtuellen Tastatur oder anderweitig implementiert.

Sie können die Autokorrektur für virtuelle Tastaturen in einer Mobilanwendung verwenden, indem Sie die Eigenschaft `autoCorrect` des Texteingabesteuerelements auf `true` setzen. Dies ist der Standardwert.

Im folgenden Beispiel kann die Autokorrektur aktiviert und deaktiviert werden:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/AutoCorrectionExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Auto-Correction">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextInput prompt="Enter your text" autoCorrect="{myCB.selected}"/>
    <s:CheckBox id="myCB" label="Enable auto-correct" enabled="true"/>

</s:View>
```

Nicht alle Geräte unterstützen Autokorrektur. Wenn Sie die Eigenschaft `autoCorrect` auf einem Gerät aktivieren oder deaktivieren, das diese Funktion nicht unterstützt, wird der Wert in der Laufzeitumgebung ignoriert und das Standardverhalten des Geräts verwendet.

Automatische Großschreibung für virtuelle Tastaturen in Flex-Mobilanwendungen verwenden

Die automatische Großschreibung ist eine Einstellung, bei der das Texteingabesteuerelement bestimmte Wörter oder Buchstaben bei der Eingabe von Text durch den Benutzer großschreibt. Beispielsweise können Sie einstellen, dass alle Buchstaben großgeschrieben werden oder dass lediglich das erste Wort eines Satzes automatisch großgeschrieben wird. So muss sich der Benutzer beim Eingeben von Text in eine Mobilanwendung keine Gedanken über die Großschreibung machen.

Sie verwenden die automatische Großschreibung in virtuellen Tastaturen, indem Sie im Texteingabesteuerelement den Wert der Eigenschaft `autoCapitalize` festlegen. Mögliche Werte sind `none`, `word`, `sentence` und `all`. Die AutoCapitalize-Klasse definiert die möglichen Werte. Der Standardwert ist `none`.

Im folgenden Beispiel können verschiedene Werte für die automatische Großschreibung ausgewählt werden:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/AutoCapitalizeExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Auto-Capitalization">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a capitalization setting:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="capTypeList" width="300" labelField="name" fontSize="12">
            <s:ArrayCollection>
                <fx:Object name="All" value="all"/>
                <fx:Object name="None" value="none"/>
                <fx:Object name="Sentence" value="sentence"/>
                <fx:Object name="Word" value="word"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput autoCapitalize="{capTypeList.selectedItem.value}"/>

</s:View>
```

Nicht alle Geräte unterstützen die automatische Großschreibung. Wenn Sie den Wert der Eigenschaft `autoCapitalize` auf einem Gerät festlegen, das diese Funktion nicht unterstützt, wird der Wert in der Laufzeitumgebung ignoriert und das Standardverhalten des Geräts verwendet.

Typen virtueller Tastaturen in Flex-Mobilanwendungen ändern

Die SoftKeyboardType-Klasse definiert die Typen virtueller Tastaturen für Mobilanwendungen. Der Tastaturtyp wird mithilfe der Eigenschaft `softKeyboardType` im Texteingabesteuerelement ausgewählt.

Die Unterschiede zwischen den meisten Tastaturen wie `email` und `contact` sind nur gering. Beispielsweise stellt die `email`-Tastatur dem Benutzer dieselben Tasten wie die `contact`-Tastatur bereit, es ersetzt lediglich das Mikrofon durch das Symbol „@“. Die `url`-Tastatur verwendet das Symbol „/“. Eine Ausnahme hierzu stellt die `number`-Tastatur dar. Diese Tastatur sieht wie ein Rechner aus und besitzt vor allem Zahlen und Operatoren.

Im folgenden Beispiel sind die verschiedenen Typen virtueller Tastaturen aufgeführt, die verfügbar sind:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/KeyboardTypes.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Keyboard Types">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a keyboard type:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="keyboardTypeList" width="300" labelField="name">
            <s:ArrayCollection>
                <fx:Object name="Contact" value="contact"/>
                <fx:Object name="Default" value="default"/>
                <fx:Object name="Email" value="email"/>
                <fx:Object name="Number" value="number"/>
                <fx:Object name="Punctuation" value="punctuation"/>
                <fx:Object name="URL" value="url"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput softKeyboardType="{keyboardTypeList.selectedItem.value}" text=""/>

</s:View>
```

Nicht alle Typen virtueller Tastaturen werden auf allen Geräten unterstützt. Wenn Sie einen Typ festlegen, der nicht unterstützt wird, wird in der Laufzeitumgebung der Wert ignoriert und das Standardverhalten des Geräts verwendet.

Eingabetastenbeschriftungen auf einer virtuellen Tastatur in Flex-Mobilanwendungen ändern

Wenn die virtuelle Tastatur eingeblendet wird und der Benutzer Text eingibt, muss der Benutzer angeben können, dass er mit der Texteingabe fertig ist und mit dem nächsten Feld fortfahren oder die eingegebenen Daten senden möchte. Auf einer virtuellen Tastatur wird hierzu für gewöhnlich die Eingabetaste verwendet. Die Taste fügt der Texteingabe kein Zeichen hinzu, sondern signalisiert dem Texteingabesteuerelement, dass der Benutzer mit der Texteingabe fertig ist.

Die ReturnKeyLabel-Klasse definiert verschiedene Beschriftungen für die Eingabetaste. Die möglichen Werte sind `default`, `done`, `go`, `next` und `search`. Die Eingabetastenbeschriftung wird mithilfe der Eigenschaft `returnKeyLabel` im Texteingabesteuerelement festgelegt.

Im folgenden Beispiel können verschiedene Beschriftungen für die Eingabetaste ausgewählt werden:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/ReturnKeyLabels.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Return Key Labels">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a return key label:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="returnKeyLabelList" width="300" labelField="name">
            <s:ArrayCollection>
                <fx:Object name="Default" value="default"/>
                <fx:Object name="Done" value="done"/>
                <fx:Object name="Go" value="go"/>
                <fx:Object name="Next" value="next"/>
                <fx:Object name="Search" value="search"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput returnKeyLabel="{returnKeyLabelList.selectedItem.value}" text=""/>

</s:View>
```

Die Ereignisse oder Interaktionen unterscheiden sich zwischen den verschiedenen Eingabetastentypen nicht. Durch Ändern der Eigenschaft `returnKeyLabel` wird lediglich die Beschriftung der Taste geändert.

Nicht auf allen Geräten kann die Beschriftung der Eingabetaste eingestellt werden. Wenn Sie den Wert der Eigenschaft `returnKeyLabel` auf einem Gerät festlegen, das diese Funktion nicht unterstützt, wird der Wert in der Laufzeitumgebung ignoriert und das Standardverhalten des Geräts verwendet.

Ereignisse mit einer virtuellen Tastatur in Mobilanwendungen verwenden

Die Interaktion mit einer virtuellen Tastatur auf einem Mobilgerät unterscheidet sich von der Interaktion mit der Tastatur in einer Desktop- oder webbasierten Anwendung. In der folgenden Tabelle werden die Ereignisse aufgeführt, die mit virtuellen Tastaturen ausgelöst werden können:

Ereignis	**Auslösungszeitpunkt**
`enter`	Wenn der Benutzer die Eingabetaste drückt.
`keyDown` und `keyUp`	Bei StageText-basierten Skins: wenn nur einige Tasten gedrückt und losgelassen werden. Bei TextField-basierten Skins: wenn alle Tasten gedrückt und losgelassen werden. Diese Ereignisse werden nicht für alle Tasten auf allen Geräten ausgelöst. Verwenden Sie diese Methoden nicht zum Erfassen von Tasteneingaben mit virtuellen Tastaturen, es sei denn, Sie verwenden die TestField-basierten Skins für Steuerelemente, die die Tastatur einblenden.
`softKeyboardActivating`	Direkt vor dem Öffnen der Tastatur
`softKeyboardActivate`	Direkt nach dem Öffnen der Tastatur

Ereignis	Auslösungszeitpunkt
`softKeyboardDeactivate`	Nach dem Schließen der Tastatur

Um zu bestimmen, wann ein Benutzer die virtuelle Tastatur nicht mehr benötigt, können Sie das `FlexEvent.ENTER`-Ereignis im Texteingabesteuerelement abhören. Das Steuerelement löst dieses Ereignis aus, sobald die Eingabetaste gedrückt wird. Indem Sie das `enter`-Ereignis abhören, können Sie eine Überprüfung durchführen, den Fokus ändern oder andere Vorgänge für den soeben eingegebenen Text ausführen.

In einigen Fällen wird das `enter`-Ereignis nicht ausgelöst. Dieser Fall tritt auf Android-Geräten auf, wenn Sie eine virtuelle Tastatur für das letzte Texteingabesteuerelement in der Ansicht verwenden. Um dieses Problem zu umgehen, legen Sie für das letzte Texteingabesteuerelement in der Ansicht die Eigenschaft `returnKeyLabel` auf `go`, `next` oder `search` fest.

Im folgenden Beispiel wird der Fokus von einem Feld auf das nächste verschoben, wenn der Benutzer auf der virtuellen Tastatur die Taste „Next“ drückt:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/UseNextLikeTab.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Change Focus">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private function changeField(ti:TextInput):void {
                // Before changing focus to a new control, set the stage's focus to null:
                stage.focus = null;

                // Set focus on the TextInput that was passed in:
                ti.setFocus();
            }
        ]]>
    </fx:Script>

    <s:HGroup>
        <s:Label text="1:" paddingTop="15"/>
        <s:TextInput id="ti1" prompt="First Name"
```

```
                    width="80%"
                    returnKeyLabel="next"
                    enter="changeField(ti2)"/>
    </s:HGroup>
    <s:HGroup>
        <s:Label text="2:" paddingTop="15"/>
        <s:TextInput id="ti2" prompt="Middle Initial"
                    width="80%"
                    returnKeyLabel="next"
                    enter="changeField(ti3)"/>
    </s:HGroup>
    <s:HGroup>
        <s:Label text="3:" paddingTop="15"/>
        <s:TextInput id="ti3" prompt="Last Name"
                    width="80%"
                    returnKeyLabel="next"
                    enter="changeField(ti1)"/>
    </s:HGroup>

</s:View>
```

Interagiert der Benutzer mit der standardmäßigen virtuellen Tastatur für die TextInput- und TextArea-Steuerelemente, werden die `keyUp`- und `keyDown`-Ereignisse nur für eine kleine Untergruppe der Tasten ausgelöst. Verwenden Sie zum Erfassen einzelner Tastendrücke für alle Tasten das `change`-Ereignis. Das `change`-Ereignis wird jedes Mal ausgelöst, wenn sich der Inhalt des Texteingabesteuerelements ändert. Der Nachteil hier ist, dass Sie keinen Zugriff auf die Eigenschaften der gedrückten Taste haben und Sie Ihre eigene Tastendrucklogik schreiben müssen.

Im folgenden Beispiel wird der Zeichencode der zuletzt gedrückten Taste angezeigt:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/CompareMobileKeyPresses.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Keyboard Events">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private var storedValueOfText:String = null;

            // Compare the new text against the stored value to see what key was pressed.
            private function compareKey(e:Event):void {
                var key:String = "";
                if (storedValueOfText == null) {
           key = ti1.text.charCodeAt(0).toString(); // Capture the first key pressed.
```

```
                } else {
                    // Compare the stored value against the current value and extract the
difference.
                    for (var i:int = 0; i<ti1.text.length; i++) {
                        if (ti1.text.charAt(i) == storedValueOfText.charAt(i)) {
                            // Do nothing if they're equal.
                        } else {
                            key = ti1.text.charCodeAt(i).toString();
                        }
                    }
                }
                ti2.text = "The '" + key + "' key was pressed.";
                storedValueOfText = ti1.text;
            }
        ]]>
    </fx:Script>

    <s:TextInput id="ti1" change="compareKey(event)"/>
    <s:TextInput id="ti2" editable="false"/>
</s:View>
```

In diesem Beispiel wird die zuletzt gedrückte Taste nur erkannt, wenn sich der Cursor am Ende des Texteingabefelds befindet.

Interagiert der Benutzer mit der virtuellen Tastatur für TextField-basierte Steuerelementen, funktionieren Ereignisse wie `keyUp` und `keyDown` für alle Tasten. Im folgenden Beispiel wird die `keyUp`-Prozedur verwendet, um die aktuelle Taste abzurufen und den Stil des Label-Steuerelements basierend auf dem Tastencode anzuwenden. Da die `requestSoftKeyboard()`-Methode die Tastatur für das Label-Steuerelement und nicht für das TextInput- oder TextArea-Steuerelement hervorruft, verwendet die Anwendung die TextField-basierte Tastatur.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/RequestSoftKeyboardExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="requestSoftKeyboard()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private function handleButtonClick():void {
                myLabel.requestSoftKeyboard();
            }
            /* Ok to use keyUp handler on limited screen keyboard. */
            private function handleKeys(event:KeyboardEvent):void {
                var c:int;
```

```
                switch(event.keyCode) {
                    case(82): // 82 = "r"
                        c = 0xFF0000;
                        break;
                    case(71): // 71 = "g"
                        c = 0x00FF00;
                        break;
                    case(66): // 66 = "b"
                        c = 0x0000FF;
                        break;
                }
                event.currentTarget.setStyle("color",c);
            }
        ]]>
    </fx:Script>
    <s:Label id="myLabel" text="This is a label." needsSoftKeyboard="true"
keyUp="handleKeys(event)"/>
    <s:Button id="b1" label="Click Me" click="handleButtonClick()"/>

</s:View>
```

Um die virtuelle Tastatur programmgesteuert zu schließen, setzen Sie `stage.focus` auf `null`. Um die virtuelle Tastatur zu schließen und den Fokus auf ein anderes Steuerelement zu verschieben, setzen Sie `stage.focus` auf `null` und verschieben Sie anschließend den Fokus auf das gewünschte Steuerelement. Sie können auch die `requestSoftKeyboard()`-Methode eines anderen Steuerelements aufrufen, um den Fokus zu verschieben und die virtuelle Tastatur in einem anderen Steuerelement zu öffnen.

Auf einige Eigenschaften der virtuellen Tastatur können über Ereignisprozeduren zugegriffen werden. Verwenden Sie zum Zugreifen auf die Größe und die Position der virtuellen Tastatur die Eigenschaft `softKeyboardRect` der flash.display.Stage-Klasse wie im folgenden Beispiel:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/SoftKeyboardEvents.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Soft Keyboard Events">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextInput prompt="Enter your text"
             softKeyboardActivate="ta1.text+=stage.softKeyboardRect + '\n'"
             softKeyboardDeactivate="ta1.text+=stage.softKeyboardRect + '\n'"
             softKeyboardActivating="ta1.text+=stage.softKeyboardRect + '\n'"/>
    <s:TextArea id="ta1" width="100%" height="100%" editable="false"/>

</s:View>
```

Anwendung für die virtuelle Tastatur konfigurieren

Zur Unterstützung der virtuellen Tastatur kann die Anwendung beim Öffnen der Tastatur die folgenden Aktionen ausführen:

- Anpassen der Anwendungsgröße an die verbleibende Bildschirmgröße, sodass die Tastatur die Anwendung nicht verdeckt
- Bildlauf im übergeordneten Container des Texteingabesteuerelements, um sicherzustellen, dass das Steuerelement sichtbar ist

System für virtuelle Tastatur konfigurieren

Die virtuelle Tastatur funktioniert nicht in Anwendungen, die im Vollbildmodus ausgeführt werden. Stellen Sie daher sicher, dass in der Datei „app.xml“ das Attribut `<fullScreen>` auf den Standardwert `false` gesetzt ist.

Stellen Sie sicher, dass der Rendermodus der Anwendung auf den CPU-Modus gestellt ist. Der Rendermodus wird in der Anwendungsbeschreibungsdatei „app.xml“ über das `<renderMode>`-Attribut gesteuert. Stellen Sie sicher, dass das Attribut `<renderMode>` auf den Standardwert `cpu` gesetzt ist – nicht auf `gpu`.

Hinweis: *Das Attribut `<renderMode>` ist nicht standardmäßig in der Datei „app.xml“ enthalten. Um die Einstellung zu ändern, fügen Sie sie im `<initialWindow>`-Attribut hinzu. Wenn das Attribut nicht in der Datei „app.xml“ enthalten ist, hat es den Standardwert `cpu`.*

Bildlauf im übergeordneten Container beim Öffnen der virtuellen Tastatur

Die `resizeForSoftKeyboard`-Eigenschaft des Application-Containers bestimmt das Größenänderungsverhalten der Anwendung. Wenn die `resizeForSoftKeyboard`-Eigenschaft `false` (Standardwert) lautet, kann die Tastatur über der Anwendung angezeigt werden. Ist als Wert `true` festgelegt, wird die Größe der Anwendung entsprechend der Größe der Tastatur angepasst.

Damit Scrolling unterstützt wird, müssen die Texteingabesteuerelemente die TextField-basierten und nicht die StageText-basierten Skins verwenden. Hierfür muss die Eigenschaft `skinClass` des TextInput- bzw. TextArea-Steuerelements auf die TextInputSkin- bzw. TextAreaSkin-Klasse zeigen.

Schließen Sie für Bildläufe den übergeordneten Container aller Texteingabesteuerelemente in einer Scroller-Komponente ein. Wenn eine Komponente, die die Tastatur öffnet, den Fokus erhält, führt der Scroller automatisch einen Bildlauf aus, sodass die Komponente sichtbar wird. Die Komponente kann zudem mehreren, verschachtelten Containern der Scroller-Komponente untergeordnet sein.

Der übergeordnete Container muss eine GroupBase- oder eine SkinnableContainer-Klasse oder eine Unterklasse eines GroupBase-oder SkinnableContainer-Containers sein. Die Komponente, die den Fokus erhält, muss die IVisualElement-Schnittstelle implementieren und den Fokus erhalten können.

Durch Einschließen des übergeordneten Containers in einer Scroller-Komponente sind Bildläufe im Container bei geöffneter Tastatur möglich. Beispielsweise kann ein Container mehrere Texteingabesteuerelemente enthalten. Dann können Sie zur Dateneingabe Bildläufe zu den einzelnen Texteingabesteuerelementen durchführen.

Nach dem Schließen der Tastatur kann der übergeordnete Container kleiner als der verfügbare Platz auf dem Bildschirm sein. Wenn der Container kleiner als der verfügbare Platz auf dem Bildschirm ist, setzt der Scroller die Bildlaufpositionen auf 0, den oberen Rand des Containers zurück.

Das folgende Beispiel veranschaulicht einen View-Container mit mehreren TextInput-Steuerelementen und einer Scroller-Komponente:

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

    <s:Scroller width="100%" top="10" bottom="50">
        <s:VGroup paddingTop="3" paddingLeft="5" paddingRight="5" paddingBottom="3">
            <!-- Use TextField-based skins so that scrolling works. -->
         <s:TextInput prompt="To" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
         <s:TextInput prompt="CC" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextInput prompt="Subject" width="100%"
skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextArea height="400" width="100%" prompt="Compose Mail"
skinClass="spark.skins.mobile.TextAreaSkin"/>
        </s:VGroup>
    </s:Scroller>

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

</s:View>
```

Der VGroup-Container ist der übergeordnete Container der TextInput-Steuerelemente. Der Scroller schließt die VGroup ein, sodass jedes TextInput-Steuerelement über der Tastatur angezeigt wird, wenn es den Fokus erhält.

Weitere Informationen zur Scroller-Komponente finden Sie unter Scrolling Spark containers.

Anwendungsgröße beim Öffnen der virtuellen Tastatur ändern

Wenn die `resizeForSoftKeyboard`-Eigenschaft des Application-Containers `true` lautet, ändert sich die Größe der Anwendung so, dass diese beim Öffnen der Tastatur in den verfügbaren Bildschirmbereich passt. Die Anwendung stellt ihre Größe nach dem Schließen der Tastatur wieder her.

Das folgende Beispiel zeigt die Hauptanwendungsdatei für eine Anwendung, die Größenänderungen für die Anwendung durch Festlegen der `resizeForSoftKeyboard`-Eigenschaft auf `true` unterstützt:

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

</s:ViewNavigatorApplication>
```

Um die Größenänderung zu aktivieren, müssen Sie das `<softKeyboardBehavior>`-Attribut in der Anwendungsbeschreibungsdatei „app.xml“ auf `none` festlegen. Standardmäßig ist das Attribut `<softKeyboardBehavior>` auf `none` gesetzt. Durch diesen Standardwert wird AIR so konfiguriert, dass die gesamte Bühne verschoben und die Textkomponente mit Fokus sichtbar wird.

In der Regel funktionieren automatische Verhaltensweisen mit Bildschirmtastaturereignissen. Vermeiden Sie aber das Platzieren wichtiger Benutzeroberflächenelemente an Stellen, an denen sie von der Bildschirmtastatur verdeckt werden, im Fall, dass ein Bildschirmtastaturereignis fehlschlägt. Vermeiden Sie es beispielsweise, die Schaltflächen „OK“, „Anmelden“, „Senden“ in der unteren Hälfte einer Ansicht zu platzieren.

Wenn eine StageText-basierte Steuerung animiert wird oder an einer Animation beteiligt ist, ersetzt die Laufzeit sie vorübergehend durch eine Bitmap, sodass der Text sich synchron mit anderen Elementen verschiebt. Wenn die Steuerung Fokus hat, verliert die Steuerung vorübergehend Fokus. In einige Fällen verdeckt die Bildschirmtastatur ein `softKeyboardDeactivate`-Ereignis oder löst es aus, ohne die Bildschirmtastatur auszublenden.

Das kann besonders problematisch sein, wenn programmgesteuert Fokus auf StageText-basierte Komponenten in Popups, die durch die Bildschirmtastatur verursachte Größenänderungen animieren, eingestellt wird. Um dies zu vermeiden, verwenden Sie StageText-basierte Komponenten in Popups, deren Größe sich aufgrund der Bildschirmtastatur nicht ändert.

Wenn Sie StageText-basierte Komponenten in Popups, deren Größe sich ändert, verwenden müssen, zeigen Sie zuerst die Bildschirmtastatur an und warten Sie, bis die Popup-Animation abgeschlossen ist, bevor Sie programmgesteuert Fokus auf die StageText-basierte Komponente einstellen. Alternativ dazu können Sie die programmgesteuerte Einstellung von Fokus in Popups vermeiden.

Popup für die Verwendung mit der virtuellen Tastatur konfigurieren

Ein Callout-Container wird als Popup über einer Mobilanwendung angezeigt. Der Callout-Container kann eine oder mehrere Komponenten enthalten, die Tastatureingaben annehmen. Weitere Informationen zum Callout-Container finden Sie unter „Callout-Container zum Erstellen eines Callouts verwenden“ auf Seite 85.

Verwenden Sie die Eigenschaften des SkinnablePopUpContainer-Containers, der übergeordneten Klasse von Callout, um die Interaktion zwischen Popup und einer Tastatur zu konfigurieren:

moveForSoftKeyboard Wenn `true` (Standardwert), wird das Popup beim Öffnen der Tastatur über die Tastatur verschoben.

resizeForSoftKeyboard Wenn `true` (Standardwert), wird die Größe des Popup beim Öffnen der Tastatur an die verfügbare Fläche über der Tastatur angepasst.

Wenn die `moveForSoftKeyboard`- oder `resizeForSoftKeyboard`-Eigenschaft eines SkinnablePopUpContainers auf `true` gesetzt sind, verschiebt sich der Container möglicherweise jedes Mal bzw. ändert seine Größe, wenn sich die Sichtbarkeit der Bildschirmtastatur ändert. Diese beiden automatischen Verhaltensweisen können bewirken, dass andere Komponenten ihre Größe ändern oder sich verschieben.

Der einfachste Weg zum Vermeiden dieses Szenarios ist es, `resizeForSoftKeyboard` nicht auf `true` zu setzen. Wenn die Größe der Anwendung nicht für die Bildschirmtastatur geändert wird, kann diese nicht bewirken, dass Komponenten sich unter dem Finger des Benutzers verschieben. Dadurch kann jedoch die Bildschirmtastatur einige Anwendungen der Benutzeroberfläche verdecken.

Wenn Ihre Anwendung automatische Größenänderung erfordert, platzieren Sie die interaktiven Komponenten so, dass durch Änderungen an der Sichtbarkeit der Bildschirmtastatur die Komponenten nicht unter dem Finger des Benutzers verschoben werden, wenn sie die Zielkomponente sind. Dazu haben Sie folgende Möglichkeiten:

- Stellen Sie für Schaltflächen, Kontrollkästchen oder für andere kleine Ziele keine Beschränkungen für den unteren Rand ein und platzieren Sie diese nicht unterhalb anderer Komponenten mit prozentualer Höhe.
- Entwerfen Sie das Layout so, dass der obere Rand der untersten Komponente feststehend bleibt, wenn die Tastatur ein- oder ausgeblendet wird.

- Stellen Sie auf Plattformen, die über keine integrierte Affordance (Angebotscharakter) zum Ausblenden der Bildschirmtastatur verfügen, ein feststehendes Benutzeroberflächenelement bereit. Das kann eine Schaltfläche sein, die speziell zum Ausblenden der Bildschirmtastatur vorgesehen ist oder nur ein feststehender leerer Rand, der breit genug ist, um darauf zu tippen, damit der Fokus von einer Textkomponente entfernt werden kann.
- Ordnen Sie Komponenten, die möglicherweise verschoben werden, nicht vertikal an. Sollten Sie diese Komponenten vertikal anordnen, treffen Gesten beim Ändern der Sichtbarkeit einer Bildschirmtastatur unter Umständen das falsche Ziel.

Schriftarten in eine Mobilanwendung einbetten

Sie können mit einigen Einschränkungen Schriftarten für die Verwendung in Mobilanwendungen einbetten.

Da das Label-Steuerelement mit FTE (und folglich auch mit CFF-Schriftarten) arbeitet, verwenden Sie zum Einbetten von Schriftarten in Mobilanwendungen stattdessen das TextArea- oder TextInput-Steuerelement mit TextField-basierten Skins. Sie können keine Schriftarten mit StageText-basierten Skins einbetten. Im Allgemeinen sollten Sie die Verwendung von FTE in Mobilanwendungen vermeiden.

Setzen Sie in der CSS `embedAsCFF` auf `false` und wenden Sie die TextField-basierte Skin an. Beispiel:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/Main.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                      xmlns:s="library://ns.adobe.com/flex/spark"
                      firstView="views.EmbeddingFontsView">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        @font-face {
            src: url("../assets/MyriadWebPro.ttf");
            fontFamily: myFontFamily;
            embedAsCFF: false;
        }
        .customStyle {
            fontFamily: myFontFamily;
            fontSize: 24;
            skinClass: ClassReference("spark.skins.mobile.TextAreaSkin");
        }
    </fx:Style>
</s:ViewNavigatorApplication>
```

Das Steuerelement TextArea in der Ansicht „EmbeddingFontView“ übernimmt die Typauswahl:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/EmbeddingFontsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Embedded Fonts">
    <s:layout>
        <s:VerticalLayout paddingTop="20" paddingLeft="20" paddingRight="20"/>
    </s:layout>
    <s:TextArea id="ta1"
                width="100%"
                styleName="customStyle"
                text="This is a TextArea control that uses an embedded font."/>
    <s:TextArea id="ta2"
                width="100%"
                text="This TextArea control does not use an embedded font."/>
</s:View>
```

Beim Übernehmen von Stilen (oder Einbetten von Schriftarten) über die Klassenauswahl (z. B. `s|TextArea`) definieren Sie die Klassenauswahl in der Hauptanwendungsdatei. Sie können keine Klassenauswahl in der Ansicht einer Mobilanwendung definieren.

Weitere Informationen finden Sie unter Schriftarten einbetten.

Kapitel 6: Skinning

Grundlagen des Mobilskinnings

Vergleich von Desktop- und Mobilskins

Mobilskins sind einfacher gestaltet als ihre Entsprechungen in Desktopanwendungen. Daher bestehen zahlreiche Unterschiede. Beispielsweise gilt für Mobilskins Folgendes:

- Mobilskins sind in ActionScript geschrieben. Reine ActionScript-Skins liefern auf Mobilgeräten die beste Leistung.
- Mobilskins erweitern die spark.skins.mobile.supportClasses.MobileSkin-Klasse. Diese Klasse erweitert UIComponent. Im Vergleich dazu erweitert die SparkSkin-Klasse die Skin-Klasse.
- Mobilskins verwenden als Grafikelemente kompilierte FXG-Dateien oder einfache ActionScript-Zeichnungen zur Leistungsoptimierung. Skins für Desktopanwendungen verwenden hingegen meist MXML-Grafiken für das Design.
- Mobilskins müssen keine Skinstatus deklarieren. Da die Skins in ActionScript geschrieben sind, müssen die Status über Prozeduren implementiert werden.
- Mobilskins unterstützen keine Statusübergänge.
- Das Layout von Mobilskins wird manuell festgelegt. „Group“ wird von Mobilskins nicht erweitert, weshalb die Spark-Layouts nicht unterstützt werden. Deshalb werden die untergeordneten Elemente in ActionScript manuell positioniert.
- Mobilskins unterstützen nicht alle Stile. Im Mobildesign fehlen aufgrund von Leistungsunterschieden oder sonstigen Unterschieden bei Mobilskins einige Stile.


Außer den Unterschieden in der Leistung verwendet Flash Builder auch einige Mobilskindateien anders. Dies trifft besonders für in Bibliotheksprojekten verwendete Mobildesigns zu. Blogger Jeffry Houser beschreibt die Problemlösung dazu.

Mobilhostkomponente

Mobilskins deklarieren normalerweise eine öffentliche `hostComponent`-Eigenschaft. Diese Eigenschaft ist nicht erforderlich, wird jedoch empfohlen. Die `hostComponent`-Eigenschaft muss denselben Typ wie die Komponente aufweisen, von der die Skin verwendet wird. Beispielsweise deklariert die ActionBarSkin die `hostComponent` als vom Typ ActionBar:

```
public var hostComponent:ActionBar;
```

Flex legt den Wert der `hostComponent`-Eigenschaft fest, wenn die Skin erstmals von der Komponente geladen wird.

Wie bei Desktopskins können Sie über die Hostkomponente auf Eigenschaften und Methoden der Komponente zugreifen, an die die Skin angefügt wurde. Beispielsweise können Sie auf die öffentlichen Eigenschaften der Hostkomponente zugreifen oder in der Skinklasse der Hostkomponente einen Ereignislistener hinzufügen.

Mobilstile

Mobilskins unterstützen im Vergleich zu Desktopskins nur einen Teil der Stileigenschaften. Diese Stile werden durch das Mobildesign definiert.

In der folgenden Tabelle sind die Stileigenschaften definiert, die bei Verwendung des Mobildesigns für Komponenten verfügbar sind:

Stileigenschaft	Unterstützung durch	Vererbung
`accentColor`	Button, ActionBar, ButtonBar	Ja
`backgroundAlpha`	ActionBar	Nein
`backgroundColor`	Anwendung	Nein
`borderAlpha`	List	Nein
`borderColor`	List	Nein
`borderVisible`	List	Nein
`chromeColor`	ActionBar, Button, ButtonBar, CheckBox, HSlider, RadioButton	Ja
`color`	Alle Komponenten mit Text	Ja
`contentBackgroundAlpha`	TextArea, TextInput	Ja
`contentBackgroundColor`	TextArea, TextInput	Ja
`focusAlpha`	Alle fokussierbaren Komponenten	Nein
`focusBlendMode`	Alle fokussierbaren Komponenten	Nein
`focusColor`	Alle fokussierbaren Komponenten	Ja
`focusThickness`	Alle fokussierbaren Komponenten	Nein
`locale`	Alle Komponenten	Ja
`paddingBottom`	TextArea, TextInput	Nein
`paddingLeft`	TextArea, TextInput	Nein
`paddingRight`	TextArea, TextInput	Nein
`paddingTop`	TextArea, TextInput	Nein
`selectionColor`	ViewMenuItem	Ja

Textbasierte Komponenten unterstützen außerdem die Standardtextstile wie `fontFamily`, `fontSize`, und `fontWeight`.

Um festzustellen, ob eine Stileigenschaft vom Mobildesign unterstützt wird, öffnen Sie die Beschreibung der Komponente in der ActionScript Language Reference. Viele dieser Stileinschränkungen sind darauf zurückzuführen, dass die textbasierten Mobilkomponenten kein TLF (Text Layout Framework) benutzen. Stattdessen ersetzen Mobilskins TLF-basierte Textsteuerelemente durch einfachere Komponenten. Weitere Informationen finden Sie unter „Text in Mobilanwendungen“ auf Seite 147.

Die Stileigenschaften `rollOverColor`, `cornerRadius` und `dropShadowVisible` werden vom Mobildesign nicht unterstützt.

Flex-Ingenieur Jason SJ beschreibt Themen und Stile für Mobilanwendungen in seinem Blog.

Teile von Mobilskins

Bei den Skinteilen müssen Mobilskins den gleichen Skinning-Vertrag wie Desktopskins verwenden. Wenn eine Komponente einen erforderlichen Skinteil aufweist, muss die Mobilskin eine öffentliche Eigenschaft des betreffenden Typs deklarieren.

Ausnahmen

Nicht alle Skinteile sind erforderlich. Beispielsweise enthält das Spark Button-Steuerelement die optionalen Skinteile `iconDisplay` und `labelDisplay`. Daher kann die ButtonSkin-Mobilklasse eine `iconDisplay`-Eigenschaft vom Typ BitmapImage deklarieren. Sie kann auch eine `labelDisplay`-Eigenschaft vom Typ StyleableTextField deklarieren.

Für den `labelDisplay`-Teil wird keine `id`-Eigenschaft festgelegt, da alle für diesen verwendeten Stile Textstile erben. Darüber hinaus ist StyleableTextField keine UIComponent und weist daher keine `id`-Eigenschaft auf. Der `iconDisplay`-Teil unterstützt keine Stile, daher wird ebenfalls keine `id`-Eigenschaft festgelegt.

Stile mit erweiterter CSS festlegen

Wenn Sie Stile für den Skinteil über die erweiterte CSS-Auswahl für `id` festlegen möchten, muss in der Skin auch die `id`-Eigenschaft des Skinteils festgelegt sein. Beispielsweise legt der titleDisplay-Skinteil der `ActionBar` eine `id`-Eigenschaft fest, sodass der Stil über erweiterte CSS festgelegt werden kann:

```
@namespace s "library://ns.adobe.com/flex/spark";
s|ActionBar #titleDisplay {
    color:red;
}
```

Mobildesign

Das Mobildesign bestimmt, welche Stile von einer Mobilanwendung unterstützt werden. Die Anzahl der für das Mobildesign verfügbaren Stile ist etwas kleiner als beim Spark-Design. Eine vollständige Liste von Stilen, die vom Mobildesign unterstützt werden, finden Sie unter „Mobilstile“ auf Seite 173.

Standarddesign für Mobilanwendungen

Das Design für Mobilanwendungen ist in der Datei „themes/Mobile/mobile.swc“ definiert. In dieser Datei sind die globalen Stile für Mobilanwendungen sowie die Standardeinstellungen für die einzelnen Mobilkomponenten definiert. Mobilskins sind in dieser Designdatei im spark.skins.mobile.*-Paket definiert sind. Dieses Paket enthält die MobileSkin-Basisklasse.

Die Designdatei „mobile.swc“ ist standardmäßig in Flash Builder-Mobilprojekten enthalten, aber die SWC-Datei ist im Paket-Explorer nicht vorhanden.

Wenn Sie in Flash Builder ein neues Mobilprojekt erstellen, wird dieses Design standardmäßig angewendet.

Design ändern

Wenn Sie das Design ändern möchten, geben Sie im `theme`-Argument des Compilers das neue Design an, z. B.:

```
-theme+=myThemes/NewMobileTheme.swc
```

Weitere Informationen zu Designs finden Sie unter Designs.

Flex-Ingenieur Jason SJ beschreibt die Erstellung und Überlagerung eines Designs in einer Mobilanwendung in seinem Blog.

Zustände von Mobilskins

Die MobileSkin-Klasse überschreibt den Statusmechanismus der UIComponent-Klasse und verwendet nicht die Ansichtsstatusimplementierung von Desktopanwendungen. Daher deklarieren Mobilskins nur die von der Skin implementierten Skinzustände der Hostkomponente. Sie ändern den Status mithilfe von Prozeduren, und zwar nur basierend auf dem Statusnamen. Demgegenüber müssen Desktopskins alle Status deklarieren, unabhängig davon, ob diese verwendet werden. Desktopskins verwenden auch die Klassen im Paket „mx.states.*“ zum Ändern der Status.

Mobilskins implementieren weniger Zustände als ihre Entsprechungen in Desktopanwendungen. Beispielsweise implementiert die spark.skins.mobile.ButtonSkin-Klasse die Zustände `up`, `down` und `disabled`. Die spark.skins.spark.ButtonSkin implementiert alle diese Status sowie den `over`-Status. Die Mobilskin definiert kein Verhalten für den `over`-Zustand, da dieser Zustand auf einem Mobilgerät normalerweise nicht verwendet wird.

commitCurrentState()-Methode

Die Mobilskinklassen definieren Zustandsverhalten über die `commitCurrentState()`-Methode. Sie können einer Mobilskin Verhalten hinzufügen, um zusätzliche Status zu unterstützen, indem Sie die `commitCurrentState()`-Methode in Ihrer benutzerdefinierten Skinklasse bearbeiten.

currentState-Eigenschaft

Die Darstellung einer Skin hängt vom Wert der `currentState`-Eigenschaft ab. Beispielsweise bestimmt in der ButtonSkin-Mobilklasse der Wert der `currentState`-Eigenschaft die als Rahmenklasse verwendete FXG-Klasse:

```
if (currentState == "down")
    return downBorderSkin;
else
    return upBorderSkin;
```

Weitere Informationen zur Eigenschaft `currentState` finden Sie unter Erstellen und Anwenden von Anzeigestatus.

Grafiken bei Mobilanwendungen

Mobilskins verwenden als Grafikelemente meist kompilierte FXG-Dateien. Skins für Desktopanwendungen verwenden hingegen meist MXML-Grafiken für das Design.

Eingebettete Bitmapgrafiken

In den Klassen können Sie auch problemlos eingebettete Bitmapgrafiken verwenden. Bitmaps werden jedoch bei unterschiedlichen Bildschirmauflösungen nicht immer gut skaliert. Wenn Sie für die einzelnen Bildschirmauflösungen unterschiedliche Elemente erstellen, werden diese besser skaliert.

Grafiken im Standardmobildesign

Die Mobilskins im Standardmobildesign verwenden FXG-Grafiken, die für den DPI-Wert des Zielgeräts optimiert sind. Die Skins laden Grafiken in Abhängigkeit vom Wert der `applicationDPI`-Eigenschaft der Stammanwendung. Wenn beispielsweise ein CheckBox-Steuerelement auf einem Gerät mit einem DPI-Wert von 320 geladen wird, verwendet die CheckBoxSkin-Klasse die Grafik „spark.skins.mobile320.assets.CheckBox_up.fxg“ für die `upIconClass`-Eigenschaft. Mit 160 DPI wird die Grafik „spark.skins.mobile160.assets.CheckBox_up.fxg“ verwendet.

Das folgende *Desktop*beispiel zeigt die verschiedenen Grafiken, die von der CheckBox-Skin für unterschiedliche DPI-Werte verwendet werden:

```
<?xml version="1.0"?>
<!-- mobile_skins/ShowCheckBoxSkins.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:skins160="spark.skins.mobile160.assets.*"
    xmlns:skins240="spark.skins.mobile240.assets.*"
    xmlns:skins320="spark.skins.mobile320.assets.*">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <!--
    NOTE: You must add the mobile theme directory to source path
    to compile this example.

    For example:
    mxmlc -source-path+=\frameworks\projects\mobiletheme\src\ ShowCheckBoxSkins.mxml
    -->
    <s:Label text="160 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins160:CheckBox_down/>
        <skins160:CheckBox_downSymbol/>
        <skins160:CheckBox_downSymbolSelected/>
        <skins160:CheckBox_up/>
        <skins160:CheckBox_upSymbol/>
        <skins160:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="240 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins240:CheckBox_down/>
        <skins240:CheckBox_downSymbol/>
        <skins240:CheckBox_downSymbolSelected/>
        <skins240:CheckBox_up/>
        <skins240:CheckBox_upSymbol/>
        <skins240:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="320 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins320:CheckBox_down/>
        <skins320:CheckBox_downSymbol/>
        <skins320:CheckBox_downSymbolSelected/>
        <skins320:CheckBox_up/>
        <skins320:CheckBox_upSymbol/>
        <skins320:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <s:Label text="down, downSymbol, downSymbolSelected, up, upSymbol, upSymbolSelected"/>
</s:Application>
```

Weitere Informationen zu Auflösungen und DPIs in Mobilanwendungen finden Sie unter „Unterstützung mehrerer Bildschirmgrößen und DPI-Werte in Mobilanwendungen“ auf Seite 132.

In ActionScript-Skins können Sie auch als Bitmaps zwischengespeicherte Vektoren verwenden. Der einzige Nachteil besteht darin, dass Sie keine Übergänge verwenden können, bei denen die Pixel neu gezeichnet werden müssen, etwa Alpha-Übergänge. Weitere Informationen finden Sie unter www.adobe.com/devnet/air/flex/articles/writing_multiscreen_air_apps.html.

Skins für Mobilanwendungen erstellen

Zum Anpassen von Mobilskins können Sie benutzerdefinierte Mobilskinklassen erstellen. In manchen Fällen können Sie auch die Elemente bearbeiten, die von einer Mobilskinklasse verwendet werden.

Wenn Sie eine Mobilskinklasse bearbeiten, können Sie zustandsbasierte Interaktionen ändern, Unterstützung für neue Stile implementieren oder der Skin untergeordnete Komponente hinzufügen oder solche entfernen. Sie beginnen in der Regel mit dem Quellcode der vorhandenen Skin und speichern diesen als neue Klasse.

Sie können auch die von Mobilskins verwendeten Elemente bearbeiten, um die visuellen Eigenschaften der Skin zu ändern, z. B. Größe, Farbe oder Verläufe und Hintergründe. In diesem Fall bearbeiten Sie auch die von den Skins verwendeten FXG-Elemente. Die von den Mobilskins verwendeten *.fxg-Quelldateien befinden sich im Verzeichnis spark/skins/mobile/assets.

Nicht alle visuellen Eigenschaften für Mobilskins sind in *.fxg-Dateien definiert. Beispielsweise ist die Hintergrundfarbe der Button-Skin in der `chromeColor`-Stileigenschaft der ButtonSkin-Klasse definiert. Sie ist nicht in einem FXG-Element definiert. In diesem Fall ändern Sie die Hintergrundfarbe der Skinklasse.

Mobilskinklasse erstellen

Wenn Sie eine benutzerdefinierte Mobilskinklasse erstellen, ist es am einfachsten, eine vorhandene Mobilskinklasse als Basis zu verwenden. Ändern Sie dann diese Klasse und verwenden Sie sie als benutzerdefinierte Skin.

Benutzerdefinierte Skinklasse erstellen:

1. Erstellen Sie in Ihrem Projekt ein Verzeichnis (z. B. „customSkins“). Dieses Verzeichnis wird als Paketname für die benutzerdefinierten Skins verwendet. Es ist nicht erforderlich, wird jedoch empfohlen, benutzerdefinierte Skins in einem eigenen Paket zu speichern.
2. Erstellen Sie im neuen Verzeichnis eine benutzerdefinierte Skinklasse. Benennen Sie die neue Klasse wie gewünscht, z. B. „CustomButtonSkin.as“.
3. Kopieren Sie den Inhalt der Skinklasse, die Sie als Grundlage für die neue Klasse verwenden. Wenn Sie beispielsweise ButtonSkin als Basisklasse verwenden, kopieren Sie in die neue benutzerdefinierte Skinklasse den Inhalt der Datei „spark.skins.mobile.ButtonSkin“.
4. Bearbeiten Sie die neue Klasse. Nehmen Sie beispielsweise an der CustomButtonSkin-Klasse die folgenden minimalen Änderungen vor:
   - Ändern Sie den Paketspeicherort:

     ```
     package customSkins
     //was: package spark.skins.mobile
     ```

   - Ändern Sie den Namen der Klasse in der Klassendeklaration. Erweitern Sie außerdem die Klasse, auf der die neue Skin basiert, nicht die Basisskinklasse:

     ```
     public class CustomButtonSkin extends ButtonSkin
     // was: public class ButtonSkin extends ButtonSkinBase
     ```

   - Ändern Sie den Klassennamen in Konstruktor:

```
    public function CustomButtonSkin()
    //was: public function ButtonSkin()
```

5 Ändern Sie die benutzerdefinierte Skinklasse. Fügen Sie beispielsweise die Unterstützung für zusätzliche Status oder neue untergeordnete Komponenten hinzu. In der Klasse selbst sind auch einige Grafikelemente definiert, also können Sie einige Elemente ändern.

Um die Lesbarkeit der Skinklasse zu erhöhen, entfernen Sie in der Regel alle Methoden aus der benutzerdefinierten Skin, die Sie nicht überschreiben.

Mit der folgenden benutzerdefinierten Skinklasse wird ButtonSkin erweitert und die `drawBackground()`-Methode durch benutzerdefinierte Logik ersetzt. Sie ersetzt den linearen Verlauf der Hintergrundfüllung durch einen radialen Verlauf.

```
package customSkins {
    import mx.utils.ColorUtil;
    import spark.skins.mobile.ButtonSkin;
    import flash.display.GradientType;
    import spark.skins.mobile.supportClasses.MobileSkin;
    import flash.geom.Matrix;
    public class CustomButtonSkin extends ButtonSkin {

        public function CustomButtonSkin() {
            super();
        }
        private static var colorMatrix:Matrix = new Matrix();
        private static const CHROME_COLOR_ALPHAS:Array = [1, 1];
        private static const CHROME_COLOR_RATIOS:Array = [0, 127.5];

        override protected function drawBackground(unscaledWidth:Number,
unscaledHeight:Number):void {
            super.drawBackground(unscaledWidth, unscaledHeight);

            var chromeColor:uint = getStyle("chromeColor");
            /*
            if (currentState == "down") {
                graphics.beginFill(chromeColor);
            } else {
            */
            var colors:Array = [];
           colorMatrix.createGradientBox(unscaledWidth, unscaledHeight, Math.PI / 2, 0, 0);
            colors[0] = ColorUtil.adjustBrightness2(chromeColor, 70);
            colors[1] = chromeColor;
            graphics.beginGradientFill(GradientType.RADIAL, colors, CHROME_COLOR_ALPHAS,
CHROME_COLOR_RATIOS, colorMatrix);
            // }
            graphics.drawRoundRect(layoutBorderSize, layoutBorderSize,
                unscaledWidth - (layoutBorderSize * 2),
                unscaledHeight - (layoutBorderSize * 2),
                layoutCornerEllipseSize, layoutCornerEllipseSize);
            graphics.endFill();
        }

    }
}
```

6 Wenden Sie in der Anwendung die benutzerdefinierte Skin mit einer der unter „Benutzerdefinierte Mobilskin anwenden“ auf Seite 185 beschriebenen Methoden an. Im folgenden Beispiel wird mithilfe der `skinClass`-Eigenschaft für das Komponententag die `customSkins.CustomButtonSkin`-Skin angewendet:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/CustomButtonSkinView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark" title="Home">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Button label="Click Me" skinClass="customSkins.CustomButtonSkin"/>

</s:View>
```

Lebenszyklusmethoden von Mobilskins

Beim Erstellen benutzerdefinierter Skinklassen sollten Sie sich mit den folgenden UIComponent-Methoden vertraut machen. Mit diesen geerbten, geschützten Methoden definieren Sie die untergeordneten Elemente und Member einer Skin und unterstützen die Interaktion mit anderen Komponenten in der Anzeigeliste.

- `createChildren()` – Hiermit erstellen Sie beliebige untergeordnete Grafiken oder Textobjekte für die Skin.
- `commitProperties()` – Hiermit kopieren Sie bei Bedarf Komponentendaten in die Skin.
- `measure()` – Hiermit messen Sie die Skin so effizient wie möglich und speichern die Ergebnisse in der `measuredWidth`-Eigenschaft und der `measuredHeight`-Eigenschaft der Skin.
- `updateDisplayList()` – Hiermit legen Sie die Position und Größe von Grafiken und Text fest. Sie können alle erforderlichen ActionScript-Zeichnungsaktionen ausführen. Mit dieser Methode werden die Methoden `drawBackground()` und `layoutContents()` in der Skin aufgerufen.

Weitere Informationen zur Verwendung dieser Methoden finden Sie unter Implementing the component.

Gängige Methoden zum Anpassen in Mobilskins

Viele Mobilskins implementieren die folgenden Methoden:

- `layoutContents()` – Positioniert die untergeordneten Elemente für die Skin, wie z. B. Schlagschatten und Beschriftungen. Mobilskinklassen unterstützen keine Spark-Layouts wie z. B. HorizontalLayout und VerticalLayout. Legen Sie das Layout der untergeordneten Elemente für die Skin manuell mithilfe einer Methode wie z. B. `layoutContents()` fest.
- `drawBackground()` – Rendert den Hintergrund für die Skin. Typische Verwendungsmöglichkeiten sind die Darstellung der Stile `chromeColor`, `backgroundColor` oder `contentBackgroundColor` basierend auf der Skinform. Diese Methode kann auch zum Einfärben verwendet werden, wie z. B. mit der `applyColorTransform()`-Methode.
- `commitCurrentState()` – Definiert Zustandsverhalten für Mobilskins. Durch Bearbeiten dieser Methode können Sie unterstützte Zustände hinzufügen oder entfernen oder auch das Verhalten vorhandener Zustände ändern. Diese Methode wird bei Zustandsänderungen aufgerufen. Die meisten Skinklassen überschreiben diese Methode. Weitere Informationen finden Sie unter „Zustände von Mobilskins“ auf Seite 176.

Benutzerdefinierte FXG-Elemente erstellen

Die meisten visuellen Elemente von Mobilskins werden als FXG definiert. FXG ist eine deklarative Syntax für die Definition statischer Grafiken. Mit einer Grafikanwendung wie Adobe Fireworks, Adobe Illustrator oder Adobe Catalyst können Sie ein FXG-Dokument exportieren. Anschließend können Sie das FXG-Dokument in der Mobilskin verwenden. Sie können FXG-Dokumente auch in einem Texteditor erstellen, jedoch kann es schwierig sein, komplexe Grafiken von Grund auf neu zu schreiben.

Die Zustände von Mobilskins werden üblicherweise in FXG-Dateien definiert. Beispielsweise verwendet die CheckBoxSkin-Klasse die folgenden FXG-Dateien, um die Darstellung von Feld und Häkchensymbol zu definieren:

- CheckBox_down.fxg
- CheckBox_downSymbol.fxg
- CheckBox_downSymbolSelected.fxg
- CheckBox_up.fxg
- CheckBox_upSymbol.fxg
- CheckBox_upSymbolSelected.fxg

Wenn Sie diese Dateien in einer Grafikanwendung öffnen, werden sie wie folgt dargestellt:

CheckBox-Zustände (down, downSymbol, downSymbolSelected, up, upSymbol und upSymbolSelected)

FXG-Dateien für mehrere Auflösungen

Die meisten Mobilskins weisen drei FXG-Grafikdateigruppen auf, eine pro Standardzielauflösung. Beispielsweise werden verschiedene Versionen aller sechs CheckBoxSkin-Klassen in den Verzeichnissen „spark/skins/mobile160“, „spark/skins/mobile240“ und „spark/skins/mobile320“ angezeigt.

Beim Erstellen einer benutzerdefinierten Skin können Sie einen der folgenden Schritte ausführen:

- Verwenden Sie eine der Standardskins als Grundlage (gewöhnlich 160 DPI). Fügen Sie Logik hinzu, durch die die benutzerdefinierte Skin skaliert und an das Gerät angepasst wird, auf dem die Anwendung ausgeführt wird. Legen Sie dazu die `applicationDPI`-Eigenschaft im Application-Objekt fest.
- Erstellen Sie alle drei Versionen der benutzerdefinierten Skin (160, 240 und 320 DPI) für die optimale Darstellung.

Manche Mobilskins verwenden einen einzigen FXG-Dateisatz für die Grafikelemente und weisen keine DPI-spezifischen Grafiken auf. Diese Elemente sind im Verzeichnis „spark/skins/mobile/assets“ gespeichert. Beispielsweise gibt es für die ViewMenuItem-Skins und TabbedViewNavigator-Schaltflächenleisten-Skins keine DPI-spezifischen Versionen, weshalb alle FXG-Elemente in diesem Verzeichnis gespeichert sind.

FXG-Datei anpassen

Sie können eine vorhandene FXG-Datei öffnen und anpassen oder aber eine FXG-Datei erstellen und in einem Grafikeditor wie z. B. Adobe Illustrator exportieren. Wenn Sie die FXG-Datei bearbeitet haben, wenden Sie sie auf die Skinklasse an.

Benutzerdefinierte Skin durch Bearbeiten einer FXG-Datei erstellen:

1. Erstellen Sie eine benutzerdefinierte Skinklasse und speichern Sie sie im Ordner „customSkins“, wie unter „Mobilskinklasse erstellen“ auf Seite 178 beschrieben.
2. Erstellen Sie im Verzeichnis „customSkins“ ein Unterverzeichnis, z. B. „assets“. Das Erstellen eines Unterverzeichnisses ist optional, trägt jedoch zur Übersichtlichkeit der FXG-Dateien und Skinklassen bei.

3 Erstellen Sie im Verzeichnis „assets“ eine Datei und kopieren Sie in diese den Inhalt einer vorhandenen FXG-Datei. Erstellen Sie beispielsweise eine Datei mit dem Namen „CustomCheckBox_upSymbol.fxg“. Kopieren Sie den Inhalt der Datei „spark/skins/mobile160/assets/CheckBox_upSymbol.fxg“ in die neue Datei „CustomCheckBox_upSymbol.fxg“.

4 Ändern Sie die neue FXG-Datei. Ersetzen Sie beispielsweise die Logik, mit der das Aktivierungs-„X“ mit Verlaufseinträgen gezeichnet wird:

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- mobile_skins/customSkins/assets/CustomCheckBox_upSymbol.fxg -->
<Graphic xmlns="http://ns.adobe.com/fxg/2008" version="2.0"
    viewWidth="32" viewHeight="32">
    <!-- Main Outer Border -->
    <Rect x="1" y="1" height="30" width="30" radiusX="2" radiusY="2">
        <stroke>
            <SolidColorStroke weight="1" color="#282828"/>
        </stroke>
    </Rect>
    <!-- Replace check mark with an "x" -->
    <Group x="2" y="2">
        <Line xFrom="3" yFrom="3" xTo="25" yTo="25">
            <stroke>
               <LinearGradientStroke caps="none" weight="8" joints="miter" miterLimit="4">
                    <GradientEntry color="#FF0033"/>
                    <GradientEntry color="#0066FF"/>
                </LinearGradientStroke>
            </stroke>
        </Line>
        <Line xFrom="25" yFrom="3" xTo="3" yTo="25">
            <stroke>
            <stroke>
               <LinearGradientStroke caps="none" weight="8" joints="miter" miterLimit="4">
                    <GradientEntry color="#FF0033"/>
                    <GradientEntry color="#0066FF"/>
                </LinearGradientStroke>
            </stroke>
            </stroke>
        </Line>
    </Group>
</Graphic>
```

5 Importieren Sie in der benutzerdefinierten Skinklasse die neue FXG-Klasse und wenden Sie sie auf eine Eigenschaft an. Dies kann beispielsweise die CustomCheckBox-Klasse sein:

   1 Importieren Sie die neue FXG-Datei:

   ```
   //import spark.skins.mobile.assets.CheckBox_upSymbol;
   import customSkins.assets.CustomCheckBox_upSymbol;
   ```

   2 Fügen Sie der benutzerdefinierten Skinklasse das neue Element hinzu. Ändern Sie z. B. den Wert der `upSymbolIconClass`-Eigenschaft so, dass diese auf das neue FXG-Element zeigt:

   ```
   upSymbolIconClass = CustomCheckBox_upSymbol;
   ```

Die vollständige benutzerdefinierte Skinklasse sieht folgendermaßen aus:

```
// mobile_skins/customSkins/CustomCheckBoxSkin.as
package customSkins {
    import spark.skins.mobile.CheckBoxSkin;
    import customSkins.assets.CustomCheckBox_upSymbol;

    public class CustomCheckBoxSkin extends CheckBoxSkin {
        public function CustomCheckBoxSkin() {
            super();
            upSymbolIconClass = CustomCheckBox_upSymbol; // was CheckBox_upSymbol
        }
    }
}
```

Informationen zum Arbeiten mit und zum Optimieren von FXG-Elementen für Skins finden Sie unter Optimizing FXG.

FXG-Dateien in Anwendungen anzeigen

Da FXG-Dateien in XML geschrieben werden, kann es schwierig sein, sich vorzustellen, wie das endgültige Produkt aussehen wird. Sie können eine Flex-Anwendung schreiben, mit der Sie FXG-Dateien importieren und rendern, indem Sie sie als Komponenten hinzufügen und in einem Spark-Container umschließen.

Zum Hinzufügen von FXG-Dateien als Komponenten einer Anwendung fügen Sie dem Quellpfad der Anwendung den Speicherort der Quelldateien hinzu. Wenn Sie z. B. die FXG-Elemente in einer webbasierten Anwendung anzeigen möchten, fügen Sie dem Quellpfad das Mobildesign hinzu. Der Compiler kann dann die FXG-Dateien finden.

Im folgenden *Desktop*beispiel werden die verschiedenen FXG-Elemente der CheckBox-Komponente gerendert, wenn sie in einer Mobilanwendung verwendet wird. Fügen Sie beim Kompilieren dieses Beispiels das Verzeichnis „frameworks\projects\mobiletheme\src\“ dem `source-path`-Argument des Compilers hinzu.

```
<?xml version="1.0"?>
<!-- mobile_skins/ShowCheckBoxSkins.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:skins160="spark.skins.mobile160.assets.*"
    xmlns:skins240="spark.skins.mobile240.assets.*"
    xmlns:skins320="spark.skins.mobile320.assets.*">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <!--
    NOTE: You must add the mobile theme directory to source path
    to compile this example.

    For example:
    mxmlc -source-path+=\frameworks\projects\mobiletheme\src\ ShowCheckBoxSkins.mxml
    -->
    <s:Label text="160 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins160:CheckBox_down/>
        <skins160:CheckBox_downSymbol/>
        <skins160:CheckBox_downSymbolSelected/>
        <skins160:CheckBox_up/>
        <skins160:CheckBox_upSymbol/>
        <skins160:CheckBox_upSymbolSelected/>
    </s:HGroup>
```

```
    <mx:Spacer height="30"/>
    <s:Label text="240 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins240:CheckBox_down/>
        <skins240:CheckBox_downSymbol/>
        <skins240:CheckBox_downSymbolSelected/>
        <skins240:CheckBox_up/>
        <skins240:CheckBox_upSymbol/>
        <skins240:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="320 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins320:CheckBox_down/>
        <skins320:CheckBox_downSymbol/>
        <skins320:CheckBox_downSymbolSelected/>
        <skins320:CheckBox_up/>
        <skins320:CheckBox_upSymbol/>
        <skins320:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <s:Label text="down, downSymbol, downSymbolSelected, up, upSymbol, upSymbolSelected"/>
</s:Application>
```

Text in benutzerdefinierten Mobilskins

Verwenden Sie für die Textdarstellung in Mobilskins die StyleableStageText- oder StyleableTextField-Klasse. Diese Textklassen sind für Mobilanwendungen optimiert.

StyleableStageText bietet Zugriff auf die nativen Texteingaben für die TextInput- und TextArea-Steuerelemente. Die Klasse erweitert die UIComponent-Klasse und implementiert die IEditableText- und ISoftKeyboardHintClient-Schnittstellen.

StyleableTextField wird ebenfalls von den TextInput- und TextArea-Steuerelementen verwendet, wenn Sie auf die nativen Eingaben nicht zugreifen möchten. Die Klasse wird außerdem von Steuerelementen verwendet, die keine Texteingabefunktion haben, wie ActionBar und Button. Sie erweitert die TextField-Klasse und implementiert die ISimpleStyleClient-Schnittstelle und die IEditableText-Schnittstelle.

Weitere Informationen zu Textsteuerelementen in Mobilanwendungen finden Sie unter „Text in Mobilanwendungen“ auf Seite 147.

TLF in Mobilskins

In Mobilanwendungen sollten Sie aus Leistungsgründen Klassen vermeiden, die Text Layout Framework (TLF) verwenden. In manchen Fällen, wie z. B. bei der Spark Label-Komponente, können Sie Klassen einsetzen, die FTE verwenden.

htmlText in Mobilskins

Sie können die Eigenschaft `htmlText` nicht in Mobilanwendungen verwenden.

Gesten mit Text

Mit Berührungs+Zieh-Gesten wird immer Text ausgewählt (sofern der Text auswählbar oder bearbeitbar ist). Wenn der Text sich innerhalb einer Bildlaufleiste befindet, wird nur dann ein Bildlauf ausgeführt, wenn die Bewegung außerhalb der Textkomponente stattfindet. Diese Gesten können nur verwendet werden, wenn der Text bearbeitbar und auswählbar ist.

Text bearbeitbar und auswählbar machen

Um den Text bearbeitbar und auswählbar zu machen, legen Sie die `editable`-Eigenschaft und die `selectable`-Eigenschaft auf `true` fest:

```
textDisplay.editable = true;
textDisplay.selectable = true;
```

Bidirektionalität

Bidirektionalität wird für Text in der StyleableStageText- oder StyleableTextField-Klasse nicht unterstützt.

Benutzerdefinierte Mobilskin anwenden

Eine benutzerdefinierte Skin können Sie auf eine Mobilkomponente auf dieselbe Weise anwenden wie auf eine Komponente in einer Desktopanwendung.

Skin in ActionScript anwenden

```
// Call the setStyle() method:
myButton.setStyle("skinClass", "MyButtonSkin");
```

Skin in MXML anwenden

```
<!-- Set the skinClass property: -->
<s:Button skinClass="MyButtonSkin"/>
```

Skin in CSS anwenden

```
// Use type selectors for mobile skins, but only in the root document:
s|Button {
    skinClass: ClassReference("MyButtonSkin");
}
```

oder

```
// Use class selectors for mobile skins in any document:
.myStyleClass {
    skinClass: ClassReference("MyButtonSkin");
}
```

Beispiele für die Anwendung einer benutzerdefinierten Mobilskin

Im folgenden Beispiel werden alle drei Methoden der Anwendung einer benutzerdefinierten Mobilskin auf Mobilkomponenten veranschaulicht:

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

    <fx:Script>
        <![CDATA[
            import customSkins.CustomButtonSkin;
            private function changeSkin():void {
                b3.setStyle("skinClass", customSkins.CustomButtonSkin);
            }
        ]]>
    </fx:Script>

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        .customButtonStyle {
            skinClass: ClassReference("customSkins.CustomButtonSkin");
        }
    </fx:Style>

    <s:Button id="b1" label="Click Me" skinClass="customSkins.CustomButtonSkin"/>
    <s:Button id="b2" label="Click Me" styleName="customButtonStyle"/>
    <s:Button id="b3" label="Click Me" click="changeSkin()"/>

</s:View>
```

Wenn Sie zum Anwenden einer benutzerdefinierten Skin eine CSS-*Typauswahl* verwenden, legen Sie diese in der Stammdatei der Mobilanwendung fest. Typauswahlen können nicht in Mobilansichten, d. h. in benutzerdefinierten Komponenten, festgelegt werden. Dennoch können Sie Stile in ActionScript, MXML oder CSS mit einer Klassenauswahl in beliebigen Ansichten oder Dokumenten Ihrer Mobilanwnedung festlegen.

Kapitel 7: Testen und Debuggen

Startkonfigurationen verwalten

Flash Builder nutzt Startkonfigurationen beim Ausführen oder Debuggen von Mobilanwendungen. Sie können angeben, ob die Anwendung auf dem Desktop oder auf einem Gerät gestartet werden soll, das an den Computer angeschlossen ist.

Führen Sie zum Erstellen einer Startkonfiguration die folgenden Schritte aus:

1 Wählen Sie „Ausführen“ > „Konfigurationen ausführen“, um das Dialogfeld „Konfigurationen ausführen“ zu öffnen.

Zum Öffnen des Dialogfelds „Debugkonfigurationen“ wählen Sie „Ausführen“ > „Debugkonfigurationen“. Siehe „Mobilanwendungen auf einem Gerät testen und debuggen“ auf Seite 188.

Sie können auf die beiden Konfigurationsmenüs auch über die Dropdownliste der Schaltfläche „Ausführen“ oder „Debuggen“ in der Symbolleiste von Flash Builder zugreifen.

2 Erweitern Sie den Mobilanwendungsknoten. Klicken Sie in der Symbolleiste des Dialogfelds auf die Schaltfläche „Neue Startkonfiguration“.

3 Geben Sie in der Dropdownliste eine Zielplattform an.

4 Geben Sie eine Startmethode an:

- **Auf Desktop**

  Führt die Anwendung auf Ihrem Desktop mit dem AIR Debug Launcher (ADL) gemäß einer angegebenen Gerätekonfiguration aus oder debuggt sie. Diese Startmethode ist keine echte Emulation der auf einem Gerät ausgeführten Anwendung. Sie können jedoch das Anwendungslayout anzeigen und mit der Anwendung interagieren. Siehe „Anwendungsvorschau mit ADL“ auf Seite 188.

  Klicken Sie zum Bearbeiten von Gerätekonfigurationen auf „Konfigurieren“. Siehe „Geräteinformationen für die Desktopvorschau konfigurieren“ auf Seite 188.

- **Auf Gerät**

  Installiert die Anwendung auf dem Gerät und führt sie aus.

  Für die Google Android-Plattform installiert Flash Builder die Anwendung auf Ihrem Gerät und führt sie aus. Flash Builder greift auf das mit dem USB-Port des Computers verbundene Gerät zu. Weitere Informationen finden Sie unter „Mobilanwendungen auf einem Gerät testen und debuggen“ auf Seite 188.

  Unter Windows ist ein USB-Treiber erforderlich, damit ein Android-Gerät an den Computer angeschlossen werden kann. Weitere Informationen finden Sie unter „USB-Gerätetreiber für Android-Geräte (Windows) installieren“ auf Seite 19.

5 Legen Sie fest, ob ggf. die Anwendungsdaten bei jedem Start gelöscht werden sollen.

Mobilanwendungen auf dem Desktop testen und debuggen

Wenn Sie erste Tests oder Debugvorgänge ausführen möchten oder nicht über ein Mobilgerät verfügen, können Sie in Flash Builder mit dem AIR Debug Launcher (ADL) Anwendungen auf dem Desktop testen und debuggen.

Vor dem erstmaligen Testen oder Debuggen einer Mobilanwendung müssen Sie eine Startkonfiguration definieren. Geben Sie die Zielplattform und „Auf Desktop“ als Startmethode an. Siehe „Startkonfigurationen verwalten“ auf Seite 187.

Geräteinformationen für die Desktopvorschau konfigurieren

Die Eigenschaften einer Gerätekonfiguration bestimmen, wie die Anwendung im AIR Debug Launcher (ADL) und im Designmodus von Flash Builder dargestellt wird.

Unter „Gerätekonfigurationen festlegen“ auf Seite 15 sind die unterstützten Konfigurationen aufgeführt. Gerätekonfigurationen haben keine Auswirkung auf die Darstellung einer Anwendung auf dem Gerät.

Bildschirmauflösung

Sie können Ihre Anwendung auf dem Entwicklungsdesktop als Vorschau betrachten oder das Layout der Anwendung im Designmodus von Flash Builder anzeigen. Flash Builder arbeitet mit einer Bildschirmauflösung von 240 DPI. Die Darstellung einer Anwendung in der Vorschau weicht manchmal von der Darstellung auf einem Gerät ab, das eine unterschiedliche Pixeldichte unterstützt.

Anwendungsvorschau mit ADL

Bei der Anwendungsvorschau auf dem Desktop startet Flash Builder die Anwendung mit ADL. ADL zeigt ein Gerätemenü mit entsprechenden Direktaufrufen zur Emulierung der Tasten auf dem Gerät an.

Beispiel: Zur Emulierung der Zurücktaste auf dem Gerät wählen Sie „Gerät“ > „Zurück“ aus. Zur Emulierung des Drehens wählen Sie „Gerät“ > „Nach links drehen“ oder „Gerät“ > „Nach rechts drehen“. Die Optionen zum Drehen sind deaktiviert, wenn Sie die automatische Ausrichtung nicht ausgewählt haben.

Ziehen Sie mit der Maus in einer Liste, um den Bildlauf in einer Liste auf dem Gerät zu emulieren.

Der zertifizierte Adobe-Experte für Flex, Brent Arnold, hat ein Videotutorial zur Verwendung von ADL für die Vorschau einer Mobilanwendung auf dem Desktop erstellt.

Mobilanwendungen auf einem Gerät testen und debuggen

Sie können eine Mobilanwendung auf Ihrem Entwicklungsdesktop oder einem Gerät mithilfe von Flash Builder testen oder debuggen.

Sie testen und debuggen Anwendungen basierend auf einer von Ihnen definierten Startkonfiguration. Flash Builder verwendet die Startkonfiguration sowohl zum Ausführen als auch zum Debuggen der Anwendung. Wenn Sie eine Anwendung mithilfe von Flash Builder auf einem Gerät debuggen, installiert Flash Builder eine Debugversion der Anwendung auf dem Gerät.

Hinweis: *Wenn Sie einen Releasebuild auf ein Gerät exportieren, installieren Sie eine Nicht-Debugversion der Anwendung. Die Nicht-Debugversion ist nicht für das Debuggen geeignet.*

Weitere Informationen finden Sie unter Startkonfigurationen verwalten.

Anwendung auf einem Google Android-Gerät debuggen

Auf einem Android-Gerät ist für das Debugging Android 2.2 oder höher erforderlich.

Sie können wie folgt debuggen:

Über USB debuggen Um eine Anwendung über eine USB-Verbindung zu debuggen, schließen Sie das Gerät über einen USB-Port an den Hostcomputer an. Beim Debuggen über USB verpackt Flash Builder die Anwendung und installiert und startet sie auf dem Gerät, bevor das Debugging beginnt. Stellen Sie sicher, dass das Gerät während der gesamten Debuggingsitzung an den USB-Port des Hostcomputers angeschlossen ist.

Debuggen über Netzwerk Wenn Sie eine Anwendung über das Netzwerk debuggen, müssen sich das Gerät und der Hostcomputer im selben Netzwerk befinden. Sie können per Wi-Fi, Ethernet-Kabel oder Bluetooth verbunden sein.

Beim Debuggen über Netzwerk können Sie mit Flash eine Anwendung debuggen, die bereits auf einem verbundenen Gerät installiert ist, ohne die Anwendung erneut zu installieren. Schließen Sie das Gerät nur über einen USB-Port an den Hostcomputer an, während die Anwendung verpackt wird und während die Anwendung auf dem Gerät installiert wird. Während des Debuggings können Sie das Gerät vom USB-Port trennen. Stellen Sie jedoch sicher, dass während der gesamten Debuggingsitzung zwischen dem Gerät und dem Hostcomputer eine Netzwerkverbindung besteht.

Debugging der Anwendung vorbereiten

Führen Sie die folgenden Schritte aus, bevor Sie mit dem Debuggen über USB oder über ein Netzwerk beginnen:

1 (Windows) Vergewissern Sie sich, dass der richtige USB-Treiber installiert ist.

Installieren Sie unter Windows den Android USB-Treiber. Weitere Informationen hierzu entnehmen Sie bitte der Dokumentation zum Android SDK Build. Weitere Informationen finden Sie unter „USB-Gerätetreiber für Android-Geräte (Windows) installieren“ auf Seite 19.

2 Stellen Sie sicher, dass das Debuggen über USB auf dem Gerät aktiviert ist.

Gehen Sie in den Geräteeinstellungen zu „Anwendungen“ > „Entwicklung“ und aktivieren Sie „USB-Debugging“.

Nach angeschlossenen Geräten suchen

Wenn Sie eine Mobilanwendung auf einem Gerät ausführen oder debuggen, sucht Flash Builder nach angeschlossenen Geräten. Wenn Flash Builder ein einziges angeschlossenes Gerät findet, stellt Flash Builder die Anwendung bereit und startet sie. Ansonsten öffnet Flash Builder das Dialogfenster zum Auswählen eines Geräts für diese Szenarien:

- Kein angeschlossenes Gerät gefunden
- Angeschlossenes Gerät mit Offline-Status gefunden oder die Betriebssystemversion wird nicht unterstützt
- Mehrere angeschlossene Geräte gefunden

Wenn mehrere Geräte gefunden wurden, werden im Dialogfeld „Gerät wählen“ die Geräte und deren Status (online oder offline) aufgeführt. Wählen Sie das Gerät aus, das gestartet werden soll.

Im Dialogfeld „Gerät wählen“ werden die Betriebssystemversion und die AIR-Version aufgeführt. Wenn Adobe AIR nicht auf dem Gerät installiert ist, wird Adobe AIR automatisch von Flash Builder installiert.

Netzwerkdebugging konfigurieren

Führen Sie die folgenden Schritte nur aus, falls Sie eine Anwendung über ein Netzwerk debuggen.

Debugging über das Netzwerk vorbereiten

Führen Sie die folgenden Schritte aus, bevor Sie eine Anwendung über das Netzwerk debuggen:

1 Öffnen Sie unter Windows Port 7935 (für den Flash Player-Debugger) und Port 7 (für Echo/Ping).

Ausführliche Anweisungen finden Sie in diesem Microsoft TechNet-Artikel.

Deaktivieren Sie unter Windows Vista das Kontrollkästchen „Drahtlosnetzwerkverbindung“ unter „Windows-Firewall“ > „Einstellungen ändern“ > „Erweitert“.

2 Konfigurieren Sie auf Ihrem Gerät Drahtloseinstellungen unter „Einstellungen“ > „Drahtlos und Netzwerk.“

Primäre Netzwerkschnittstelle auswählen

Der Hostcomputer kann mit mehreren Netzwerkschnittstellen gleichzeitig verbunden sein. Sie können jedoch eine primäre Netzwerkschnittstelle für das Debugging auswählen. Dazu fügen Sie eine Hostadresse in der Android APK-Paketdatei hinzu.

1 Öffnen Sie in Flash Builder die Option „Voreinstellungen“.

2 Wählen Sie „Flash Builder“ > „Zielplattformen“.

Im Dialogfeld werden alle auf dem Hostcomputer verfügbaren Netzwerkschnittstellen aufgeführt.

3 Wählen Sie die in das Android APK-Paket einzubettende Netzwerkschnittstelle aus.

Stellen Sie sicher, dass auf dem Gerät der Zugriff auf die ausgewählte Netzwerkschnittstelle möglich ist. Wenn das Gerät beim Herstellen einer Verbindung nicht auf die ausgewählte Netzwerkschnittstelle zugreifen kann, zeigt Flash Builder ein Dialogfeld an, in dem die IP-Adresse des Hostcomputers angefordert wird.

Anwendung debuggen

1 Verbinden Sie das Gerät über einen USB-Port oder über ein Netzwerk.

2 Wählen Sie zur Konfiguration einer Startkonfiguration für das Debuggen „Ausführen“ > „Debugkonfigurationen“.

- Wählen Sie als Startmethode „Auf Gerät“.
- Wählen Sie „Über USB debuggen“ oder „Debuggen über Netzwerk“.

  Wenn Sie die Anwendung zum ersten Mal über ein Netzwerk debuggen, können Sie die Anwendung per USB auf dem Gerät installieren. Hierzu wählen Sie „Anwendung auf Gerät über USB installieren“ und schließen das Gerät über einen USB-Port am Hostrechner an.

  Wenn die Anwendung installiert wurde und Sie für nachfolgende Debuggingsitzungen nicht über USB eine Verbindung herstellen möchten, deaktivieren Sie „Anwendung auf Gerät über USB installieren“.

- (Optional) Anwendungsdaten bei jedem Start löschen

  Aktivieren Sie diese Option, wenn Sie den Anwendungsstatus bei jeder Debug-Sitzung beibehalten wollen. Diese Option wird nur dann angewendet, wenn sessionCachingEnabled in Ihrer Anwendung auf „true“ gesetzt ist.

3 Wählen Sie zum Starten einer Debug-Sitzung „Debuggen“.

Der Debugger startet und wartet auf das Starten der Anwendung. Die Debug-Sitzung beginnt, wenn der Debugger eine Verbindung zum Gerät hergestellt hat.

Wenn Sie versuchen, auf einem Gerät über das Netzwerk zu debuggen, wird in der Anwendung möglicherweise ein Dialogfeld angezeigt, in dem Sie zur Eingabe einer IP-Adresse aufgefordert werden. Dieses Dialogfeld ist ein Hinweis darauf, dass der Debugger keine Verbindung herstellen konnte. Stellen Sie sicher, dass das Gerät ordnungsgemäß mit dem Netzwerk verbunden ist und auf den Computer, auf dem Flash Builder ausgeführt wird, von diesem Netzwerk aus zugegriffen werden kann.

Hinweis: *In einem Unternehmens-, Hotel- oder sonstigen Gastnetzwerk kann das Gerät manchmal keine Verbindung zum Computer herstellen, selbst wenn sich beide im selben Netzwerk befinden.*

Wenn Sie über das Netzwerk debuggen und die Anwendung zuvor auf dem Gerät installiert wurde, beginnen Sie das Debuggen mit der Eingabe der IP-Adresse des Hostcomputers.

Der zertifizierte Adobe-Experte für Flex, Brent Arnold, hat ein Videotutorial über das Debuggen einer Anwendung über USB für ein Android-Gerät erstellt.

Verwandte Themen

Debuggen und Verpacken von Anwendungen für Geräte (engl. Video)

Anwendung auf einem Apple iOS-Gerät debuggen

Zum Debuggen einer Anwendung auf einem Apple iOS-Gerät müssen Sie das Debug-iOS-Paket (IPA-Datei) manuell auf dem iOS-Gerät bereitstellen und installieren. Die automatische Bereitstellung wird für die Apple iOS-Plattform nicht unterstützt.

Wichtig: *Führen Sie vor dem Debuggen einer Anwendung auf einem iOS-Gerät die Schritte unter „Erstellen, Debuggen oder Bereitstellen einer iOS-Anwendung vorbereiten“ auf Seite 22 aus.*

1. Schließen Sie das Apple iOS-Gerät an den Entwicklungscomputer an.
2. Starten Sie auf Ihrem iOS-Gerät iTunes.

   Hinweis: *Zum Installieren Ihrer Anwendung auf dem iOS-Gerät und zum Abrufen der Geräte-ID des iOS-Geräts ist iTunes erforderlich.*
3. Wählen Sie in Flash Builder „Ausführen“ > „Debugging konfigurieren“ aus.
4. Führen Sie im Dialogfeld „Debugging konfigurieren“ die folgenden Schritte aus:
   a. Wählen Sie die Anwendung aus, die Sie debuggen möchten.
   b. Wählen Sie als Zielplattform „Apple iOS“ aus.
   c. Wählen Sie als Startmethode „Auf Gerät“ aus.
   d. Wählen Sie eine der folgenden Verpackungsoptionen aus:

      Standard Mit dieser Methode verpacken Sie eine veröffentlichungsreife Version Ihrer Anwendung, die auf Apple iOS-Geräten ausgeführt werden kann. Die Anwendungsleistung mit dieser Methode kommt dem Paket der Endversion sehr nahe und kann an den Apple App Store übermittelt werden.

      Diese Methode zur Erstellung einer Debug-iOS-Datei (IPA-Datei) nimmt jedoch mehrere Minuten in Anspruch.

      Schnell Mit dieser Methode können Sie eine IPA-Datei schnell erstellen und anschließend die Datei auf dem Gerät ausführen und debuggen. Diese Methode ist zum Testen von Anwendungen geeignet. Diese Methode liefert kein Ergebnis in Veröffentlichungsqualität und eignet sich nicht zur Übermittlung an den Apple App Store.

   e. Klicken Sie auf „Konfigurieren“, um das entsprechende Codesignierungszertifikat, die Bereitstellungsdatei und die Paketinhalte auszuwählen.
   f. Klicken Sie auf „Netzwerkdebugging konfigurieren“, um die Netzwerkschnittstelle auszuwählen, die Sie im Debug-iOS-Paket hinzufügen möchten.

      Hinweis: *Der Hostcomputer kann mit mehreren Netzwerkschnittstellen gleichzeitig verbunden sein. Sie können jedoch eine primäre Netzwerkschnittstelle für das Debugging auswählen.*

g Klicken Sie auf „Debuggen“. Flash Builder zeigt ein Dialogfeld zur Eingabe eines Kennworts an. Geben Sie das Kennwort für das P12-Zertifikat ein.

Flash Builder generiert die Debug-IPA-Datei und speichert sie im Ordner „bin-debug.

5 Führen Sie auf Ihrem iOS-Gerät die folgenden Schritte aus:

1 (Optional) Wählen Sie in iTunes „Ablage“ > „Zur Bibliothek hinzufügen“ aus und navigieren Sie zur mobilen Provisioning-Profildatei (Dateinamenerweiterung .mobileprovision), die Sie von Apple erhalten haben.

2 (Optional) Wählen Sie in iTunes „Ablage“ > „Zur Bibliothek hinzufügen“ aus und navigieren Sie zur Debug-IPA-Datei, die Sie in Schritt 4 erstellt haben.

3 Synchronisieren Sie Ihr iOS-Gerät mit iTunes, indem Sie „Ablage“ > „Synchronisieren“ auswählen.

4 Flash Builder versucht eine Verbindung zu der in der Debug-IPA-Datei angegebenen Hostadresse herzustellen. Wenn die Anwendung keine Verbindung zur Hostadresse herstellen kann, zeigt Flash Builder ein Dialogfeld an, in dem die IP-Adresse des Hostcomputers angefordert wird.

Hinweis: *Wenn Sie den Code oder die Elemente seit dem Erstellen des letzten Debug-IPA-Pakets nicht geändert haben, überspringt Flash Builder das Verpacken und debuggt die Anwendung. Das heißt, Sie können die installierte Anwendung auf dem Gerät starten und auf „Debuggen“ klicken, um eine Verbindung zum Flash Builder-Debugger herzustellen. Auf diese Weise können Sie wiederholt debuggen, ohne die Anwendung jedes Mal zu verpacken.*

Kapitel 8: Installation auf Geräten

Anwendung auf einem Google Android-Gerät installieren

Während der Entwicklungs-, Test- und Bereitstellungsphase Ihres Projekts können Sie Ihre Anwendung direkt auf einem Gerät installieren.

Mithilfe von Flash Builder können Sie eine Anwendung direkt auf einem Android-Gerät installieren. Beim Installieren eines Pakets auf einem Gerät, auf dem Adobe AIR nicht installiert ist, wird AIR von Flash Builder automatisch installiert.

1 Schließen Sie das Google Android-Gerät an den Entwicklungscomputer an.

   Flash Builder greift auf das mit dem USB-Port des Computers verbundene Gerät zu. Vergewissern Sie sich, dass Sie die erforderlichen USB-Gerätetreiber konfiguriert haben. Siehe „Google Android-Geräte anschließen“ auf Seite 18.

2 Wählen Sie in Flash Builder „Ausführen“ > „Konfigurationen ausführen“ aus. Wählen Sie im Dialogfeld „Konfigurationen ausführen“ die Mobilanwendung aus, die Sie bereitstellen möchten.

3 Wählen Sie als Startkonfigurationsmethode „Auf Gerät“ aus.

4 (Optional) Legen Sie fest, ob die Anwendungsdaten bei jedem Start gelöscht werden sollen.

5 Klicken Sie auf „Anwenden“.

Die Anwendung wird von Flash Builder auf Ihrem Android-Gerät installiert und gestartet. Wenn Sie das Paket auf einem Gerät installiert haben, auf dem Adobe AIR nicht installiert ist, wird AIR von Flash Builder automatisch installiert.

Der zertifizierte Adobe-Experte für Flex, Brent Arnold, hat ein Videotutorial zum Einrichten und Ausführen der Anwendung auf einem Android-Gerät erstellt.

Anwendung auf einem Apple iOS-Gerät installieren

Auf einem iOS-Gerät müssen Sie eine Anwendung (IPA-Datei) manuell installieren, da die automatische Bereitstellung von der Apple iOS-Plattform nicht unterstützt wird.

Wichtig: *Bevor Sie eine Anwendung auf einem iOS-Gerät installieren, benötigen Sie das Apple iOS-Entwicklungszertifikat (im P12-Format) und eine Entwicklungsversion des Bereitstellungsprofils. Führen Sie dafür die Schritte unter „Erstellen, Debuggen oder Bereitstellen einer iOS-Anwendung vorbereiten“ auf Seite 22 aus.*

1 Schließen Sie das Apple iOS-Gerät an den Entwicklungscomputer an.

2 Starten Sie auf dem Entwicklungscomputer iTunes.

   Hinweis: *Zum Installieren Ihrer Anwendung auf dem iOS-Gerät und zum Abrufen der Geräte-ID des iOS-Geräts ist iTunes erforderlich.*

3 Wählen Sie in Flash Builder „Ausführen“ > „Konfigurationen ausführen“ aus.

4 Führen Sie im Dialogfeld „Konfigurationen ausführen“ die folgenden Schritte aus:

   a Wählen Sie die Anwendung aus, die Sie installieren möchten.

b Wählen Sie als Zielplattform „Apple iOS“ aus.

c Wählen Sie als Startmethode „Auf Gerät“ aus.

d Wählen Sie eine der folgenden Verpackungsoptionen aus:

Standard Mit dieser Methode verpacken Sie eine veröffentlichungsreife Version Ihrer Anwendung, die auf Apple iOS-Geräten ausgeführt werden kann.

Bei der Standardverpackungsmethode wird vor der Verpackung der Bytecode der SWF-Datei der Anwendung in ARM-Anweisungen konvertiert. Aufgrund dieses zusätzlichen Konvertierungsschritts vor der Verpackung nimmt diese Methode zum Erstellen einer (IPA-)Anwendungsdatei mehrere Minuten im Anspruch. Die Standardmethode dauert länger als die schnelle Methode. Anwendungen mit dieser Standardmethode sind dennoch veröffentlichungsreif und eignen sich zur Übermittlung an den Apple App Store.

Schnell Mit dieser Methode können Sie eine IPA-Datei schnell erstellen.

Bei der schnellen Verpackungsmethode wird die Konvertierung des Bytecodes übersprungen und es werden lediglich die SWF-Anwendungsdatei und die Anwendungselemente mit der vorkompilierten AIR-Laufzeitumgebung zusammengefasst. Die schnelle Verpackungsmethode ist schneller als die Standardmethode. Anwendungen mit der Standardmethode sind jedoch nicht veröffentlichungsreif und eignen sich nicht zur Übermittlung an den Apple App Store.

***Hinweis:** Zwischen der Standard- und der schnellen Verpackungsmethode gibt es keine Unterschiede in der Laufzeitumgebung oder in den Funktionen.*

e Klicken Sie auf „Konfigurieren“, um das entsprechende Codesignierungszertifikat, die Bereitstellungsdatei und die Paketinhalte auszuwählen.

f Klicken Sie auf „Ausführen“. Flash Builder zeigt ein Dialogfeld zur Eingabe eines Kennworts an. Geben Sie das Kennwort für das P12-Zertifikat ein.

Flash Builder generiert die IPA-Datei und speichert sie im Ordner „bin-debug“.

5 Gehen Sie auf Ihrem Entwicklungscomputer wie folgt vor:

1 Wählen Sie in iTunes „Ablage“ > „Zur Bibliothek hinzufügen“ aus und navigieren Sie zur mobilen Provisioning-Profildatei (Dateinamenerweiterung .mobileprovision), die Sie von Apple erhalten haben.

Sie können außerdem die Drag & Drop-Funktion verwenden, um die mobile Provisioning-Profildatei in iTunes zu verschieben.

2 (Optional) Wählen Sie in iTunes „Ablage“ > „Zur Bibliothek hinzufügen“ aus und navigieren Sie zur IPA-Datei, die Sie in Schritt 4 erstellt haben.

Des Weiteren können Sie die Drag & Drop-Funktion verwenden, um die IPA-Datei in iTunes zu verschieben.

3 Synchronisieren Sie Ihr iOS-Gerät mit iTunes, indem Sie „Ablage“ > „Synchronisieren“ auswählen.

Die Anwendung wird auf Ihrem iOS-Gerät bereitgestellt und kann gestartet werden.

Kapitel 9: Verpacken und Exportieren

Mobilanwendung verpacken und in einen Onlineshop exportieren

Mit der Funktion „Releasebuild exportieren“ von Flash Builder verpacken und exportieren Sie den Releasebuild einer Mobilanwendung. Ein Releasebuild ist im Allgemeinen die endgültige Version der Anwendung, die Sie in einen Onlineshop wie Android Market, Amazon App Store oder Apple App Store hochladen möchten.

Beim Exportieren einer Anwendung können Sie diese auch auf einem Gerät installieren. Wenn das Gerät während des Exports an den Computer angeschlossen ist, wird die Anwendung von Flash Builder auf dem Gerät installiert. Außerdem kann ein plattformspezifisches Anwendungspaket zur späteren Installation auf einem Gerät exportiert werden. Das erstellte Paket kann wie eine native Anwendung bereitgestellt und installiert werden.

Ausführliche Informationen zum Exportieren einer Android-Anwendung in den Android Market oder Amazon App Store finden Sie unter „Android APK-Pakete für die Veröffentlichung exportieren“ auf Seite 195.

Ausführliche Informationen zum Exportieren einer iOS-Anwendung in den Apple App Store finden Sie unter „Apple iOS-Pakete für die Veröffentlichung exportieren“ auf Seite 196.

Anwendung mit gekoppelter Laufzeitumgebung exportieren

Falls Sie eine Mobilanwendung mithilfe der Funktion zum Exportieren von Releasebuilds exportieren, können Sie die AIR-Laufzeitumgebung in das Anwendungspaket einbetten.

Benutzer können die Anwendung dann sogar auf einem Gerät ausführen, auf dem AIR noch nicht installiert ist. Je nachdem, auf welche Plattform Sie das Paket exportieren, können Sie eine gekoppelte Laufzeitumgebung oder eine freigegebene Laufzeitumgebung verwenden.

Android APK-Pakete für die Veröffentlichung exportieren

Vor dem Export einer Mobilanwendung können Sie die Android-Berechtigungen anpassen. Passen Sie die Einstellungen manuell in der Anwendungsbeschreibungsdatei an. Diese Einstellungen befinden sich im Block `<android>` der Datei „bin-debug/*app_name*-app.xml“. Weitere Informationen finden Sie unter Setting AIR application properties.

Wenn die Anwendung für die spätere Installation auf einem Gerät exportiert werden soll, installieren Sie das Anwendungspaket mit den Tools des Betriebssystemanbieters des betreffenden Geräts.

1. Wählen Sie in Flash Builder „Projekt“ > „Releasebuild exportieren“.
2. Wählen Sie das Projekt und die Anwendung aus, die Sie exportieren möchten.
3. Wählen Sie die Zielplattformen und den Speicherort zum Exportieren des Projekts aus.
4. Exportieren und signieren Sie ein plattformspezifisches Anwendungspaket.

   Sie können Ihre Anwendung mit einer digitalen Signatur für jede Zielplattform oder als digital signierte AIR-Anwendung für den Desktop verpacken.

Darüber hinaus können Sie die Anwendung als AIRI-Zwischendatei exportieren, die später signiert werden kann. Wenn Sie diese Option auswählen, verpacken Sie die AIRI-Datei mithilfe des AIR-Befehlszeilenprogramms adt als APK-Datei. Installieren Sie anschließend mithilfe plattformspezifischer Tools (z. B. mit adb für das Android SDK) die APK-Datei auf dem Gerät. Informationen zu Befehlszeilenprogrammen zum Verpacken von Anwendungen finden Sie unter „Mobilanwendungen über die Befehlszeile entwickeln und bereitstellen“ auf Seite 197.

5 Auf der Seite „Verpackungseinstellungen“ können Sie das digitale Zertifikat, den Paketinhalt und native Erweiterungen auswählen.

Bereitstellung Soll die Anwendung außerdem auf einem Gerät installiert werden, klicken Sie auf die Seite „Bereitstellung“ und wählen Sie „Anwendung auf angeschlossenen Geräten installieren und starten“ aus. Stellen Sie sicher, dass Sie ein oder mehr Geräte an die USB-Ports Ihres Computers angeschlossen haben.

- **Anwendung mit gekoppelter Laufzeitumgebung exportieren**

  Wählen Sie diese Option, wenn die AIR-Laufzeit beim Export des Anwendungspakets in der APK-Datei eingebettet werden soll. Benutzer können die Anwendung dann sogar auf einem Gerät ausführen, auf dem AIR noch nicht installiert ist.

- **Anwendung mit freigegebener Laufzeit exportieren**

  Wählen Sie diese Option, wenn die AIR-Laufzeit beim Export des Anwendungspakets nicht in der APK-Datei eingebettet werden soll. Sie können eine URL auswählen oder angeben, um Adobe AIR für das Anwendungspaket herunterzuladen, falls AIR noch nicht auf einem Benutzergerät installiert ist.

  Die Standard-URL führt zum Android Market. Sie können die Standard-URL jedoch überschreiben und die URL auswählen, die auf eine Seite im Amazon App Store verweist, oder Sie geben Ihre eigene URL ein.

Digitale Signatur Klicken Sie auf die Registerkarte „Digitale Signatur“, um ein digitales Zertifikat zu erstellen oder zu suchen, das die Identität des Anwendungsentwicklers darstellt. Sie können für das Zertifikat auch ein Kennwort angeben.

Wenn Sie ein Zertifikat erstellen, handelt es sich um ein *selbst signiertes* Zertifikat. Ein kommerziell signiertes Zertifikat kann von einem Zertifikatsanbieter erworben werden. Siehe Digitally sign your AIR applications.

Paketinhalte (Optional) Klicken Sie auf die Registerkarte „Inhalt verpacken“, um auszuwählen, welche Dateien in das Paket einbezogen werden sollen.

Native Erweiterungen (Optional) Wählen Sie die nativen Erweiterungen aus, die dem Anwendungspaket hinzugefügt werden sollen.

Weitere Informationen zu nativen Erweiterungen finden Sie unter „Verwenden nativer Erweiterungen“ auf Seite 13.

6 Klicken Sie auf „Fertig stellen“.

Flash Builder erstellt die Datei `Anwendungsname.apk` im Verzeichnis, das im ersten Bedienfeld (standardmäßig die oberste Ebene in Ihrem Projekt) angegeben wurde. Wenn das Gerät während des Exports an den Computer angeschlossen wurde, wird die Anwendung von Flash Builder auf dem Gerät installiert.

Apple iOS-Pakete für die Veröffentlichung exportieren

Sie können ein iOS-Paket zur sofortigen Verteilung oder zur Übermittlung an den Apple App Store erstellen und exportieren.

Wichtig: *Stellen Sie vor dem Exportieren eines iOS-Pakets sicher, dass Sie über die erforderlichen Zertifikate und ein Distributions-Bereitstellungsprofil von Apple verfügen. Führen Sie dafür die Schritte unter „Erstellen, Debuggen oder Bereitstellen einer iOS-Anwendung vorbereiten“ auf Seite 22 aus.*

1 Wählen Sie in Flash Builder „Projekt“ > „Releasebuild exportieren“.

2 Wählen Sie Apple iOS als Zielplattform für den Export und signieren Sie ein IPA-Paket.

Klicken Sie auf „Weiter“.

3 Wählen Sie das P12-Zertifikat und das Distributions-Bereitstellungsprofil von Apple aus.

4 Auf der Seite „Verpackungseinstellungen“ können Sie das Bereitstellungszertifikat, das digitale Zertifikat, den Paketinhalt und native Erweiterungen auswählen.

Bereitstellung Beim Exportieren eines iOS-Pakets wird die AIR-Laufzeit standardmäßig in die IPA-Datei eingebettet.

Digitale Signatur Wählen Sie das P12-Zertifikat und das Distributions-Bereitstellungsprofil von Apple aus.

Sie können einen der folgenden Verpackungstypen auswählen:

- **Adhoc Distribution For Limited Distribution** für eine begrenzte Verteilung der Anwendung
- **Final Release Package For Apple App Store** zur Übermittlung der Anwendung an den Apple App Store

Paketinhalte (Optional) Klicken Sie auf die Registerkarte „Inhalt verpacken“, um auszuwählen, welche Dateien in das Paket einbezogen werden sollen.

Native Erweiterungen (Optional) Wählen Sie die nativen Erweiterungen aus, die dem Anwendungspaket hinzugefügt werden sollen.

Verwendet die native Erweiterung iOS5 SDK-Funktionen, wählen Sie den Speicherort des iOS SDK. Weitere Informationen finden Sie unter „Unterstützung für native iOS5-Erweiterungen“ auf Seite 14.

5 Klicken Sie auf „Fertig stellen“.

Flash Builder überprüft die Konfiguration der Paketeinstellungen und kompiliert anschließend die Anwendung. Nach Abschluss des Verpackens können Sie die IPA-Datei auf einem verbundenen Apple iOS-Gerät installieren oder an den Apple App Store senden.

Informationen zum Verpacken der IPA-Datei mit dem AIR Developer Tool (ADT) finden Sie im Abschnitt iOS-Pakete unter *Erstellen von AIR-Anwendungen*.

Mobilanwendungen über die Befehlszeile entwickeln und bereitstellen

Mobilanwendungen können auch ohne Flash Builder erstellt werden. Sie verwenden stattdessen die Befehlszeilenprogramme mxmlc, adl und adt.

Die allgemeine Vorgehensweise zum Entwickeln und Installieren einer Mobilanwendung auf einem Gerät mithilfe von Befehlszeilenprogrammen wird im Folgenden beschrieben. Die einzelnen Schritte werden später ausführlicher erläutert:

1 Die Anwendung wird mit dem Programm mxmlc kompiliert.

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

Hierzu muss dem Parameter `configname` der Wert „airmobile“ übergeben werden.

2 Testen Sie die Anwendung mit dem adl-Programm im AIR Debug Launcher (ADL).

```
adl MyMobileApp-app.xml -profile mobileDevice
```

Es muss eine Deskriptordatei für die Anwendung erstellt und als Argument an das adl-Programm übergeben werden. Außerdem geben Sie das `mobileDevice`-Profil an.

3 Verpacken Sie die Anwendung mit dem adt-Programm.

```
adt -package -target apk SIGN_OPTIONS MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

Hierfür muss zunächst ein Zertifikat erstellt werden.

4 Installieren Sie die Anwendung auf dem Mobilgerät. Zum Installieren der Anwendung auf einem Android-Gerät verwenden Sie das adb-Programm.

```
adb install -r MyMobileApp.apk
```

Dazu muss das Mobilgerät an den USB-Port des Computers angeschlossen werden.

5 Stellen Sie die Mobilanwendung in Onlineshops bereit.

Mobilanwendung mit mxmlc kompilieren

Mobilanwendungen können mit dem Befehlszeilencompiler mxmlc kompiliert werden. Übergeben Sie für die Verwendung von mxmlc dem Parameter `configname` den Wert `airmobile`. Beispiel:

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

Durch `+configname=airmobile` weisen Sie den Compiler an, die Datei „airmobile-config.xml“ zu verwenden. Die Datei befindet sich im Verzeichnis sdk/frameworks. Von dieser Datei werden die folgenden Aufgaben ausgeführt:

- Die übernimmt das Design mobile.swc.
- Sie ändert den Bibliothekspfad wie folgt:
  - libs/air wird aus dem Pfad entfernt. Mobilanwendungen verstehen nicht die Klassen Window und WindowedApplication.
  - libs/mx wird aus dem Bibliothekspfad entfernt. Abgesehen von Diagrammen unterstützen Mobilanwendungen keine MX-Komponenten.
  - libs/mobile wird dem Bibliothekspfad hinzugefügt.
- Sie entfernt die Namensräume ns.adobe.com/flex/mx und www.adobe.com/2006/mxml. Abgesehen von Diagrammen unterstützen Mobilanwendungen keine MX-Komponenten.
- Sie deaktiviert die Barrierefreiheit.
- Sie entfernt RSL-Einträge, da sie von Mobilanwendungen nicht unterstützt werden.

Der mxmlc-Compiler erzeugt eine SWF-Datei.

Mobilanwendung mit adl testen

Eine Mobilanwendung kann mit AIR Debug Launcher (ADL) getestet werden. Verwenden Sie ADL zum Ausführen und Testen einer Anwendung, ohne sie vorher verpacken und auf einem Gerät installieren zu müssen.

Mit dem adl-Programm debuggen

ADL fügt der Standardausgabe Trace-Anweisungen und Laufzeitfehler hinzu, unterstützt jedoch keine Haltepunkte oder andere Debugging-Funktionen. Sie können eine integrierte Entwicklungsumgebung wie beispielsweise Flash Builder für komplexe Debugging-Probleme verwenden.

adl-Programm starten

Um das adl-Programm über die Befehlszeile zu starten, übergeben Sie die Deskriptordatei Ihrer Mobilanwendung und legen den Parameter `profile` auf `mobileDevice` fest. Beispiel:

```
adl MyMobileApp-app.xml -profile mobileDevice
```

Im `mobileDevice`-Profil werden eine Reihe von Funktionen für Anwendungen auf Mobilgeräten definiert. Genaue Informationen zum `mobileDevice`-Profil finden Sie unter Funktionen verschiedener Profile.

Anwendungsbeschreibungsdatei erstellen

Wenn Sie Ihre Anwendung nicht mit Flash Builder kompiliert haben, erstellen Sie die Anwendungsbeschreibungsdatei manuell. Als Grundlage können Sie die Datei /sdk/samples/descriptor-sample.xml verwenden. Im Allgemeinen nehmen Sie mindestens die folgenden Änderungen vor:

- Das Element `<initialWindow><content>` muss auf den Namen der SWF-Datei Ihrer Mobilanwendung verweisen:

```
<initialWindow>
    <content>MyMobileApp.swf</content>
    ...
</initialWindow>
```

- Ändern Sie den Titel der Anwendung. Dieser wird auf dem Mobilgerät unter dem Anwendungssymbol angezeigt. Den Titel geben Sie im Element `<name><text>` ein:

```
<name>
    <text xml:lang="en">MyMobileApp by Nick Danger</text>
</name>
```

- Fügen Sie einen `<android>`-Block ein, um Android-spezifische Berechtigungen für die Anwendung zu setzen. Je nach den von Ihrem Gerät verwendeten Diensten reicht meist die folgende Berechtigung aus:

```
<application>
    ...
    <android>
        <manifestAdditions>
            <![CDATA[<manifest>
                <uses-permission android:name="android.permission.INTERNET"/>
            </manifest>]]>
        </manifestAdditions>
    </android>
</application>
```

Sie können mithilfe der Beschreibungsdatei auch die Höhe und Breite der Anwendung, den Speicherort der Symboldateien, Versionsinformationen und andere Details zum Installationspfad festlegen.

Weitere Informationen zum Erstellen und Bearbeiten der Deskriptordateien für Anwendungen finden Sie unter Deskriptordateien für Adobe AIR-Anwendung.

Mobilanwendung mit adt verpacken

Mit dem AIR Developer Tool (ADT) verpacken Sie Mobilanwendungen über die Befehlszeile. Mit dem Programm adt kann eine APK-Datei zur Bereitstellung auf einem Android-Gerät erstellt werden.

Zertifikat erstellen

Vor der Erstellung einer APK-Datei muss ein Zertifikat erstellt werden. Zu Entwicklungszwecken reicht ein selbst signiertes Zertifikat aus. Dieses können Sie mit dem adt-Programm erstellen. Beispiel:

```
adt -certificate -cn SelfSign -ou QE -o "Example" -c US 2048-RSA newcert.p12 password
```

Das adt-Programm erstellt die neueste newcert.p12-Datei im aktuellen Verzeichnis. Dieses Zertifikat übergeben Sie beim Verpacken der Anwendung an adt. Verwenden Sie keine selbst signierten Zertifikate für Endversionen der Anwendung. Sie bieten den Benutzern nur begrenzte Sicherheit. Informationen zum Signieren der AIR-Installationsdateien mit einem von einer vertrauenswürdigen Zertifizierungsstelle ausgestellten Zertifikat finden Sie unter Signing AIR applications.

Paketdatei erstellen

Zum Erstellen der APK-Datei für Android übergeben Sie dem adt-Programm die Anwendungsdetails, u. a. das Zertifikat. Hier ein Beispiel:

```
adt -package -target apk -storetype pkcs12 -keystore newcert.p12 -keypass password
MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

Das adt-Programm erzeugt eine Datei namens „*Anwendungsname*.apk“.

Für iOS verpacken

Zum Verpacken von Mobilanwendungen für iOS benötigen Sie ein Entwicklerzertifikat von Apple sowie eine Bereitstellungsdatei. Zu diesem Zweck müssen Sie dem Entwicklerprogramm von Apple beitreten. Weitere Informationen finden Sie unter „Erstellen, Debuggen oder Bereitstellen einer iOS-Anwendung vorbereiten“ auf Seite 22.

Der Flex-Guru Piotr Walczyszyn erläutert das Verpacken einer AIR-Anwendung für iOS-Geräte mithilfe des ADT-Befehls und des ANT-Skripts.

Der Blogger Valentin Simonov liefert zusätzliche Informationen zum Veröffentlichen Ihrer Anwendung für iOS.

Eine Mobilanwendung auf einem Gerät mit adb installieren

Installieren Sie mit Android Debug Bridge (adb) die Anwendung (APK-Datei) auf einem Android-Mobilgerät. Das Programm adb ist eine Komponente des Android SDK.

Gerät an einen Computer anschließen

Bevor Sie adb zum Installieren der APK-Datei auf dem Mobilgerät ausführen können, muss das Gerät an den Computer angeschlossen werden. Unter Windows und Linux sind dazu entsprechende USB-Treiber erforderlich.

Informationen zum Installieren von USB-Treibern für Ihr Gerät finden Sie unter Using Hardware Devices.

Anwendung auf einem angeschlossenen Gerät installieren

Wenn Sie das Gerät an den Computer angeschlossen haben, können Sie die Anwendung auf dem Gerät installieren. Verwenden Sie die Option `install` und übergeben Sie den Namen der APK-Datei. Ein Beispiel illustriert dies:

```
adb install -r MyMobileApp.apk
```

Mit der Option `-r` überschreiben Sie eine zuvor installierte Anwendung. Wenn Sie diese Option nicht verwenden, müssen Sie die vorhandene Anwendung vor jeder Installation einer neueren Version auf dem Mobilgerät deinstallieren.

Verwandte Themen

Android Debug Bridge

Anwendung in Onlineshops bereitstellen

Sie können Ihre Anwendung in Online-App-Shops wie Android Market, Amazon App Store oder Apple App Store bereitstellen.


Lee Brimlow zeigt, wie eine neue AIR für Android-Anwendung für den Android-Markt bereitgestellt wird.


Christian Cantrell erläutert, wie die Anwendung im Amazon App Store für Android bereitgestellt wird.