Разработка мобильных приложений в ADOBE® FLEX® 4.6 и ADOBE® FLASH® BUILDER™ 4.6

Юридическая информация

Последнее обновление 6.12.2011.

Содержание

Глава 1. Начало работы

Начало работы с мобильными приложениями

Выпуск Adobe Flex предоставляет возможность использования Flex и Adobe Flash Builder на смартфонах и планшетах. С помощью Adobe AIR можно разрабатывать мобильные приложения в Flex так же легко, как и на настольных платформах.

Функциональность многих существующих компонентов Flex расширена с учетом использования на мобильных устройствах, в том числе добавлена поддержка прокрутки посредством касаний. Flex также содержит набор новых компонентов для облегчения создания приложений с учетом стандартных шаблонов дизайна для телефонов и планшетов.

Также в Flash Builder добавлены новые возможности поддержки разработки приложений для мобильных устройств. В Flash Builder можно разрабатывать, тестировать и отлаживать приложения на настольном компьютере или непосредственно на мобильном устройстве.

В видеоролике специалиста по продукции Adobe Mark Doherty рассматривается создание приложений для настольных компьютеров, мобильных телефонов и планшетных устройств.

В видеоролике специалиста по продукции Adobe James Ward рассматривается создание мобильных приложений с помощью Flex.

Специалист Adobe Community Joseph Labrecque делится своими впечатлениями о Mobile Flex Demonstration.

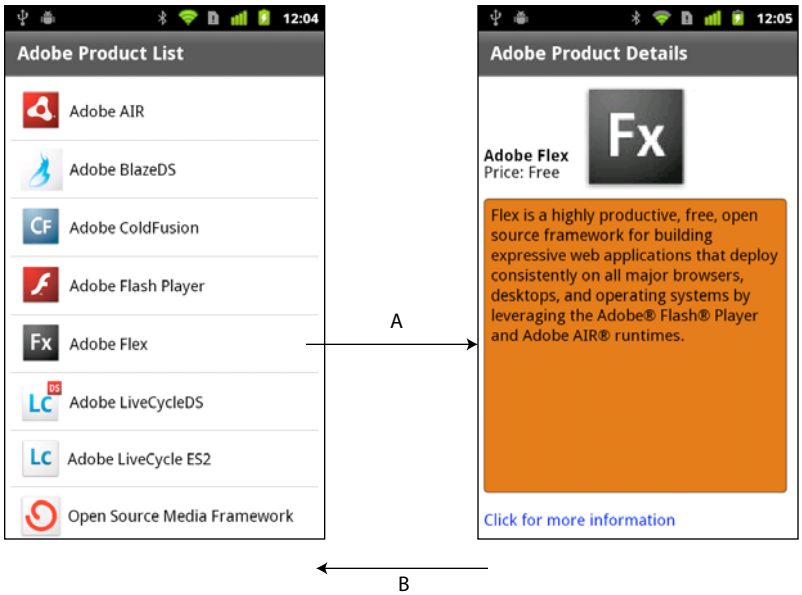
Разработчик Flash Fabio Biondi рассказывает о создании YouTube Player для устройств Android на основе AIR в программе Flash Builder.

Разработка мобильного приложения

В мобильных устройствах размер экрана невелик, поэтому в разработке мобильных приложений используются шаблоны проектирования, отличающиеся от шаблонов приложений для браузера. Разработка мобильного приложения обычно заключается в том, что содержимое разделяется на последовательность представлений, отображаемых на мобильном устройстве.

Каждое представление содержит компоненты, предназначенные для одной задачи или содержащие один набор данных. Пользователь обычно переходит от вышестоящего к нижестоящему представлению, прикасаясь к компонентам на экране. Пользователь может нажать кнопку возврата для перехода к предыдущему представлению или встроить функции навигации в приложение.

В следующем примере первоначальное представление приложения отображает список продуктов:



A. Выберите элемент списка, чтобы изменить представления в приложении. B. Нажмите кнопку возврата для перехода к предыдущему представлению.

Пользователь выбирает продукт в списке для получения подробной информации. После выбора элемента представление изменяется и отображает подробную информацию о продукте.

Если приложение разрабатывается как для мобильной платформы, так и для веб- или настольных платформ, обычно создаются отдельные пользовательские интерфейсы для этих платформ. При этом приложения могут использовать общую базовую модель и основной код доступа для всех платформ.

Разработка приложений для телефонов и планшетов

При разработке приложений для планшетных устройств ограничения размера экрана не столь важны, как в телефонах. Поэтому нет необходимости учитывать малый размер экрана в структуре приложений для планшетов. В разрабатываемых приложениях можно использовать стандартный контейнер Spark Application с поддерживаемыми мобильными компонентами и темами оформления.

Примечание. *Приложение для мобильного телефона можно создать с использованием контейнера Spark Application. Однако обычно в мобильных приложениях используются контейнеры ViewNavigatorApplication и TabbedViewNavigatorApplication.*

Процесс создания мобильных проектов в Flash Builder аналогичен для планшетных и мобильных устройств. Как в планшетном приложении, так и в приложении для телефона используется аналогичная мобильная тема с компонентами и темами оформления Flex, оптимизированными для мобильных устройств.

Разработка мобильных приложений в Flash Builder

В Flash Builder можно создавать, компоновать и отлаживать приложения для мобильных устройств. Добавление мобильных возможностей в Flash Builder нацелено на упрощение разработки мобильных приложений на основе ActionScript или Flex аналогично созданию веб- и настольных приложений.

Flash Builder предлагает два способа тестирования и отладки. Приложение можно запустить и отладить на настольном компьютере с помощью инструмента AIR Debug Launcher (ADL). Расширенное управление возможно при непосредственном запуске и отладке приложения на мобильном устройстве. Независимо от выбранного способа, пользователю доступны такие возможности отладки в Flash Builder, как установка точек прерывания и проверка состояния приложения с помощью панелей «Переменные» и «Выражения».

Когда приложение готово для развертывания, используется процесс экспорта сборки выпуска, который аналогичен процессу для настольный и веб-приложений. Основным отличием является то, что при экспорте сборки выпуска мобильного проекта Flash Builder упаковывает сборку как собственную программу установки, а не как файл .air. Например, для платформы Android Flash Builder создает файл .apk, который аналогичен собственному пакету приложений для Android. Собственная программа установки позволяет распространять приложения на основе AIR аналогично тому, как распространяются собственные приложения для каждой платформы.

Развертывание мобильных приложений в AIR

Мобильные приложения, созданные в Flex, развертываются в Adobe AIR для мобильных устройств. Любое устройство, на котором необходимо развернуть мобильное приложение, должно поддерживать AIR.

Интеграция AIR с мобильной платформой поможет значительно расширить функциональные возможности приложений. Например, в мобильном приложении можно использовать аппаратную кнопку возврата и меню или доступ к локальному хранилищу. Также в приложении доступны все функции AIR для мобильных устройств, например геопозиционирование, измерение ускорения и интегрированная камера.

На мобильном устройстве необязательно устанавливать AIR перед выполнением приложения, созданного в Flex. При первом запуске приложения, созданного в Flex, пользователю предлагается установить AIR.

Для получения подробной информации о возможностях AIR см. перечисленные ниже темы.

- Знакомство с Adobe AIR
- Запуск и завершение приложения AIR
- Работа с информацией среды выполнения AIR и операционной системы
- Работа с собственными окнами AIR
- Работа с локальными базами данных SQL в AIR

При разработке мобильных приложений не допускается использование компонентов Flex для AIR WindowedApplication и Window. Вместо этих компонентов используются контейнеры ViewNavigatorApplication и TabbedViewNavigatorApplication. При разработке мобильных приложений для планшетов также используется контейнер Spark Application.

Подробную информацию см. в разделах Использование компонентов Flex AIR и «Определение мобильного приложения и заставки экрана» на странице 37.

Использование мобильной темы в приложении

Тема определяет внешний вид визуальных компонентов приложения, таких как цветовая схема, общий шрифт приложения или изменение внешнего вида всех компонентов, используемых в приложении.

Стили CSS можно установить для компонентов Flex только в том случае, если они содержатся в текущей теме. Для получения информации о том, поддерживается ли свойство стиля CSS в используемой теме, см. данные о стиле в документе Справочник ActionScript 3.0 для платформы Adobe Flash Platform.

Flex поддерживает три основных темы: Mobile, Spark и Halo. Мобильная тема определяет стандартный внешний вид компонентов Flex при создании мобильного приложения. Чтобы обеспечить совместимость компонентов Flex и мобильной темы, компания Adobe разработала новые темы оформления для компонентов. Поэтому некоторые компоненты содержат темы оформления, специфичные для темы.

Приложения, созданные в Flex, могут использоваться на многочисленных мобильных устройствах с различным размером и разрешением экрана. Процесс создания приложений, независимых от разрешения, упрощается с помощью Flex, поскольку эта программа предоставляет темы оформления, независимые от DPI, для мобильных компонентов. Для получения подробной информации о мобильных темах оформления см. раздел «Основы создания мобильных тем оформления» на странице 179.

Для получения подробной информации о стилях и темах см. в разделах Стили и темы и «Мобильные стили» на странице 180.

Ресурсы сообщества

Ниже представлены описания новых возможностей в Flex и Flash Builder

- Introducing Adobe Flex 4.5 SDK, автор: Deepa Subramaniam, менеджер по продукции Adobe.
- Mobile development using Adobe Flex SDK and Flash Builder , автор: дизайнер продуктов Adobe Narciso Jaramillo.
- What's new in Flex 4.6 SDK, автор: менеджер по продукции Adobe Jacob Surber и What's New in Flash Builder 4.6, автор: менеджер по продукции Adobe Adam Lehman.

Портал Центр разработчиков Flex содержит разнообразные ресурсы, предназначенные для упрощения процесса создания мобильных приложений в Flex:

- статьи, ссылки и руководства по началу работы;
- образцы действительных приложений, созданных в Flex;
- ресурс Flex Cookbook, содержащий ответы на наиболее распространенные вопросы о неполадках при создании кода;
- ссылки на сайт сообщества Flex и другие сайты, предлагающие информацию о Flex.

Ресурс Adobe TV содержит видеоролики, созданные инженерами, специалистами и клиентами Adobe, о разработке приложений в Flex. Одним из таких роликов является Build your first mobile application in Flash Builder 4.5.

Различия в разработке мобильных и настольных приложений, а также приложений для браузера

В Flex разрабатываются приложения для следующих сред развертывания:

Браузер: приложение развертывается как файл SWF для использования в Flash Player браузера.

Настольный компьютер: автономное приложение AIR развертывается на настольном компьютере, например компьютере с установленной ОС Windows или Macintosh.

Мобильная тема: автономное приложение AIR развертывается на мобильном устройстве, например телефоне или планшете.

Flash Player и AIR используют подобные среды выполнения. Большинство аналогичных операций может выполняться в любой из этих сред. Помимо развертывания автономных приложений вне браузера, AIR обеспечивает тесную интеграцию с платформой хоста. Посредством этой интеграции доступны такие функции, как доступ к файловой системе устройства, возможность создавать и использовать локальные базы данных SQL и другие возможности.

Рекомендации по дизайну и разработке мобильных приложений

Приложения для мобильных устройств с сенсорным управлением отличаются от приложений для настольного компьютера или браузера:

- Мобильные компоненты должны обеспечить легкое управление сенсорным вводом и поэтому размер областей взаимодействия в них гораздо больше, чем размер областей в приложениях для браузера или настольного компьютера.
- Также различаются способы взаимодействия, например прокрутка.
- Поскольку размер экрана таких устройств невелик, в создаваемых мобильных приложениях обычно отображается лишь малая часть пользовательского интерфейса.
- В структуре пользовательского интерфейса необходимо учесть различия в разрешениях экрана для различных устройств.
- В отличие от настольных устройств на телефонах и планшетах ограничена производительность центрального и графического процессоров.
- На мобильных устройствах также ограничен объем памяти, поэтому приложения должны обеспечивать экономное использование памяти.
- Мобильные приложения можно закрыть или перезапустить в любое время, например при получении входящего звонка или текстового сообщения.

Поэтому создание мобильного приложения не является простым изменением настольного приложения для отображения на экране устройства с другим размером. В Flex можно создавать отдельные пользовательские интерфейсы для любых параметров формы, при этом базовая модель и код доступа к данным будут общими для мобильных и настольных приложений, а также приложений для браузера.

Ограничения использования компонентов Spark и MX в мобильных приложениях

При создании мобильных приложений Flex используйте набор компонентов Spark. Эти компоненты определены в пакетах spark.components.*. Однако мобильные приложения не поддерживают полный набор компонентов Spark, что связано с обеспечением производительности или с тем, что не все компоненты Spark имеют темы оформления для мобильной темы.

Помимо элементов управления диаграмм MX и MX Spacer, мобильные приложения не поддерживают набор компонентов MX, определенный в пакетах mx.*.

В таблице ниже перечислены компоненты, которые допустимы или недопустимы для использования, а также компоненты, которые следует использовать с осторожностью в мобильных приложениях.

Компонент	**Компонент**	**Использование в мобильном приложении**	**Примечания**
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	Да	Эти новые компоненты поддерживают мобильные приложения.
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	Да	Многие из этих компонентов используют темы оформления для мобильной темы. Элементы Label, Image и BitmapImage также могут применяться, хотя для них не определена мобильная тема оформления. У некоторых контейнеров макета Spark, например Group и его подклассов, отсутствуют темы оформления, поэтому их можно использовать в мобильном приложении.
Другие компоненты Spark с изменяемым оформлением		Не рекомендуется	Не рекомендуется использовать компоненты Spark с изменяемым оформлением, за исключением перечисленных выше, поскольку в них отсутствует тема оформления для мобильной темы. Если в компоненте не определена тема оформления для мобильной темы, пользователь может создать ее для своего приложения.
Spark DataGrid	Spark RichEditableText Spark RichText	Не рекомендуется	Эти компоненты не рекомендуется использовать для обеспечения высокой производительности. Использование этих компонентов в мобильном приложении может снизить его производительность. Производительность элемента управления DataGrid основана на количестве обрабатываемых данных. Производительность элементов управления RichEditableText и RichText зависит от объема текста и количества элементов управления в приложении.

Компонент	**Компонент**	**Использование в мобильном приложении**	**Примечания**
Компоненты MX, кроме Spacer и диаграмм		Нет	Мобильные приложения не поддерживают такие компоненты MX, как MX Button, CheckBox, List или DataGrid. Эти компоненты соответствуют компонентам Flex 3 в пакетах mx.controls.* и mx.containers.*.
MX Spacer		Да	Spacer не применяет тему оформления и поэтому может использоваться в мобильном приложении.
Компоненты диаграмм MX		Да, но с ограничением производительности	В мобильном приложении могут использоваться элементы управления диаграмм MX, например AreaChart и BarChart. Эти элементы управления диаграмм MX определены в пакетах mx.charts.*. При этом производительность мобильного устройства может быть снижена в зависимости от размера и типа данных, используемых в диаграммах. В Flash Builder компоненты MX не включены по умолчанию в путь библиотеки мобильных проектов. Чтобы использовать компоненты диаграмм MX в приложении, добавьте mx.swc и charts.swc в путь библиотеки.

Некоторые функции Flex не поддерживаются в мобильных приложениях.

- Операции перетаскивания не поддерживаются.
- Элемент управления ToolTip не поддерживается.
- RSL не поддерживается.

Рекомендации по повышению производительности мобильных приложений

Производительность мобильных устройств обусловлена существующими ограничениями, поэтому некоторые функции мобильных приложений разрабатываются иначе, чем в приложениях для браузера или рабочего стола. Ниже перечислены рекомендации по повышению производительности.

- **Создание средств визуализации элементов в ActionScript**

  При прокрутке списка в мобильных приложениях необходимо обеспечить максимальную производительность. Для этого следует создавать средства визуализации элементов в ActionScript. При создании средств визуализации элементов в MXML производительность приложения может быть снижена.

  Flex предоставляет два средства визуализации элементов, оптимизированных для использования в мобильном приложении: spark.components.LabelItemRenderer и spark.components.IconItemRenderer. Для получения подробной информации об этих средствах визуализации элементов см. раздел Использование мобильного средства визуализации с элементом управления Spark на основе списка.

  Подробную информацию о создании пользовательских средств визуализации элементов в ActionScript см. в разделе Пользовательские средства визуализации элементов Spark. Для получения подробной информации о различиях между мобильными и настольными средствами визуализации элементов см. раздел Различия между мобильными и настольными средствами визуализации элементов.

- **Использование ActionScript и скомпилированных графических и растровых объектов FXG для создания пользовательских тем оформления**

  Мобильные темы оформления, поставляемые с Flex, созданы в ActionScript с помощью скомпилированных графических объектов FXG с целью обеспечения максимальной производительности. При создании тем оформления в MXML производительность приложения может быть снижена в зависимости от количества компонентов, использующих темы оформления MXML. Для максимальной производительности рекомендуется создавать темы оформления в ActionScript и использовать скомпилированные графические объекты FXG. Для получения подробной информации см. разделы Создание тем оформления Spark и Графические объекты FXG и MXML.

- **Использование компонентов текстового ввода на основе StageText**

  При добавлении компонентов текстового ввода, таких как TextInput и TextArea, используйте настройки по умолчанию. Эти элементы управления используют StageText в качестве основного механизма ввода текста, в котором применяются собственные классы текстового ввода. Это позволяет повысить производительность и использовать такие собственные функции, как автоматическое исправление, автоматический верхний регистр, ограничение текста и пользовательские типы виртуальной клавиатуры.

  Среди недостатков использования StageText можно отметить невозможность прокрутки представления, в котором находятся элементы управления. Кроме того, невозможно использовать встроенные шрифты или изменять размеры элементов управления на основе StageText. Если эти функции являются необходимыми, можно использовать элементы управления вводом текста на основе TextField.

  Подробную информацию см. в разделе «Использование текста в мобильном приложении» на странице 152.

- **Целесообразность использования компонентов диаграмм MX в мобильном приложении**

В мобильном приложении могут использоваться элементы управления диаграмм MX, например AreaChart и BarChart. При этом производительность может быть снижена в зависимости от размера и типа данных, используемых в диаграммах.


Nahuel Foronda является автором ряда статей, посвященных Mobile ItemRenderer в ActionScript.


Автор блога Rich Tretola предлагает подробные инструкции по созданию списка с помощью ItemRenderer для мобильного приложения.

Глава 2. Среда разработки

Создание приложения для Android в Flash Builder

Ниже рассматривается общий процесс создания мобильного приложения Flex для платформы Google Android. Для выполнения этого рабочего процесса необходимо сначала создать мобильное приложение. Для получения подробной информации см. раздел «Разработка мобильного приложения» на странице 1.

Специалист по продукции Adobe Mike Jones делится опытом, который он приобрел во время разработки своей межплатформенной игры Mode, и предлагает 10 советов по разработке приложения для нескольких устройств.

Требования AIR

Для создания мобильных проектов Flex и ActionScript требуется AIR 2.6 или более новой версии. Для запуска мобильных проектов могут использоваться физические устройства, поддерживающие AIR 2.6 или более новую версию AIR.

Установить AIR 2.6 или более новую версию можно только на поддерживаемых устройствах Android, которые работают под управлением Android 2.2 или более новой версии. Полный список поддерживаемых устройств Android см. на странице сертифицированных устройств. Также ознакомьтесь с минимальными системными требованиями для выполнения Adobe AIR на устройствах Android, приведенными на странице требований к мобильным системам.

Примечание. *Если используемое устройство не поддерживает AIR 2.6 или более новую версию, для запуска и отладки приложений на настольном компьютере используется Flash Builder.*

Каждая версия Flex SDK содержит необходимую версию Adobe AIR. Если для установки мобильных приложений на устройстве использовалась предыдущая версия Flex SDK, удалите AIR с устройства. Flash Builder устанавливает правильную версию AIR, когда пользователь запускает или отлаживает мобильное приложение на устройстве.

Создание приложения

1 В меню Flash Builder выберите пункты «Файл» > «Создать» > «Мобильный проект Flex».

Мобильный проект Flex - это особый тип проектов AIR. Следуйте инструкциям в мастере создания проектов, как и при создании любого другого проекта AIR в Flash Builder. Для получения подробной информации см. раздел Мобильные проекты Flex.

Подробную информацию о настройке мобильных проектов для Android см. в разделе «Настройка параметров мобильных проектов» на странице 16.

При создании мобильного проекта Flex в Flash Builder создаются следующие файлы:

- *ProjectName*.mxml

  Это стандартный файл приложения для проекта.

  По умолчанию в имени этого файла Flash Builder использует имя проекта. Если в имени проекта содержатся недопустимые символы ActionScript, названием файла будет Main.mxml. Этот файл MXML содержит базовый тег приложения Spark для проекта, который может быть представлен ViewNavigatorApplication или TabbedViewNavigatorApplication.

  Как правило, содержимое не добавляется непосредственно в стандартный файл приложения, кроме содержимого ActionBar, которое отображается во всех представлениях. Для добавления содержимого в ActionBar установите свойства `navigatorContent`, `titleContent` или `actionContent`.

- *ProjectName*HomeView.mxml

  Этот файл определяет начальное представление проекта. Файл помещается в пакет представлений Flash Builder. Атрибут `firstView` тега ViewNavigatorApplication в *ProjectName*.mxml определяет этот файл в качестве начального представления по умолчанию в приложении.

Для получения подробной информации см. раздел «Определение представлений в мобильном приложении» на странице 42.

Также можно создать мобильный проект, использующий только ActionScript. См. раздел «Создание мобильного проекта ActionScript» на странице 14.

2 Добавьте содержимое в элемент ActionBar в файле главного приложения (необязательно).

ActionBar отображает содержимое и функциональные возможности, применяемые к приложению или текущему представлению приложения. Добавляемое содержимое будет отображаться во всех представлениях приложения. См. раздел «Определение элементов управления навигацией, заголовком и действиями в мобильном приложении» на странице 66.

3 Создайте макет содержимого в начальном представлении приложения.

Для добавления компонентов к представлению откройте режим «Дизайн» или «Код» в Flash Builder.

Используйте только такие компоненты, которые поддерживаются в Flex при разработке мобильных приложений. В режимах «Дизайн» или «Код» в Flash Builder отображается подсказка по использованию поддерживаемых компонентов. См. раздел «Макет и интерфейс пользователя» на странице 27.

В ActionBar представления добавьте содержимое, которое будет отображаться только в этом представлении.

4 (Дополнительно) Добавьте другие представления в приложение.

Откройте проводник пакетов Flash Builder из контекстного меню пакета представлений проекта и выберите пункт меню «Новый MXML-компонент». В мастере создания MXML-компонентов представлены этапы создания представления.

Для получения подробной информации см. раздел «Определение представлений в мобильном приложении» на странице 42.

5 Добавьте средства визуализации элементов для компонентов списка, оптимизированные для мобильных устройств (необязательно).

Компания Adobe предоставляет IconItemRenderer - средство визуализации элементов на основе ActionScript для мобильных приложений. См. раздел Использование мобильного средства визуализации с элементом управления Spark на основе списка.

6 Настройте конфигурации запуска для выполнения и отладки приложений.

Запуск и отладка приложения выполняются на настольном компьютере или другом устройстве.

Конфигурация запуска должна выполнять запуск и отладку приложения из Flash Builder. Перед первым запуском или отладкой мобильного приложения в Flash Builder предлагается настройка конфигурации запуска.

При запуске или отладке мобильного приложения на устройстве Flash Builder устанавливает приложение на этом устройстве.

См. раздел «Тестирование и отладка» на странице 194.

7 Экспортируйте приложение как установочный пакет.

Для создания пакетов, которые можно установить на мобильном устройстве, используйте функцию «Экспорт сборки выпуска». Flash Builder создает пакеты для выбранной пользователем платформы экспорта. См. раздел «Экспорт пакетов Android APK для выпуска» на странице 202.


Сертифицированный специалист Adobe по продуктам Flex Brent Arnold предлагает следующие видеоруководства:

- Create a Flex mobile application with multiple views
- Create a Flex mobile application using a Spark-based List control

Создание приложения для iOS в Flash Builder

Ниже рассматривается общий процесс создания мобильного приложения для платформы Apple iOS.

1 Прежде чем приступать к созданию приложения, выполните шаги, указанные в разделе «Процесс разработки приложения для Apple iOS в Flash Builder» на странице 22.

2 В меню Flash Builder выберите пункты «Файл» > «Создать» > «Мобильный проект Flex».

В качестве целевой платформы укажите Apple iOS и определите настройки мобильного проекта.

Следуйте инструкциям в мастере создания проектов, как и при создании любого другого проекта в Flash Builder. Для получения подробной информации см. раздел «Создание приложения» на странице 11.

Также можно создать мобильный проект, использующий только ActionScript. Для получения подробной информации см. раздел Создание мобильных проектов ActionScript.

3 Настройте конфигурации запуска для выполнения и отладки приложений. Запуск и отладка приложения выполняются на настольном компьютере или другом подключенном устройстве.

Подробную информацию см. в разделе «Отладка приложений на устройстве Apple iOS» на странице 198.

4 Экспортируйте приложение в Apple App Store или разверните приложение пакета iOS (IPA) на устройстве.

Подробную информацию см. в разделах «Экспорт пакетов Apple iOS для выпуска» на странице 203 и «Установка приложения на устройстве Apple iOS» на странице 200.

Дополнительные разделы справки

Beginning a Mobile Application (видеоролик)

Создание приложения для BlackBerry Tablet OS в Flash Builder

В Flash Builder включен дополнительный модуль от компании Research In Motion (RIM), с помощью которого создаются и упаковываются приложения Flex и ActionScript для BlackBerry® Tablet OS.

Создание приложения

Ниже рассматриваются основные этапы создания приложений для BlackBerry Tablet OS.

1 Перед созданием мобильного приложения установите BlackBerry Tablet OS SDK для AIR, доступный для загрузки на веб-сайте BlackBerry Tablet OS Application Development.

Приложение BlackBerry Tablet OS SDK для AIR содержит API, необходимые для создания приложений Flex и ActionScript на основе AIR.

Подробную информацию по установке BlackBerry Tablet OS SDK см. в руководстве BlackBerry Tablet OS Getting Started Guide.

2 Для создания приложения AIR на основе Flex в меню Flash Builder выберите пункты «Файл» > «Создать» > «Мобильный проект Flex».

Следуйте инструкциям в мастере создания проектов, как и при создании любого другого проекта AIR в Flash Builder. Убедитесь, что в качестве целевой платформы указано BlackBerry Tablet OS.

Для получения подробной информации см. раздел Мобильные проекты Flex.

3 Для создания приложения AIR на основе ActionScript в меню Flash Builder выберите пункты «Файл» > «Создать» > «Мобильный проект ActionScript».

Следуйте инструкциям в мастере создания проектов, как и при создании любого другого проекта AIR в Flash Builder. Убедитесь, что в качестве целевой платформы указано BlackBerry Tablet OS.

Для получения подробной информации см. раздел Создание мобильных проектов ActionScript.

Подписание, упаковка и развертывание приложения

Подробную информацию по подписанию, упаковке и развертыванию приложения см. в руководстве BlackBerry Tablet OS SDK for Adobe AIR Development Guide, предоставленном компанией RIM.

Для получения дополнительной информации по разработке BlackBerry Tablet OS, предоставленной компаниями Adobe и RIM, перейдите на веб-сайт Adobe Developer Connection.

Создание мобильного проекта ActionScript

Программа Flash Builder позволяет создавать мобильные приложения ActionScript. Приложения создаются на основе Adobe AIR API.

1 Выберите пункты «Файл» > «Создать» > «Мобильный проект ActionScript».

2 Введите имя и местоположение проекта. Местоположением по умолчанию является текущая рабочая область.

3 Используйте стандартный Flex 4.6 SDK, который поддерживает разработку мобильных приложений.

Нажмите кнопку «Далее».

4 Выберите целевые платформы для приложения и настройте мобильный проект для каждой платформы.

Для получения подробной информации см. раздел «Настройка параметров мобильных проектов» на странице 16.

5 Нажмите кнопку «Готово» для завершения или кнопку «Далее» для выбора дополнительных параметров конфигурации и путей сборки.

Для получения подробной информации о параметрах конфигурации проекта и путях сборки см. раздел Пути сборки, собственные расширения и другие параметры конфигурации проекта.

Использование собственных расширений

Собственные расширения позволяют встраивать функции собственной платформы в мобильное приложение.

Собственное расширение содержит классы ActionScript и собственный код. Реализация собственного кода предоставляет доступ к определенным функциям устройства, которые недоступны при использовании обычных классов ActionScript, например к функциям вибрации устройства.

Собственный код можно определить как код, выполняемый вне среды AIR. В расширении можно определить классы и собственный код ActionScript для различных платформ. Классы расширения ActionScript используют и передают данные в собственный код с помощью класса ActionScript ExtensionContext.

Расширения зависят от аппаратной платформы устройства. Пользователь может создать расширения как для одной, так и для нескольких платформ. Например, можно создать собственное расширение ActionScript для платформ Android и iOS. Собственные расширения поддерживаются следующими мобильными устройствами:

- Android с установленным приложением Android 2.2 или новее;
- iOS с установленным приложением iOS 4.0 или новее.

Подробную информацию о создании кросс-платформенных собственных расширений см. на странице Разработка собственных расширений для Adobe AIR.

Коллекция примеров собственных расширений, созданных сотрудниками компании Adobe и сообществом, представлена на веб-сайте Native extensions for Adobe AIR.

Упаковка собственных расширений

Чтобы отправить собственное расширение разработчикам приложений, необходимо упаковать все важные файлы в файл ANE (ActionScript Native Extension). Для этого выполните указанные ниже действия.

1 Встройте библиотеку ActionScript расширения в файл SWC.

2 Встройте собственные библиотеки расширения. Если расширение предназначено для нескольких платформ, встройте одну библиотеку для каждой платформы.

3 Создайте подписанный сертификат для расширения. Если расширение не подписано, Flash Builder выводит предупреждение при добавлении расширения к проекту.

4 Создайте файл дескриптора расширения.

5 Добавьте в расширение необходимые внешние ресурсы, например изображения.

6 Создайте пакет расширения с помощью инструмента Air Developer Tool. Для получения подробной информации см. документацию по Adobe AIR.

Подробную информацию об упаковке расширений ActionScript см. на странице Разработка собственных расширений для Adobe AIR.

Добавление собственных расширений в проект

Для добавления файла ANE (ActionScript Native Extension) в путь сборки проекта используется тот же способ, что и для добавления файла SWC.

1 При создании мобильного проекта Flex в Flash Builder откройте вкладку «Собственные расширения» на странице настроек путей сборки.

   Также расширения можно добавить в диалоговом окне «Свойства проекта», выбрав пункт «Путь сборки Flex».

2 Перейдите к файлу ANE или к папке, содержащей файлы ANE, чтобы добавить их в проект. При добавлении файла ANE в файл дескриптора приложения (*имя проекта*-app.xml) проекта по умолчанию добавляется ID расширения.

В Flash Builder отображается символ ошибки для добавленного расширения в следующих случаях:

- Версия среды AIR расширения новее версии среды в приложении.
- Расширение не содержит все выбранные платформы, для которых предназначено приложение.

Примечание. *Пользователь может создать собственное расширение ActionScript, предназначенное для нескольких платформ. Чтобы протестировать приложение, содержащее файл ANE, на компьютере разработки с помощью симулятора AIR, убедитесь, что файл ANE поддерживает платформу компьютера. Например, чтобы протестировать приложение, содержащее симулятор AIR, в ОС Windows, убедитесь, что файл ANE поддерживает Windows.*

Добавление собственных расширений ActionScript в пакет приложения

Если для экспорта мобильного приложения используется функция «Экспорт сборки выпуска», входящие в проект расширения добавляются в пакет приложения по умолчанию.

Чтобы изменить выбор по умолчанию, выполните следующие шаги:

1 В диалоговом окне «Экспорт сборки выпуска» откройте вкладку «Собственные расширения» в разделе настроек пакета.

2 В окне отобразится список файлов собственных расширений ActionScript, на которые определены ссылки в проекте, и сведения о том, используется ли файл ANE в проекте.

Если файл ANE используется в проекте, он по умолчанию выбирается в пакете приложения.

Если файл ANE содержится, но не используется в проекте, компилятор не распознает этот файл ANE. Следовательно, этот файл не добавляется в пакет приложения. Чтобы добавить файл ANE в пакет приложения, выполните следующие действия:

a В диалоговом окне «Свойства проекта» выберите пункт «Упаковка сборки Flex» и укажите требуемую платформу.

b Выберите расширения, которые требуется добавить в пакет приложения.

Поддержка собственных расширений iOS5

Чтобы упаковать собственные расширения, использующие функции iOS5 SDK, необходимо указать местоположение iOS5 SDK для AIR Developer Tool (ADT).

В Mac OS программа Flash Builder предоставляет возможность выбора местоположения iOS5 SDK в диалоговом окне «Параметры пакета». Выбранное местоположение iOS SDK передается с помощью команды ADT `-platformsdk` .

Примечание. *В настоящее время эта функция не поддерживается в ОС Windows.*

Подробную информацию см. на странице Разработка собственных расширений для Adobe AIR.

Настройка параметров мобильных проектов

Настройка конфигурации устройства

Flash Builder использует конфигурации устройства для отображения в Design View предварительного просмотра, соответствующего размеру экрана устройства, или для запуска приложений на рабочем столе с помощью AIR Debug Launcher (ADL). См. раздел «Настройка данных устройства для просмотра на настольном компьютере» на странице 195.

Для установки конфигурации устройства откройте диалоговое окно «Параметры» и выберите)пункты «Flash Builder» > «Конфигурации устройств».

В Flash Builder предусмотрено несколько стандартных конфигураций устройств. Дополнительные конфигурации устройств можно добавлять, изменять или удалять. Стандартные конфигурации, которые предоставляет Flash Builder, изменять нельзя.

Параметр «Восстановить значения по умолчанию» восстанавливает стандартные конфигурации устройств, при этом добавленные пользователем конфигурации не удаляются. Если имя добавленной конфигурации устройств совпадает с именем стандартной конфигурации Flash Builder, добавленная конфигурация будет заменена стандартной.

В таблице ниже перечислены свойства конфигураций устройств.

Свойство	Описание
Имя устройства	Уникальное имя устройства..
Платформа	Платформа устройства. Список поддерживаемых платформ для выбора.
Размер в полноэкранном режиме	Ширина и высота экрана устройства.
Используемый размер экрана	Стандартный размер приложения на устройстве. Этот размер является ожидаемым размером приложения, запущенного не в полноэкранном режиме и с использованием системного хрома, например строки состояния.
Пикселы на дюйм	Количество пикселов на дюйм на экране устройства.

Выбор целевых платформ

Flash Builder поддерживает целевые платформы на основе типа приложения.

Чтобы выбрать платформу, откройте диалоговое окно «Параметры» и выберите пункты «Flash Builder» > «Целевые платформы».

Для получения информации о модулях сторонних поставщиков см. соответствующую документацию.

Выбор шаблона приложения

Для создания мобильного приложения используются приведенные ниже шаблоны приложений.

Пустой документ В качестве базового элемента приложения используется тег Spark Application.

Этот параметр позволяет создать пользовательское приложение, не используя стандартную навигацию представления.

Приложение на основе представления В качестве базового элемента приложения с одним представлением используется тег Spark ViewNavigatorApplication.

Пользователь может указать имя исходного представления.

Приложение с вкладками В качестве базового элемента приложения с вкладками используется тег Spark TabbedViewNavigatorApplication.

Чтобы добавить вкладку, укажите имя вкладки и нажмите кнопку «Добавить». Порядок вкладок изменяется с помощью клавиш со стрелками вверх и вниз. Чтобы удалить вкладку из приложения, выберите требуемую вкладку и нажмите кнопку «Удалить».

Имя представления - это имя вкладки, к которой добавлено слово «View». Например, если именем вкладки является FirstTab, Flash Builder создаст представление с именем FirstTabView.

Файл MXML для каждой создаваемой вкладки находится в пакете views.

Примечание. *Имя этого пакета невозможно изменить в мастере создания мобильных проектов Flex.*

При создании файлов MXML применяются следующие правила:

- Если имя вкладки является действительным именем класса ActionScript, Flash Builder создает файл MXML, используя имя вкладки с добавлением слова «View».

- Если имя вкладки не является действительным именем класса, Flash Builder удаляет из него недопустимые символы и при необходимости добавляет действительные начальные символы. Если измененное имя недопустимо, Flash Builder изменяет имя файла MXML на «ViewN», где N - это положение представления начиная с N=1.


Сертифицированный специалист Adobe по продуктам Flex Brent Arnold предлагает видеоруководство по использованию шаблона Tabbed Application.

Выбор разрешений для мобильного приложения

При создании мобильного приложения пользователь может установить или изменить разрешения по умолчанию для целевой платформы. Разрешения указываются при компиляции и не могут быть изменены во время выполнения.

Выберите целевую платформу и укажите необходимые разрешения для каждой платформы. Разрешения можно отредактировать позднее в файле XML дескриптора приложения.

Модули сторонних поставщиков предоставляют поддержку дополнительных платформ для проектов Flex и ActionScript. Для получения информации о разрешениях, зависящих от платформы, см. документацию к устройству.

Разрешения для платформы Google Android

Для платформы Google Android устанавливаются следующие разрешения:

INTERNET: позволяет выполнять сетевые запросы и удаленную отладку.

Разрешение INTERNET выбрано по умолчанию. В случае отмены выбора этого разрешения отладка приложения на устройстве будет невозможна.

WRITE_EXTERNAL_STORAGE: позволяет выполнять запись на внешнее устройство.

Это разрешение используется для записи из приложения на внешнюю карту памяти устройства.

READ_PHONE_STATE: отключает звук аудиозаписи в случае получения входящего звонка.

Это разрешение позволяет приложению отключать звук аудиозаписи на время телефонных звонков. Например, это разрешение можно выбрать, если приложение воспроизводит аудиозаписи в фоновом режиме.

ACCESS_FINE_LOCATION: предоставляет доступ к местоположению GPS.

Выбор этого разрешения делает возможным доступ приложения к данным GPS с помощью класса Geolocation.

DISABLE_KEYGUARD и WAKE_LOCK: не позволяет устройству переходить в режим сна.

Это разрешение использует параметры класса SystemIdleMode для предотвращения перехода устройства в режим сна.

CAMERA: предоставляет доступ к камере.

Это разрешение позволяет приложению использовать камеру.

RECORD_AUDIO: Предоставляет доступ к микрофону.

Это разрешение позволяет приложению использовать микрофон.

ACCESS_NETWORK_STATE и ACCESS_WIFI_STATE: предоставляет доступ к информации о сетевых интерфейсах, связанных с устройством.

Это разрешение позволяет приложению получать доступ к сетевой информации с помощью класса NetworkInfo.

Для получения более подробных сведений о настройке свойств мобильного приложения см. Документацию по Adobe AIR .

Разрешения для платформы Apple iOS

На платформе Apple iOS вместо предварительно определенных разрешений используется проверка разрешений во время выполнения. Если приложению необходимо получить доступ к определенной функции платформы Apple iOS, для которой требуется разрешение пользователя, на экране отобразится всплывающее окно с запросом разрешения.

Выбор параметров платформы

Параметры платформ позволяют выбрать группу целевых устройств. Различные платформы предлагают возможность выбора отдельного целевого устройства или группы целевых устройств. При этом можно выбрать отдельное устройство или все устройства, поддерживаемые данной платформой.

Модули сторонних поставщиков предоставляют поддержку дополнительных платформ для проектов Flex и ActionScript. Для получения информации о параметрах, зависящих от платформы, см. документацию к устройству.

Параметры для платформы Google Android

Платформа Google Android не требует использования специфических параметров.

Параметры для платформы Apple iOS

В мобильных проектах Flex или ActionScript определяются следующие целевые устройства для платформы Apple iOS:

iPhone/iPod Touch: приложения для этой группы целевых устройств совместимы только с устройствами iPhone и iPod Touch в Apple App Store.

iPad: приложения для этой группы целевых устройств совместимы только с устройствами iPad в Apple App Store.

Все: приложения для этой группы целевых устройств совместимы с устройствами iPhone/iPod Touch и iPad в Apple App Store. Этот параметр задан по умолчанию.

Выбор параметров приложения

Автоматическое изменение ориентации: приложение меняет ориентацию в зависимости от поворотов устройства. Если этот параметр не выбран, ориентация приложения на экране будет фиксированной.

Полноэкранный режим: отображает приложение на устройстве в полноэкранном режиме. Если этот параметр активирован, над приложением не отображается строка состояния устройства. Окно приложения заполняет всю область экрана.

Если приложение предназначается для нескольких типов устройств с различной плотностью экрана, выберите параметр «Автоматически определять масштаб приложения для различных показателей плотности экрана». При выборе этого параметра автоматически изменяется масштаб приложения и обрабатываются различные показатели плотности экрана устройства при необходимости. См. раздел «Установка масштабирования приложения» на странице 20.

Установка масштабирования приложения

Функция масштабирования позволяет создавать мобильные приложения, совместимые с устройствами, имеющими различные размеры экранов и плотность.

Экраны мобильных устройств имеют разные уровни плотности или DPI (точки на дюйм). В зависимости от плотности экрана целевого устройства значение DPI может составлять 160, 240 или 320. После активации автоматического масштабирования Flex оптимизирует способ отображения приложения в соответствии с плотностью экрана каждого устройства.

Например, если указанным целевым значением DPI является 160 и активировано автоматическое масштабирование, при запуске приложения на устройстве со значением DPI 320 Flex автоматически изменит масштаб приложения, увеличив его в 2 раза. Это значит, что выполняемое Flex увеличение составляет 200%.

Чтобы указать целевое значение DPI, установите его в качестве свойства `applicationDPI` тега `<s:ViewNavigatorApplication>` или `<s:TabbedViewNavigatorApplication>` в файле главного приложения:

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

Если автоматическое масштабирование приложения выключено, показатели плотности необходимо изменять в макете вручную. Тем не менее, Flex адаптирует темы оформления к плотности каждого устройства.

Для получения подробной информации о создании мобильных приложений, независимых от плотности экрана устройств, см. раздел «Поддержка различных размеров экрана и значений DPI в мобильном приложении» на странице 137.

Подключение устройств Google Android

Подключение устройств Google Android к компьютеру разработки обеспечивает возможности просмотра или отладки приложения на устройстве Android.

Поддерживаемые устройства Android

Для создания мобильных проектов Flex и ActionScript требуется наличие AIR 2.6 или более новой версии. Для запуска мобильных проектов могут использоваться физические устройства, поддерживающие AIR 2.6 или более новую версию.

Установить AIR 2.6 можно только на поддерживаемых устройствах Android, которые работают под управлением Android 2.2 или более новой версии. Список поддерживаемых устройств см. на веб-сайте http://www.adobe.com/flashplatform/certified_devices/. Также ознакомьтесь с минимальными системными требованиями для выполнения Adobe AIR на устройствах Android, приведенными на странице требований к мобильным системам.

Настройка устройств Android

Для запуска и отладки мобильных приложений на устройстве Android выполните следующие шаги для включения отладки с использованием USB.

1 Убедитесь, что на устройстве включена отладка с использованием USB, выполнив следующие действия:

   a Нажмите кнопку Home для отображения начального экрана.

b В меню Settings выберите пункты Applications > Development.

c Включите отладку с использованием USB.

2 Подключите устройство к компьютеру посредством кабеля USB.

3 Разверните область уведомлений в верхней части экрана. На экране отображается сообщение USB Connected или USB Connection.

a Нажмите строку USB Connected или USB Connection.

b Если на экране отображается набор параметров, включающий режим Charge Only, выберите пункт Charge Only и нажмите кнопку «OK».

c Нажмите кнопку для отключения режима массового сохранения (при ее наличии).

4 (Только для Windows) Установите соответствующий драйвер USB для устройства. См. раздел «Установка драйверов устройств USB для устройств Android (Windows)» на странице 21.

5 Разверните область уведомлений в верхней части экрана.

Если параметр USB Debugging не отображается, проверьте режим USB, как указано в шаге 3 выше. Убедитесь, что режим USB не установлен в режим PC.

Примечание. *При отладке требуется дополнительная конфигурация. «Тестирование и отладка мобильного приложения на устройстве» на странице 195*

Установка драйверов устройств USB для устройств Android (Windows)

Драйверы и конфигурации устройств

Для подключения устройства Android к компьютеру разработки с платформой Windows необходимо установить драйвер USB. Flash Builder предоставляет различные драйверы и конфигурации для устройств Android.

Эти конфигурации драйверов устройств перечислены в файле `android_winusb.inf`. Диспетчер устройств Windows считывает этот файл при установке драйвера устройства. Для установки `android_winusb.inf` Flash Builder использует следующий путь:

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

Полный список всех поддерживаемых устройств см. на странице сертифицированных устройств. Для устройств Android, отсутствующих в списке, можно добавить драйверы USB в файл `android_winusb.inf`. См. раздел «Добавление конфигураций драйверов устройств USB Android» на странице 22.

Установка драйвера устройства USB

1 Подключите устройство Android к компьютеру через порт USB.

2 Перейдите по следующему пути:

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

Для установки драйвера можно использовать мастер нового оборудования Windows или диспетчер устройств Windows.

Важная информация. *Если системе Windows не удается распознать устройство, необходимо установить соответствующий драйвер USB, предоставленный производителем устройства. Ссылки на веб-сайты различных производителей устройств для загрузки требуемых драйверов USB перечислены в списке драйверов OEM USB.*

Добавление конфигураций драйверов устройств USB Android

Если поддерживаемое устройство Android отсутствует в таблице, приведенной в разделе «Установка драйверов устройств USB для устройств Android (Windows)» на странице 21, добавьте это устройство в файл `android_winusb.inf`.

1 Подключите устройство к порту USB компьютера. Система Windows сообщит, что ей не удается найти драйвер.

2 В диспетчере устройств Windows откройте вкладку «Подробности» на странице свойств устройства.

3 Выберите свойство «Идентификатор оборудования» для просмотра соответствующих данных.

4 Откройте `android_winusb.inf` в текстовом редакторе. Перейдите к файлу `android_winusb.inf` по следующему пути:

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

5 В файле перейдите к спискам, соответствующим используемой архитектуре: `[Google.NTx86]` или `[Google.NTamd64]`. Список содержит описательный комментарий и одну или несколько строк с идентификатором оборудования, как показано ниже:

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

6 Скопируйте и вставьте комментарий и список оборудования. Измените список, как представлено в примере ниже, чтобы добавить драйвер устройства.

a В комментарии укажите имя устройства.

b Замените идентификатор оборудования на идентификатор, указанный в шаге 3 выше.

Например:

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

7 Установите устройство с помощью диспетчера устройств Windows, как описано в разделе «Установка драйверов устройств USB для устройств Android (Windows)» на странице 21.

Во время установки система Windows выведет предупреждение о том, что издатель драйвера неизвестен. Однако с помощью этого драйвера выполняется передача данных между Flash Builder и устройством.

Процесс разработки приложения для Apple iOS в Flash Builder

Прежде чем начинать разработку приложения iOS в Flash Builder, важно ознакомиться с процессом разработки iOS и узнать, что требуется для получения необходимых сертификатов Apple.

Обзор процессов разработки и развертывания iOS

В этой таблице приведен короткий перечень шагов процесса разработки iOS, рассматриваются предварительные условия для каждого шага, а также указаны способы получения необходимых сертификатов.

Подробные сведения о каждом из шагов см. в разделе «Подготовка к созданию, отладке или развертыванию приложения для iOS» на странице 24.

№ шага	Шаг	Местоположение	Предварительные условия
1.	Станьте участником программы для разработчиков Apple.	Веб-сайт для разработчиков Apple	Отсутствует
2.	Зарегистрируйте уникальный идентификатор (UDID) используемого устройства iOS.	iOS Provisioning Portal	Идентификатор разработчика Apple (шаг 1)
3.	Создайте файл запроса на подпись сертификата (CSR) (*.certSigningRequest).	• В Mac OS используйте программу Keychain Access. • В Windows используйте OpenSSL.	Отсутствует
4.	Создайте сертификат разработчика/распространения iOS (*.cer).	iOS Provisioning Portal	• Идентификатор разработчика Apple (шаг 1) • Файл CSR (шаг 3)
5.	Преобразуйте сертификат разработчика/распространения iOS в формат P12.	• В Mac OS используйте программу Keychain Access. • В Windows используйте OpenSSL.	• Идентификатор разработчика Apple (шаг 1) • Сертификат разработчика/распространения iOS (шаг 4)
6.	Создайте идентификатор приложения.	iOS Provisioning Portal	Идентификатор разработчика Apple (шаг 1)
7.	Создайте профиль обеспечения (*.mobileprovision)	iOS Provisioning Portal	• Идентификатор разработчика Apple (шаг 1) • UDID используемого устройства iOS (шаг 2) • Идентификатор устройства (шаг 6)
8.	Создайте приложение.	Flash Builder	• Идентификатор разработчика Apple (шаг 1) • Сертификат разработчика/распространения в формате P12 (шаг 5) • Идентификатор устройства (шаг 6)
9.	Разверните приложение.	iTunes	• Профиль обеспечения (шаг 7) • Пакет приложения (шаг 8)

Подготовка к созданию, отладке или развертыванию приложения для iOS

Прежде чем приступить к созданию приложения для iOS в Flash Builder, его развертыванию на устройстве iOS или отправке в магазин Apple App Store, выполните следующие шаги:

1 Станьте участником программы для разработчиков Apple iOS.

Для входа в систему можно использовать существующий идентификатор Apple или создать новый. В окне регистрации разработчиков Apple будут представлены инструкции по завершению всех необходимых шагов.

2 Зарегистрируйте уникальный идентификатор (UDID) устройства.

Этот шаг следует выполнять только в случае, если приложение необходимо развернуть на устройстве iOS, а не в магазине Apple App Store. Для развертывания приложения на нескольких устройствах iOS зарегистрируйте UDID каждого из устройств.

Получите UDID используемого устройства iOS

a Подключите устройство iOS к компьютеру, на котором выполнялась разработка, и запустите iTunes. Подключенное устройство iOS отобразится в разделе Devices приложения iTunes.

b Щелкните имя устройства, чтобы отобразить сводные данные об устройстве iOS.

c На вкладке Summary щелкните Serial Number, чтобы отобразить 40-значный UDID устройства iOS.

UDID из iTunes можно скопировать, используя комбинацию клавиш Ctrl+C (Windows) или Cmd+C (Mac).

Зарегистрируйте UDID используемого устройства

Используя идентификатор Apple, войдите в систему iOS Provisioning Portal и зарегистрируйте UDID устройства.

3 Создайте файл запроса на подпись сертификата (CSR) (*.certSigningRequest).

Создайте CSR для получения сертификата разработчика/распространения iOS. Создать CSR можно с помощью приложения Keychain Access (Mac) или OpenSSL (Windows). Для создания CSR необходимо предоставить только имя пользователя и адрес электронной почты. Сведения о приложении или устройстве не требуются.

При генерации CSR создаются открытый и закрытый ключи, а также файл *.certSigningRequest. Открытый ключ содержится в CSR, а закрытый ключ используется для подписания запроса.

Для получения подробной информации о создании CSR см. статью Generating a certificate signing request.

4 При необходимости создайте сертификат разработчика или распространения iOS (*.cer).

Примечание. *Для развертывания приложения на устройстве необходим сертификат разработчика. Для развертывания приложения в магазине Apple App Store требуется сертификат распространения.*

Создайте сертификат разработчика iOS

a Используя идентификатор Apple, войдите в систему iOS Provisioning Portal и перейдите на вкладку Development.

b Щелкните Request Certificate и перейдите к созданному и сохраненному на компьютере CSR-файлу (шаг 3).

c Выберите файл CSR и нажмите кнопку Submit.

d На странице Certificates нажмите кнопку Download.

e Сохраните загруженный файл (*.developer_identity.cer).

Создайте сертификат распространения iOS

f Используя идентификатор Apple, войдите в систему iOS Provisioning Portal и перейдите на вкладку Distribution.

g Щелкните Request Certificate и перейдите к созданному и сохраненному на компьютере CSR-файлу (шаг 3).

h Выберите файл CSR и нажмите кнопку Submit.

i На странице Certificates нажмите кнопку Download.

j Сохраните загруженный файл (*.distribution_identity.cer).

5 Преобразуйте файл сертификата разработчика или распространения iOS в формат P12 (*.p12).

Сертификат разработчика или распространения iOS преобразуется в формат P12 для того, чтобы приложение Flash Builder могло создать цифровую подпись для приложения iOS. Во время преобразования в формат P12 сертификат разработчика/распространения iOS и соответствующий закрытый ключ объединяются в один файл.

Примечание. *Для тестирования приложения на настольном компьютере с помощью AIR Debug Launcher (ADL) преобразование сертификата разработчика/распространения в формат P12 не требуется.*

Для создания файла обмена личными данными (*.p12) используйте программу Keychain Access (Mac) или OpenSSL (Windows). Для получения подробной информации см. статью Преобразование сертификата разработчика в файл P12.

6 Чтобы создать идентификатор приложения, выполните следующие шаги:

a Используя идентификатор Apple, войдите в систему iOS Provisioning Portal.

b Перейдите на страницу App IDs и щелкните ссылку New App ID.

c На вкладке Manage введите описание приложения, создайте новый Bundle Seed ID и введите значение для Bundle Identifier.

Для каждого приложения определен уникальный идентификатор, указанный в файле XML дескриптора приложения. Идентификатор приложения состоит из предоставляемого компанией Apple десятизначного Bundle Seed ID и суффикса Bundle Identifier, который указывает пользователь. Указываемый пользователем Bundle Identifier должен совпадать с идентификатором приложения, указанным в файле дескриптора приложения. Например, если идентификатором приложения является com.myDomain.*, идентификатор, содержащийся в файле дескриптора приложения, должен начинаться с com.myDomain.

Важная информация. *Подстановочные идентификаторы пакета удобны для разработки и тестирования приложений iOS, но не подходят для развертывания приложений в магазине Apple App Store.*

7 Создайте файл профиля обеспечения разработчика или файл профиля обеспечения распространения (*.mobileprovision).

Примечание. *Для развертывания приложения на устройстве необходим профиль обеспечения разработчика. Для развертывания приложения в магазине Apple App Store требуется профиль обеспечения распространения. Профиль обеспечения распространения используется для подписания приложения.*

Создайте профиль обеспечения разработчика

a Используя идентификатор Apple, войдите в систему iOS Provisioning Portal.

b Перейдите к пункту меню Certificate > Provisioning и нажмите кнопку New Profile.

c Укажите имя профиля, выберите сертификат разработчика iOS, идентификатор приложения и идентификаторы UDID устройств, на которых требуется установить приложение.

d Нажмите кнопку Submit.

e Загрузите созданный файл профиля обеспечения разработчика (*.mobileprovision) и сохраните его на компьютере.

Создайте профиль обеспечения распространения

f Используя идентификатор Apple, войдите в систему iOS Provisioning Portal.

g Перейдите к пункту меню Certificate > Provisioning и нажмите кнопку New Profile.

h Введите имя профиля, выберите сертификат распространения iOS и идентификатор приложения. Если приложение перед развертыванием необходимо протестировать, укажите идентификаторы UDID устройств, на которых будет выполняться тестирование.

i Нажмите кнопку Submit.

j Загрузите созданный файл профиля обеспечения распространения (*.mobileprovision) и сохраните его на компьютере.

Дополнительные разделы справки

«Создание приложения для iOS в Flash Builder» на странице 12

Файлы, выбираемые при тестировании, отладке или установке приложения для iOS

Чтобы запустить, отладить или установить приложение для тестирования на устройстве iOS, выберите следующие файлы в диалоговом окне «Конфигурации запуска/отладки»:

- сертификат разработчика iOS в формате P12 (шаг 5);
- файл XML дескриптора приложения, который содержит идентификатор приложения (шаг 6);
- профиль обеспечения разработчика (шаг 7).

Для получения подробной информации см. разделы «Отладка приложений на устройстве Apple iOS» на странице 198 и «Установка приложения на устройстве Apple iOS» на странице 200.

Файлы, выбираемые при развертывании приложения в магазине Apple App Store

Чтобы развернуть приложение в магазине Apple App Store, в пункте «Тип пакета» диалогового окна «Экспорт сборки выпуска» необходимо указать значение Final Release Package For Apple App Store, а затем выбрать следующие файлы:

- сертификат распространения iOS в формате P12 (шаг 5);
- файл XML дескриптора приложения, который содержит идентификатор приложения (шаг 6);

  Примечание. *Подстановочный идентификатор приложения не подходит для отправки приложения в магазин Apple App Store.*

- профиль обеспечения распространения (шаг 7).

Для получения подробной информации см. раздел «Экспорт пакетов Apple iOS для выпуска» на странице 203.

Глава 3. Макет и интерфейс пользователя

Создание макета мобильного приложения

Использование представлений и разделов для создания макета мобильного приложения

Мобильное приложение состоит из одного или нескольких экранов, которые называются *представлениями*. Например, мобильное приложение может иметь три представления:

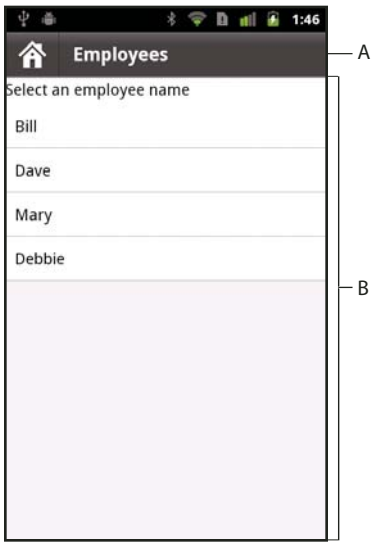
1 начальное представление для добавления контактной информации;

2 представление со списком существующих контактов;

3 представление для поиска в списке контактов.

Простое мобильное приложение

На изображении ниже представлен основной экран мобильного приложения, созданного в Flex:



***A.** Элемент управления ActionBar **B.** Область содержимого*

На изображении представлены основные области мобильного приложения:

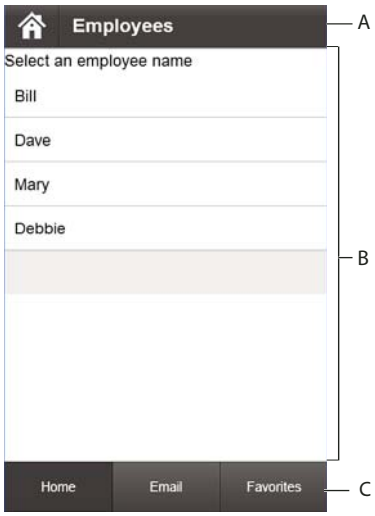
Элемент управления ActionBar: элемент управления ActionBar отображает контекстную информацию о текущем состоянии приложения. Эта информация включает в себя три области: заголовок, элементы управления навигацией в приложении и элементы управления выполнением действий. В элемент управления ActionBar можно добавлять как глобальное содержимое для всего приложения, так и элементы для отдельного представления.

Область содержимого: область содержимого отображает отдельные экраны, или *представления*, которые составляют приложение. Для навигации по представлениям приложения пользователи могут использовать компоненты, встроенные в приложение, и элементы управления для ввода данных в мобильное устройство.

Мобильное приложение с разделами

В приложении с более сложной структурой можно определить несколько областей, или *разделов*. Например, приложение может включать в себя разделы контактов, электронной почты, избранных материалов и т. п. В каждом разделе приложения содержится одно или несколько представлений. Отдельные представления могут быть доступны из различных разделов, поэтому нет необходимости определять одно и то же представление несколько раз.

На изображении ниже представлено окно мобильного приложения, в нижней части которого находится панель вкладок.



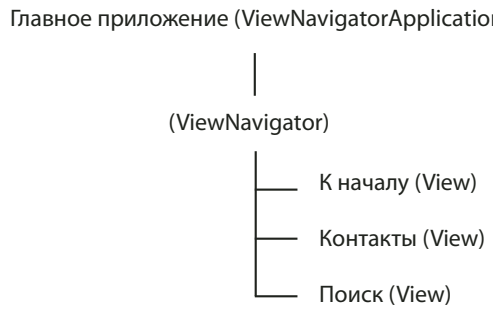
***A.** Элемент управления ActionBar **B.** Область содержимого **C.** Панель вкладок*

Для реализации панели вкладок Flex использует элемент управления ButtonBarBase. Каждая кнопка на панели вкладок соответствует определенному разделу. Для перехода к другому разделу выберите соответствующую кнопку на панели вкладок.

Для каждого раздела приложения определен собственный элемент ActionBar. Таким образом, панель вкладок - это глобальный элемент управления для всего приложения, в то время как ActionBar - это определенный элемент управления для каждого раздела.

Создание макета простого мобильного приложения

На изображении ниже представлена архитектура простого мобильного приложения.



На изображении представлено приложение, состоящее из четырех файлов. Мобильное приложение содержит файл главного приложения и отдельные файлы для каждого представления. Для ViewNavigator отдельный файл не предусмотрен; этот файл создает контейнер ViewNavigatorApplication.

Примечание. *Архитектура приложения на диаграмме не отображает приложение во время выполнения, когда только одно представление активно и находится в памяти. Для получения подробной информации см. раздел «Переход между представлениями в мобильном приложении» на странице 32.*

Классы, используемые в мобильном приложении

Для определения мобильного приложения используются указанные ниже классы.

Класс	Описание
ViewNavigatorApplication	Определяет файл главного приложения. Для контейнера ViewNavigatorApplication не создаются нижестоящие элементы.
ViewNavigator	Управляет переходом между представлениями в приложении. ViewNavigator также создает элемент управления ActionBar. Контейнер ViewNavigatorApplication автоматически создает один контейнер ViewNavigator для всего приложения. Методы контейнера ViewNavigator используются для переключения между различными представлениями.
View	Определяет представления приложения, причем каждое представление определяется в отдельном файле MXML или ActionScript. Экземпляр контейнера View реализует каждое представление приложения. Каждое представление определяется в отдельном файле MXML или ActionScript.

Контейнер ViewNavigatorApplication используется для определения основного файла приложения, как показано в следующем примере.

```
<?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>
```

Контейнер ViewNavigatorApplication автоматически создает один объект ViewNavigator, определяющий элемент ActionBar. ViewNavigator используется для перехода между представлениями в приложении.

Добавление контейнера View в мобильное приложение

В любом мобильном приложении используется как минимум одно представление. Файл главного приложения создает ViewNavigator, но при этом не определяет представления, используемые в приложении.

Каждое представление в приложении соответствует контейнеру View, определенному в файле ActionScript или MXML. Каждый контейнер View содержит свойство `data`, определяющее данные, связанные с этим представлением. Контейнеры View используют свойство `data` для передачи информации друг другу, когда пользователь выполняет навигацию по приложению.

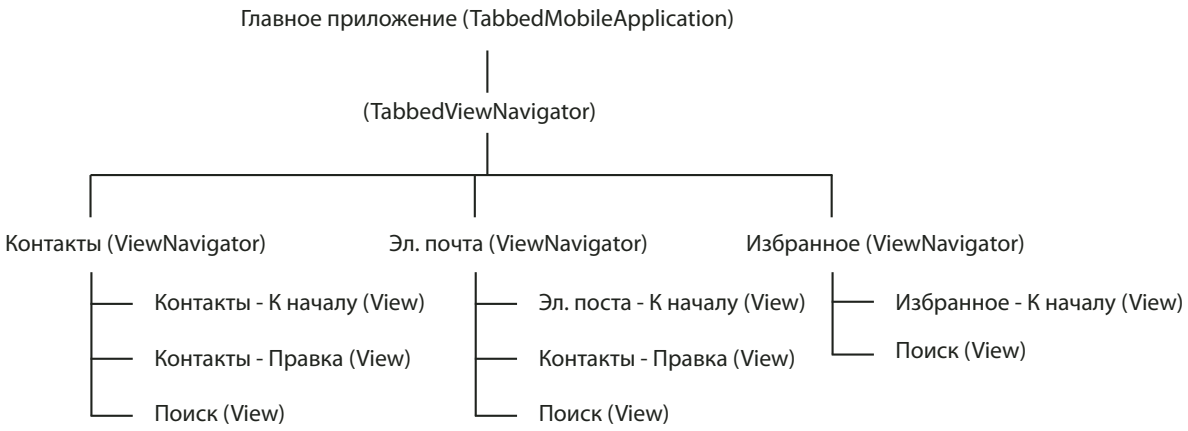
Свойство `ViewNavigatorApplication.firstView` указывает файл, который определяет первое представление в приложении. В предыдущем приложении свойство `firstView` определяет `views.HomeView`. В примере ниже файл HomeView.mxml определяет это представление.

```
<?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>
```

Автор блога David Hassoun предоставляет основные сведения о ViewNavigator.

Создание макета мобильного приложения с несколькими разделами

Разделы мобильного приложения могут содержать связанные представления. В примере ниже представлена структура мобильного приложения с тремя разделами.



Любой раздел может содержать любые представления View. Таким образом, представление не принадлежит определенному разделу. В разделе определяется способ упорядочения и перехода между представлениями. На иллюстрации представление «Поиск» является частью каждого раздела приложения.

Во время выполнения только одно представление может быть активным и находиться в памяти. Для получения подробной информации см. раздел «Переход между представлениями в мобильном приложении» на странице 32.

Классы, используемые в мобильном приложении с несколькими разделами

В таблице ниже перечислены классы, используемые для создания мобильного приложения с несколькими разделами.

Класс	Описание
TabbedViewNavigatorApplication	Определяет файл главного приложения. Единственным допустимым нижестоящим элементом контейнера TabbedViewNavigatorApplication является ViewNavigator. Определяет один ViewNavigator для каждого раздела приложения.
TabbedViewNavigator	Управляет переходом между разделами, составляющими приложение. Контейнер TabbedViewNavigatorApplication автоматически создает один контейнер TabbedViewNavigator для всего приложения. Для поддержки навигации по разделам контейнер TabbedViewNavigator создает панель вкладок.
ViewNavigator	Определяет один контейнер ViewNavigator для каждого раздела. ViewNavigator управляет переходом между представлениями, составляющими приложение. Также он создает элемент управления ActionBar для раздела.
View	Определяет представления приложения. Экземпляр контейнера View реализует каждое представление приложения. Каждое представление определяется в отдельном файле MXML или ActionScript.

Мобильное приложение с разделами содержит файл главного приложения и отдельные файлы для каждого представления. Контейнер TabbedViewNavigatorApplication используется для определения основного файла приложения, как показано в следующем примере:

```
<?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>
```

Использование ViewNavigator в приложении с несколькими разделами

Единственным допустимым нижестоящим компонентом контейнера TabbedViewNavigatorApplication является ViewNavigator. Каждому разделу приложения соответствует отдельный контейнер ViewNavigator.

Контейнер ViewNavigator используется для перехода между представлениями каждого раздела и определения элемента управления ActionBar для раздела. Свойство `ViewNavigator.firstView` указывает файл, который определяет первое представление в разделе.

Использование TabbedViewNavigator в приложении с несколькими разделами

Контейнер TabbedViewNavigatorApplication автоматически создает один контейнер типа TabbedViewNavigator. Контейнер TabbedViewNavigator затем создает панель вкладок в нижней части окна приложения. Добавлять логику перехода между разделами в приложении нет необходимости.

Переход между представлениями в мобильном приложении

Переход в мобильном приложении контролирует группа объектов View. Верхний объект View в стеке определяет текущее видимое представление.

Контейнер ViewNavigator поддерживает стек. Для изменения представлений поместите новый объект View в стек или удалите текущий объект View из стека. Если текущий видимый объект View больше не отображается в стеке, он будет удален и в стеке будет отображаться предыдущее представление.

В приложении с разделами используйте панель вкладок для перехода между разделами. Поскольку каждый раздел определяется отдельным ViewNavigator, изменение разделов соответствует изменению текущего ViewNavigator и стека. Объект View в начале стека нового ViewNavigator становится текущим представлением.

Чтобы снизить нагрузку на память, ViewNavigator по умолчанию использует только одно представление в памяти. При этом он сохраняет данные предыдущих представлений в стеке. Когда пользователь переходит к предыдущему представлению, то для этого представления повторно создается экземпляр с соответствующими данными.

Примечание. *Контейнер View определяет свойство* `destructionPolicy`*. При установленном значении* `auto` *(по умолчанию) ViewNavigator удаляет представление, которое становится неактивным. Если установлено значение* `none`*, представление кэшируется в памяти.*

Автор блога Mark Lochrie рассказывает о ViewNavigator.

Методы навигации ViewNavigator

Управление навигацией выполняется с помощью указанных ниже методов класса ViewNavigator:

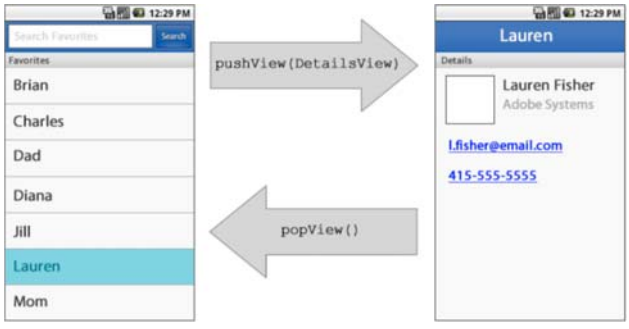
pushView() добавляет объект View в стек. View, который передается как аргумент в метод `pushView()`, становится текущим представлением.

popView() удаляет текущий объект View из стека навигации и удаляет объект View. Предыдущий объект View в стеке становится текущим представлением.

popToFirstView() удаляет все объекты View из стека, кроме первого объекта View в стеке, который становится текущим представлением.

popAll() очищает стек ViewNavigator и удаляет все объекты View. В приложении отображается пустое представление.

В примере ниже показаны два представления. Чтобы изменить текущее представление, метод `ViewNavigator.pushView()` добавляет в стек объект View, который определяет новое представление. При использовании метода `pushView()` ViewNavigator отображает новый объект View.



Добавление и удаление объектов View для изменения представлений.

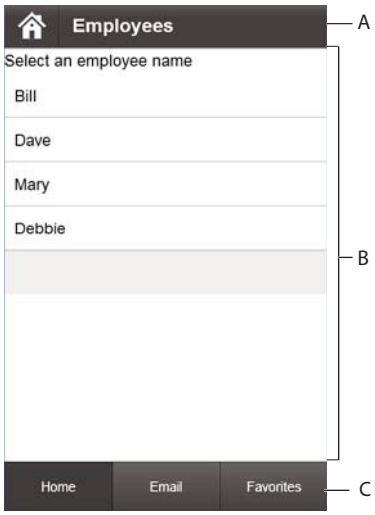
Метод `ViewNavigator.popView()` удаляет текущий объект View из стека. ViewNavigator возвращает представление к предыдущему объекту View в стеке.

Примечание. *Большинством переходов в мобильном приложении управляет само мобильное устройство. Например, мобильные приложения, созданные в Flex, автоматически обрабатывают кнопку возврата на мобильном устройстве. Поэтому для кнопки возврата не требуется добавлять поддержку в приложении. Когда пользователь нажимает кнопку возврата на мобильном устройстве, Flex автоматически вызывает метод `popView()` для восстановления предыдущего представления.*

Автор блога David Hassoun разъясняет особенности управления данными в представлении.

Навигация в приложении с несколькими разделами

В примере ниже представления находятся в различных разделах. Отдельный контейнер ViewNavigator определяет каждый раздел и содержит одно или несколько представлений:



A. *ActionBar* ***B.*** *Область содержимого* ***C.*** *Панель вкладок*

Для изменения представления в текущем разделе, соответствующем текущему ViewNavigator, используются методы `pushView()` и `popView()`.

Для изменения текущего раздела используйте панель вкладок. При переходе между разделами контейнер ViewNavigator переключается на новый раздел. В представлении отображается объект View, который в настоящее время находится в начале стека нового ViewNavigator.

Для изменения разделов программным способом используется свойство `TabbedViewNavigator.selectedIndex`. Это свойство содержит отсчитываемый от нуля индекс выбранного навигатора представлений.

Обработка введенных пользователем данных в мобильном приложении

Обработка введенных пользователем данных в мобильном приложении отличается от обработки этих данных в настольном приложении или приложении для браузера. В настольном приложении, созданном для AIR, или в приложении для браузера, созданном для Flash Player, основными устройствами ввода являются клавиатура и мышь. В мобильных устройствах в качестве устройства ввода используется сенсорный экран, а также специализированная клавиатура и устройства с методом ввода в пяти направлениях (налево, направо, вверх, вниз и выбор).

Класс mx.core.UIComponent определяет свойство стиля `interactionMode`, которое используется для настройки компонентов типа ввода в приложении. В темах Halo и Spark значение по умолчанию (`mouse`) указывает, что мышь является основным устройством ввода. В мобильной теме значение по умолчанию (`touch`) указывает, что основным устройством ввода является сенсорный экран.

Поддержка аппаратных клавиш в мобильном приложении

Приложения, определенные контейнерами ViewNavigatorApplication или TabbedViewNavigatorApplication, используют аппаратные кнопки возврата и меню, находящиеся на устройстве. При нажатии кнопки возврата приложение переходит к предыдущему представлению. Если предыдущее представление отсутствует, выполняется выход из приложения и отображается начальный экран устройства.

При нажатии кнопки возврата активное представление приложения получает событие `backKeyPressed`. Для отмены действия клавиши возврата вызывается `preventDefault()` в обработчике события `backKeyPressed`.

При нажатии кнопки меню появляется предварительно определенный контейнер ViewMenu текущего представления. Контейнер ViewMenu определяет меню в нижней части контейнера View. Каждый контейнер View определяет собственное меню, специфическое для данного представления.

Текущий контейнер View отправляет событие `menuKeyPressed`, когда пользователь нажимает кнопку меню. Чтобы отменить действие кнопки меню и предотвратить отображение ViewMenu, вызовите метод `preventDefault()` в обработчике события `menuKeyPressed`.

Для получения более подробной информации см. раздел «Определение меню в мобильном приложении» на странице 78.

Обработка событий аппаратной клавиатуры в мобильном приложении

В мобильном приложении, которое было создано в Flex, можно определить нажатие пользователем аппаратной клавиши на мобильном устройстве. Например, на устройстве Android можно определить, когда пользователь нажимает кнопку Home, Back или Menu.

Для определения нажатия пользователем аппаратной клавиши создайте обработчики события `KEY_UP` или `KEY_DOWN`. Как правило, обработчики событий назначаются объектам приложения, определяемым контейнером Application, ViewNavigatorApplication или TabbedViewNavigatorApplication.

Объект Stage определяет область рисования приложения. Каждое приложение имеет один объект Stage. Следовательно, контейнер приложения фактически является нижестоящим контейнером объекта Stage.

Свойство `Stage.focus` указывает компонент, который в данный момент имеет фокус клавиатуры. Если ни один из компонентов не имеет фокус, свойству присваивается значение `null`. Компонент с фокусом клавиатуры получает соответствующее уведомление о событии, когда пользователь использует клавиатуру. Следовательно, если `Stage.focus` назначается объекту приложения, то запускаются обработчики событий объекта приложения.

На мобильном устройстве работа приложения может быть прервана другим приложением. Например, во время выполнения приложения мобильное устройство может получить входящий звонок или пользователь может переключиться на другое приложение. Когда пользователь возвращается к приложению, для свойства `Stage.focus` устанавливается значение null. В результате назначенные объекту приложения обработчики событий не отвечают на события клавиатуры.

Поскольку свойство `Stage.focus` мобильного приложения может иметь значение null, необходимо прослушивать события клавиатуры самого объекта Stage, чтобы обеспечить распознавание приложением необходимого события. В следующем примере рассматривается назначение обработчиков событий клавиатуры объекту Stage:

```
<?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>
```

Обработка событий мыши и прикосновений в мобильном приложении

Для обозначения типов ввода AIR создает различные события, некоторые из которых перечислены ниже.

События мыши - события, создаваемые при взаимодействии пользователя с мышью или сенсорным экраном. События мыши включают в себя `mouseOver`, `mouseDown` и `mouseUp`.

События касания создаются на устройстве, которое распознает действие касания, например когда пользователь касается экрана пальцем. События касания включают в себя `touchTap`, `touchOver` и `touchMove`. Взаимодействие пользователя с сенсорным экраном устройства заключается в том, что пользователь касается экрана пальцем или указателем.

События жестов создаются для нескольких касаний, например при одновременном касании экрана двумя пальцами. События жестов включают в себя `gesturePan`, `gestureRotate` и `gestureZoom`. Например, для уменьшения масштаба изображения на некоторых устройствах используется сжатие пальцев.

Встроенная поддержка событий мыши

В инфраструктуру и набор компонентов Flex встроена поддержка событий мыши, но не событий жестов или касания. Например, если пользователь взаимодействует с компонентами Flex в мобильном приложении, используя сенсорный экран, компоненты реагируют на события мыши (`mouseDown` и `mouseOver`), но не на события касания или жестов.

Например, пользователь нажимает элемент управления Flex Button на сенсорном экране. Чтобы сообщить о взаимодействии пользователя с элементом управления, Button использует события `mouseUp` и `mouseDown`. Элемент управления Scroller использует события `mouseMove` и `mouseUp` для указания прокрутки экрана, выполняемой пользователем.

Разработчик и евангелист компании Adobe Paul Trani объясняет процесс обработки событий жестов и касаний в статье Touch Events and Gesture on Mobile.

Управление событиями, созданными в AIR

Свойство `flash.ui.Multitouch.inputMode` управляет событиями, создаваемыми в AIR и Flash Player. Свойство `flash.ui.Multitouch.inputMode` может иметь одно из следующих значений:

- `MultitouchInputMode.NONE` AIR отправляет события мыши, но не события касания или жестов.
- `MultitouchInputMode.TOUCH_POINT` AIR отправляет события мыши и касания, но не события жестов. В этом режиме инфраструктура Flex получает те же события мыши, что и при `MultitouchInputMode.NONE`.
- `MultitouchInputMode.GESTURE` AIR отправляет события мыши и жестов, но не события касания. В этом режиме инфраструктура Flex получает те же события мыши, что и при `MultitouchInputMode.NONE`.

Как видно из списка, независимо от настройки свойства `flash.ui.Multitouch.inputMode` AIR всегда отправляет события мыши. Поэтому компоненты Flex всегда реагируют на взаимодействие пользователя с сенсорным экраном.

При работе с Flex в приложении может использоваться любое значение свойства `flash.ui.Multitouch.inputMode`. Хотя компоненты Flex не реагируют на события касания или жестов, в приложение можно добавить дополнительные функции, обеспечивающие ответ на любые события. Например, для обработки событий касания в элемент управления Button добавляется обработчик таких событий, как `touchTap`, `touchOver` и `touchMove`.

В «Руководстве разработчика по ActionScript 3.0» содержится дополнительная информация об обработке способов ввода данных на различных устройствах и о работе с событиями касания, множественного касания и жестов. Дополнительные сведения см. в разделе:

- Основы взаимодействия пользователя с системой
- Ввод с помощью прикосновения, нескольких прикосновений или жестов

Определение мобильного приложения и заставки экрана

Создание контейнера мобильного приложения

В мобильном приложении может использоваться один из первых тегов, указанных ниже.

- Тег `<s:ViewNavigatorApplication>` определяет мобильное приложение с одним разделом.
- Тег `<s:TabbedViewNavigatorApplication>` определяет мобильное приложение с несколькими разделами.

При разработке приложений для планшета ограничения размера экрана не так важны, как для телефонов. Поэтому нет необходимости учитывать малый размер экрана в структуре приложений для планшетов. В разрабатываемых приложениях можно использовать стандартный контейнер Spark Application с поддерживаемыми мобильными компонентами и темами оформления.

Примечание. *При разработке любого мобильного приложения (даже для телефонов) можно использовать контейнер Spark Application. Однако контейнер Spark Application не поддерживает навигацию представлений, сохраняемость данных и работу кнопок возврата и меню устройства. Для получения подробной информации см. разделы «Различия между контейнерами мобильного приложения и контейнером Spark Application» на странице 37 и О контейнере Application.*

Контейнеры мобильного приложения могут обладать следующими характеристиками:

Характеристика	**Контейнеры Spark ViewNavigatorApplication и TabbedViewNavigatorApplication**
Размер по умолчанию	Высота 100% и ширина 100%, что позволяет заполнить всю доступную область экрана.
Нижестоящий макет	Определяется отдельными контейнерами View, которые составляют представления приложения.
Заполнение по умолчанию	0 пикселов.
Полосы прокрутки	Отсутствует. При добавлении полосы прокрутки в тему оформления контейнера приложения пользователи могут прокручивать все приложение, в том числе область ActionBar и панель вкладок приложения. Обычно прокручивание этих областей не требуется. Поэтому полосы прокрутки следует добавлять к отдельным контейнерам View приложения, а не к теме оформления контейнера приложения.

Различия между контейнерами мобильного приложения и контейнером Spark Application

Контейнеры мобильного приложения Spark и контейнер Spark Application имеют множество общих функций. Например, в контейнерах мобильного приложения и контейнере Spark Application стили применяются аналогичным способом.

Контейнеры мобильного приложения Spark имеют несколько характеристик, которые отличаются от контейнера Spark Application.

- **Поддержка сохраняемости**

  Поддерживает сохранение и загрузку данных с диска. Механизм сохраняемости позволяет восстановить состояние приложения после прерывания его выполнения, например если пользователю необходимо ответить на телефонный звонок.

- **Поддержка навигации по представлениям**

  Контейнер ViewNavigatorApplication автоматически создает один контейнер ViewNavigator для управления переходом между представлениями.

Контейнер TabbedViewNavigatorApplication автоматически создает один контейнер TabbedViewNavigator для управления переходом между разделами.

- **Поддержка кнопок возврата и меню устройства**

  Когда пользователь нажимает кнопку возврата, приложение открывает предыдущее представление в стеке. При нажатии кнопки меню появляется предварительно определенный контейнер ViewMenu текущего представления.

Для получения более подробной информации о контейнере Spark Application см. раздел О контейнере Application.

Обработка событий на уровне приложения

Класс NativeApplication представляет приложение AIR. Он предоставляет сведения о приложении и его функциях, а также отправляет события на уровне приложения. Для доступа к экземпляру класса NativeApplication, соответствующего мобильному приложению, используется статическое свойство `NativeApplication.nativeApplication`.

Например, класс NativeApplication определяет события `invoke` и `exiting`, обрабатываемые в мобильном приложении. В примере ниже класс NativeApplication определяет обработчик для события `exiting`:

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

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

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

</s:ViewNavigatorApplication>
```

Обратите внимание, что для доступа к ViewNavigator требуется установить свойство `ViewNavigatorApplication.navigator`.

Добавление экрана заставки в приложение

Контейнер Spark Application является базовым классом для контейнеров ViewNavigatorApplication и TabbedViewNavigatorApplication. Контейнер Spark Application при использовании с темой Spark поддерживает средство предварительной загрузки приложения, отображающее ход выполнения загрузки и инициализации файла SWF приложения.

При использовании с мобильной темой может отображаться заставка экрана. Заставка экрана отображается при запуске приложения.

Примечание. *Чтобы использовать заставку экрана в настольном приложении, установите для свойства* `Application.preloader` *значение spark.preloaders.SplashScreen. Также необходимо добавить frameworks\libs\mobile\mobilecomponents.swc в путь к библиотеке приложения.*


Автор блога Joseph Labrecque рассказывает об использовании AIR для заставки экрана Android в Flex.


В видеоролике автора Brent Arnold предлагаются способы добавления заставки экрана в приложение Android.

Добавление заставки экрана из файла изображения

Заставку экрана можно загрузить непосредственно из файла изображения. Для настройки заставки экрана используются свойства `splashScreenImage`, `splashScreenScaleMode` и `splashScreenMinimumDisplayTime` класса приложения.

Например, в следующем примере рассматривается загрузка заставки экрана из файла JPG в формате Letterbox:

```
<?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>
```

Добавление заставки экрана из пользовательского компонента

В примере, приведенном в предыдущем разделе, рассматривалась загрузка заставки экрана из файла JPG. Недостатком такого механизма является то, что приложение использует одно и то же изображение независимо от возможностей мобильного устройства, на котором оно запущено.

Мобильные устройства имеют разные разрешения и размеры экранов. Вместо использования одного изображения в качестве заставки экрана можно определить пользовательский компонент. Компонент будет определять возможности мобильного устройства и использовать в качестве заставки экрана соответствующее изображение.

Для определения пользовательского компонента используйте класс SplashScreenImage, как показано в следующем примере:

```
<?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>
```

В определении компонента используйте класс SplashScreenImageSource для определения каждого из изображений заставки экрана. Свойство `SplashScreenImageSource.source` указывает файл изображения. Свойства `dpi`, `aspectRatio` и `minResolution` класса SplashScreenImageSource определяют возможности мобильного устройства, необходимые для вывода изображения.

Например, первое определение SplashScreenImageSource указывает только свойство `source` изображения. Поскольку настройки для свойств `dpi`, `aspectRatio` и `minResolution` отсутствуют, это изображение можно использовать на любом устройстве. Следовательно, это изображение является изображением по умолчанию, которое отображается в случаях, когда ни одно из изображений не соответствует возможностям устройства.

Второе и третье определения SplashScreenImageSource указывают изображение для устройства с 240 DPI в книжной или альбомной ориентации.

Конечное определение SplashScreenImageSource указывает изображение для устройства с 160 DPI в книжной ориентации с минимальным разрешением экрана в 960 пикселов. Значение свойства `minResolution` сравнивается с большим из значений свойств `Stage.stageWidth` и `Stage.stageHeight`. Большее из двух значений должно быть равным или превышать значение свойства `minResolution`.

Следующее мобильное приложение использует этот компонент:

```
<?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>
```

Класс SplashScreenImage автоматически определяет изображение, которое наилучшим образом соответствует возможностям устройства. Поиск соответствия основывается на таких свойствах каждого определения SplashScreenImageSource, как `dpi`, `aspectRatio` и `minResolution`.

Ниже представлены действия по поиску наиболее точного соответствия:

1 Укажите все определения SplashScreenImageSource, которые соответствуют параметрам мобильного устройства. Соответствие определяется в следующих случаях:
   - **a** Этот параметр не указан явно в определении SplashScreenImageSource. Например, ни одно из значений свойства `dpi` не соответствует DPI ни одного устройства.
   - **b** Свойства `dpi` или `aspectRatio` должны точно соответствовать параметрам мобильного устройства.
   - **c** Свойство `minResolution` соответствует параметру устройства, если наиболее высокое из значений свойств `Stage.stageWidth` и `Stage.stageHeight` больше или равно `minResolution`.
2 Если устройству соответствует более одного определения SplashScreenImageSource:
   - **a** Выберите определение с наибольшим количеством явных параметров. Например, определение SplashScreenImageSource, указывающее свойства `dpi` и `aspectRatio`, является более точным соответствием, чем определение, указывающее только свойство `dpi`.
   - **b** При наличии нескольких совпадений выберите определение с наиболее высоким значением `minResolution`.
   - **c** При наличии нескольких совпадений выберите то, которое указано первым в компоненте.

Явный выбор изображения для заставки экрана

Метод `SplashScreenImage.getImageClass()` выбирает определение SplashScreenImageSource, которое наилучшим образом соответствует возможностям мобильного устройства. Этот метод можно переопределить, добавив собственную пользовательскую логику, как показано в следующем примере.

В этом примере рассматривается добавление определения SplashScreenImageSource для заставки экрана iOS. В теле метода, переопределяющего метод `getImageClass()`, необходимо определить, выполняется ли приложение в iOS. Если это так, то отображается изображение, определенное для iOS.

Если приложение выполняется не в iOS, вызовите метод `super.getImageClass()`. Этот метод использует реализацию по умолчанию, чтобы определить экземпляр SplashScreenImageSource для отображения:

```
<?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>
```

Определение представлений в мобильном приложении

Мобильное приложение обычно определяет несколько экранов или представлений. Навигация по приложению представляет собой переходы между представлениями.

Сделайте навигацию интуитивно понятной для пользователя вашего приложения. Т.е. при переходе из одного представления в другое пользователи ожидают, что также будет доступна возможность вернуться к предыдущему представлению. Приложение может также содержать кнопку Home или другие инструменты верхнего уровня для перехода к определенным экранам из любого представления в приложении.

Для определения представлений в мобильном приложении используется контейнер View. Управление переходами между представлениями в мобильном приложении выполняется с помощью контейнера ViewNavigator.

Использование pushView() для изменения представлений

Метод `ViewNavigator.pushView()` добавляет новое представление в стек. Для доступа к ViewNavigator используется свойство `ViewNavigatorApplication.navigator`. При этом в приложении отображается новое представление в стеке.

Метод `pushView()` использует следующий синтаксис:

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

где:

- `viewClass` указывает имя класса представления. Как правило, этот класс соответствует файлу MXML, в котором определяется представление.

- `data` указывает любые данные, передаваемые в представление. Этот объект записывается в свойстве `View.data` нового представления.
- `context` указывает произвольный объект, который записывается в свойстве `ViewNavigator.context`. Новое представление может создать ссылку на это свойство и выполнить действие, указанное в значении свойства. Например, от значения `context` зависят различные способы отображения данных в представлении.
- `transition` указывает переход, воспроизводимый при изменении представлений. Для получения подробной информации о переходах представлений см. раздел «Определение переходов в мобильном приложении» на странице 100.

Использование аргумента data для передачи одного Object

Аргумент `data` передает один Object, содержащий все данные, необходимые для нового представления. Для доступа к объекту представление использует свойство `View.data`, как показано в следующем примере:

```
<?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>
```

В этом примере EmployeeView определяется в файле EmployeeView.mxml. Свойство `data` позволяет использовать в представлении имя и фамилию сотрудника, а также идентификатор сотрудника, определенный переданным Object.

Свойство `View.data` будет действительным во время события `add` для объекта View. Для получения подробной информации о жизненном цикле контейнера View см. раздел «Жизненный цикл контейнеров Spark ViewNavigator и View» на странице 51.

Передача данных в первое представление в приложении

Свойства `ViewNavigatorApplication.firstView` и `ViewNavigator.firstView` определяют первое представление в приложении. Для передачи данных в первое представление используется свойство `ViewNavigatorApplication.firstViewData` или `ViewNavigator.firstViewData`.

Передача данных в представление

В следующем примере для определения мобильного приложения используется контейнер ViewNavigatorApplication, который автоматически создает один экземпляр класса ViewNavigator для перехода между представлениями, указанными в приложении.

```
<?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>
```

Этот пример определяет кнопку Home в области навигации элемента управления ActionBar. Когда пользователь нажимает кнопку Home, все представления удаляются из стека и отображается первое представление. Это приложение показано на следующем изображении:

Файл EmployeeMainView.mxml определяет первое представление в приложении, как показано в следующем примере:

```
<?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>
```

Представление определяет элемент управления List, с помощью которого пользователь выбирает имя сотрудника. При выборе имени пользователем обработчик события `change` добавляет экземпляр другого представления с именем EmployeeView в стек. Когда экземпляр EmployeeView добавляется в стек, приложение открывает представление EmployeeView.

В этом примере метод `pushView()` может принимать два аргумента: новое представление и объект, который определяет данные, передаваемые в новое представление. Передаваемый объект данных соответствует текущему выбранному объекту в элементе управления List.

В следующем примере показано определение 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>
```

EmployeeView отображает три поля из поставщика данных элемента управления List. Для доступа к переданным данным EmployeeView использует свойство `View.data`.

Автор блога Steve Mathews предлагает подробные инструкции по передаче данных между представлениями.

Возврат данных из представления

Метод `ViewNavigator.popView()` возвращает элемент управления из текущего представления в предыдущее представление в стеке. При выполнении метода `popView()` текущее представление удаляется и восстанавливается предыдущее представление в стеке. Восстановление предыдущего представления предусматривает сброс свойства `data` из стека.

Для получения подробной информации о жизненном цикле представления, включая события, отправляемые во время создания, см. раздел «Жизненный цикл контейнеров Spark ViewNavigator и View» на странице 51.

Новое представление восстанавливается с оригинальным объектом `data`, который использовался при выключении представления. Поэтому оригинальный объект `data`, как правило, не используется для передачи данных из старого представления в новое. Для этой цели необходимо изменить метод `createReturnObject()` предыдущего представления. Метод `createReturnObject()` возвращает один объект.

Тип возвращаемого объекта

Этот объект, возвращаемый методом `createReturnObject()`, записывается в свойстве `ViewNavigator.poppedViewReturnedObject`. Типом данных свойства `poppedViewReturnedObject` является ViewReturnObject.

ViewReturnObject определяет два свойства: `context` и `object`. Свойство `object` содержит объект, возвращенный методом `createReturnObject()`. Свойство `context` содержит значение аргумента `context`, которое было передано представлению при помещении этого представления в стек навигации с помощью `pushView()`.

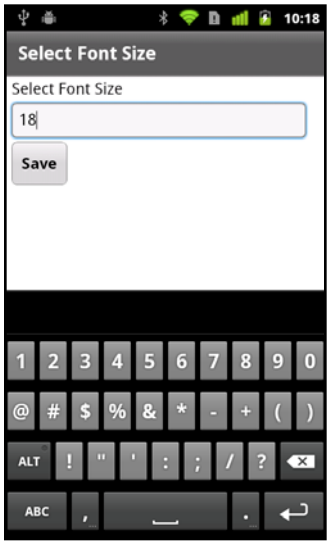
Свойство `poppedViewReturnedObject` будет обязательно установлено в новом представлении до того, как оно получит событие `add`. Если для свойства `poppedViewReturnedObject.object` указано значение null, данные не возвращаются.

Пример: передача данных в представление

В следующем примере SelectFont.mxml содержит представление, в котором можно определить размер шрифта. При изменении метода `createReturnObject()` возвращается значение Number. Поле `fontSize` свойства `data`, которое было передано из предыдущих наборов представлений, устанавливает первоначальное значение элемента контроля TextInput:

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

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

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

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

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

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

На следующем изображении показано представление, определенное SelectFont.mxml:



В следующем примере представление MainFontView.mxml использует представление, определенное в SetFont.mxml. Следующие функции определяются в представлении MainFontView.mxml:

- элемент управления Button в ActionBar, используемый для изменения представления, которое определено в SetFont.mxml;
- Обработчик события `add`, который первым определяет, указано ли для свойства `View.data` значение null. Если указано значение null, обработчик события добавляет поле `data.fontSize` к свойству `View.data`.

  Если для свойства `data` не указано значение null, то обработчик устанавливает размер шрифта в соответствии со значением в поле `data.fontSize`.

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

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

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

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

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

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

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

Настройка приложения для книжной и альбомной ориентации

При изменении положения мобильного устройства ориентация приложения изменяется автоматически. Для настройки отображения приложения при изменении ориентации Flex определяет два состояния представлений, которые соответствуют книжной и альбомной ориентации: `portrait` и `landscape`. Эти состояния представлений позволяют определить характеристики приложения на основе ориентации.

В следующем примере состояние представления управляет свойством `layout` контейнера Group на основе текущей ориентации:

```
<?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>
```

В примере используется представление Search. Контейнер Group управляет макетом вводимых поисковых терминов и кнопкой поиска. В режиме книжной ориентации контейнер Group использует вертикальный макет. Если режим изменяется на альбомную ориентацию, контейнер Group использует горизонтальный макет.

Определение пользовательской темы оформления для поддержки режимов макета

Для мобильного приложения можно указать пользовательский класс темы оформления. Если тема оформления поддерживает макеты книжной и альбомной ориентации, то она должна обрабатывать состояния представлений `portrait` и `landscape`.

Приложение можно настроить таким образом, чтобы ориентация макета не изменялась при повороте устройства. Для этого измените следующие свойства в файле XML приложения, имя которого заканчивается на -app.xml:

- чтобы выключить изменение ориентации макета приложения, укажите для свойства `<autoOrients>` значение `false`;
- чтобы установить ориентацию, укажите для свойства `<aspectRatio>` значение `portrait` или `landscape`.

Установка режима перекрытия для контейнера Spark ViewNavigator

По умолчанию панель вкладок и элемент управления ActionBar мобильного приложения определяют область, которая не может использоваться представлениями приложения. Это означает, что содержимое не может использовать полноэкранный режим мобильного приложения.

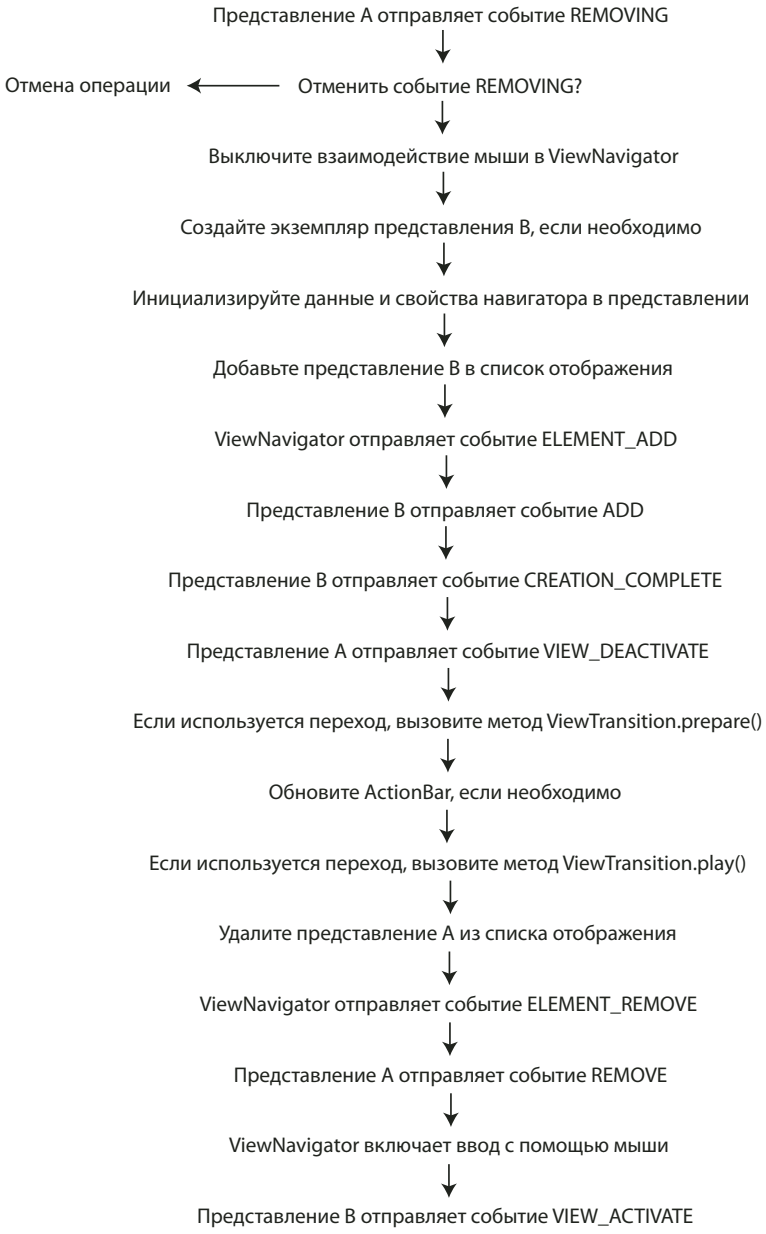
Однако для изменения стандартного макета компонентов можно использовать свойство `ViewNavigator.overlayControls`. Если для свойства `overlayControls` указано `true`, размеры области содержимого приложения совпадают с шириной и высотой экрана. Элемент управления ActionBar и панель вкладок размещаются над областью содержимого и отображаются частично прозрачными в соответствии с установленным значением их альфа-каналов.

Класс темы оформления spark.skins.mobile.ViewNavigatorSkin для контейнера ViewNavigator определяет состояния представлений, используемые для обработки различных значений свойства `overlayControls`. Если для свойства `overlayControls` указано значение `true`, к имени текущего представления добавляется AndOverlay. Например, тема оформления ViewNavigator по умолчанию использует состояние в книжной ориентации. Если для свойства `overlayControls` указано значение `true`, состояние темы оформления навигатора изменяется на portraitAndOverlay.

Жизненный цикл контейнеров Spark ViewNavigator и View

Для переключения представлений в мобильном приложении Flex выполняет последовательность операций. В ходе этого переключения Flex отправляет события, которые пользователь может отслеживать, чтобы управлять действиями при переключении. Например, событие `removing` может отменить переход между представлениями.

Следующая диаграмма отображает процесс перехода от текущего представления A к представлению B.



Определение вкладок в мобильном приложении

Определение разделов приложения

Контейнер TabbedViewNavigatorApplicationопределяет мобильное приложение с несколькими разделами. Контейнер TabbedViewNavigatorApplication автоматически создает контейнер TabbedViewNavigator. Для поддержки навигации по разделам приложения контейнер TabbedViewNavigator создает панель вкладок.

Каждый контейнер ViewNavigator определяет отдельный раздел приложения. Для указания контейнеров ViewNavigator используется свойство `navigators` контейнера TabbedViewNavigatorApplication.

В следующем примере определяются три раздела в соответствии с тремя тегами ViewNavigator. Каждый ViewNavigator определяет первое представление, которое отображается при переходе к этому экрану:

```
<?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>
```

Примечание. *Нижестоящий тег `navigators` не требуется определять в MXML, так как он является стандартным свойством TabbedViewNavigator.*

Каждый ViewNavigator обрабатывает отдельный стек навигации. Таким образом, методы ViewNavigator, например `pushView()` и `popView()`, зависят от раздела, который активен в настоящее время. Кнопка возврата на мобильном устройстве возвращает элемент управления к предыдущему представлению в стеке текущего ViewNavigator. Изменение представления не изменяет текущий раздел.

Поэтому нет необходимости добавлять логику перехода между разделами в приложении. Для управления навигацией по разделам контейнер TabbedViewNavigator автоматически создает панель вкладок в нижней части окна приложения.

Необязательным действием является добавление программного элемента управления текущим разделом. Чтобы изменить разделы программным путем, установите для свойства `TabbedViewNavigator.selectedIndex` индекс требуемого раздела. Индексы разделов отсчитываются от 0, т. е. индексом первого раздела в приложении является 0, индексом второго раздела - 1 и т. д.


Сертифицированный специалист Adobe по продуктам Flex Brent Arnold предлагает видеоролик об использовании стека навигации ViewNavigator.


Специалист по продукции Adobe Holly Schinsky в своей статье Flex Mobile Development - Passing Data Between Tabs описывает способы передачи данных между вкладками мобильного приложения.


Ознакомьтесь с предоставленным video2brain видеороликом о контейнере TabbedViewNavigator в разделе Creating a Tabbed View Navigator Application.

Обработка событий изменения разделов

При изменении раздела контейнер TabbedViewNavigator отправляет перечисленные ниже события.

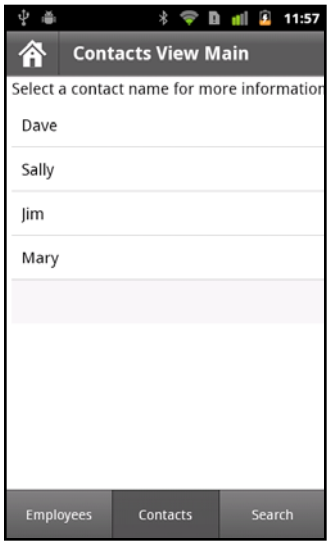
- Событие `changing` отправляется непосредственно перед изменением раздела. Для предотвращения изменения вызовите метод `preventDefault()` в обработчике события `changing`.
- Событие `change` отправляется непосредственно после изменения раздела.

Настройка ActionBar с несколькими разделами

Элемент управления ActionBar связан с ViewNavigator. Поэтому настройку ActionBar для каждого раздела можно выполнять при определении ViewNavigator раздела. В следующем примере ActionBar настраивается отдельно для каждого контейнера ViewNavigator, который определяет три различные раздела приложения:

```
<?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>
```

На следующем рисунке показано это приложение (на панели вкладок этого приложения выбрана вкладка «Contacts»):



Элемент ActionBar также можно определить в каждом представлении приложения. Таким образом, каждое представление использует одинаковое содержимое ActionBar независимо от того, где оно используется в приложении.

Управление панелью вкладок

Скрытие элемента управления панелью вкладок в представлении

Чтобы скрыть панель вкладок в любом представлении, укажите для свойства `View.tabBarVisible` значение `false`. По умолчанию панель вкладок отображается, т. е. для свойства `tabBarVisible` указано значение `true`.

Для управления значениями видимости используйте методы `TabbedViewNavigator.hideTabBar()` и `TabbedViewNavigator.showTabBar()`.

В видеоролике сертифицированного специалиста Adobe по продуктам Flex Brent Arnold рассматриваются способы скрытия панели вкладок.

Применение эффекта к панели вкладок контейнера TabbedViewNavigator

По умолчанию панель вкладок для своих эффектов скрытия и отображения использует эффект перемещения. Если изменяется текущая выбранная вкладка, эффекты для панели вкладок не выполняются.

Чтобы изменить стандартный эффект панели вкладок для эффектов отображения или скрытия, переопределите методы `TabbedViewNavigator.createTabBarHideEffect()` и `TabbedViewNavigator.createTabBarShowEffect()`. После скрытия панели вкладок не забудьте установить для ее свойств `visible` и `includeInLayout` значения `false`.

Создание нескольких панелей в мобильном приложении

SplitViewNavigator – это контейнер с изменяемым оформлением, который на одном экране мобильного устройства отображает два или более нижестоящих навигаторов представлений. Каждый навигатор представлений отображается в отдельной панели, управляемой контейнером SplitViewNavigator.

Нижестоящим элементом контейнера SplitViewNavigator может быть любой компонент, который расширяет ViewNavigatorBase. Таким образом, в качестве нижестоящих элементов контейнера SplitViewNavigator можно использовать контейнеры ViewNavigator и TabbedViewNavigator.

Примечание. *Поскольку для одновременного отображения нескольких панелей требуется достаточно большая область экрана, корпорация Adobe рекомендует использовать контейнер SplitViewNavigator только на планшетных устройствах.*

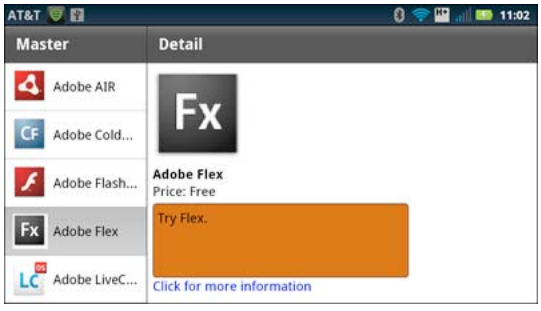
По умолчанию контейнер SplitViewNavigator располагает соответствующие его нижестоящим элементам панели горизонтально. Вместо этого можно указать использование вертикального или пользовательского макета.

Создание контейнера SplitViewNavigator

Стандартным шаблоном интерфейса для планшетных устройств является шаблон «Основной-подробности». Этот шаблон разделяет экран на две основные области содержимого: основную панель и панель подробностей. Как правило, пользователь работает с основной панелью для отображения содержимого на панели подробностей.

Каждая панель отвечает нижестоящему элементу контейнера SplitViewNavigator, при этом нижестоящими элементами являются контейнеры ViewNavigator или TabbedViewNavigator. Навигатор представлений для каждой панели содержит собственный стек представлений и строку меню, поскольку его нижестоящие элементы являются навигаторами представлений.

На следующем изображении показан контейнер SplitViewNavigator в приложении с основной панелью и панелью подробностей:



В этом примере расположенная слева основная панель содержит элемент управления Spark List, который отображает набор продуктов Adobe. Расположенная справа панель подробностей отображает дополнительные сведения о выбранном на основной панели продукте.

Ниже приведен файл главного приложения для этого примера:

```
<?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>
```

Контейнер SplitViewNavigator может быть нижестоящим элементом контейнера Application или TabbedViewNavigatorApplication. В этом примере контейнер SplitViewNavigator является единственным нижестоящим элементом контейнера Application. Обратите внимание, что для высоты и ширины контейнера SplitViewNavigator указано значение 100%, что позволяет ему занимать всю область экрана устройства.

В этом примере нижестоящими элементами контейнера SplitViewNavigator являются контейнеры ViewNavigator. Первый контейнер ViewNavigator определяет основную панель, а второй – панель подробностей.

Примечание. *Контейнер SplitViewNavigator может иметь более двух нижестоящих элементов. Таким образом, можно создать контейнер SplitViewNavigator с тремя , четырьмя или большим количеством панелей.*

Доступ к панелям и представлениям контейнера SplitViewNavigator

Контейнер SplitViewNavigator определяет методы и свойства, которые используются для доступа к его нижестоящим элементам. Например, свойство `SplitViewNavigator.numViewNavigators` используется для определения количества нижестоящих навигаторов представлений контейнера SplitViewNavigator.

Метод `SplitViewNavigator.getViewNavigatorAt()` используется для получения доступа к нижестоящим элементам контейнера SplitViewNavigator на основе их индексов. В приведенном выше примере контейнер ViewNavigator основной панели имеет индекс 0, а контейнер ViewNavigator панели подробностей - индекс 1.

Примечание. *Контейнер SplitViewNavigator наследует методы getElementAt() и getElementIndex(). Не используйте эти методы с SplitViewNavigator. Вместо этого используйте getViewNavigatorAt().*

Контейнер SplitViewNavigator получает доступ к представлениям отдельной панели из ссылки этой панели на контейнер ViewNavigator.

Доступ к контейнеру SplitViewNavigator из нижестоящего элемента можно получить с помощью свойства `parentNavigator` нижестоящего элемента. Например, `ViewNavigator.parentNavigator` содержит ссылку на вышестоящий контейнер SplitViewNavigator.

Контейнер View получает доступ к вышестоящему навигатору представлений при помощи свойства `View.navigator`. Таким образом, представление может получить доступ к SplitViewNavigator при помощи `View.navigator.parentNavigator`.

В приведенном ранее примере контейнер ViewNavigator основной панели указывает MasterCategory как свое первое представление. Это представление определяется в файле MasterCategory.mxml, как указано ниже:

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

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

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

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

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

MasterCategory.mxml определяет один элемент управления List, который содержит сведения о продуктах Adobe. Элемент управления List использует пользовательское средство визуализации для отображения метки и значка для каждого продукта. Для получения подробной информации об определении средств визуализации см. раздел Использование мобильного средства визуализации с элементом управления Spark на основе списка.

Элемент управления List основной панели использует событие `change` для обновления панели подробностей в ответ на действие пользователя. Обработчик событий сначала получает ссылку на контейнер SplitViewNavigator. Затем из полученной ссылки обработчик событий получает ссылку на контейнер ViewNavigator кадра подробностей.

После этого обработчик событий вызывает метод `push()` для контейнера ViewNavigator кадра подробностей. Метод `push()` принимает два аргумента – представление, перемещенное в стек контейнера ViewNavigator, и объект, содержащий информацию о выбранном элементе List.

Обновление панели подробностей контейнера SplitViewNavigator

В приведенном выше примере панель подробностей отображает информацию о пункте, выбранном в элементе управления List основной панели. Панель подробностей называется DetailView и определяется в файле DetailView.mxml, как указано ниже:

```
<?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>
```

Основная панель передает объект в файл DetailView.mxml, соответствующий пункту, выбранному в элементе управления List. Панель подробностей получает доступ к этим данным при помощи свойства `View.data`. Затем панель подробностей выводит изображение и сведения о продукте, а также создает гиперссылку на страницу с более подробной информацией.

Для получения подробной информации о передаче данных в контейнер View см. раздел «Передача данных в представление» на странице 43.

Отображение панелей в зависимости от ориентации устройства

При разработке приложения для планшетного устройства можно использовать макет, основанный на ориентации. Например, в альбомной ориентации планшетное устройство имеет широкую доступную область экрана, на которой могут отображаться несколько панелей. В книжной ориентации, когда доступная область экрана достаточно узкая, можно выбрать скрытие панели.

Контейнер SplitViewNavigator определяет свойство `autoHideFirstViewNavigator`, используемое для управления видимостью первой панели для различных ориентаций. По умолчанию для свойства `autoHideFirstViewNavigator` указано значение `false`, поэтому контейнер отображает первую панель независимо от ориентации.

При изменении значения свойства `autoHideFirstViewNavigator` на `true` контейнер отображает первую панель в альбомной ориентации и скрывает ее в книжной. Контейнер SplitViewNavigator скрывает первую панель, указывая для свойства `visible` соответствующего навигатора представлений значение `false`.

Чтобы открыть скрытую в книжной ориентации первую панель, используйте метод `SplitViewNavigator.showFirstViewNavigatorInPopUp()`. При вызове этого метода первая панель открывается в контейнере Callout. Контейнер выноски является всплывающим контейнером, который отображается поверх приложения, как показано на следующем изображении:

В этом примере к панели действий на панели подробностей SplitViewNavigator добавлена кнопка «Показать навигатор». Когда первая панель контейнера скрыта, пользователь может использовать эту кнопку, чтобы открыть основную панель.

Примечание. *Для открытия первой панели создайте пользовательскую тему оформления для SplitViewNavigator в SkinnablePopUpContainer или в подклассе SkinnablePopUpContainer.*

Метод `showFirstViewNavigatorInPopUp()` игнорируется, если компонент Callout уже открыт. При изменении ориентации устройства на альбомную выноска автоматически закрывается и снова отображается первая панель.

Чтобы закрыть контейнер Callout, щелкните область за его пределами. Также этот контейнер можно закрыть, вызвав метод `SplitViewNavigator.hideViewNavigatorPopUp()`.

Для получения подробной информации о контейнере Callout см. раздел «Добавление контейнера выноски в мобильное приложение» на странице 87.

Добавление панели действий на панель, отображаемую в контейнере Callout

Ниже приведен файл главного приложения, который для свойства `autoHideFirstViewNavigator` контейнера SplitViewNavigator указывает значение `true`. В этом примере рассматривается использование состояний представлений для добавления кнопки к панели действий панели подробностей при книжной ориентации устройства:

```
<?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>
```

Приложение добавляет обработчик событий для события `resize` в контейнер Application. При изменении ориентации планшетного устройства Flex передает событие `resize`. В обработчике событий для события resize состояние представления приложения определяется на основе текущей ориентации. Для получения подробной информации о состояниях представлений см. раздел Состояния представлений.

Навигатор представлений для панели подробностей использует текущее состояние для управления внешним видом элемента управления Button на панели действий. В альбомной ориентации кнопка скрыта, поскольку отображается основная панель.

В книжной ориентации, когда основная панель скрыта, элемент управления Button отображается на панели действий панели подробностей. Выбор пользователем элемента управления Button позволяет открыть контейнер Callout, содержащий основную панель.

Ссылка на элемент управления Button передается как аргумент в метод `showFirstViewNavigatorInPopUp()`. Этот аргумент определяет хост контейнера Callout. Это значит, что контейнер Callout размещается относительно элемента управления Button.

Закрытие выноски SplitViewNavigator в ответ на действия пользователя

Чтобы закрыть контейнер Callout, щелкните область за его пределами. Щелчок внутри контейнера Callout не приводит к закрытию.

В этом примере рассматривается закрытие контейнера Callout при выборе пользователем элемента List с помощью вызова метода `SplitViewnavigator.hideViewNavigatorPopUp()`. Этот метод вызывается в обработчике событий для события `change` элемента управления List, как показано далее:

```
<?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>
```

Реализация сохраняемости для контейнера SplitViewNavigator

Выполнение приложения на мобильном устройстве часто прерывается другими функциями, например входящими текстовыми сообщениями, телефонными звонками или действиями других мобильных приложений. Приложение, выполнение которого прервано, обычно перезапускается, при этом пользователь ожидает, что будет восстановлено предыдущее состояние приложения. Механизм сохраняемости позволяет устройству восстановить приложение в его предыдущем состоянии. Для получения подробной информации см. раздел «Включение функции сохраняемости в мобильном приложении» на странице 133.

SplitViewNavigator реализует методы `loadViewData()` и `saveViewData()`, унаследованные из базового класса ViewNavigatorBase. Следовательно, SplitViewNavigator может выполнять сериализацию и десериализацию стека навигации и данных представления для каждого из своих нижестоящих навигаторов.

Однако в случае прерывания работы приложения эти методы необходимо вызвать вручную, как показано в следующем примере:

```
<?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>
```

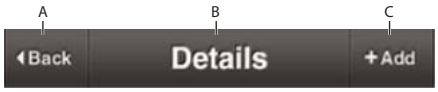
Определение элементов управления навигацией, заголовком и действиями в мобильном приложении

Настройка элемента управления ActionBar

Контейнер ViewNavigator определяет элемент управления ActionBar, который предоставляет стандартную область для заголовка и элементов управления навигацией и действиями. Он позволяет определить глобальные элементы управления, которые могут использоваться в любых компонентах приложения, и элементы управления, которые зависят от представления. Например, с помощью элемента управления ActionBar можно добавить кнопку возврата к началу, кнопку поиска и другие функции.

В мобильном приложении с одним разделом, т. е. с одним контейнером ViewNavigator, для всех представлений используется одна панель действий. В мобильном приложении с несколькими разделами, т. е. с несколькими контейнерами ViewNavigator, в каждом разделе определяется собственная панель действий.

Элемент управления ActionBar определяет область панели действий. При этом элемент управления ActionBar определяет три различных области, как указано в примере ниже:



***A**. Область навигации **B**. Область заголовка **C**. Область действий*

Области ActionBar

- **Область навигации**

  Содержит компоненты для навигации по разделу. Например, можно указать кнопку возврата в области навигации.

  Свойство `navigationContent` используется для определения компонентов, которые отображаются в области навигации. Для определения макета области навигации используется свойство `navigationLayout`.

- **Область заголовка**

  Содержит строку с текстом заголовка или компоненты. Если указаны компоненты, необходимо определить строку заголовка.

  Свойство `title` используется для указания строки, отображаемой в области заголовка. Для определения компонентов, отображаемых в области заголовка, используется свойство `titleContent`. С помощью свойства `titleLayout` можно определить макет области заголовка. Если указывается значение для свойства `titleContent`, тема оформления ActionBar игнорирует свойство `title`.

- **Область действий**

  Содержит компоненты, определяющие действия, которые пользователь может выполнять в представлении. Например, в качестве компонента области действий можно указать кнопку «Поиск» или «Обновить».

  Для определения компонентов, отображаемых в области действий, используется свойство `actionContent`. С помощью свойства `actionLayout` можно определить макет области действий.

Компания Adobe рекомендует использовать области навигации, заголовка и действий указанными выше способами, при этом пользователи могут размещать любые компоненты в пределах этих областей.

Установка свойств ActionBar в контейнерах ViewNavigatorApplication, ViewNavigator или View

Свойства, определяющие содержимое элемента управления ActionBar, можно установить в контейнерах ViewNavigatorApplication, ViewNavigator или отдельных контейнерах View. Наиболее высокий приоритет определен для контейнера View, после которого следует ViewNavigator и затем ViewNavigatorApplication. Таким образом, свойства, установленные в контейнере ViewNavigatorApplication, применяются ко всему приложению, но могут быть изменены в контейнерах ViewNavigator или View.

Примечание. *Элемент управления ActionBar связан с контейнером ViewNavigator и поэтому является специфическим для отдельного раздела мобильного приложения. Поэтому для настройки ActionBar не могут использоваться контейнеры TabbedViewNavigatorApplication и TabbedViewNavigator.*

Пример настройки элемента управления Spark ActionBar в приложении

Ниже приведен пример файла главного приложения в мобильном проекте:

```
<?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>
```

Пример определяет кнопку Home в области содержимого навигации элемента управления ActionBar и кнопку Refresh в области содержимого действий.

В примере ниже представлен контейнер MobileViewHome View, с помощью которого создается начальное представление приложения. Контейнер View определяет строку заголовка Home View, но не изменяет области содержимого навигации или действий в элементе управления ActionBar:

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

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

Пример настройки элемента управления ActionBar в контейнере View

В примере используется файл главного приложения с одним разделом, который определяет кнопку Home в области навигации контейнера ViewNavigatorApplication и кнопку Search в области действий.

```
<?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>
```

Первым представлением этого приложения является MobileViewHomeOverride, которое определяет элемент управления Button, используемый для перехода ко второму контейнеру View, который содержит информацию о странице Search:

```
<?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>
```

Контейнер View, который определяет страницу Search, изменяет области заголовка и действий элемента управления ActionBar, как указано ниже:

```
<?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>
```

На изображении ниже представлен элемент управления ActionBar для этого представления.

Поскольку представление Search не изменяет область навигации элемента управления ActionBar, в этой области по-прежнему отображается кнопка Home.

Скрытие элемента управления ActionBar

Чтобы скрыть элемент управления ActionBar в любом представлении, укажите для свойства `View.actionBarVisible` значение `false`. По умолчанию элемент управления ActionBar отображается, т. е. для свойства `actionBarVisible` указано значение `true`.

В примере ниже метод `ViewNavigator.hideActionbar()` используется для скрытия элемента управления ActionBar во всех представлениях, управляемых контейнером ViewNavigator:

```
<?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>
```

При отображении или скрытии элемента управления ActionBar можно применять пользовательские эффекты. По умолчанию эффект Animate используется для скрытия или отображения элемента управления, ActionBar. Для изменения эффекта по умолчанию отредактируйте методы `ViewNavigator.createActionBarHideEffect()` и `ViewNavigator.createActionBarShowEffect()`. После воспроизведения эффекта, при котором скрывается ActionBar, укажите для его свойств `visible` и `includeInLayout` значение `false`, чтобы предотвратить его включение в макет представления.

Использование полос прокрутки в мобильном приложении

Рекомендации по использованию полос прокрутки в мобильном приложении

Как правило, в приложении отображаются полосы прокрутки, если размер содержимого превышает видимую область экрана. Для добавления полос прокрутки к приложению используется элемент управления Scroller. Некоторые компоненты, например элемент управления Spark List, поддерживают прокрутку без необходимости использования компонента Scroller. Более подробные сведения см. в разделеПрокрутка контейнеров Spark.

Областью щелчка полосы прокрутки является область экрана, в которую помещается курсор мыши для выполнения прокрутки. В настольном приложении или приложении на основе обозревателя областью щелчка является видимая область полосы прокрутки. В мобильных приложениях полосы прокрутки скрываются, даже если размер содержимого превышает видимую область экрана. Это необходимо для того, чтобы приложение заполняло весь экран по высоте и ширине.

Мобильное приложение должно отличать взаимодействие пользователя с элементом управления, таким как Button, от выполнения прокрутки. При использовании полос прокрутки в мобильном приложении также необходимо учитывать, что компоненты Flex часто изменяют свой внешний вид в ответ на действия пользователя.

Например, когда пользователь нажимает элемент управления Button, внешний вид кнопки изменяется и отображает нажатое состояние. Когда пользователь отпускает кнопку, ее внешний вид отображает ненажатое состояние. Однако если пользователь касается элемента Button на экране во время прокрутки, внешний вид кнопки не изменяется.

Инженер Adobe Steven Shongrunden демонстрирует работу с полосами прокрутки в статье Saving scroll position between views in a mobile Flex Application.

Термины, описывающие функцию прокрутки

Следующие термины используются для описания прокрутки в мобильном приложении

Содержимое - для прокручиваемого компонента (например, контейнера Group или элемента управления List) – вся область компонента. В зависимости от размера экрана и макета приложения видимым может быть только подмножество содержимого.

Область просмотра - подмножество области содержимого отображаемого компонента.

Перетаскивание - пользователь прикасается к прокручиваемой области и передвигает содержимое пальцем.

Скорость - скорость и направление перемещения содержимого при перетаскивании. Измеряется в пикселах в миллисекунду по осям X и Y.

Бросок - когда перемещение содержимого достигает определенной скорости, пользователь прекращает прикосновение, в то время как прокручиваемое содержимое продолжает движение.

Отскакивание - пользователь перемещает область просмотра прокручиваемого компонента за пределы содержимого этого компонента. В результате отображается пустая область просмотра. Если пользователь прекратил прикосновение или скорость броска достигла нуля, область просмотра отскакивает в точку покоя, где снова заполняется содержимым. При возвращении области просмотра в точку покоя скорость движения снижается, и область просмотра плавно останавливается.

Режимы прокрутки в мобильном приложении

Некоторые прокручиваемые компоненты (например, List и Scroller) поддерживают различные типы прокрутки в зависимости от значений таких свойств компонента, как `pageScrollingEnabled` и `scrollSnappingMode`. Эти свойства доступны только в случае, когда для стиля `interactionMode` указано значение `touch`.

Следующая таблица содержит описания режимов прокрутки.

pageScrollingEnabled	scrollSnappingMode	Режим
false (по умолчанию)	none (по умолчанию)	По умолчанию прокрутка выполняется на основе пикселов. Окончательным положением прокрутки является любое расположение пиксела в результате жеста перетаскивания или броска. Например, при прокручивании элемента управления List прокручивание завершается, когда пользователь прекращает прикосновение, даже если отображается только часть элемента List.
false	leadingEdge, center, trailingEdge	Прокрутка выполняется на основе пикселов, однако содержимое перемещается в окончательное положение в зависимости от значения `scrollSnappingMode`. Например, при вертикальной прокрутке элемента управления List, когда для `scrollSnappingMode` установлено значение `leadingEdge`, элемент управления List переместится в окончательное положение прокрутки, где расположен верхний элемент списка.
true	none	Прокрутка выполняется постранично. Размер области просмотра прокручиваемого компонента определяет размер страницы. Независимо от выполняемого жеста одновременно прокручивается только одна страница. Чтобы перейти на следующую страницу, необходимопрокрутить не менее 50% видимой области компонента. Если прокручено менее 50%, компонент остается на текущей странице. Кроме того, следующая страница может отображаться при достаточно высокой скорости прокрутки. Если скорость прокрутки невысока, компонент остается на текущей странице. Если размер содержимого не полностью соответствует размеру области просмотра, к последней странице будет добавлено необходимое заполнение.
true	leadingEdge, center, trailingEdge	Прокрутка выполняется постранично, однако компонент перемещается в окончательное положение в зависимости от значения `scrollSnappingMode`. Чтобы обеспечить соответствие режиму привязки, расстояние прокрутки не всегда точно соответствует размеру страницы.

Примеры прокрутки в мобильном приложении

В следующем примере рассматривается использование компонента Scroller, в который вложен контейнер Group мобильного приложения. Контейнер Group имеет нижестоящий элемент управления Image, который содержит большое изображение. После вложения контейнера Group в компонент Scroller изображение становится доступным для прокрутки:

```
<?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>
```

Обратите внимание, что в этом примере пропускаются настройки свойств `pageScrollingEnabled` и `scrollSnappingMode`. Таким образом, в этом примере используется стандартный режим прокрутки на основе пикселов и пользователь может перейти к любому пикселу на изображении.

В следующем примере рассматривается элемент управления List, который определяет свойства `pageScrollingEnabled` и `scrollSnappingMode`:

```
<?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>
```

В этом примере используется постраничная прокрутка со значением привязки `leadingEdge`. Таким образом, прокрутка элемента управления List выполняется постранично. После перехода на следующую страницу элемент управления List перемещается в окончательное положение прокрутки, где расположен верхний элемент списка.

Рекомендации по использованию прокрутки и StageText

StageText позволяет использовать собственные функции ввода текста в мобильном приложении вместо стандартных элементов управления текстовыми полями. Однако прокручиваемый контейнер не может содержать элементы управления вводом текста, такие как TextInput или TextArea, которые используют StageText. Таким образом, чтобы использовать элемент управления вводом текста в прокручиваемом контейнере, необходимо изменить тему оформления так, чтобы в ней не использовался StageText.

Flex поставляется с темами оформления для элементов управления TextInput и TextArea, которые не используют StageText. Используйте указанные ниже темы оформления с этими элементами управления в прокручиваемом контейнере:

- spark.skins.mobile.TextInputSkin – тема оформления для TextInput, которая не использует StageText;
- spark.skins.mobile.TextAreaSkin – тема оформления для TextArea, которая не использует StageText.

В примере ниже представлен контейнер View с элементами управления TextInput и TextArea и прокручиваемым контейнером:

```
<?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>
```

События и полосы прокрутки

Компоненты Flex зависят от событий, указывающих на возникновение действий пользователя. В ответ на действие пользователя компонент может изменить свой вид или выполнить какое-либо другое действие.

Разработка приложений также зависит от событий, определяющих действия пользователя. Например, событие `click` элемента управления Button обычно используется для вызова собственного обработчика событий в ответ на выбор пользователем этой кнопки. Событие `change` элемента управления List запускает обработчик событий, когда пользователь выбирает элемент в списке.

Механизм прокрутки в Flex основан на событии `mouseDown`. Это значит, что механизм прокрутки прослушивает события `mouseDown`, чтобы определить необходимость запуска операции прокрутки.

Определение жеста пользователя в качестве операции прокрутки

Приложение содержит несколько элементов управления Button в прокручиваемом контейнере:

1. Нажмите элемент управления Button пальцем. Элемент Button отправляет событие `mouseDown`.
2. Flex реагирует на действие пользователя с задержкой, длительность которой предварительно определена. Период задержки позволяет установить, что пользователь выбирает кнопку, а не выполняет прокрутку.

   Если во время периода задержки пользователь перемещает палец на расстояние, которое превышает заранее установленное значение, Flex определяет это действие как прокрутку. Расстояние, на которое можно передвинуть палец, чтобы это действие рассматривалось как событие прокрутки, составляет 0,08 пикселов. Расстояние соответствует приблизительно 20 пикселам на устройстве с 252 DPI.

   Поскольку пользователь переместил палец до истечения периода задержки, элемент управления Button не определяет это событие как взаимодействие. Кнопка не отправляет событие и не изменяет свой внешний вид.
3. После истечения периода задержки элемент управления Button распознает действие пользователя. Внешний вид Button изменяется и указывает, что данная кнопка выбрана.

Продолжительность периода задержки определяется в свойстве `touchDelay` элемента управления. Стандартная продолжительность составляет 100 миллисекунд. Если для свойства `touchDelay` указано значение 0, задержка отсутствует и прокрутка выполняется немедленно.

4 После истечения периода задержки и отправки событий мыши в Flex пользователь может передвинуть палец на расстояние, превышающее 20 пикселов. Элемент управления Button возвращает нормальное состояние и начинает действие прокрутки.

В этом случае внешний вид кнопки изменяется, поскольку истек период задержки. Однако если пользователь перемещает палец на расстояние, которое превышает 20 пикселов, даже после истечения периода задержки, Flex определяет это действие как прокрутку.

Примечание. *Компоненты Flex поддерживают различные типы событий, помимо событий мыши. При использовании компонентов необходимо определить, как приложение будет реагировать на эти события. При выполнении события* `mouseDown` *действия пользователя могут определяться неоднозначно. Пользователь может взаимодействовать с компонентом или выполнять прокрутку. Ввиду этой неоднозначности компания Adobe рекомендует прослушивание событий* `click` *или* `mouseUp` *вместо события* `mouseDown`*.*

Обработка событий прокрутки в мобильном приложении

Чтобы сигнализировать о начале операции прокрутки, компонент, отправляющий событие `mouseDown`, отправляет событие восходящей цепочки `touchInteractionStarting`. Если это событие не отменено, компонент отправляет событие восходящей цепочки `touchInteractionStart`.

При обнаружении события `touchInteractionStart` компонент не должен реагировать на жест пользователя. Например, если элемент управления Button обнаруживает событие `touchInteractionStart`, он выключает все визуальные индикаторы, установленные на основе исходного события `mouseDown`.

Если для компонента не требуется запуск прокрутки, он может вызвать метод `preventDefault()` в обработчике события `touchInteractionStarting`.

После завершения операции прокрутки компонент, отправляющий событие `mouseDown`, отправляет событие восходящей цепочки `touchInteractionEnd`.

Поведение прокрутки на основе исходного положения точки касания

В таблице ниже представлены способы обработки прокрутки на основе исходного положения точки касания:

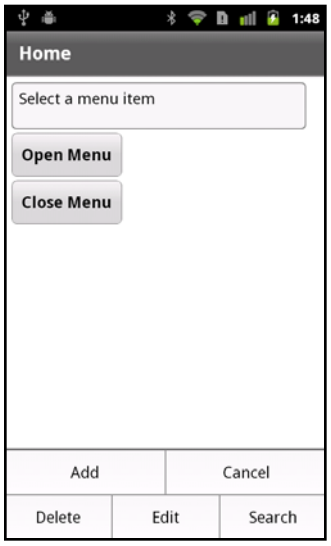
Выбранный элемент	**Поведение**
Пустая область, неизменяемый текст, текст без возможности выбора	Жест не распознает ни один компонент. Перед инициацией прокрутки компонент Scroller ожидает, пока пользователь переместит точку касания более чем на 20 пикселов.

Выбранный элемент	Поведение
Объект в элементе управления List	После периода задержки средство визуализации выбранного элемента изменяет отображение выбранного состояния. Однако если в любое время пользователь перемещает палец на расстояние, превышающее 20 пикселов, внешний вид элемента отображается как обычное состояние и начинается процесс прокрутки.
Button, CheckBox, RadioButton, DropDownList	После истечения периода задержки отображает свое состояние `mouseDown`. Однако, если пользователь перемещает точку касания более чем на 20 пикселов, элемент управления изменяет свой вид для отображения нормального состояния и начинает прокрутку.
Компонент Button в средстве визуализации элементов List	Для средства визуализации элементов никогда не выделяется. Это действие обрабатывается элементами управления Button или Scroller, как и при обычной обработке элемента Button.

Определение меню в мобильном приложении

Контейнер ViewMenu определяет меню в нижней части контейнера View мобильного приложения. Каждый контейнер View определяет собственное меню, специфическое для данного представления.

На следующем изображении показан контейнер ViewMenu в приложении:



Контейнер ViewMenu определяет меню с отдельной иерархией кнопок меню. Поэтому невозможно создать меню с вложенными элементами.

Нижестоящие элементы контейнера ViewMenu рассматриваются как элементы управления ViewMenuItem. Каждый элемент управления ViewMenuItem представляет одну кнопку в меню.

Взаимодействие пользователя с контейнером ViewMenu

Откройте окно меню с помощью аппаратной кнопки меню на мобильном устройстве. Его также можно открыть программным способом.

Выбор кнопки меню закрывает окно полного меню. Элемент управления ViewMenuItem отправляет событие `click`, когда пользователь нажимает кнопку меню.

В открытом окне меню нажмите кнопку возврата или кнопку меню на устройстве, чтобы закрыть меню. Также окно меню закроется, если пользователь коснется области экрана вне окна меню.

Символ вставки представляет собой кнопку меню, которая в настоящее время имеет фокус. Для изменения положения символа вставки используется элемент управления с возможностью выбора пяти направлений или клавиши со стрелками на устройстве. Чтобы выбрать элемент символа вставки, нажмите клавишу Enter устройства или элемент управления с возможностью выбора пяти направлений и затем и закройте меню.

Создание меню в мобильном приложении

С помощью свойства `View.viewMenuItems` можно определить меню для представления. Свойство `View.viewMenuItems` принимает Vector элементов управления ViewMenuItem, как в примере ниже:

```
<?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>
```

В этом примере свойство `View.viewMenuItems` используется для добавления пяти элементов меню, каждый из которых представлен элементом управления ViewMenuItem. Текст, отображаемый в меню для этого элемента, определяется в свойстве `label` всех элементов управления ViewMenuItem.

Учтите, что контейнер ViewMenu не определяется явным образом. Контейнер View автоматически создает экземпляр контейнера ViewMenu, в котором содержатся элементы управления ViewMenuItem.

Использование стиля `icon` элемента управления ViewMenuItem

Элемент управления ViewMenuItem определяет свойство стиля `icon`, которое используется для добавления изображения. Стиль `icon` можно использовать совместно со свойством `label` или независимо от него.

Обработка события `click` элемента управления ViewMenuItem

Каждый элемент управления ViewMenuItem также определяет обработчик события `click`. Когда пользователь выбирает объект, элемент управления ViewMenuItem отправляет событие `click`. В примере ниже для всех элементов меню используется один обработчик событий. Кроме того, для каждого события `click` можно указать отдельный обработчик.

Открытие элемента управления ViewMenuItem программным способом

Окно меню открывается с помощью аппаратной кнопки меню на устройстве. Это приложение также определяет два элемента управления Button, которые открывают и закрывают окно меню программным способом.

Чтобы открыть окно меню программным способом, установите для свойства `viewMenuOpen` контейнера приложения значение `true`. Чтобы закрыть окно меню, установите для этого свойства значение `false`. Свойство `viewMenuOpen` определяется в классе ViewNavigatorApplicationBase, который является базовым классом контейнеров ViewNavigatorApplication и TabbedViewNavigatorApplication.

Применение темы оформления к компонентам ViewMenu и ViewMenuItem

Темы оформления управляют внешним видом компонентов ViewMenu и ViewMenuItem. Классом темы оформления по умолчанию ViewMenu является spark.skins.mobile.ViewMenuSkin. Классом темы оформления по умолчанию ViewMenuItem является spark.skins.mobile.ViewMenuItemSkin.

Автор блога Daniel Demmel рассказывает о создании тем оформления элемента управления ViewMenu в стиле Gingerbread Black.

Для управления внешним видом темы оформления классы тем оформления используют состояния тем оформления, например обычное, закрытое или отключенное состояния. Темы оформления также определяют переходы, т. е. управляют внешним видом меню при изменении состояний.

Для получения подробной информации см. раздел «Основы создания мобильных тем оформления» на странице 179.

Определение макета контейнера ViewMenu

Класс ViewMenuLayout определяет макет меню представления. В зависимости от количества включенных элементов меню может содержать определенное количество строк.

Правила создания макетов ViewMenuItem

Свойство `requestedMaxColumnCount` класса ViewMenuLayout определяет максимальное количество элементов меню в строке. По умолчанию для свойства `requestedMaxColumnCount` указано значение 3.

Ниже представлены правила создания макета в классе ViewMenuLayout.

- Если определено три или менее элементов меню и свойство `requestedMaxColumnCount` по умолчанию содержит значение 3, элементы меню отображаются в одной строке. При этом размеры всех элементов меню совпадают.

  Если определено четыре или более элементов меню, т. е. количество элементов превышает указанное в свойстве `requestedMaxColumnCount` значение, контейнер ViewMenu создает несколько строк.

- Если количество элементов меню кратно значению свойства `requestedMaxColumnCount`, каждая строка содержит равное количество элементов меню. При этом размеры всех элементов меню совпадают.

  Например, для свойства `requestedMaxColumnCount` указано значение 3 и пользователь определяет 6 элементов меню. При этом в меню отображаются две строки, каждая из которых содержит три элемента.

- Если количество элементов меню не является кратным значению свойства `requestedMaxColumnCount`, строки могут содержать различное количество элементов меню. При этом размер элементов меню зависит от количества элементов в строке.

  Например, для свойства `requestedMaxColumnCount` по умолчанию указано значение 3 и пользователь определяет 8 элементов меню. Меню отображает три строки. Первая строка содержит два элемента меню. Вторая и третья строки содержат по три элемента каждая.

Создание пользовательского макета ViewMenuItem

Класс ViewMenuLayout содержит свойства, которые используются для изменения пробелов между элементами меню и определения стандартного количества элементов меню в каждой строке. Если требуется применить пользовательский макет для меню, создайте собственный класс макета.

По умолчанию класс spark.skins.mobile.ViewMenuSkin определяет тему оформления контейнера ViewMenu. Чтобы применить настроенный класс ViewMenuLayout к контейнеру ViewMenu, определите новый класс темы оформления для контейнера ViewMenu.

Стандартный класс ViewMenuSkin включает в себя определение для контейнера Group с именем `contentGroup`, как показано в примере ниже:

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

Класс темы оформления должен также определить контейнер с именем `contentGroup`. Этот контейнер использует свойство `layout` для указания настроенного класса макета.

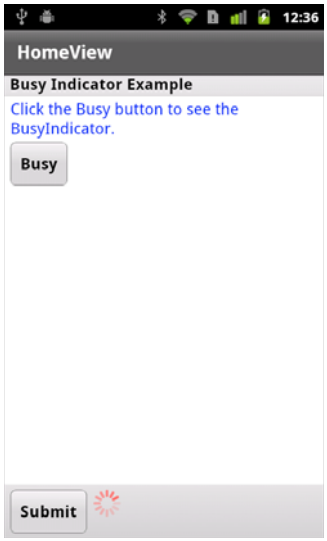
Затем пользовательский класс темы оформления применяется в приложении, как показано в примере ниже:

```
<?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>
```

Отображение индикатора выполнения длительной операции в мобильном приложении

Элемент управления Spark BusyIndicator отображает вращающийся счетчик с 12 линиями. Элемент управления BusyIndicator визуально отображает ход выполнения длительной операции.

В следующем примере элемент управления BusyIndicator расположен в области панели управления контейнера Spark Panel рядом с кнопкой «Submit»:



Для визуализации хода выполнения элемент управления BusyIndicator должен отображаться на экране. После завершения операции этот элемент управления можно скрыть.

Например, создайте экземпляр элемента управления BusyIndicator в обработчике событий, который может запустить длительную операцию. Чтобы добавить элемент управления в контейнер, вызовите метод `addElement()` в обработчике событий. После завершения процесса вызовите метод `removeElement()`, который удалит элемент управления BusyIndicator из контейнера.

Для отображения или скрытия элемента управления также используется соответствующее свойство `visible`. В следующем примере добавленный элемент управления BusyIndicator определяет область панели контейнера Spark Panel в контейнере View:

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

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

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

В этом примере для свойства `visible` элемента управления BusyIndicator изначально установлено значение `false`, которое скрывает этот элемент. Для отображения этого элемента управления нажмите кнопку Busy и установите для свойства `visible` значение `true`.

Элемент управления BusyIndicator вращается только в том случае, если он отображается на экране. Поэтому если для свойства `visible` установлено значение `false`, этот элемент управления не требует циклов обработки.

Примечание. *Если для свойства `visible` установлено значение `false`, элемент управления скрывается, но по-прежнему содержится в макете вышестоящего контейнера. Чтобы исключить этот элемент управления из макета, укажите для свойств `visible` и `includeInLayout` значение `false`.*

Элемент управления Spark BusyIndicator не поддерживает создание тем оформления. Однако для установки цвета и интервала поворота счетчика можно использовать стили. В предыдущем примере цвет индикатора установлен с помощью свойства `symbolColor`.

Добавление переключателя в мобильное приложение

Элемент управления Spark ToggleSwitch определяет простой двоичный переключатель. Элемент управления состоит из ползунка и полосы прокрутки, вдоль которой перемещается ползунок.

Элемент управления ToggleSwitch аналогичен элементам управления ToggleButton и CheckBox. Все эти элементы управления предоставляют выбор между значениями «выбрано» и «не выбрано».

На следующем изображении показан элемент управления ToggleSwitch в приложении:

Элемент управления ToggleSwitch может находиться в двух положениях: выбранном или невыбранном. Элемент управления находится в невыбранном положении, когда ползунок перемещен влево. Выбранному положению соответствует перемещение ползунка вправо. На рисунке переключатель находится в невыбранном положении.

Щелчок в любой области элемента управления переключает его положение. Чтобы изменить положение, можно также перемещать ползунок вдоль дорожки. При отпускании ползунок перемещается в выбранное или невыбранное положение, которое находится ближе к точке, где был отпущен ползунок.

По умолчанию метка OFF соответствует невыбранному положению, а метка ON – выбранному.

Создание элемента управления ToggleSwitch

Ниже рассматривается контейнер View, который определяет элемент управления ToggleSwitch, показанный на предыдущем изображении:

```
<?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>
```

В этом примере первый элемент управления Label отображает положение ползунка с помощью меток ON и OFF. Второй элемент управления Label отображает текущее положение ползунка как значение от 0,0 («не выбрано») до 1,0 («выбрано»).

В этом примере для стиля `slideDuration` также установлено значение 1000. Этот стиль определяет длительность анимации ползунка (в миллисекундах) при перемещении между выбранным и невыбранным положением.

Ниже приведен файл главного приложения:

```
<?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>
```

Изменение стандартной выноски элемента управления ToggleSwitch

В предыдущем примере элемент управления ToggleSwitch использовал значения по умолчанию для меток OFF («не выбрано») и ON («выбрано»). Для настройки меток или других визуальных характеристик элемента управления класс тем оформления необходимо определить как подкласс spark.skins.mobile.ToggleSwitchSkin или создать собственный класс тем оформления.

Следующий класс тем оформления изменяет метки на Yes и 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";
        }
    }
}
```

Следующий контейнер View использует этот класс тем оформления:

```
<?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>
```

Добавление контейнера выноски в мобильное приложение

В мобильном приложении выноска представляет собой контейнер, который отображается в верхней области приложения. Контейнер поддерживает различные типы макетов и может содержать один или несколько типов компонентов.

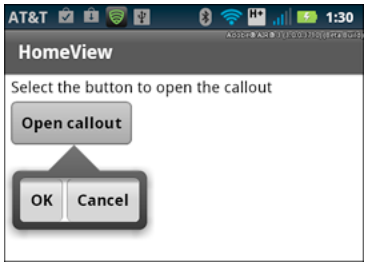
Контейнер выноски может быть модальным или немодальным. Модальный контейнер вплоть до момента своего закрытия принимает все входные данные клавиатуры и мыши. Немодальный контейнер даже в открытом состоянии разрешает другим компонентам приложения принимать входные данные.

В Flex предусмотрены два компонента, которые можно использовать для добавления контейнеров выноски в мобильном приложении: CalloutButton и Callout.

Использование элемента управления CalloutButton для создания контейнера выноски

Элемент управления CalloutButton предоставляет простой способ создания контейнера выноски. Этот компонент позволяет определять компоненты в выноске и устанавливать макет контейнера.

При выборе элемента управления CalloutButton в мобильном приложении открывается контейнер выноски. Flex автоматически проводит стрелку от контейнера выноски к элементу управления CalloutButton, как показано на приведенном далее изображении:



В следующем примере рассматривается мобильное приложение, которое создает показанный на предыдущем изображении элемент управления CalloutButton:

```
<?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>
```

Элемент управления CalloutButton определяет два элемента управления Button, которые отображаются внутри контейнера выноски. Элемент управления CalloutButton также указывает на необходимость использования HorizontalLayout в качестве макета контейнера выноски. По умолчанию контейнер использует BasicLayout.

Открытие и закрытие контейнера выноски с помощью элемента управления CalloutButton

Контейнер выноски открывается при выборе пользователем элемента управления CalloutButton или при вызове метода `CalloutButton.openDropDown()`. Свойства `horizontalPosition` и `verticalPosition` определяют положение контейнера выноски относительно элемента управления CalloutButton. См. пример в разделе «Размер и положение контейнера выноски» на странице 98.

Контейнер выноски, открытый элементом управления CalloutButton, всегда является немодальным. Это означает, при открытой выноске другие компоненты приложения могут получать входные данные. Контейнер Callout используется для создания модальной выноски.

Контейнер выноски остается открытым до щелчка за его пределами или до вызова метода `CalloutButton.closeDropDown()`. В этом примере метод `closeDropDown()` вызывается в обработчике события `click` для двух элементов управления Button, отображаемых в контейнере выноски.

Использование контейнера Callout для создания выноски

Элемент управления CalloutButton объединяет в себе контейнер выноски и все логические функции, необходимые для открытия и закрытия выноски. Поэтому элемент управления CalloutButton рассматривается как *host* контейнера выноски.

В мобильном приложении также можно использовать контейнер Callout. Преимущество контейнера Callout состоит в том, что он не связан с одним хостом и может использоваться в любом компоненте приложения.

Методы `Callout.open()` и `Callout.close()` используются для открытия контейнера Callout (как правило, в ответ на событие). При вызове метода `open()` можно передать дополнительный аргумент, указывающий на то, что контейнер выноски является модальным. По умолчанию контейнер выноски является немодальным.

Положение контейнера выноски определяется относительно компонента хоста. Свойства `horizontalPosition` и `verticalPosition` определяют расположение контейнера относительно хоста. См. пример в разделе «Размер и положение контейнера выноски» на странице 98.

Поскольку используется всплывающее окно, контейнер Callout не создается как часть обычного кода макета MXML приложения. Вместо этого контейнер Callout следует определить как пользовательский компонент MXML в файле MXML.

В следующем примере контейнер Callout определяется в файле MyCallout.mxml каталога comps приложения:

```
<?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>
```

Компонент MyCallout.mxml определяет обычное всплывающее окно, где пользователю необходимо ввести имя и фамилию. Обратите внимание, что кнопки вызывают метод `close()` для закрытия выноски в ответ на событие `click`.

В следующем примере рассматривается контейнер View, который открывает компонент MyCallout.mxml в ответ на событие `click`:

```
<?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>
```

Сначала в приложение импортируется компонент MyCallout.mxml. В ответ на событие `click` кнопка calloutB создает экземпляр компонента MyCallout.mxml и вызывает метод `open()`.

Метод `open()` указывает два аргумента. Первый аргумент указывает, что calloutB является компонентом хоста выноски. Следовательно, расположение выноски в приложении определяется расположением calloutB. Вторым аргументом является значение `true` для создания модальной выноски.

Определение встроенного контейнера Callout

Контейнер Callout не требуется определять в отдельном файле. В следующем примере используется тег `<fx:Declaration>` для определения этого контейнера как встроенного компонента контейнера View:

```
<?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>
```

Возврат данных из контейнера Callout

Для возврата данных в главное приложение используется метод `close()` контейнера Callout. Метод `close()` имеет следующую подпись:

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

где:

- `commit` имеет значение `true`, если приложение должно сохранить возвращаемые данные.
- `data` определяет возвращаемые данные.

Вызов метода `close()` передает событие `close`. Объект события, связанный с событием `close`, является объектом типа spark.events.PopUpEvent. Класс PopUpEvent определяет два свойства (`commit` и `data`), которые содержат значения соответствующих аргументов метода `close()`. Эти свойства используются в обработчике события `close` для проверки возвращаемых из выноски данных.

Контейнер выноски является подклассом класса SkinnablePopUpContainer, который использует такой же механизм возврата данных в главное приложение. Пример возврата данных из контейнера SkinnablePopUpContainer см. в разделе Возврат данных из контейнера Spark SkinnablePopUpContainer.

В следующем примере описанный ранее компонент Callout был изменен, чтобы выполнять возврат значений имени и фамилии:

```
<?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>
```

В этом примере создается элемент String для возврата имени и фамилии в ответ на нажатие пользователем кнопки OK.

Затем контейнер View использует событие `close` на контейнере Callout для отображения возвращаемых данных:

```
<?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>
```

Добавление компонента ViewNavigator к контейнеру Callout

Компонент ViewNavigator можно использовать в контейнере Callout. ViewNavigator позволяет добавить к выноске панель действий и несколько представлений.

Например, следующий контейнер View открывает контейнер Callout, определенный в файле MyCalloutPassBackVN:

```
<?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>
```

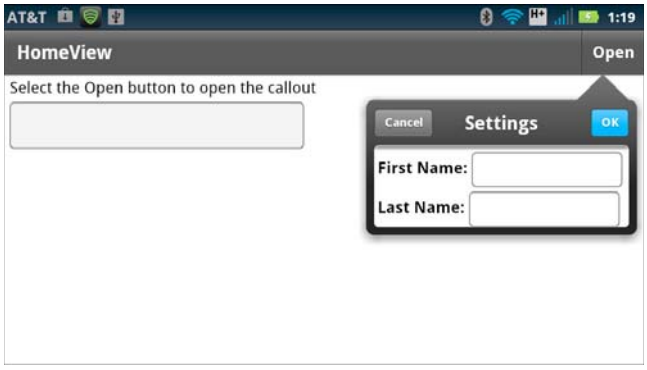
Файл MyCalloutPassBackVN.mxml определяет контейнер Callout, который содержит контейнер ViewNavigator:

```
<?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>
```

В файле MyCalloutPassBackVN.mxml необходимо указать, что первым представлением контейнера ViewNavigator является SettingsView. SettingsView определяет элементы управления TextInput для имени и фамилии пользователя. После того как пользователь нажимает кнопку OK, контейнер Callout закрывается и возвращаемые данные передаются в файл MyCalloutPassBackVN.

Примечание. *При отображении ViewNavigator в контейнере Callout для элемента управления ActionBar используется прозрачный фон. В этом примере для* `contentBackgroundAppearance` *установлено значение* `none` *в контейнере Callout. Если выбран этот параметр, стандартный белый фон* `contentBackgroundColor` *для Callout не отображается в прозрачной области элемента ActionBar.*

В следующем примере показано приложение с открытым контейнером Callout:



Далее показан файл SettingsView.mxml:

```
<?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>
```

Примечание. *Элемент управления ActionBar, определенный контейнером ViewNavigator в контейнере Callout, имеет прозрачный фон. По умолчанию переход от одного контейнера View к другому корректно отображается в контейнере Callout. Однако в случае указания нестандартного перехода, например CrossFadeViewTransition или ZoomViewTransition, может произойти наложение областей ActionBar двух представлений. Чтобы решить эту проблему, для элемента управления ActionBar и контейнера Callout необходимо создать пользовательский класс темы оформления, который использует непрозрачный фон.*

Размер и положение контейнера выноски

Элемент управления CalloutButton и контейнер Callout используют два свойства для указания расположения контейнера выноски по отношению к его хосту: `horizontalPosition` и `verticalPosition`. Эти свойства могут иметь следующие значения: `before`, `start`, `middle`, `end`, `after` и `auto` (по умолчанию).

Например, значения этих свойств можно указать следующим образом:

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

Контейнер выноски открывается левее и ниже компонента хоста. Если значения свойств указать, как показано ниже:

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

Контейнер выноски открывается выше компонента хоста, при этом центр выноски выравнивается по центру компонента хоста.

Создание стрелки от выноски к хосту

Для всех (кроме пяти) комбинаций свойств `horizontalPosition` и `verticalPosition` выноска рисует стрелку, указывающую на хост. Стрелка не отображается, если выноска выровнена по центру хоста или расположена в углу. При следующих комбинациях стрелка не отображается:

```
// 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"
```

Для контейнера Callout свойства `horizontalPosition` и `verticalPosition` также определяют значение свойства `Callout.arrowDirection`, доступного только для чтения. Положение контейнера выноски по отношению к хосту определяет значение свойства `arrowDirection`. Возможные значения: `up`, `left` и другие.

Элемент темы оформления `Callout.arrow` использует значение свойства `arrowDirection` для отображения стрелки на основе положения выноски.

Управление памятью, используемой контейнером выноски

При использовании контейнера выноски очень важно определить способ управления памятью, которую использует выноска. Например, для снижения объема используемой приложением памяти необходимо создавать экземпляр выноски при каждом запуске приложения. После закрытия приложения выноска удаляется. Однако следует помнить, что также необходимо удалить все ссылки на выноску, особенно обработчики событий, поскольку в таком случае выноска удалена не будет.

Кроме того, если контейнер выноски относительно небольшой, его можно многократно использовать в том же приложении. В этой конфигурации приложение создает один экземпляр выноски. Затем созданный экземпляр используется повторно, при этом выноска между периодами использования остается в памяти. Эта конфигурация снижает время выполнения приложения, поскольку повторное создание выноски при каждом запуске не требуется.

Управление памятью с помощью элемента управления CalloutButton

Для настройки выноски, которую использует элемент управления CalloutButton, укажите значение свойства `CalloutButton.calloutDestructionPolicy`. Значение `auto` предполагает удаление выноски после закрытия. Если указано значение `never`, элемент управления кэширует выноску в памяти.

Управление памятью с помощью контейнера Callout

Контейнер Callout не определяет свойство `calloutDestructionPolicy`. Вместо этого отследите использование памяти этим контейнером, создав экземпляр контейнера выноски в приложении. В следующем примере экземпляр контейнера выноски создается при каждом запуске:

```
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);
}
```

Также можно определить один экземпляр контейнера выноски, который при каждом запуске используется повторно:

```
// 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);
}
```

Определение переходов в мобильном приложении

Переходы представлений Spark определяют способ изменения контейнеров View на экране. Для реализации переходов во время изменения представления используется анимация. Переходы используются для создания привлекательных интерфейсов для мобильных приложений.

По умолчанию существующий контейнер View перемещается за пределы экрана, а новое представление отображается на экране. Пользователь может настроить этот способ изменения контейнеров. Например, один контейнер View приложения использует форму, в которой отображается только несколько полей, в то время как последующий контейнер View содержит дополнительные поля этой формы. Кроме надвигающихся представлений, можно использовать переходы с эффектами зеркального отображения или затухания.

Flex предоставляет возможность использовать следующие классы переходов представлений при изменении контейнеров View:

Переход	**Описание**
CrossFadeViewTransition	Выполняет переход с эффектом плавного затухания существующего и нового представлений. Существующее представление исчезает, а новое представление постепенно становится видимым.
FlipViewTransition	Выполняет переход с эффектом зеркального отображения существующего и нового представлений. Пользователь может определить направление и тип зеркального отображения.
SlideViewTransition	Выполняет переход со сдвигом между существующим и новым представлением. Существующее представление отодвигается, а новое представление надвигается. Пользователь может определить направление и тип сдвига. Этот переход представлений используется в Flex по умолчанию.
ZoomViewTransition	Выполняет переход с изменением размера существующего и нового представления. Существующее представление можно уменьшить, а новое - увеличить.

Примечание. *Переходы между представлениями в мобильных приложениях не связаны со стандартными переходами Flex. Стандартные переходы Flex определяют эффекты, воспроизводимые во время изменения состояния. Операции навигации контейнера ViewNavigator инициируют переходы View. Переходы View невозможно определить в MXML и они не взаимодействуют с состояниями.*

Применение перехода

Переход применяется при изменении активного контейнера View. Поскольку переходы представлений выполняются при изменении контейнеров View, для управления этими переходами используется контейнер ViewNavigator.

Например, следующие методы класса ViewNavigator используются для изменения текущего представления:

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

Все эти методы принимают дополнительный аргумент, который определяет воспроизводимый переход при изменении представлений.

Кроме того, текущее представление можно изменить посредством аппаратных клавиш навигации, например кнопки возврата на устройстве. Если для изменения представления используется аппаратная клавиша, ViewNavigator использует стандартные переходы, определенные в свойствах `ViewNavigator.defaultPopTransition` и `ViewNavigator.defaultPushTransition`. По умолчанию эти свойства используют класс SlideViewTransition.

В следующем примере файл главного приложения инициализирует свойства `defaultPopTransition` и `defaultPushTransition` для использования FlipViewTransition:

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

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

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

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

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

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

Первое представление EmployeeMainViewTrans.mxml определяет CrossFadeViewTransition. Затем CrossFadeViewTrainsition передается в качестве аргумента метода `pushView()` при изменении EmployeeView:

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

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

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

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

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

EmployeeView определяется в файле EmployeeView.mxml, как показано в следующем примере:

```
<?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>
```

Применение перехода к элементу управления ActionBar

По умолчанию переходы между представлениями не включают в себя ActionBar. При изменении представлений и вне зависимости от указанного вида перехода элемент управления ActionBar использует переход со сдвигом. Чтобы добавить ActionBar в переход при изменении представления, укажите для свойства `transitionControlsWithContent` класса перехода значение `true`.

Использование класса замедления в переходе

Воспроизведение перехода выполняется в два этапа: *ускорение* и последующее *торможение*. Для изменения свойств ускорения и торможения перехода используется класс замедления, позволяющий создавать реалистичные эффекты изменения скорости воспроизведения. Кроме того, класс замедления используется для создания скачкообразного движения и управления другими эффектами движения.

Flex предоставляет классы замедления Spark в пакете spark.effects.easing. Этот пакет содержит наиболее распространенные типы замедления, например Bounce, Linear и Sine. Для получения подробной информации об использовании этих классов см. документ Использование классов замедления Spark.

В следующем примере изменяется приложение, определенное в предыдущем разделе. В этой версии в переход FlipViewTransition добавляется класс замедления Bounce:

```
<?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>
```

Для отображения эффекта скачкообразного движения нажмите кнопку возврата на устройстве.

Жизненный цикл перехода представлений

Двумя основными этапами перехода представлений являются *подготовка* и *выполнение*.

Этап подготовки определяется тремя методами класса перехода. Эти методы вызываются в следующем порядке.

1 `captureStartValues()`

   При вызове этого метода ViewNavigator создает новое представление, но не проверяет его и не обновляет содержимое элемента управления ActionBar и панели вкладок. Этот метод принимает начальные значения компонентов, которые участвуют в переходе.

2 `captureEndValues()`

   При вызове этого метода выполняется полная проверка нового представления, а элемент управления ActionBar и панель вкладок отражают его состояние. Этот метод перехода позволяет принимать любые требуемые значения из нового представления.

3 `prepareForPlay()`

   Этот метод позволяет переходу активировать экземпляр эффекта, который используется для анимации компонентов перехода.

Этап выполнения начинается с вызова метода `play()` перехода в ViewNavigator. В этот момент создание и проверка нового представления уже завершены, а также активированы элемент управления ActionBar и панель вкладок. Переход отправляет событие `start`, и все экземпляры эффектов, созданные на этапе подготовки, вызываются посредством метода `play()` эффекта.

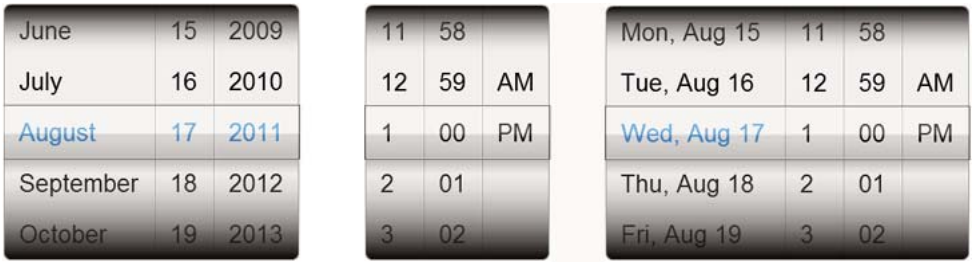
После завершения перехода представлений отправляется событие `end`. Базовый класс переходов ViewTransitionBase определяет метод `transitionComplete()`, который вызывается для отправки события `end`. При переходе необходимо удалить все временные объекты и прослушиватели, которые были созданы перед отправкой события завершения.

После вызова метода `transitionComplete()` контейнер ViewNavigator завершает процесс изменения представления и восстанавливает начальное состояние перехода.

Выбор даты и времени в мобильном приложении

Элемент управления DateSpinner позволяет пользователям выбирать дату и время в мобильном приложении. Этот элемент управления использует обычный мобильный интерфейс, состоящий из наборов смежных колес прокрутки, где каждое колесо отображает определенную часть даты и/или времени.

Для использования доступны три основных типа элементов управления DateSpinner, которые представлены на следующем изображении:



А. Дата. Б. Время. В. Дата и время.

Следующая таблица содержит описания каждого из типов DateSpinner:

Тип	**Константа (эквивалент String)**	**Описание**
Дата	`DateSelectorDisplayMode.DATE` (date)	Отображает месяц, день месяца и год. Например: `|| June || 11 || 2011 ||` Date является типом по умолчанию. Если значение свойства `displayMode` элемента управления DateSpinner не указано пользователем, Flex использует значение date. Текущая дата выделяется цветом, который определяется свойством стиля `accentColor`. Самая ранняя поддерживаемая дата: 1 января 1601 г. Самая поздняя поддерживаемая дата: 31 декабря 9999 г.
Время	`DateSelectorDisplayMode.TIME` (time)	Отображает часы и минуты. Для локалей, в которых используется 12-часовой формат времени, также отображается индикатор AM/PM. Например: `|| 2 || 57 || PM ||` Текущее время не выделяется. В элементе управления DateSpinner секунды не отображаются. Переключение между 12- и 24-часовым форматами времени невозможно. Элемент управления DateSpinner использует формат, стандартный для текущей локали.

Тип	Константа (эквивалент String)	Описание
Дата и время	`DateSelectorDisplayMode.DATE_AND_TIME` (dateAndTime)	Отображает день, часы и минуты. Для локалей, в которых используется 12-часовой формат времени, также отображается индикатор AM/PM. Например: `|| Mon Jun 13 || 2 || 57 || PM ||` Текущая дата выделяется цветом, который определяется свойством стиля `accentColor`. Текущее время не выделяется. В элементе управления DateSpinner секунды не отображаются. Название месяца отображается в сокращенной форме. Год не отображается.

Элемент управления DateSpinner состоит из нескольких элементов управления SpinnerList. Каждый элемент управления SpinnerList отображает список допустимых значений для определенной части элемента управления DateSpinner. Например, элемент управления DateSpinner, который отображает дату, состоит из трех элементов управления SpinnerList: один из них отображает дату, второй – месяц, а третий – год. Элемент управления DateSpinner, отображающий только время, будет состоять из двух или трех элементов управления SpinnerList: один из них отображает часы, второй – минуты, а третий (дополнительный) указывает AM или PM (если время отображается в 12-часовом формате).

Изменение типа элемента управления DateSpinner

Тип элемента управления DateSpinner выбирается при указании значения его свойства `displayMode`. В качестве значений свойства `displayMode` можно использовать определяемые классом DateSelectionDisplayMode константы или их строчные эквиваленты.

В следующем примере рассматривается переключение различных типов элемента управления DateSpinner:

```
<?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>
```

Когда с элементом управления DateSpinner работают пользователи, счетчики переключаются на ближайший элемент в списке. В неактивном состоянии счетчики никогда не отображают промежуточные значения.

Привязывание выбранного элемента управления DateSpinner к другим элементам управления

Свойство `selectedDate` элемента управления DateSpinner можно привязать к другим элементам управления мобильного приложения. Свойство `selectedDate` является указателем на объект Date, в результате чего любые методы объекта Date являются доступными.

В следующем примере день, месяц и год привязываются к элементам управления Label:

```
<?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>
```

Программный способ выбора дат в элементе управления DateSpinner

Изменить дату в элементе управления DateSpinner можно программным способом. Для этого новый объект Date необходимо назначить значению свойства `selectedDate`.

В следующем примере необходимо ввести день, месяц и год. При нажатии кнопки элемент управления DateSpinner переходит к новой дате:

```
<?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>
```

При изменении даты программным способом элемент управления DateSpinner передает событие `change` и событие `valueCommit`. При изменении даты в результате действий пользователя элемент управления DateSpinner передает событие `change`.

Если дата изменяется программным способом, выбранные значения отображаются без анимации промежуточных значений.

Ограничение диапазонов дат в элементе управления DateSpinner

Для ограничения дат, которые пользователи могут выбрать в элементе управления DateSpinner, используются свойства `minDate` и `maxDate`. Эти свойства работают с объектами Date. В элементе управления DateSpinner недоступны любые даты раньше значения, указанного в свойстве `minDate`, и любые даты позже значения, указанного в свойстве `maxDate`. Кроме того, недопустимые годы не отображаются в режиме date, а недопустимые даты – в режиме dateAndTime.

В следующем примере рассматривается создание двух элементов управления DateSpinner, для которых указаны различные диапазоны доступных дат:

```
<?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>
```

Можно указать только одну минимальную и одну максимальную дату. Указание массива дат или нескольких диапазонов выбора невозможно.

Свойства `minDate` и `maxDate` также можно использовать для ограничения значений элемента управления DateSpinner в режиме time. В следующем примере рассматривается ограничение выбора времени значениями от 8 AM до 2 PM:

```
<?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>
```

Реакция на события элемента управления DateSpinner

При изменении даты пользователем элемент управления DateSpinner передает событие `change`. Свойство `target` этого события `change` содержит ссылку на элемент управления DateSpinner, которую можно использовать для получения доступа к выбранной дате, как показано в следующем примере:

```
<?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>
```

Передача события `change` и обновление значения свойства `selectedDate` происходят только после прекращения вращения всех счетчиков в ответ на действия пользователей.

Для захвата изменения даты, выполненного программным способом, необходимо прослушивать событие `value_commit`.

Изменение минутного интервала элемента управления DateSpinner

Для изменения интервала отображаемых элементом управления DateSpinner минут используется свойство `minuteStepSize`. Это свойство применимо только в случае, если для параметра `displayMode` элемента управления DateSpinner установлено значение time или dateAndTime. Например, если для свойства `minuteStepSize` было указано значение 10, элемент управления DateSpinner на счетчике минут будет отображать только значения 0, 10, 20, 30, 40 и 50.

В следующем примере рассматривается указание значения свойства `minuteStepSize`. Счетчик минут обновляется соответственно.

```
<?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>
```

Допустимыми значениями для свойства `minuteStepSize` являются значения, на которые можно разделить 60 без остатка. В случае указания значения, на которое 60 не делится без остатка (например, 25), свойство `minuteStepSize` будет использовать стандартное значение 1.

Если минутный интервал был указан, когда текущее время не совпадало со значением на счетчике минут, элемент управления DateSpinner округлит текущий выбор до ближайшего интервала. Например, если отображается время 10:29, а для свойства `minuteStepSize` указано значение 15, элемент управления DateSpinner изменит отображаемое время на 10:15 при условии, что значение 10:15 не противоречит настройкам свойства `minDate`.

Настройка внешнего вида элемента управления DateSpinner

Элемент управления DateSpinner поддерживает большинство стилей текста, среди которых `fontSize`, `color` и `letterSpacing`. Кроме того, он также поддерживает новый стиль – `accentColor`. Этот стиль изменяет цвет текущей даты или времени в списках счетчика. В следующем примере для текущей даты или времени установлен красный цвет:

```
<?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>
```

Элемент управления DateSpinner не поддерживает свойство `textAlign`. Выравнивание текста выполняется другим элементом управления.

Для настройки других аспектов внешнего вида элемента управления DateSpinner можно создать пользовательскую тему оформления, определяющую или изменяющую некоторые подкомпоненты с помощью CSS.

Класс DateSpinnerSkin определяет размер элемента управления DateSpinner. Каждый счетчик, действующий в элементе управления DateSpinner является объектом SpinnerList с собственным классом SpinnerListSkin. Все счетчики одного элемента управления DateSpinner являются нижестоящими элементами одного контейнера SpinnerListContainer, который имеет собственную тему оформления – SpinnerListContainerSkin.

Свойство `height` элемента управления DateSpinner можно установить явным образом. При установке свойства `width` элемент управления располагается по центру области, имеющей запрашиваемую ширину.

Для изменения настроек счетчиков элемента управления DateSpinner также можно использовать такие CSS-селекторы типа, как SpinnerList, SpinnerListContainer и SpinnerListItemRenderer. Например, селектор типа SpinnerList определяет свойства отступов счетчиков.

В следующем примере рассматривается изменение отступов счетчиков:

```
<?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>
```

В мобильных приложениях селекторы типа должны находиться в файле верхнего уровня, а не в нижестоящем ракурсе приложения. При попытке помещения селектора типа SpinnerListItemRenderer в блок стиля внутри представления приложение Flex выводит предупреждение компилятора.

Класс SpinnerListContainerSkin можно расширить для дальнейшей настройки внешнего вида счетчиков, которые используются элементом управления DateSpinner. В следующем примере рассматривается применение пользовательской темы оформления к контейнеру SpinnerListContainer:

```
<?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>
```

Следующий класс CustomSpinnerListContainerSkin уменьшает размер индикатора выбора для более точного отражения нового размера шрифтов и заполнения в строках счетчика:

```
// 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;
                }
            }
        }
    }
}
```

Для получения подробной информации о создании тем оформления мобильных компонентов см. раздел «Основы создания мобильных тем оформления» на странице 179.

Использование локализованных дат и времени в элементе управления DateSpinner

Элемент управления DateSpinner поддерживает все локали, поддерживаемые устройством, на котором выполняется приложение. При изменении локали на ja-JP элемент управления DateSpinner будет отображать даты в формате, стандартном для японской локали.

Свойство `locale` можно определить непосредственно в элементе управления DateSpinner или в таком контейнере, как Application. Элемент управления DateSpinner наследует значение этого свойства. Локалью по умолчанию является локаль устройства, на котором выполняется приложение, кроме случаев, когда локаль была переопределена при помощи свойства `locale`.

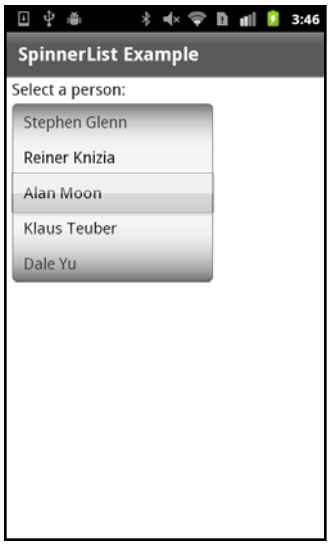
В следующем примере в качестве локали по умолчанию выбрана локаль ja-JP. Выбор локали позволяет изменить формат элемента управления DateSpinner:

```
<?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>
```

Использование списка счетчика в мобильном приложении

Компонент SpinnerList является специальным элементом управления List, который обычно используется для выбора данных в мобильных приложениях. Как только во время прокрутки пользователем элементов списка достигается конец списка, счетчик по умолчанию возвращается в начало. Элемент управления SpinnerList обычно используется в мобильных приложениях как компонент NumericStepper.

На следующем изображении показан типичный вид элемента управления SpinnerList в мобильном приложении:



Элемент управления SpinnerList

Компонент SpinnerList действует как вращающийся цилиндрический барабан. Пользователи могут прокручивать список с помощью «бросков» и перетаскивания вверх и вниз, а также посредством щелчков на элементах списка.

Элемент управления SpinnerList обычно вкладывается в элемент управления SpinnerListContainer. Этот класс предоставляет большинство дополнительных элементов оформления для элемента управления SpinnerList и определяет макет. Дополнительные элементы оформления включают границы, тени и внешний вид индикатора выбора.

Данные элемента управления SpinnerList сохраняются в виде списка. Этот список затем визуализируется в счетчике с помощью SpinnerListItemRenderer. Чтобы настроить внешний вид или содержимое элементов списка, можно переопределить средство визуализации.

Элемент управления DateSpinner представляет собой набор элементов управления SpinnerList с пользовательским средством визуализации.

В настоящее время невозможно отключить компоненты элемента управления SpinnerList, не отключая весь элемент управления. Это ограничение не действует на элемент управления DateSpinner, который предоставляет дополнительные логические операции для настройки диапазонов деактивированных дат.

Определение данных для списка счетчика

Чтобы определить данные для элемента управления SpinnerList, выполните одно из следующих действий:

- Определите данные непосредственно в свойстве `dataProvider` элемента управления SpinnerList .
- Определите данные как нижестоящие теги тега `<s:SpinnerList>`.
- Определите данные в ActionScript или MXML и свяжите их с элементом управления SpinnerList. Эти данные могут быть получены от внешней службы, из вложенного ресурса (например, файла XML) или любого другого источника данных.
- Свяжите элемент управления SpinnerList с операцией службы данных, выполняемой мастером служб Flash Builder. Для получения подробной информации о создании ориентированных на данные приложений при помощи Flash Builder см. раздел Подключение к службам данных.

Элемент управления SpinnerList может принять как поставщика данных любой класс, который реализует интерфейс IList. Среди принимаемых классов: ArrayCollection, ArrayList, NumericDataProvider и XMLListCollection.

Если при создании элемента управления SpinnerList поставщик данных не был определен, SpinnerList отображается с одной пустой строкой. После добавления поставщика данных элемент управления SpinnerList изменяет свой размер для стандартного отображения пяти элементов списка.

В следующем примере рассматривается определение данных для элемента управления SpinnerList в нижестоящих тегах тега `<s:SpinnerList>`:

```
<?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>
```

В следующем примере рассматривается определение данных SpinnerList в теге `<s:SpinnerList>`:

```
<?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>
```

В следующем примере рассматривается определение данных SpinnerList в ActionScript:

```
<?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>
```

При наличии в ActionScript таких сложных объектов, как данные, необходимо определить свойство `labelField`, чтобы элемент управления SpinnerList отображал соответствующие метки, как показано в следующем примере:

```
<?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>
```

Также можно использовать удобный класс NumericDataProvider, предоставляющий элементу управления SpinnerList числовые данные. Этот класс позволяет легко определять минимальное значение, максимальное значение и размер шага для набора числовых данных.

В следующем примере рассматривается использование класса NumericDataProvider как источника данных для элементов управления SpinnerList:

```
<?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>
```

Значением свойства `stepSize` может быть отрицательное число. В таком случае в первую очередь отображается максимальное значение. Сначала счетчик отображает максимальное значение, а затем постепенно переходит к минимальному.

Выбор элементов в списке счетчика

Элемент управления SpinnerList поддерживает одновременный выбор только одного элемента. Выбранный элемент всегда располагается в центре компонента и по умолчанию отображается под индикатором выбора. Когда счетчики не вращаются, в SpinnerList всегда должен быть выбран элемент. Нельзя выбрать отключенный элемент или строку без элементов.

Для отображения в элементе управления SpinnerList выбранного в настоящий момент элемента используется свойство `selectedIndex` или `selectedItem`.

Для установки выбранного в настоящий момент элемента в элементе управления SpinnerList необходимо указать значение для свойства `selectedIndex` или `selectedItem`. Эти свойства обычно определяются в теге `<s:SpinnerList>`, чтобы элемент выбирался при создании элемента управления SpinnerList.

Если значение свойства `selectedIndex` или `selectedItem` в элементе управления SpinnerList не определено явно, первым в списке будет элемент, выбранный по умолчанию.

Свойства `selectedIndex` и `selectedItem` можно использовать для программного изменения выбранного элемента в счетчике. При определении одного из этих свойств элемент управления мгновенно переходит к указанному элементу без соответствующей анимации (или «вращения») счетчика.

В следующем примере рассматривается использование элемента управления SpinnerList в качестве таймера обратного отсчета. Объект Timer изменяет выбранный элемент счетчика посредством ежесекундного изменения значения свойства `selectedIndex`:

```
<?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>
```

Список счетчика: действия пользователя и события

При изменении выбранного элемента в SpinnerList элемент управления передает события `change` и `valueCommit`. Это, как правило, происходит в ответ на такое действие пользователя, как выделение. Если пользователь выделяет элемент, коснувшись его, элемент управления передает события `click`, `change` и `valueCommit`.

Если выделенный элемент изменяется программным способом, элемент управления SpinnerList передает только событие `valueCommit`.

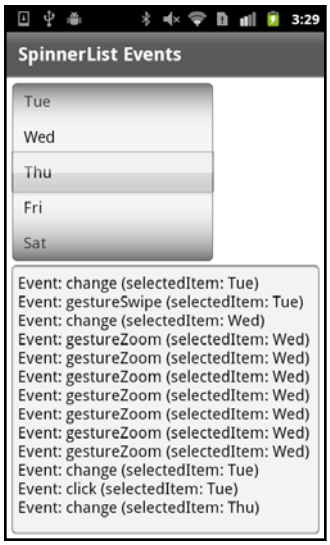
Во время вращения элемента управления SpinnerList события для каждого пройденного элемента не передаются. Событие `change` или `valueCommit` передается только при остановке элемента управления на новом элементе.

Если элемент управления SpinnerList был первоначально создан с поставщиком данных, он передает как событие `change`, так и событие `valueCommit`.

В следующем примере рассматриваются общие события, которые передаются при использовании элемента управления SpinnerList:

```
<?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>
```

На следующем изображении показан вывод после взаимодействия с элементом управления SpinnerList:



События элемента управления SpinnerList

Настройка возврата в начало списка счетчика

По умолчанию если проводник данных элемента управления SpinnerList содержит меньше элементов, чем предусматривается в счетчике, то счетчик не возвращается, а останавливается на последнем элементе списка. В противном случае после прохождения последнего элемента счетчик возвращается в начало списка.

Стандартное количество отображаемых в списке элементов – пять. Если необходимо изменить количество элементов, создайте пользовательскую тему оформления. Для получения подробной информации см. раздел «Создание пользовательской темы оформления для списка счетчика» на странице 130.

Значение свойства `wrapElements` определяет, начинает ли элемент управления SpinnerList работу с первого элемента после достижения последнего элемента списка. Если для свойства `wrapElements` указано значение `false`, то счетчик останавливается при достижении конца списка независимо от количества отображаемых и содержащихся в списке элементов.

Если для свойства `wrapElements` указано значение `true`, то счетчик снова начинает работу с первого элемента, но только в случае, если количество элементов в списке хотя бы на один превышает количество отображаемых элементов. Например, если элемент управления SpinnerList отображает пять элементов, а список содержит только четыре элемента, то счетчик не вернется к первому элементу независимо от значения свойства `wrapElements`.

Переопределить стандартное поведение возврата SpinnerList в начало можно, установив для свойства `wrapElements` значение `true` или `false`.

В следующем примере рассматривается переключение значения свойства `wrapElements`:

```
<?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>
```

Обычно ожидается, что счетчик вернется в начало, если список содержит больше элементов, чем одновременно отображаются. Если список содержит меньше элементов, чем может отобразить счетчик, ожидается, что счетчик не вернется в начало.

Установка стилей для списка счетчика

Элемент управления SpinnerList поддерживает все текстовые стили, используемые в мобильной теме Spark. Среди этих стилей: `fontSize`, `fontWeight`, `color`, `textDecoration` и свойства выравнивания. Эти свойства стиля можно указать непосредственно для элемента управления в MXML или CSS. Элемент управления SpinnerList также наследует эти свойства, если они определены в вышестоящем контейнере.

Кроме того, можно определить свойства заполнения SpinnerList, изменив свойства стиля SpinnerListItemRenderer.

В следующем примере рассматривается определение свойств связанного с текстом стиля в селекторе типа SpinnerList и свойств заполнения в селекторе типа SpinnerListItemRenderer:

```
<?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>
```

Если в мобильном приложении используются селекторы типа, необходимо определить блок `<fx:Style>` в файле верхнего уровня приложения. В противном случае компилятор выведет предупреждение и стили не будут применены.

Элемент управления SpinnerList не поддерживает такие свойства стиля, как `accentColor`, `backgroundAlpha`, `backgroundColor` или `chromeColor`.

Создание пользовательской темы оформления для списка счетчика

Для элемента управления SpinnerList или SpinnerListContainer можно создать пользовательскую тему оформления. Для этого необходимо скопировать исходные данные SpinnerListSkin или SpinnerListContainerSkin для использования в качестве основы для создания пользовательской темы оформления.

Пользовательские темы оформления SpinnerList обычно создаются для изменения следующих аспектов элемента управления SpinnerList или его контейнера:

- Изменение размера или формы рамки вокруг выбранного в данный момент элемента (`selectionIndicator`). Для этого необходимо создать пользовательский класс SpinnerListContainerSkin.
- Определение высоты каждой строки (`rowHeight`). Для этого необходимо создать пользовательский класс SpinnerListSkin.
- Определение количества отображаемых строк (`requestedRowCount`). Для этого необходимо создать пользовательский класс SpinnerListSkin.
- Определение внешнего вида контейнера (например, радиуса скругления и толщины границы) Для этого необходимо создать пользовательский класс SpinnerListContainerSkin.

Примеры пользовательских классов SpinnerListSkin и SpinnerListContainerSkin см. в разделе «Настройка внешнего вида элемента управления DateSpinner» на странице 114.

Использование изображений в списке счетчика

Изображения в элементе управления SpinnerList можно использовать вместо текстовых меток, определив IconItemRenderer как средство визуализации SpinnerList.

Изображения, которые необходимо использовать в объекте IconItemRenderer, следует встроить или загрузить в среду выполнения. Для пользователей мобильных устройств более оптимальным решением может быть встраивание изображений с целью минимизации использования сети данных.

В следующем примере рассматривается использование встроенных изображений в элементе управления SpinnerList:

```
<?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>
```

На следующем изображении показано, как это приложение выглядит на мобильном устройстве:

Элемент управления SpinnerList со встроенными изображениями

Глава 4. Процесс создания приложений

Включение функции сохраняемости в мобильном приложении

Выполнение приложения на мобильном устройстве часто прерывается другими функциями, например входящими текстовыми сообщениями, телефонными звонками или действиями других мобильных приложений. Приложение, выполнение которого прервано, обычно перезапускается, при этом пользователь ожидает, что будет восстановлено предыдущее состояние приложения. Механизм сохраняемости позволяет устройству восстановить приложение в его предыдущем состоянии.

Инфраструктура Flex обеспечивает два вида сохраняемости в мобильном приложении. Сохраняемость *в памяти* хранит данные представления во время навигации по приложению. *Сохраняемость сеанса* восстанавливает данные при закрытии и повторном запуске приложения. Сохраняемость сеанса важна в мобильных приложениях, так как операционная система мобильного устройства может закрыть приложение в любое время, например при недостаточном объеме свободной памяти.

Автор блога Steve Mathews делится советами по обеспечению сохраняемости данных в мобильном приложении Flex .

Автор блога Holly Schinsky является автором статьи о сохраняемости и обработке данных в Flex .

Сохраняемость в памяти

Для поддержки сохраняемости в памяти контейнеры View используют свойство `View.data`. Свойство `data` существующего представления автоматически сохраняется при изменении выбранного раздела или добавлении нового представления в стек ViewNavigator, в результате чего удаляется существующее представление. Свойство `data` представления восстанавливается, когда элемент управления возвращается к представлению и активируется новый созданный экземпляр представления. Таким образом, сохраняемость в памяти обрабатывает информацию о состоянии представления во время выполнения.

Сохраняемость сеанса

Сохраняемость сеанса хранит информацию о состоянии приложения между выполнениями этого приложения. Контейнеры ViewNavigatorApplication и TabbedViewNavigatorApplication определяют свойство `persistNavigatorState` для реализации сохраняемости сеанса. Чтобы включить сохраняемость сеанса, установите для свойства `persistNavigatorState` значение `true`. По умолчанию для свойства `persistNavigatorState` указано значение `false`.

Включенная функция сохраняемости сеанса записывает состояние приложения на диск с помощью локального общедоступного объекта с именем `FxAppCache`. В приложении также можно использовать методы spark.managers.PersistenceManager для записи дополнительной информации в локальный общедоступный объект.

Сохраняемость сеанса ViewNavigator

Для поддержки сохраняемости сеанса контейнер ViewNavigator записывает состояние своего стека представлений на диск при закрытии приложения. Сохраненные данные содержат свойство `data` текущего представления.

После перезапуска приложения повторно создается экземпляр стека ViewNavigator и пользователь видит то же представление и содержимое, которое отображалось при выходе из приложения. Поскольку для каждого представления в стеке содержится копия свойства `data`, предыдущие представления в стеке восстанавливаются, когда становятся активными.

Сохраняемость сеанса TabbedViewNavigator

Для контейнера TabbedViewNavigator функция сохраняемости сеанса записывает текущую выбранную вкладку на панели вкладок при выходе из приложения. Эта вкладка соответствует ViewNavigator и стеку представлений, определяющему вкладку. Сохраненные данные содержат свойство `data` текущего представления. Таким образом, при повторном запуске приложения для активной вкладки и связанного ViewNavigator устанавливается состояние, присвоенное при закрытии приложения.

Примечание. *В приложениях, определенных контейнером TabbedViewNavigatorApplication, сохраняется только стек текущего ViewNavigator. Поэтому при перезапуске приложения восстанавливается только состояние текущего ViewNavigator.*

Представление данных при сохраняемости сеанса

Механизм сохраняемости, который использует Flex, не является зашифрованным или защищенным. Поэтому соответствующие данные хранятся в формате, который может интерпретироваться другой программой или пользователем. Этот механизм не предназначен для сохранения конфиденциальных данных, таких как учетные данные пользователя. Пользователь может создать собственный диспетчер сохраняемости, обеспечивающий более высокий уровень защиты. Подробную информацию см. в разделе «Настройка механизма сохраняемости» на странице 137.

Использование сохраняемости сеанса

В примере ниже для свойства `persistNavigatorState` устанавливается значение `true`, чтобы включить сохраняемость сеанса:

```
<?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>
```

В качестве первого представления в приложении используется файл EmployeeMainView.mxml, определяющий элемент управления List, в котором пользователь выбирает имя сотрудника.

```
<?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>
```

Для просмотра сохраняемости сеанса и перехода к представлению EmployeeView.mxml откройте приложение и выберите параметр Dave в элементе управления List:

```
<?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>
```

Представление EmployeeView.mxml отображает данные о параметре Dave. После этого закройте приложение. При повторном запуске приложения представление EmployeeView.mxml отображает те же данные, что и при выходе из приложения.

Доступ к данным в локальном общедоступном объекте

Информация в локальном общедоступном объекте сохраняется в виде пары ключ-значение. Для доступа к связанному значению в локальном общедоступном объекте методы PeristenceManager, такие как `setPropery()` и `getProperty()`, используют ключ.

Для записи пользовательских пар ключ-значение в локальный общедоступный объект используется метод `setProperty()`. Метод `setProperty()` имеет следующую подпись:

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

Метод `getProperty()` позволяет использовать значение определенного ключа. Метод `getProperty()` имеет следующую подпись:

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

Если для свойства `persistNavigatorState` установлено значение `true`, диспетчер сохраняемости автоматически сохраняет две пары ключ-значение в локальном общедоступном объекте при выходе из приложения:

- `applicationVersion`

  Версия приложения в соответствии с файлом application.xml.

- `navigatorState`

  Состояние представления навигатора в соответствии со стеком текущего ViewNavigator.

Выполнение функции сохраняемости вручную

Если для свойства `persistNavigatorState` установлено значение `true`, Flex автоматически выполняет сохраняемость сеанса. Сохраняемость данных приложения можно обеспечить и в том случае, если для свойства `persistNavigatorState` указано значение `false`. Для этого создайте пользовательский механизм сохраняемости с помощью метода PeristenceManager.

Для записи и чтения информации в локальном общедоступном объекте используйте методы `setProperty()` и `getProperty()`. Вызовите метод `load()` для запуска PeristenceManager. Для записи данных на диск вызовите методы `save()`.

Примечание. *Если для свойства `persistNavigatorState` указано значение `false`, в Flex не выполняется автоматическое сохранение стека представлений текущего ViewNavigator при выходе из приложения или восстановление стека при запуске приложения.*

Обработка событий сохраняемости

Для разработки пользовательского механизма сохраняемости используются следующие события контейнеров мобильного приложения:

- `navigatorStateSaving`;
- `navigatorStateLoading`.

Чтобы отменить сохранение состояний приложения на диск, вызовите метод `preventDefault()` в обработчике события `navigatorStateSaving`. Для отмены загрузки приложения при перезапуске вызовите метод `preventDefault()` в обработчике события `navigatorStateLoading`.

Настройка механизма сохраняемости

При включении функции сохраняемости сеанса приложение открывает представление, которое отображалось при выходе из приложения. Для полного восстановления состояния приложения необходимо сохранить достаточное количество информации в свойстве `data` представления или в любом другом объекте, таком как общедоступный объект.

Например, в восстановленном представлении может потребоваться выполнить расчет на основе свойства `data` представления. Для выполнения требуемых расчетов приложение должно распознавать событие перезапуска. Для выполнения пользовательских действий при закрытии или перезапуске приложения можно изменить методы `serializeData()` и `deserializePersistedData()` в View.

Поддержка встроенных типов данных для сохраняемости сеанса

Механизм сохраняемости автоматически поддерживает все встроенные типы данных, включая Number, String, Array, Vector, Object, uint, int и Boolean. Эти типы данных автоматически сохраняются механизмом сохраняемости.

Поддержка пользовательских классов для сохраняемости сеанса

Для определения данных во многих приложениях используются пользовательские классы. Если пользовательский класс содержит свойства, определенные встроенными типами данных, механизм сохраняемости может автоматически сохранить и загрузить этот класс. Однако сначала необходимо зарегистрировать этот класс в механизме сохраняемости посредством вызова метода flash.net.registerClassAlias(). Этот метод обычно вызывается для события `preinitialize` приложения перед определением соответствующего хранилища и записи в нем данных.

Если определяется сложный класс, в котором используются отличные от встроенных типы данных, необходимо преобразовать эти данные в поддерживаемый тип, например String. Кроме того, любые частные переменные, определяемые в классе, не сохраняются автоматически. Сложный класс, который необходимо поддержать в механизме сохраняемости, должен реализовать интерфейс flash.utils.IExternalizable. Чтобы сохранить и восстановить экземпляр класса в этом интерфейсе, класс должен реализовать методы `writeExternal()` и `readExternal()`.

Поддержка различных размеров экрана и значений DPI в мобильном приложении

Рекомендации по поддержке различных размеров экрана и значений DPI

При разработке независимого от платформы приложения следует учитывать различия между устройствами вывода. В устройствах могут использоваться экраны различных размеров или разрешений, а также различные значения DPI или плотности.

Специалист по Flex Jason SJ рассматривает два подхода к созданию мобильных приложений, независимых от разрешения, в своем блоге.

Терминология

Разрешение - это соотношение количества пикселов в значениях высоты и ширины, т. е. общее количество пикселов, поддерживаемое устройством.

DPI - это количество точек на квадратный дюйм, т. е. плотность пикселов на экране устройства. Вместо термина DPI может использоваться термин PPI (пикселы на дюйм).

Поддержка DPI в Flex

Для упрощения создания приложений, независимых от разрешения и DPI, Flex предоставляет указанные ниже возможности.

Темы оформления. Независимые от DPI темы оформления для мобильных компонентов. Масштабирование стандартных мобильных тем оформления не требует дополнительного кодирования при использовании большинства разрешений устройств.

applicationDPI. Свойство, определяющее размер разрабатываемых пользовательских тем оформления. Например, для этого свойства установлено определенное значение DPI, в то время как пользователь выполняет данное приложение на устройстве с другим значением DPI. Flex масштабирует все компоненты приложения с учетом DPI используемого устройства.

Стандартные масштабируемые и немасштабируемые мобильные темы оформления не зависят от DPI. Поэтому установка свойства `applicationDPI` требуется только в том случае, если используются компоненты со статическим размером или пользовательскими темами оформления.

Динамические макеты

Проблема различия в разрешениях решается посредством динамического макета. Например, если для элемента управления установлена ширина 100%, этот элемент всегда заполняет экран по ширине вне зависимости от указанного разрешения: 480x854 или 480x800.

Установка свойства applicationDPI

При создании приложений, независимых от плотности, можно установить целевое значение DPI в корневом теге приложения. Для мобильных приложений корневым тегом является `<s:ViewNavigatorApplication>`, `<s:TabbedViewNavigatorApplication>` или `<s:Application>`.

Значение свойства `applicationDPI` может быть указано как 160, 240 или 320 в зависимости от приблизительного разрешения целевого устройства. Например:

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

Установка свойства `applicationDPI` позволяет определить масштаб приложения при его сравнении с действительным разрешением целевого устройства (`runtimeDPI`) во время выполнения. Например, если для свойства `applicationDPI` указано значение 160 и значением свойства `runtimeDPI` целевого устройства является 160, то коэффициент масштабирования составляет 1 (т. е. масштабирование не выполняется). Если для свойства `applicationDPI` указано значение 240, то коэффициент масштабирования составит 1,5 (Flex увеличивает масштаб до 150%). При значении 320 коэффициент масштабирования составит 2, т. е. Flex увеличивает масштаб до 200%.

Если коэффициент масштабирования представлен нецелым числом, то в некоторых случаях при интерполяции могут отображаться артефакты, например нечеткие линии.

Выключение масштабирования DPI

Если требуется выключить масштабирование DPI для приложения, не устанавливайте значение для свойства `applicationDPI`.

Основные сведения о applicationDPI и runtimeDPI

В таблице ниже описываются два свойства класса Application, которые требуются для работы приложений с различными разрешениями.

Свойство	Описание
`applicationDPI`	Целевая плотность или DPI приложения. При установке этого свойства Flex применяет коэффициент масштабирования к корневому приложению, что позволяет эффективно масштабировать приложение с одним DPI на устройстве с другим DPI. Коэффициент масштабирования вычисляется на основе сравнения значения этого свойства со свойством `runtimeDPI`. Коэффициент масштабирования применяется ко всем компонентам приложения, включая средство предварительной загрузки, всплывающие экраны и другие компоненты в рабочей области. Если значение свойства не установлено, то оно будет аналогично значению свойства `runtimeDPI`. Значение этого свойства устанавливается только в MXML и не может быть определено кодом ActionScript. Значение этого свойства невозможно изменить во время выполнения.
`runtimeDPI`	Плотность, или DPI устройства, на котором в настоящее время выполняется приложение. Возвращает значение свойства `Capabilities.screenDPI`, округленное до значения одной из констант, определенных в классе DPIClassification. Это свойство предназначено только для чтения.

Создание приложений, независимых от разрешения и DPI

Приложения, независимые от разрешения и DPI, имеют следующие характеристики:

Изображения. Векторные изображения эффективно масштабируются в соответствии с действительным разрешением целевого устройства. Однако масштабирование растровых изображений может быть неэффективным. В этом случае загрузите растровые изображения с различными разрешениями, соответствующими разрешению устройства, используя класс MultiDPIBitmapSource.

Текст. Размер шрифта текста (но не самого текста) масштабируется с учетом разрешения.

Макеты. Динамические макеты позволяют эффективно масштабировать приложения. Как правило, не рекомендуется использовать макеты на основе ограничений, в которых указаны абсолютные значения для границ пикселов. Если требуется применить ограничения, включите для значения свойства `applicationDPI` функцию масштабирования.

Масштабирование. Не используйте свойства `scaleX` и `scaleY` для объекта Application. Если свойство `applicationDPI` установлено, Flex автоматически выполнит масштабирование.

Стили. В таблицах стилей можно настроить свойства стилей на основе ОС целевого устройства и параметров DPI приложения.

Темы оформления. Для определения ресурсов, которые необходимо использовать во время выполнения, темы оформления Flex в мобильной теме используют значения DPI приложения. Все визуальные ресурсы темы оформления, определенные файлами FXG, соответствуют целевому устройству.

Размер приложения. Не требуется явно указывать ширину и высоту приложения. Также при вычислении размеров пользовательских компонентов или всплывающих окон не используйте свойства `stageWidth` и `stageHeight`. Вместо этого установите свойство `SystemManager.screen`.

Определение среды выполнения DPI

При запуске приложения используется значение свойства `runtimeDPI`, полученное из свойства `Capabilities.screenDPI` Flash Player. Это свойство сопоставляется с одной из констант, определенных в классе DPIClassification. Например, значение DPI для Droid, которое равно 232, сопоставляется со значением DPI среды выполнения, указанным как 240. Значения DPI устройств не всегда аналогичны константам DPIClassification (160, 240 или 320). Более того, они сопоставляются с этими классификациями на основе диапазона целевых значений.

Допускаются следующие сопоставления:

Константа DPIClassification	**160 DPI**	**240 DPI**	**320 DPI**
Действительное значение DPI устройства	< 200	>= 200 и < 280	>= 280

Настройка этих значений позволяет изменить поведение по умолчанию или отрегулировать устройства, сообщающие недействительные значения DPI. Для получения подробной информации см. раздел «Изменение DPI по умолчанию» на странице 149.

Выбор или отмена автоматического масштабирования

Решение о выборе автоматического масштабирования (при установке значения свойства `applicationDPI`) принимается с учетом требований к удобству и визуальной точности изображения на уровне пикселов. Если в свойстве `applicationDPI` установлено автоматическое масштабирование приложения, в Flex используются темы оформления, предназначенные для `applicationDPI`. Flex увеличивает или уменьшает размер темы оформления в соответствии с действительной плотностью устройства. Также масштабируются другие ресурсы в приложении и положение макета.

Указанные ниже действия выполняются для автоматического масштабирования при создании пользовательских тем оформления или ресурсов, предназначенных для одного значения DPI.

- Создайте один набор тем оформления и макетов представлений и компонентов, предназначенный для указанного `applicationDPI`.
- Создайте несколько версий любого растрового ресурса, используемого в теме оформления или в приложении, и определите их в классе MultiDPIBitmapSource. Векторные и текстовые ресурсы, используемые в темах оформления, не зависят от плотности при автоматическом масштабировании.
- Не используйте правило `@media` в таблицах стилей, поскольку приложение учитывает только одно целевое значение DPI.
- Протестируйте приложение на устройствах с различной плотностью, чтобы проверить правильное отображение внешнего вида масштабированного приложения на каждом устройстве. В частности, проверьте отображение на устройствах, для которых используется нецелочисленный коэффициент масштабирования. Например, если значение `applicationDPI` равно 160, протестируйте приложение на устройствах с DPI, равным 240.

Если автоматическое масштабирование не используется (т. е. свойство `applicationDPI` не установлено), необходимо получить значение `applicationDPI` для определения действительной классификации DPI устройства и отредактировать приложение во время выполнения. Для этого выполните следующие действия:

- Создайте несколько наборов тем оформления и макетов с учетом каждой используемой спецификации DPI или создайте один набор тем оформления и макетов с динамической адаптацией к различным показателям плотности. Встроенные темы оформления Flex используют второй подход: класс каждой темы оформления настраивается в соответствии со свойством `applicationDPI`.

- Применяйте правила `@media` в таблицах стилей для фильтрации правил CSS на основе классификации DPI устройства. Как правило, пользователь настраивает размер шрифта и значения заполнения для каждого значения DPI.
- Протестируйте приложение на устройствах с различной плотностью, чтобы проверить правильное отображение тем оформления и макетов.

Выбор стилей на основе DPI

В Flex предусмотрена поддержка применения стилей на основе целевой ОС и значения DPI приложения в CSS. Для применения стилей используется правило `@media` в таблице стилей. Правило `@media` включено в спецификацию CSS. Flex расширяет это правило для включения дополнительных свойств: `application-dpi` и `os-platform`. Эти свойства позволяют выборочно применять стили на основе DPI приложения и платформы, на которой запускается данное приложение.

В следующем примере в элементе управления Spark Button стандартному свойству стиля `fontSize` присваивается значение 12. Если устройство использует значение 240 DPI и выполняется в операционной системе Android, свойство `fontSize` принимает значение 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>
```

Значения свойства application-dpi

Свойство CSS `application-dpi` сравнивается со значением свойства стиля `applicationDPI`, которое установлено в корневом приложении. Допустимые значения свойства CSS `application-dpi`:

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

Для каждого из поддерживаемых значений `application-dpi` указана соответствующая константа в классе DPIClassification.

Значения свойства os-platform

Свойство CSS `os-platform` сопоставляется со значением свойства `flash.system.Capabilities.version` проигрывателя Flash Player. Допустимые значения свойства CSS `os-platform`:

- `Android`

- `iOS`
- `Macintosh`
- `Linux`
- `QNX`
- `Windows`

Сопоставление выполняется без учета регистра.

Если соответствующие записи отсутствуют, Flex выполняет повторное сопоставление по первым трем буквам в списке поддерживаемых платформ.

Значения по умолчанию для свойств application-dpi и os-platform

Если выражение, содержащее свойства `application-dpi` или `os-platform` не определено явным образом, предполагается, что все выражения совпадают.

Операторы в правиле @media

Правило `@media` поддерживает общие операторы and и not. Также это правило поддерживает списки с разделителем-запятой. При разделении выражений с помощью запятой выполняется условие or.

Используемый оператор not должен являться первым ключевым словом в выражении. Этот оператор отменяет все выражение, а не только свойство, следующее за not. С учетом ошибки SDK-29191 за оператором not должен следовать тип media, например all, перед одним или несколькими выражениями.

Следующий пример иллюстрирует использование некоторых общих операторов:

```
<?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>
```

Выбор растровых ресурсов на основе DPI

Визуализация растровых графических ресурсов наиболее эффективна в разрешении, для которого они созданы. Поэтому при использовании этих ресурсов в приложении, предназначенном для различных разрешений, могут возникать проблемы. Для устранения этих проблем необходимо создать несколько растровых изображений для различных разрешений и загрузить соответствующее изображение в соответствии со значением свойства `runtimeDPI` приложения.

Компоненты Spark BitmapImage и Image содержат свойство `source` типа Object. Благодаря этому свойству, можно передавать класс, который определяет используемые ресурсы. В этом случае класс MultiDPIBitmapSource передается для сопоставления различных источников в зависимости от значения свойства `runtimeDPI`.

В следующем примере на основании значения DPI загружаются различные изображения:

```
<?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>
```

Если классы BitmapImage и Image используются с MultiDPIBitmapSource в *настольном* приложении, в качестве источника выбирается свойство`source160dpi`.

Свойство `icon` элемента управления Button также принимает класс в качестве аргумента. Поэтому объект MultiDPIBitmapSource также может использоваться в качестве источника для значка Button. Источник значка можно определить и на внутреннем уровне, как показано в следующем примере:

```
<?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>
```

Также можно объявить значки в блоке `<fx:Declarations>` и присвоить источнику привязку данных, как иллюстрирует следующий пример:

```
<?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>
```

Если свойство `runtimeDPI` сопоставляется со свойством `source`*`XXX`*`dpi`, для которого указано значение `null` или пустая строка (""), Flash Player использует последующую плотность с наиболее высоким значением в качестве источника. Если это значение также является `null` или пустой строкой, используется последующая плотность с более низким значением. Если значения этой плотности *также* являются `null` или пустой строкой, Flex назначает значение `null` в качестве источника, в результате чего графический объект не отображается. Это означает, что невозможно явно указать отсутствие изображения для определенного DPI.

Выбор ресурсов тем оформления на основе DPI

Логика в конструкторах стандартных мобильных тем оформления выбирает ресурсы на основе значения свойства `applicationDPI`. Эти классы выбирают ресурсы, которые максимально соответствуют целевому значению DPI. При создании пользовательских тем оформления, которые могут применять или не применять масштабирование DPI, используется свойство `applicationDPI`, но не свойство `runtimeDPI`.

Например, класс spark.skins.mobile.ButtonSkin использует инструкцию switch/case, которая выбирает ресурсы FXG, созданные для определенного значения DPI, как показано в следующем примере:

```
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;
    }
}
```

Помимо определенного выбора ресурсов FXG, классы мобильной темы оформления также устанавливают значения других свойств стилей, например пробелов или заполнения в макетах. Эти настройки основаны на DPI целевого устройства.

Отсутствие установленного значения applicationDPI

Если свойство `applicationDPI` не установлено, темы оформления по умолчанию используют свойство `runtimeDPI`. Этот подход гарантирует, что тема оформления, значения которой основаны на свойстве `applicationDPI` вместо свойства `runtimeDPI`, использует соответствующий ресурс независимо от наличия или отсутствия масштабирования DPI.

При создании пользовательских тем оформления можно выбрать игнорирование установки `applicationDPI`. Однако хотя эта тема оформления будет масштабироваться в соответствии с DPI целевого устройства, при ее визуализации могут возникать ошибки, так как ресурсы этой темы не созданы с учетом этого значения DPI.

Использование applicationDPI в CSS

Значение свойства `applicationDPI` в селекторе CSS `@media` используется для настройки стилей мобильного или планшетного приложения без необходимости создания пользовательских тем оформления. Для получения подробной информации см. раздел «Выбор стилей на основе DPI» на странице 141.

Определение коэффициента масштабирования и текущего DPI вручную

Для ручной настройки выбора ресурсов в мобильном приложении на основе значения DPI целевого мобильного или планшетного устройства вычисляется коэффициент масштабирования в среде выполнения. Для этого необходимо разделить значение свойства `runtimeDPI` на значение свойства стиля `applicationDPI`:

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

Вычисленный коэффициент масштабирования применяется для выбора ресурсов вручную. В следующем примере определены пользовательские папки растровых ресурсов для каждого значения DPI. Затем загружается изображение из пользовательской папки:

```
<?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>
```

В примере ниже показано представление, которое использует этот коэффициент масштабирования:

```
<?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>
```

Изменение DPI по умолчанию

После установки значения DPI приложение масштабируется на основе значения DPI устройства, на котором запускается это приложение. В некоторых случаях устройства могут передавать недействительные значения DPI или требуется изменить стандартный метод выбора DPI с учетом пользовательского метода масштабирования.

Чтобы изменить стандартное поведение масштабирования приложения, переопределите сопоставление DPI по умолчанию. Например, если устройство сообщает недействительные значения (240 DPI вместо 160 DPI), создайте пользовательское сопоставление для этого устройства и определите его DPI со значением 160.

Чтобы изменить значение DPI определенного устройства, свяжите свойство `runtimeDPIProvider` класса Application с подклассом RuntimeDPIProvider. В этом подклассе измените получатель `runtimeDPI` и добавьте логику для пользовательского сопоставления DPI. Не добавляйте зависимости от других классов в инфраструктуре, например от UIComponent. Этот подкласс может вызываться только в API Flash Player.

В следующем примере устанавливается пользовательское сопоставление DPI для устройства, свойство `Capabilities.os` которого соответствует значению Mac 10.6.5:

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

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

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

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

        return DPIClassification.DPI_320;
    }
}
}
```

В следующем приложении используется DPITestClass для определения значения DPI среды выполнения, которое используется при масштабировании. Оно определяет ViewNavigatorApplication свойства класса `runtimeDPIProvider`:

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

</s:ViewNavigatorApplication>
```

Ниже приведен еще один пример подкласса RuntimeDPIProvider. В этом случае пользовательский класс проверяет значения x и y разрешения устройства для определения того, сообщает ли устройство действительные значения своего значения DPI:

```
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;
    }
}
}
```

Глава 5. Текст.

Использование текста в мобильном приложении

Рекомендации относительно текста в мобильном приложении

Некоторые текстовые элементы Spark оптимизированы для использования в мобильных устройствах. Рекомендуется по возможности использовать следующие текстовые элементы управления:

- Spark TextArea
- Spark TextInput
- Spark Label

В качестве основного механизма ввода элементы управления текстом, предоставляющие возможности взаимодействия (TextArea и TextInput), используют класс StageText. StageText встраивается в собственные элементы управления текстом используемой ОС. В результате эти элементы управления текстом функционируют как собственные элементы управления, а не как стандартные элементы управления Flex.

Ниже перечислены преимущества использования StageText на поддерживаемых этот класс устройствах:

- высокая производительность и функциональность виртуальной клавиатуры;
- автозаполнение;
- автоматическое исправление;
- выбор текста посредством прикосновений;
- настраиваемая виртуальная клавиатура;
- ограничение клавиш.

Специалист по продукции Adobe Christian Cantrell рассматривает преимущества и недостатки использования элементов управления на основе StageText.

Также доступны версии элементов управления TextArea и TextInput на основе TextField. Эти версии используются для встраивания шрифтов или выполнения других функций, которые недоступны в версиях на основе StageText.

Темы оформления для мобильных элементов управления текстом

При создании мобильного приложения Flex автоматически использует мобильную тему. Элементы управления Spark TextInput и TextArea по умолчанию используют следующие мобильные темы оформления на основе StageText:

- StageTextAreaSkin
- StageTextInputSkin

Классы StageTextAreaSkin и StageTextInputSkin оптимизированы для мобильных приложений и основаны на классе StageTextSkinBase. Они функционируют в качестве оболочки для собственных классов текстового ввода. Однако они не поддерживают следующие функции тем оформления, которые не основаны на TextField:

Поддерживается элементами управления на основе TextField	**Не поддерживается элементами управления на основе TextField**
Формы с прокруткой	Текст Layout Framework (TLF)
Анализ текста	Двунаправленность и зеркальное отображение
Отсечение	Compact Font Format (CFF)
Встроенные шрифты	RichEditableText для визуализации текста
Дробные значения альфа-канала	Текст HTML
Flash Text Engine (FTE)	
Доступ к низкоуровневым событиям клавиатуры, например `keyUp` и `keyDown`	

Альтернативным решением для некоторых из этих ограничений является использование версий на основе TextField. Чтобы использовать версии элементов управления текстовым вводом на основе TextField, укажите в их классах тем оформления версии TextField: TextInputSkin и TextAreaSkin, например:

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin" text="TextField-based Skin"/>
```

Элемент управления Spark Label не использует темы оформления, но также не использует TLF.

TLF в мобильном приложении

Как правило, в мобильных приложениях не рекомендуется использовать текстовые элементы управления на основе Text Layout Framework (TLF). Мобильные темы оформления элементов управления TextArea и TextInput оптимизированы для мобильных приложений и, в отличие от своих настольных и веб-аналогов, не используют TLF. В приложениях TLF предоставляет расширенные функции управления визуализацией текста.

Избегайте применения в мобильном приложении следующих элементов управления текстом, поскольку они используют TLF и их темы оформления не оптимизированы для мобильных приложений:

- Spark RichText
- Spark RichEditableText

Ввод с помощью виртуальной клавиатуры

Если для ввода информации пользователь перемещает фокус на текстовый элемент управления, на экране мобильных устройств без клавиатуры отображается виртуальная клавиатура. Пользователь может настроить некоторые функции доступных клавиш и другие свойства виртуальной клавиатуры. Например, можно включить функции автоматического исправления и верхнего регистра, а также выбрать любую из предварительно установленных раскладок клавиатуры.

Для получения подробной информации см. раздел «Использование виртуальной клавиатуры в мобильном приложении» на странице 163.

Прокрутка с помощью элементов управления вводом текста

Стандартные мобильные темы оформления для элементов управления TextInput и TextArea не поддерживают формы с прокруткой. Другими словами, эти элементы управления невозможно отобразить в формах или представлениях, использующих прокрутку. При попытке отображения этих элементов возникают визуальные помехи, являющиеся следствием особенностей реализации этих элементов управления.

Чтобы использовать элементы управления вводом текста в прокручиваемом контейнере, выберите темы оформления на основе TextField вместо тем оформления на основе StageText. Более подробные сведения см. в разделе «Рекомендации по использованию прокрутки и StageText» на странице 75.

Переходы с элементами управления вводом текста

Для выполнения плавной анимации среда выполнения заменяет элементы управления StageText захваченными растровыми изображениями при воспроизведении анимации. При этом в начале анимации перехода может возникнуть небольшая задержка, также возможны некоторые визуальные артефакты в начале и конце анимации.

Задержка соответствует времени, которое требуется для захвата растрового изображения компонентов текста. Чем больше площадь и количество элементов управления на основе StageText, тем длиннее может быть задержка. Чтобы сократить время задержки, постарайтесь не анимировать крупные или многочисленные компоненты на основе StageText.

Всплывающие объекты с элементами управления вводом текста

Компоненты управления вводом текста на основе StageText, расположенные вне самого верхнего всплывающего объекта, замещаются растровыми изображениями при отображении всплывающего объекта. В результате:

- Используйте модальные всплывающие объекты вместо немодальных. При отображении модального всплывающего объекта ожидается, что компоненты, расположенные вне этого объекта, не будут интерактивными. В этом случае замена StageText растровыми объектами практически незаметна.
- Компоненты на основе StageText необходимо использовать только внутри всплывающих объектов, реализующих интерфейс IFocusManagerContainer. Например, используйте SkinnableContainer или его производные объекты в качестве основы для всплывающих объектов.
- Используйте контейнер Callout, если требуется сохранить активность текстовых компонентов в нижних слоях. Если компонент текста является владельцем выноски, то он остается активным и устанавливает стрелку выноски на этот текстовый компонент. Для пользователя это может стать естественной подсказкой о том, что данный компонент можно использовать.
- Избегайте одновременного отображения нескольких всплывающих объектов. Если одновременно отображается несколько всплывающих объектов, интерактивным будет только самое верхнее из них. Однако если всплывающие объекты не перекрывают друг друга, пользователю трудно определить, который объект является самым верхним.
- Когда немодальные всплывающие объекты накладываются на элементы управления текстом, разместите всплывающий объект так, чтобы наложение было практически полным. Если пользователь не может видеть текст, то трудно определить, что элемент управления текстом является интерактивным.

Встраивание шрифтов в мобильное приложение

В элементах управления вводом текста на основе StageText невозможно использовать встроенные шрифты. Вместо этого выберите для элементов управления вводом текста темы оформления на основе TextField. Кроме того, встроенные шрифты невозможно использовать в элементе управления Label, поскольку он использует FTE, для которого требуются шрифты на основе CFF. Шрифты CFF не рекомендуется использовать в мобильных приложениях.

Для получения подробной информации см. раздел «Встраивание шрифтов в мобильное приложение» на странице 177.

Иерархия классов элемента управления StageText

Классы StageText (TextInput и TextArea) имеют сложную иерархию. Ниже представлена иерархия базовых классов:

```
    UIComponent
        |
SkinnableComponent
        |
SkinnableTextBase
        |
TextInput/TextArea
```

Как и все классы Spark, классы тем оформления имеют собственную иерархию:

```
    UIComponent
        |
    MobileSkin    StyleableStageText
        |              |
StageTextSkinBase: textDisplay
        |
StageTextInputSkin/StageTextAreaSkin
```

В базовом классе темы оформления свойство `textDisplay` определяет подключение к собственным функциям ввода текста в виде объекта StyleableStageText. Этот класс также определяет стили, доступные в элементах управления вводом текста на основе StageText.

Использование элемента управления Label в мобильном приложении

Элемент управления Spark Label предназначен для использования в отдельных строках текста, который невозможно изменять или выбирать.

Ниже приведен пример мобильного приложения, в котором используется элемент управления Label:

```
<?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>
```

Элемент управления Label использует FTE, функциональность которого снижена по сравнению с текстовыми элементами управления, которые оптимизированы для мобильных приложений, например TextInput и TextArea. Однако элемент управления Label не использует TLF, поэтому его эффективность может быть выше, чем эффективность таких элементов управления, как RichText и RichEditableText, которые реализуют TLF.

Как правило, элементы управления Spark Label следует использовать в мобильных приложениях с осторожностью. Не используйте элемент управления Spark Label в темах оформления или средствах визуализации элементов. При создании средства визуализации элементов на основе ActionScript используйте для визуализации текста класс StyleableTextField. Для компонентов на основе MXML можно также использовать Label.

Поскольку элемент управления Label использует CFF, не применяйте его при встраивании шрифтов в мобильное приложение. Вместо него используйте версию элемента управления TextArea на основе TextField. Для получения подробной информации см. раздел «Встраивание шрифтов в мобильное приложение» на странице 177.

Использование элемента управления TextArea в мобильном приложении

Spark TextArea является элементом управления вводом текста, позволяющим вводить и изменять многострочный текст. Этот класс оптимизирован для мобильных приложений.

Стандартной функцией элемента управления TextArea является использование виртуальной клавиатуры, которая встраивается в собственные методы используемой ОС. Поэтому этот элемент поддерживает такие функции, как автоматическое исправление и заполнение, а также настройку виртуальной клавиатуры.

В примере мобильного приложения ниже используется элемент управления TextArea:

```
<?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>
```

В мобильных приложениях элемент управления TextArea использует класс StageTextAreaSkin для своей темы оформления. Визуализация текста в этой теме оформления выполняется с помощью класса StyleableStageText вместо класса RichEditableText. Поэтому элемент управления TextArea не поддерживает TLF. Он поддерживает только подмножество стилей, доступных в элементе управления TextArea с немобильной темой оформления.

Если требуется использовать многострочные блоки текста без возможности взаимодействия, установите для свойства `editable` элемента управления TextArea значение `false`. (Во время выполнения не учитывается свойство `selectable`.) Чтобы удалить рамку, укажите для свойства `borderVisible` значение `false`. Если требуется изменить цвет фона, установите свойства `contentBackgroundColor` и `contentBackgroundAlpha`.

В следующем примере показано создание блока текста без возможности взаимодействия, который сливается с фоном приложения:

```
<?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>
```

Поскольку элемент управления TextArea не поддерживает TLF, в нем не могут использоваться свойства `textFlow`, `content` или `selectionHighlighting`. Кроме того, невозможно использовать указанные ниже методы:

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

Использование элемента управления TextInput в мобильном приложении

Spark TextInput является элементом управления вводом текста, который позволяет вводить и редактировать отдельную строку текста. Этот класс оптимизирован для мобильных приложений.

Стандартной функцией элемента управления TextInput является использование виртуальной клавиатуры, которая встраивается в собственные методы используемой ОС. Поэтому этот элемент поддерживает такие функции, как автоматическое исправление и заполнение, а также настройку виртуальной клавиатуры.

В примере мобильного приложения ниже показаны элементы управления TextInput с текстом запроса и пользовательскими кольцами фокуса:

```
<?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>
```

В мобильных приложениях элемент управления TextInput по умолчанию использует класс StageTextInputSkin для своей темы оформления. Визуализация текста в этой теме оформления выполняется с помощью класса StyleableStageText вместо класса RichEditableText. Поэтому элемент управления TextInput не поддерживает TLF. Он поддерживает только подмножество стилей, доступных в элементе управления TextInput с немобильной темой оформления.

Использование элементов управления RichText и RichEditableText в мобильном приложении

В мобильном приложении не рекомендуется использовать элементы управления RichText и RichEditableText. Эти элементы управления не содержат мобильные темы оформления и не оптимизированы для мобильных приложений. В случае использования этих элементов управления используется TLF, приводящий к большим объемам вычислений.

Текстовые элементы управления MX

Текстовые элементы управления MX, такие как MX Text и MX Label, не используются в мобильных приложениях. Вместо них используйте эквиваленты Spark.

Установка стилей для элементов управления вводом текста в мобильном приложении

Элементы управления TextInput и TextArea поддерживают только подмножество стилей с мобильной теме. Эти стили определяет класс StyleableStageText.

Ниже перечислены стили, которые поддерживаются TextInput и TextArea в мобильном приложении:

- `color`
- `contentBackgroundAlpha`
- `contentBackgroundColor`
- стили кольца фокуса: `focusAlpha`, `focusBlendMode`, `focusColor` и `focusThickness`
- `fontFamily`
- `fontStyle`
- `fontSize`
- `fontWeight`
- `locale`
- стили заполнения: `paddingBottom`, `paddingLeft`, `paddingRight` и `paddingTop`
- `showPromptWhenFocused`
- `textAlign`

В мобильном приложении элемент управления Label поддерживает эти стили, а также стиль `textDecoration`.

Использование стиля fontFamily

В стандартных мобильных темах оформления свойство `fontFamily` не поддерживает список шрифтов с разделителем-запятой. Пользователь может указать только один шрифт, и среда выполнения сопоставит эти данные со шрифтом, установленным на устройстве.

Например, если указан шрифт Arial, устройство применяет шрифт Arial к тексту, если этот шрифт доступен. Среда выполнения пытается определить наиболее подходящий шрифт для замены, если указанный шрифт недоступен. Если среде выполнения не удается распознать указанное имя шрифта, устройство визуализирует текст с использованием шрифта по умолчанию. В качестве шрифта по умолчанию на мобильных устройствах обычно используется sans-serif.

Чтобы всегда использовать шрифты sans-serif, serif или шрифт кода на мобильном устройстве, установите значения `_sans`, `_serif` или `_typewriter` соответственно.

Ниже приведен пример визуализации текста на основе значения стиля `fontFamily`:

```
<?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>
```

Создание пользовательской мобильной темы оформления для элемента управления вводом текста

Возможности MXML и ActionScript позволяют изменить внешний вид и функции элементов управления вводом текста в мобильных приложениях. Например, можно изменить цвет рамки или переключить отображение границ элементов управления TextArea и TextInput. В некоторых случаях необходимо создать пользовательскую тему оформления, чтобы изменить внешний вид определенных компонентов элементов управления текстом.

Классы StageTextAreaSkin и StageTextInputSkin определяют стандартные темы оформления TextInput и TextArea в мобильной теме. Большинство функций макета и хрома этих тем оформления определяется в классе StageTextSkinBase.

Чтобы создать пользовательскую тему оформления, создайте пользовательский класс StageTextSkinBase, который будет определять новый внешний вид. Затем создайте пользовательский класс StageTextAreaSkin или StageTextInputSkin, расширяющий этот пользовательский класс.

Для получения подробной информации о создании пользовательских тем оформления в мобильной теме см. раздел «Основы создания мобильных тем оформления» на странице 179.

Ограничение клавиш в элементе управления вводом текста

Если в элементах управления вводом текста используются темы оформления на основе StageText, можно ограничить разрешенные символы с помощью свойства `restrict` элементов управления TextInput или TextArea.

Для свойства `restrict` по умолчанию установлено значение `null`, т. е. пользователь может по умолчанию вводить любой символ.

Свойство `restrict` принимает строку разрешенных символов. Для диапазонов используется дефис (-). Не разделяйте диапазоны пробелом, запятой или другими символами, если не требуется включить этот символ в определение. В следующем примере приведены различные способы использования синтаксиса ограничений:

```
<?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>
```

Строковое значение свойства `restrict` читается слева направо; все символы после символа вставки (^) являются недопустимыми. Например:

```
ta1.restrict = "A-Z^Q"; // All uppercase alpha characters, but exclude Q
```

Если первым символом в строке является символ вставки (^), разрешены все символы, кроме символов, следующих за символом вставки. Например:

```
ta1.restrict = "^a-z"; // All characters, but exclude lowercase alpha
```

Чтобы экранировать специальные символы, можно использовать обратную косую черту. Например, можно ограничить использование символа вставки:

```
ta1.restrict = "^\^"; // All characters, but exclude the caret
```

Для ввода кодов клавиш ASCII можно использовать \u, например:

```
ta1.restrict = "\u0020-\u007E"; // ASCII chars 32 through 126 only
```

Взаимодействие пользователя с текстом в мобильном приложении

Текстовые элементы управления допускают использование жестов, например манипуляцию «удар». В примере ниже прослушивается событие манипуляции «удар» и определяется ее направление:

```
<?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>
```

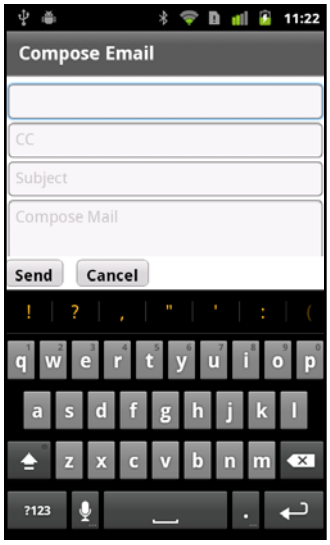
При использовании жестов касания и перетаскивания текст всегда выбирается, но только если его можно выбирать или изменять. Если не требуется выделять текст, когда пользователь выполняет манипуляцию касания и перетаскивания или удара по текстовому элементу управления, укажите для свойств `selectable` и `editable` значение `false` или сбросьте значение выбранного фрагмента, вызвав метод `selectRange(0,0)` в обработчике события удара.

Если текст расположен в элементе Scroller, то прокрутка Scroller возможна только в том случае, если жест выполнен за пределами текстового компонента.

Использование виртуальной клавиатуры в мобильном приложении

Многие устройства не используют аппаратную клавиатуру. Функции ввода на таких устройствах выполняет экранная клавиатура, которая отображается при необходимости. Окно экранной, или виртуальной, клавиатуры закрывается после того, как пользователь завершил ввод информации или отменил операцию.

На следующей иллюстрации показано приложение, использующее виртуальную клавиатуру:



Функции виртуальной клавиатуры могут различаться в зависимости от компонента, вызывающего эту клавиатуру:

- Собственные функции – клавиатура, используемая со стандартными элементами управления текстовым вводом TextArea и TextInput, подключается к собственному интерфейсу для выполнения таких функций, как автоматическое исправление и заполнение, а также создание пользовательской раскладки клавиатуры. Поддержка полного набора функций встроена в стандартные классы тем оформления на основе StageText в элементах управления вводом текста. Не все устройства поддерживают полный набор собственных функций.
- Ограниченные функции – клавиатура, используемая с любыми элементами управления, кроме TextArea и TextInput, или с элементами управления TextArea и TextInput в случаях, когда они используют темы оформления на основе TextField. Ограниченный набор функций не поддерживает собственные функции ОС, такие как автоматическое исправление и заполнение, а также создание пользовательской раскладки клавиатуры.

Так как окно клавиатуры занимает часть экрана, Flex необходимо обеспечить функционирование приложения в уменьшенной области экрана. Например, пользователь выбирает элемент управления TextInput, после чего открывается окно виртуальной клавиатуры. При этом Flex автоматически изменяет размер приложения в соответствии с доступной областью экрана. Затем Flex изменяет положение выбранного элемента управления TextInput и отображает его над клавиатурой.

Автор блога Peter Elst рассказывает об управлении виртуальной клавиатурой в мобильных приложениях Flex.

Открытие виртуальной клавиатуры в мобильном приложении Flex

Виртуальную клавиатуру можно открыть в мобильном приложении тремя способами:

- поместить фокус на элемент с элементом управления текстовым вводом, таким как TextInput или TextArea;
- установить для свойства `needsSoftKeyboard` элемента управления значение `true` и поместить фокус на этот элемент управления;
- вызвать метод `requestSoftKeyboard()` для элемента управления (но не в iOS).

Окно клавиатуры остается открытым до наступления одного из указанных ниже событий.

- Пользователь перемещает фокус на элемент управления, который не получает ввод текста. Такая ситуация возникает, когда пользователь вручную выбирает другой элемент управления текстовым вводом или нажимает кнопку возврата на клавиатуре, чтобы переместить фокус на другой элемент управления в приложении.

  Если фокус перемещается на другой элемент управления вводом текста или элемент, для свойства `needsSoftKeyboard` которого указано значение `true`, окно клавиатуры остается открытым.
- Пользователь отменяет ввод, нажав кнопку возврата на устройстве.
- Фокус можно переместить на неинтерактивный элемент управления программным способом или установить для `stage.focus` значение `null`.

Представление элемента управления вводом текста пользователям

Если управление вводом текста предоставляется пользователю, виртуальная клавиатура отображается, когда пользователь перемещает фокус на этот элемент управления; при этом для свойства `editable` не должно быть установлено значение `false`.

По умолчанию элементы управления TextInput и TextArea используют класс StageText для визуализации текста. Клавиатура, отображаемая для этих элементов управления, поддерживает такие собственные функции, как автоматическое исправление и автоматический верхний регистр, а также различные типы клавиатур. Не все устройства поддерживают полный набор функций.

Если в классах тем оформления для элементов управления вводом текста определены темы оформления на основе TextField, только некоторые функции будут доступны. Клавиатура останется без изменений, но не будет поддерживать собственные функции.

Установка свойства `needsSoftKeyboard`

Чтобы открыть окно виртуальной клавиатуры, можно использовать элементы управления без ввода данных, например Button или ButtonBar. Если требуется открыть окно клавиатуры, когда элемент управления без ввода текста получает фокус, установите для свойства `needsSoftKeyboard` этого элемента управления значение `true`. Все компоненты Flex наследуют это свойство из класса InteractiveObject.

Клавиатура, открываемая для любых элементов управления, кроме TextInput и TextArea, не поддерживает такие собственные функции, как автоматическое исправление и автоматический верхний регистр, а также настраиваемые типы клавиатур.

Примечание. *Элементы управления вводом текста всегда открывают окно клавиатуры при получении фокуса. Они игнорируют свойство `needsSoftKeyboard`, поэтому его установка не влияет на элементы управления вводом текста.*

Вызов метода `requestSoftKeyboard()`

Чтобы открыть клавиатуру программным способом, вызовите метод `requestSoftKeyboard()`. Для свойства `needsSoftKeyboard` объекта, вызывающего этот метод, необходимо также указать значение `true`. Этот метод помещает фокус на объект, который вызывает данный метод и открывает виртуальную клавиатуру, если устройство не оснащено аппаратной клавиатурой.

Класс InteractiveObject определяет метод `requestSoftKeyboard()`. Поэтому этот метод можно вызвать для любого компонента, являющего подклассом InteractiveObject.

Если метод `requestSoftKeyboard()` вызывается для элементов управления TextArea или TextInput, поддерживаются такие собственные функции клавиатуры, как автоматическое исправление и автоматический верхний регистр (при условии, что устройство поддерживает эти функции).

Метод `requestSoftKeyboard()` не работает на устройствах iOS.

Использование собственных функций с виртуальной клавиатурой

Классы StageTextInputSkin и StageTextAreaSkin определяют темы оформления для элементов управления TextInput и TextArea в мобильном приложении. Эти темы оформления используют класс StageText для визуализации текста и встраиваются в собственные функции виртуальной клавиатуры. Эти функции приведены ниже:

- автоматическое исправление;
- автоматический верхний регистр;
- пользовательские метки кнопки возврата;
- пользовательские типы клавиатуры.

Элементы управления вводом текста также могут использовать темы оформления на основе TextField для визуализации текста. Эти темы оформления не поддерживают собственные функции. Однако они предоставляют дополнительные функциональные возможности для основного элемента управления текстом, такие как формы с прокруткой, встраивание шрифтов, доступ к событиям `keyUp` и `keyDown`, отсечение, анализ текста и дробные значения альфа-канала.

Чтобы использовать темы оформления на основе TextField, укажите в свойстве `skinClass` элемента управления вводом текста связь с классами TextInputSkin иTextAreaSkin. Например:

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin"/>
<s:TextArea skinClass="spark.skins.mobile.TextAreaSkin"/>
```

Функция автоматического исправления для виртуальной клавиатуры в мобильном приложении Flex

Автоматическое исправление представляет собой функцию операционной системы, выполняющую коррекцию ошибок и предварительное заполнение строки пользовательского ввода. В зависимости от используемого устройства эта функция может быть реализована как всплывающий над текстом объект, расширение виртуальной клавиатуры или другими способами.

Чтобы использовать функцию автоматического исправления для виртуальной клавиатуры в мобильном приложении, установите для свойства `autoCorrect` элемента управления вводом текста значение `true`. Это значение является значением по умолчанию.

В примере ниже показан способ включения и выключения функции автоматического исправления:

```
<?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>
```

Не все устройства поддерживают функцию автоматического исправления. Если вы включаете или выключаете свойство `autoCorrect` на устройстве, которое его не поддерживает, среда выполнения будет игнорировать это значение и использовать стандартные функции устройства.

Функция автоматического верхнего регистра для виртуальной клавиатуры в мобильном приложении Flex

Автоматический верхний регистр – это параметр, который отправляет элементу управления вводом текста инструкции о преобразовании определенных слов или букв в верхний регистр, когда пользователь вводит текст. Например, можно автоматически преобразовать в верхний регистр все буквы в слове или только начальные слова в каждом предложении. Это очень удобная функция, поскольку пользователи могут не беспокоиться о преобразовании в верхний регистр при вводе текста в мобильном приложении.

Чтобы использовать автоматический верхний регистр для виртуальной клавиатуры, определите в значении свойства `autoCapitalize` элемент управления вводом текста. Возможные значения: `none`, `word`, `sentence` и `all`. Возможные значения определяет класс AutoCapitalize. Значение по умолчанию – `none`.

В примере ниже представлена возможность выбора различных значений для автоматического верхнего регистра:

```
<?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>
```

Не все устройства поддерживают функцию автоматического верхнего регистра. Если значение свойства `autoCapitalize` установлено на устройстве, которое его не поддерживает, среда выполнения будет игнорировать это значение и использовать стандартные функции устройства.

Изменение типов виртуальной клавиатуры в мобильном приложении Flex

Класс SoftKeyboardType определяет типы виртуальной клавиатуры в мобильных приложениях. Для выбора типа клавиатуры используется свойство `softKeyboardType` элемента управления вводом текста.

Как правило, различия между типами клавиатур, например `email` и `contact`, минимальны. Например, клавиатура типа `email` содержит те же клавиши, что и клавиатура типа `contact`, за исключением того, что вместо значка микрофона используется символ @. В клавиатуре типа `url` используется символ «/». Наибольшее количество отличий имеет клавиатура типа `number`. Эта клавиатура выглядит как калькулятор с цифрами и операторами.

В следующем примере показаны различные типы доступных виртуальных клавиатур:

```
<?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>
```

Не все устройства поддерживают полный набор типов виртуальных клавиатур. Если указан неподдерживаемый тип клавиатуры, среда выполнения будет игнорировать это значение и использовать стандартные функции устройства.

Изменение меток клавиш возврата на виртуальной клавиатуре в мобильном приложении Flex

Когда пользователи вводят текст на виртуальной клавиатуре, им необходимо сообщить приложению о том, что ввод завершен и требуется перейти к следующему полю или отправить введенные данные. На виртуальной клавиатуре для этого обычно используется клавиша возврата. Эта клавиша не вводит текст в поле, но сообщает элементу управления вводом текста о том, что пользователь завершил ввод данных.

Возможные метки клавиши возврата определяет класс ReturnKeyLabel. Возможные значения: `default`, `done`, `go`, `next` и `search`. Метка клавиши возврата определяется в свойстве `returnKeyLabel` элемента управления вводом текста.

В примере ниже представлена возможность выбора различных меток клавиши возврата:

```
<?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>
```

Различные типы клавиши возврата используют идентичные события или типы взаимодействия. Изменение свойства `returnKeyLabel` влияет только на метку клавиши.

Не все устройства поддерживают установку меток для клавиши возврата. Если значение свойства `returnKeyLabel` указано на устройстве, которое его не поддерживает, среда выполнения будет игнорировать это значение и использовать стандартные функции устройства.

Использование событий для виртуальной клавиатуры в мобильном приложении

Взаимодействие с виртуальной клавиатурой на мобильном устройстве отличается от взаимодействия в настольном или веб-приложении. В таблице ниже перечислены события, используемые при взаимодействии с виртуальной клавиатурой.

Событие	**Передача**
`enter`	Когда пользователь нажимает клавишу возврата.
`keyDown` и `keyUp`	Для тем оформления на основе StageText: когда пользователь нажимает и отпускает только определенные клавиши. Для тем оформления на основе TextField: когда пользователь нажимает и отпускает любые клавиши. Эти события отправляются только для некоторых клавиш на определенных устройствах. Используйте эти методы для захвата введенных данных на виртуальной клавиатуре только в том случае, если для элементов управления, открывающих клавиатуру, определены темы оформления на основе TestField.
`softKeyboardActivating`	Непосредственно перед открытием клавиатуры.

Событие	Передача
`softKeyboardActivate`	Непосредственно после открытия клавиатуры.
`softKeyboardDeactivate`	После закрытия клавиатуры.

Чтобы определить, когда пользователь завершил ввод данных на виртуальной клавиатуре, можно прослушать событие `FlexEvent.ENTER` для элемента управления вводом текста. Элемент управления отправляет это событие, когда пользователь нажимает клавишу возврата. При прослушивании события `enter` можно выполнить проверку, перемещение фокуса или другие действия с новым введенным текстом.

В некоторых случаях событие `enter` не отправляется. Это ограничение действует на устройствах Android, когда виртуальная клавиатура используется с последним введенным текстом в представлении. Чтобы избежать эту проблему, установите для свойства `returnKeyLabel` значение `go`, `next` или `search` в последнем введенном тексте в представлении.

В примере ниже фокус перемещается из текущего поля в следующее поле, когда пользователь нажимает клавишу «Далее» на виртуальной клавиатуре.

```
<?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>
```

Когда пользователь взаимодействует со стандартной виртуальной клавиатурой, использующей элементы управления TextInput и TextArea, события `keyUp` и `keyDown` отправляются только для небольшого подмножества клавиш. Если требуется захватить отдельные нажатия всех клавиш, используйте событие `change`. Событие `change` отправляется при каждом изменении содержимого элемента управления вводом текста. Недостатком этого способа является то, что пользователь не получает доступ к свойствам клавиши и должен создать собственную логику нажатия клавиши.

В следующем примере приведен код символа последней нажатой клавиши:

```
<?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>
```

Определение последней нажатой клавиши в этом примере выполняется только в том случае, если курсор находится в конце поля текстового ввода.

Если пользователь взаимодействует с виртуальной клавиатурой, использующей элементы управления на основе TextField, такие события, как `keyUp` и `keyDown`, работают для всех клавиш. В следующем примере обработчик `keyUp` используется для доступа к текущей клавише и применения стиля к элементу управления Label на основе кода клавиши. Поскольку метод `requestSoftKeyboard()` открывает клавиатуру с использованием элемента управления Label вместо элементов TextInput или TextArea, приложение использует клавиатуру на основе TextField.

```
<?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>
```

Чтобы закрыть виртуальную клавиатуру программным способом, установите для свойства `stage.focus` значение `null`. Если требуется закрыть виртуальную клавиатуру и переместить фокус на другой элемент управления, установите для свойства `stage.focus` значение `null` и затем переместите фокус на целевой элемент управления. Чтобы изменить фокус и открыть виртуальную клавиатуру для другого элемента управления, также можно вызвать метод `requestSoftKeyboard()` другого элемента управления.

Некоторые свойства виртуальной клавиатуры доступны из обработчиков событий. Чтобы изменить размер и местоположение виртуальной клавиатуры, используйте свойство `softKeyboardRect` класса flash.display.Stage, как показано в следующем примере:

```
<?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>
```

Настройка приложения для использования виртуальной клавиатуры

Приложение, поддерживающее виртуальную клавиатуру, может выполнить следующие действия при открытии окна клавиатуры:

- изменить размер приложения в соответствии с доступной областью экрана, чтобы окно клавиатуры не перекрывало окно приложения;
- прокрутить вышестоящий контейнер элемента управления вводом текста с фокусом, чтобы отобразить этот элемент управления.

Настройка системы для использования виртуальной клавиатуры

Виртуальная клавиатура не поддерживается в приложениях, запускаемых в полноэкранном режиме. Поэтому убедитесь, что в файле app.xml для атрибута `<fullScreen>` указано значение по умолчанию `false`.

Также убедитесь, что для режима визуализации приложения выбран режим ЦП. Режим визуализации контролируется в файле дескриптора приложения app.xml с помощью атрибута `<renderMode>`. Убедитесь, что для атрибута `<renderMode>` указано значение по умолчанию `cpu` вместо `gpu`.

Примечание. *Атрибут `<renderMode>` не включен по умолчанию в файл app.xml. Для изменения значений необходимо добавить соответствующую запись в атрибут `<initialWindow>`. Если атрибут не включен в файл app.xml, это значит, что он по умолчанию имеет значение `cpu`.*

Прокрутка вышестоящего контейнера при открытии окна виртуальной клавиатуры

Свойство `resizeForSoftKeyboard` контейнера Application определяет способ изменения размера приложения. Если для свойства `resizeForSoftKeyboard` указано значение по умолчанию `false`, клавиатура может отобразиться поверх приложения. При значении `true` приложение изменяет свой размер, чтобы заполнить область, доступную после открытия окна клавиатуры.

Для поддержки прокрутки элементы управления вводом текста должны использовать темы оформления на основе TextField вместо тем оформления на основе StageText. Для этого свойство `skinClass` элемента управления TextInput или TextArea, необходимо связать с классом TextInputSkin или TextAreaSkin соответственно.

Для выполнения прокрутки добавьте вышестоящий контейнер любых элементов управления вводом текста в компонент Scroller. Когда компонент, который открывает окно клавиатуры, получает фокус, Scroller автоматически прокручивает этот компонент в область видимости. Этот компонент может также являться нижестоящим элементом нескольких вложенных контейнеров компонента Scroller.

Вышестоящий контейнер должен быть представлен классом GroupBase или SkinnableContainer или их подклассами. Компонент, получающий фокус, должен реализовать интерфейс IVisualElement и иметь возможность использовать этот фокус.

Если вышестоящий контейнер включается в компонент Scroller, пользователь может прокручивать этот контейнер при открытом окне клавиатуры. Например, контейнер может содержать несколько элементов управления вводом текста. Для ввода данных прокрутите контейнер, чтобы отобразить каждый из этих элементов.

Когда окно клавиатуры закрывается, размер вышестоящего контейнера может быть меньше доступной области экрана. В этом случае Scroller устанавливает для положения средства прокрутки исходное значение 0, т. е. помещает его в верхнюю часть контейнера.

В примере ниже представлен контейнер View с несколькими элементами управления TextInput и компонентом Scroller:

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

    <s:Scroller width="100%" top="10" bottom="50">
        <s:VGroup paddingTop="3" paddingLeft="5" paddingRight="5" paddingBottom="3">
            <!-- 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>
```

Контейнер VGroup является вышестоящим контейнером элементов управления TextInput. Scroller включает в себя VGroup, поэтому каждый элемент управления TextInput отображается над клавиатурой при получении фокуса.

Для получения подробной информации о компоненте Scroller см. раздел Прокрутка контейнеров Spark.

Изменение размера приложения при открытии окна виртуальной клавиатуры

Если для свойства `resizeForSoftKeyboard` контейнера Application указано значение `true`, то при открытии клавиатуры приложение изменяет свои размеры в соответствии с доступной областью экрана. Когда окно клавиатуры закрывается, восстанавливается исходный размер приложения.

В приведенном ниже примере показан основной файл приложения, которое поддерживает изменение размера с помощью установки для свойства `resizeForSoftKeyboard` значения `true`:

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

</s:ViewNavigatorApplication>
```

Чтобы включить функцию изменения размера приложения, укажите для атрибута `<softKeyboardBehavior>` значение `none` в файле дескриптора app.xml приложения. Значением атрибута `<softKeyboardBehavior>` по умолчанию является `none`. Если установлено это значение по умолчанию, AIR перемещает весь объект Stage для отображения компонента текста с фокусом.

Хотя события виртуальной клавиатуры достаточно надежны для практически постоянного автоматического функционирования, старайтесь не размещать критически важные элементы пользовательского интерфейса в местах, где они могут быть скрыты под виртуальной клавиатурой в случае ошибки при выполнении события виртуальной клавиатуры. Например, не размещайте кнопки OK, Login или Submit в нижней части представления.

При анимации или участии в анимации элемента управления на основе StageText среда выполнения временно замещает его растровым изображением, чтобы обеспечить синхронизацию движения текста и других элементов. Если этот элемент управления находится в фокусе, то он временно теряет фокус. В некоторых случаях виртуальная клавиатура скрывает или запускает событие `softKeyboardDeactivate` без скрытия экранной клавиатуры.

При этом могут возникнуть трудности с программной установкой фокуса на компоненты, использующие StageText и находящиеся во всплывающих окнах, которые анимируют изменения размеров, вызванные виртуальной клавиатурой. Чтобы избежать этих проблем, используйте компоненты на основе StageText в таких всплывающих окнах, которые не изменяют размер в зависимости от экранной клавиатуры.

Если необходимо использовать компоненты на основе StageText во всплывающих окнах, размер которых меняется в зависимости от экранной клавиатуры, сначала выведите виртуальную клавиатуру и затем дождитесь завершения анимации всплывающего окна, перед тем как программным способом установить фокус на компонент, использующий StageText. В качестве альтернативы можно не устанавливать фокус во всплывающих объектах программным путем.

Настройка всплывающих элементов в виртуальной клавиатуре

Контейнер Callout выводится как всплывающий элемент поверх мобильного приложения. Контейнер Callout может содержать один или несколько компонентов, которые принимают ввод с клавиатуры. Для получения подробной информации о контейнере Callout см. раздел «Использование контейнера Callout для создания выноски» на странице 89.

Свойства контейнера SkinnablePopUpContainer, который является вышестоящим классом Callout, используются для настройки взаимодействия всплывающего элемента с клавиатурой:

moveForSoftKeyboard - если указано значение по умолчанию `true`, всплывающий элемент перемещается поверх открытой клавиатуры.

resizeForSoftKeyboard - если указано значение по умолчанию `true`, всплывающий элемент изменяет свои размеры в соответствии с доступным пространством над открытой клавиатурой.

Если для свойств `moveForSoftKeyboard` или `resizeForSoftKeyboard` в SkinnablePopUpContainer установлено значение `true`, контейнер может перемещаться или изменять размер при любом изменении видимости виртуальной клавиатуры. Следствием этого автоматического поведения может стать перемещение или изменение размера других компонентов.

Чтобы избежать такие изменения, не устанавливайте для `resizeForSoftKeyboard` значение `true`. Если приложение не изменяет размер в соответствии с виртуальной клавиатурой, то клавиатура не может влиять на перемещение компонента из-под нее, выполняемое пользователем. Однако при этом некоторые пользовательские интерфейсы могут быть скрыты под виртуальной клавиатурой.

Если для приложения требуется автоматическое изменение размера, поместите интерактивные элементы таким образом, чтобы изменение видимости виртуальной клавиатуры не вызывало их перемещения, когда пользователь прикасается к ним пальцем. Для этого можно выполнить следующие действия:

- Не определяйте для кнопок, флажков или других небольших объектов ограничения по нижнему краю или размещайте их под другими компонентами с указанием высоты в процентах.

- При создании макета определите стационарное размещение для верхнего из самых нижних компонентов при открытии или скрытии клавиатуры.
- Если на платформе отсутствует встроенная поддержка скрытия виртуальной клавиатуры, задайте эту функцию стационарному элементу пользовательского интерфейса. Таким элементом может стать кнопка, назначенная для скрытия клавиатуры, или стационарная пустая граница достаточно большого размера, чтобы прикосновение к ней позволяло удалить фокус с компонента текста.
- Не размещайте компоненты, которые могут передвигаться, по вертикали. В противном случае возможны ошибки при определении правильной цели прикосновением, когда изменяется видимость виртуальной клавиатуры.

Встраивание шрифтов в мобильное приложение

При встраивании шрифтов в мобильные приложения применяются определенные ограничения.

Поскольку в элементе управления Label применяется FTE и, следовательно, шрифты на основе CFF, его необходимо заменить на элементы управления TextArea или TextInput с темами оформления на основе TextField при встраивании шрифтов в мобильное приложение. Шрифты с темами оформления на основе StageText не могут быть встроены. Как правило, не рекомендуется использовать FTE в мобильных приложениях.

В CSS укажите для `embedAsCFF` значение `false` и примените тему оформления на основе TextField, как показано в примере ниже:

```
<?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>
```

Элемент управления TextArea в представлении EmbeddingFontView применяет селектор типа:

```
<?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>
```

Если используется селектор класса, такой как `s|TextArea`, для применения стилей или встраивания шрифтов, необходимо определить селектор класса в файле главного приложения. Селекторы класса не могут определяться в представлении мобильного приложения.

Для получения подробной информации см. раздел Встраивание шрифтов.

Глава 6. Создание тем оформления

Основы создания мобильных тем оформления

Сравнение мобильных и настольных тем оформления

Мобильные темы оформления представляют собой облегченный вариант своих настольных аналогов. Отличия между этими темами оформления заключаются в следующем:

- Для написания мобильных тем оформления используется ActionScript. Темы, созданные с помощью ActionScript, имеют наивысшую производительность в мобильных устройствах.
- Мобильные темы оформления расширяют класс spark.skins.mobile.supportClasses.MobileSkin. Этот класс расширяет UIComponent, в отличие от класса SparkSkin, который расширяет класс Skin.
- Для повышения производительности мобильные темы оформления используют для своих графических ресурсов скомпилированные FXG или простые рисунки ActionScript. В темах оформления для настольных приложений чаще всего используются графические объекты MXML.
- Мобильным темам оформления не нужно объявлять состояния. Поскольку темы оформления созданы в ActionScript, состояния должны реализоваться в процессе.
- Мобильные темы оформления не поддерживают переходы состояний.
- Мобильные темы оформления располагаются вручную. Поскольку мобильные темы оформления не расширяют Group, они не поддерживают макеты Spark. В результате их нижестоящие элементы вручную располагаются в ActionScript.
- Мобильные темы оформления поддерживают не все стили. Мобильная тема оформления пропускает некоторые стили, основанные на производительности или других различиях в мобильных темах оформления.

Помимо различий, связанных с производительностью, в Flash Builder иначе используются некоторые файлы мобильных тем оформления. Это относится, в частности, к мобильным темам, используемым в проектах библиотеки. Автор блога Jeffry Houser предлагает решение этих проблем.

Мобильный компонент хоста

Мобильные темы оформления обычно объявляют общедоступное свойство `hostComponent`. Это свойство не является обязательным, но рекомендуется для использования. Тип свойства `hostComponent` должен соответствовать типу компонента, который использует тему оформления. Например, ActionBarSkin объявляет тип ActionBar для компонента `hostComponent`:

```
public var hostComponent:ActionBar;
```

Flex устанавливает значение свойства `hostComponent`, когда компонент впервые загружает тему оформления.

Как и при работе с настольными темами оформления, компонент хоста можно использовать для доступа к свойствам и методам компонента, к которому присоединяется тема оформления. Например, можно использовать общедоступные свойства компонента хоста или добавить прослушиватель событий к компоненту хоста из класса темы оформления.

Мобильные стили

Мобильные темы оформления поддерживают подмножество свойств стиля, которое поддерживают их настольные аналоги. Этот набор стилей определяет мобильная тема.

В таблице ниже приведены свойства стилей, доступные для компонентов в мобильной теме:

Свойство стиля	Поддерживающий объект	Наследует/не наследует
`accentColor`	Button, ActionBar, ButtonBar	Наследует
`backgroundAlpha`	ActionBar	Не наследует
`backgroundColor`	Application	Не наследует
`borderAlpha`	List	Не наследует
`borderColor`	List	Не наследует
`borderVisible`	List	Не наследует
`chromeColor`	ActionBar, Button, ButtonBar, CheckBox, HSlider, RadioButton	Наследует
`color`	Все компоненты с текстом	Наследует
`contentBackgroundAlpha`	TextArea, TextInput	Наследует
`contentBackgroundColor`	TextArea, TextInput	Наследует
`focusAlpha`	Все компоненты с фокусом	Не наследует
`focusBlendMode`	Все компоненты с фокусом	Не наследует
`focusColor`	Все компоненты с фокусом	Наследует
`focusThickness`	Все компоненты с фокусом	Не наследует
`locale`	Все компоненты	Наследует
`paddingBottom`	TextArea, TextInput	Не наследует
`paddingLeft`	TextArea, TextInput	Не наследует
`paddingRight`	TextArea, TextInput	Не наследует
`paddingTop`	TextArea, TextInput	Не наследует
`selectionColor`	ViewMenuItem	Наследует

Все компоненты на основе текста также поддерживают стандартные стили текста, такие как `fontFamily`, `fontSize` и `fontWeight`.

Чтобы определить, поддерживает ли мобильная тема свойство стиля, см. описание компонента в справочнике по языку ActionScript. Многие ограничения стиля основаны на том, что текстовые мобильные компоненты не используют TLF (Text Layout Framework). В мобильных темах оформления текстовые элементы управления на основе TLF замещены облегченными компонентами. Подробную информацию см. в разделе «Использование текста в мобильном приложении» на странице 152.

Мобильная тема не поддерживает такие свойства стиля, как `rollOverColor`, `cornerRadius` и `dropShadowVisible`.

Специалист по Flex Jason SJ рассматривает создание тем и стилей для мобильных приложений в своем блоге.

Компоненты мобильных тем оформления

Мобильные темы оформления должны выполнять условия соглашения о создании темы оформления в отношении компонентов тем оформления, как и их настольные аналоги. Если компонент содержит необходимый компонент темы оформления, мобильная тема оформления должна объявить общедоступное свойство соответствующего типа.

Исключения

При этом не все компоненты тем оформления могут быть востребованы. Например, Spark Button содержит необязательные компоненты `iconDisplay` и `labelDisplay` темы оформления, поэтому мобильный класс ButtonSkin может объявить свойство `iconDisplay` типа BitmapImage или свойство `labelDisplay` типа StyleableTextField.

Компонент `labelDisplay` не устанавливает свойство `id`, поскольку все используемые ним стили наследуют стили текста. Кроме того, StyleableTextField не является UIComponent и поэтому не содержит свойство `id`. Компонент `iconDisplay` не поддерживает стили и поэтому также не устанавливает свойство `id`.

Установка стилей с помощью расширенного CSS

Если для компонента темы оформления требуется установить стили с использованием расширенного селектора CSS `id`, эта тема оформления должна также установить свойство `id` компонента темы оформления. Например, в ActionBar компонент темы оформления `titleDisplay` устанавливает свойство `id` для использования в расширенном CSS:

```
@namespace s "library://ns.adobe.com/flex/spark";
s|ActionBar #titleDisplay {
    color:red;
}
```

Мобильная тема

Мобильная тема определяет, какие стили поддерживает мобильное приложение. Количество стилей, доступных в мобильной теме, является подмножеством темы Spark (с небольшими дополнениями). Полный список стилей, поддерживаемых мобильными темами, представлен в разделе «Мобильные стили» на странице 180.

Тема по умолчанию для мобильных приложений

Темы мобильных приложений определяются в файле themes/Mobile/mobile.swc. Этот файл определяет глобальные стили мобильных приложений, а также настройки по умолчанию для всех мобильных компонентов. Мобильные темы оформления в этом файле определяются в пакете spark.skins.mobile.*. Этот пакет включает классы Базовый класс MobileSkin.

Файл темы mobile.swc по умолчанию входит в мобильные проекты Flash Builder, однако файл SWC в проводнике пакетов не отображается.

При создании мобильного проекта в Flash Builder эта тема применяется по умолчанию.

Изменение темы

Для изменения темы необходимо указать новую тему в аргументе компилятора `theme`, например:

```
-theme+=myThemes/NewMobileTheme.swc
```

Для получения дополнительной информации см. раздел О темах.

Специалист по Flex Jason SJ рассматривает создание и перекрытие тем мобильных приложений в своем блоге.

Состояния мобильных тем оформления

Класс MobileSkin переопределяет механизм состояний класса UIComponent и не использует реализацию состояний представлений настольных приложений. Поэтому мобильные темы оформления объявляют только состояния тем оформления компонента хоста, которые реализует эта тема оформления. Они изменяют состояние в процессе, основываясь только на имени состояния. Настольные темы оформления, наоборот, должны объявлять все состояния независимо от их использования. Настольные темы оформления также используют классы в mx.states.* (пакете для изменения состояний).

Как правило, мобильные темы оформления реализуют меньше состояний, чем их настольные аналоги. Например, класс spark.skins.mobile.ButtonSkin реализует состояния `up`, `down` и `disabled`. Класс spark.skins.spark.ButtonSkin реализует все эти состояния, а также состояние `over`. Мобильная тема оформления не определяет поведение состояния `over`, поскольку это состояние обычно не используется для устройств с сенсорным экраном.

Метод commitCurrentState()

Классы мобильных тем оформления определяют поведение своих состояний в методе `commitCurrentState()`. Если требуется поддержка дополнительных состояний, соответствующее поведение можно добавить в мобильную тему оформления, отредактировав метод `commitCurrentState()` в классе пользовательской темы оформления.

Свойство currentState

Внешний вид темы оформления зависит от значения свойства `currentState`. Например, в мобильном классе ButtonSkin значение свойства `currentState` определяет класс FXG, который используется в качестве класса границы:

```
if (currentState == "down")
    return downBorderSkin;
else
    return upBorderSkin;
```

Для получения информации о свойстве `currentState` см. раздел Создание и применение состояний представлений.

Мобильные графические объекты

Мобильные темы оформления обычно используют скомпилированные FXG для своих графических ресурсов. В темах оформления для настольных приложений чаще всего используются графические объекты MXML.

Встроенные растровые объекты

Приемлемая производительность также достигается при использовании встроенных растровых объектов в классах. Однако масштабирование растровых объектов не всегда выполняется безупречно на экранах с различной плотностью. Улучшить выполнение масштабирования можно, создав несколько различных ресурсов, по одному для каждой плотности экрана.

Графические объекты в стандартных мобильных темах

Мобильные темы оформления в стандартных мобильных темах используют графические объекты FXG, которые оптимизируются для DPI целевого устройства. Темы оформления загружают графические объекты в зависимости от значения свойства `applicationDPI` корневого приложения. Например, если элемент управления CheckBox используется в устройстве с DPI 320, класс CheckBoxSkin использует графический объект spark.skins.mobile320.assets.CheckBox_up.fxg для свойства `upIconClass`. При 160 DPI используется графический объект spark.skins.mobile160.assets.CheckBox_up.fxg.

Следующий пример *desktop* демонстрирует различные графические объекты используемые темой оформления CheckBox при различных DPI:

```
<?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>
```

Для получения подробной информации о разрешениях и DPI в мобильных приложениях см. раздел «Поддержка различных размеров экрана и значений DPI в мобильном приложении» на странице 137.

В темах оформления ActionScript можно использовать векторы, захваченные как растровые изображения. Однако при этом способе невозможно использовать переходы, в которых требуется перерисовка пикселов, например переходы альфа-каналов. Для получения дополнительной информации см. веб-страницу www.adobe.com/devnet/air/flex/articles/writing_multiscreen_air_apps.html.

Создание тем оформления для мобильного приложения

Настройка пользовательской мобильной темы оформления представляет собой создание класса этой темы оформления. В некоторых случаях также требуется изменить ресурсы, используемые в классе мобильной темы оформления.

При редактировании класса мобильной темы оформления можно изменить типы взаимодействия на основе состояний, внедрить поддержку для новых стилей, а также добавить или удалить нижестоящие компоненты этой темы оформления. Процесс редактирования обычно начинается с создания исходного кода для существующей темы оформления, который сохраняется как новый класс.

Также можно отредактировать ресурсы мобильной темы оформления, чтобы изменить ее визуальные свойства, например размер, цвет градиента или фон. Для этого можно изменить ресурсы FXG, используемые в теме оформления. Исходные файлы *.fxg, используемые в мобильной теме оформления, находятся в каталоге spark/skins/mobile/assets.

Файлы *.fxg определяют не все визуальные свойства мобильных тем оформления. Например, цвет фона темы оформления Button определяется в свойстве стиля `chromeColor` класса ButtonSkin, а не в ресурсе FXG. Чтобы изменить цвет фона в этом случае, отредактируйте класс темы оформления.

Создание класса мобильной темы оформления

Класс пользовательской мобильной темы оформления обычно создается на основе существующего класса мобильной темы оформления. Затем следует изменить этот класс и использовать его как пользовательскую тему оформления.

Для создания класса пользовательской темы оформления выполните перечисленные ниже действия.

1. Создайте каталог в проекте (например, customSkins). Имя каталога соответствует имени пакета пользовательских тем оформления. Хотя создание пакета не требуется, упорядочение пользовательских тем оформления в отдельных пакетах будет целесообразным.
2. Создайте класс пользовательской темы оформления в этом новом каталоге. Присвойте имя этому классу, например CustomButtonSkin.as.
3. Скопируйте содержимое класса темы оформления, который используется как база для нового класса. Например, если в качестве базового класса используется ButtonSkin, скопируйте содержимое файла spark.skins.mobile.ButtonSkin в новый пользовательский класс темы оформления.
4. Отредактируйте созданный класс. Например, внесите как минимум указанные ниже изменения в класс CustomButtonSkin:
   - измените местоположение пакета:

     ```
     package customSkins
     //was: package spark.skins.mobile
     ```

- в объявлении класса измените имя класса; также расширьте класс, на котором основывается новая тема оформления (не базовый класс темы оформления):

```
public class CustomButtonSkin extends ButtonSkin
// was: public class ButtonSkin extends ButtonSkinBase
```

- измените имя класса в конструкторе:

```
public function CustomButtonSkin()
//was: public function ButtonSkin()
```

5 измените пользовательский класс темы оформления (например, добавьте поддержку дополнительных состояний или новых нижестоящих компонентов). Также можно изменить ресурсы, поскольку некоторые графические объекты определяются непосредственно в классе темы оформления.

Чтобы упростить чтение класса тем оформления, из пользовательской темы оформления следует удалить не переопределяемые методы.

Следующий пользовательский класс темы оформления расширяет ButtonSkin и заменяет метод `drawBackground()` на методы пользовательской логики. При этом линейный градиент заменяется на радиальный градиент в заливке фона.

```
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 Примените пользовательскую тему оформления в приложении, используя один из способов, указанных в разделе «Применение пользовательской мобильной темы оформления» на странице 192. В примере ниже свойство `skinClass` используется в теге компонента для применения темы оформления `customSkins.CustomButtonSkin`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/CustomButtonSkinView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark" title="Home">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Button label="Click Me" skinClass="customSkins.CustomButtonSkin"/>

</s:View>
```

Методы жизненного цикла мобильных тем оформления

При создании классов пользовательских тем оформления ознакомьтесь со следующими методами UIComponent. Эти унаследованные защищенные методы определяют нижестоящие элементы и компоненты темы оформления и способствуют их взаимодействию с другими компонентами в списке отображения.

- `createChildren()` создает все нижестоящие графические или текстовые объекты, требуемые в теме оформления.
- `commitProperties()` копирует данные компонента в тему оформления при необходимости.
- `measure()` определяет размеры темы оформления с наиболее приближенными значениями и сохраняет эти данные в свойствах `measuredWidth` и `measuredHeight` темы оформления.
- `updateDisplayList()` определяет расположение и размер графических и текстовых объектов, а также создает необходимые рисунки ActionScript. Этот метод вызывает методы `drawBackground()` и `layoutContents()` для темы оформления.

Для получения подробной информации об использовании этих методов см. раздел Реализация компонента.

Общие методы для настройки мобильных тем оформления

Многие мобильные темы оформления использую следующие методы:

- `layoutContents()` — располагает такие нижестоящие элементы темы оформления, как отбрасываемые тени и метки. Классы мобильных тем оформления не поддерживают такие макеты Spark, как HorizontalLayout и VerticalLayout. Расположите нижестоящие элементы темы оформления вручную, используя такой метод, как `layoutContents()`.
- `drawBackground()` — отображает фон для темы оформления. Примеры типичного использования включают использование стилей `chromeColor`, `backgroundColor` или `contentBackgroundColor` на основе формы темы оформления. Также может использоваться для добавления оттенков, например с помощью метода `applyColorTransform()`.
- `commitCurrentState()` определяет поведение состояний для мобильных тем оформления. С помощью этого метода можно добавлять или удалять поддерживаемые состояния, а также изменять поведение существующих состояний. Этот метод вызывается при изменении состояния. Многие классы тем оформления переопределяют этот метод. Для получения подробной информации см. раздел «Состояния мобильных тем оформления» на странице 182.

Создание пользовательских ресурсов FXG

Многие визуальные ресурсы мобильных тем оформления определяются в файлах FXG. FXG представляет собой декларативный синтаксис для определения статических графических объектов. Для экспорта документа FXG можно использовать такие графические инструменты, как Adobe Fireworks, Adobe Illustrator или Adobe Catalyst. Затем документ FXG можно использовать в мобильных темах оформления. Также можно создавать документы FXG в текстовом редакторе, хотя запись комплексных графических объектов «с нуля» может представлять собой сложную задачу.

В мобильных темах оформления файлы FXG обычно определяют состояния темы оформления. Например, в классе CheckBoxSkin следующие файлы FXG определяют внешний вид рамки и символа флажка:

- CheckBox_down.fxg
- CheckBox_downSymbol.fxg
- CheckBox_downSymbolSelected.fxg
- CheckBox_up.fxg
- CheckBox_upSymbol.fxg
- CheckBox_upSymbolSelected.fxg.

В графическом редакторы эти файлы будут отображаться следующим образом:

Состояния флажка (down, downSymbol, downSymbolSelected, up, upSymbol и upSymbolSelected)

Файлы FXG для различных разрешений

Большинство мобильных тем оформления имеют три набора графических файлов FXG, по одному для каждого стандартного целевого разрешения. Например, в каталогах spark/skins/mobile160, spark/skins/mobile240 и spark/skins/mobile320 содержатся различные версии всех шести классов CheckBoxSkin.

При создании пользовательской темы оформления можно выполнить одно из следующих действий.

- Использование одной из тем оформления по умолчанию в качестве основы (как правило, 160 DPI). Добавьте логику, изменяющую масштаб пользовательской темы оформления в соответствии с устройством, на котором работает приложение. Для этого необходимо указать настройки свойства `applicationDPI` для объекта Application.
- Для оптимального отображения создайте все три версии пользовательской темы оформления (160, 240 и 320 DPI).

Некоторые мобильные темы оформления используют для своих графических ресурсов только один набор файлов FXG и не содержат графики, требующей определенного DPI. Эти ресурсы хранятся в каталоге spark/skins/mobile/assets. Например, темы оформления ViewMenuItem и темы оформления панели кнопок TabbedViewNavigator не содержат версий, требующих определенного DPI, поэтому все их ресурсы FXG хранятся в этом каталоге.

Настройка файла FXG

Вы можете открыть и настроить существующий файл FXG либо создать новый файл и экспортировать его из графического редактора, например Adobe Illustrator. Отредактированный файл FXG применяется к классу темы оформления.

Чтобы создать пользовательскую тему оформления с изменением файла FXG, выполните перечисленные ниже действия.

1. Создайте пользовательский класс темы оформления и поместите его в каталог customSkins, как описано в разделе «Создание класса мобильной темы оформления» на странице 184.
2. Создайте подкаталог в каталоге customSkins, например с именем assets. Это необязательный шаг, который, однако, обеспечивает упорядоченность файлов FXG и классов тем оформления.
3. Создайте файл в каталоге assets и скопируйте в него содержимое существующего файла FXG. Например, создайте файл с именем CustomCheckBox_upSymbol.fxg. Скопируйте содержимое spark/skins/mobile160/assets/CheckBox_upSymbol.fxg в новый файл CustomCheckBox_upSymbol.fxg.
4. Измените новый файл FXG. Например, замените логику, предусматривающую отображение значка «X» для флажка с заполнением градиентом:

```
<?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 В классе пользовательской темы оформления импортируйте новый класс FXG и примените его к свойству. Например, выполните указанные ниже действия в классе CustomCheckBox.

1 Импортируйте новый файл FXG:

```
//import spark.skins.mobile.assets.CheckBox_upSymbol;
import customSkins.assets.CustomCheckBox_upSymbol;
```

2 Добавьте новый ресурс в класс пользовательской темы оформления. Например, свяжите значение свойства `upSymbolIconClass` с новым ресурсом FXG:

```
upSymbolIconClass = CustomCheckBox_upSymbol;
```

Ниже представлен полный код класса пользовательской темы оформления:

```
// 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
        }
    }
}
```

Дополнительные сведения о работе с ресурсами FXG и их оптимизации для тем оформления см. в разделе Оптимизация FXG.

Просмотр файлов FXG в приложениях

Файлы FXG создаются с помощью XML, поэтому просмотр внешнего вида окончательной версии продукта может быть затруднен. Для импорта и визуализации файлов FXG можно использовать приложение Flex, при этом файлы добавляются в приложение в качестве компонентов и включаются в контейнер Spark.

Чтобы добавить файлы FXG в качестве компонентов, добавьте местоположение исходных файлов в исходный путь приложения. Например, для отображения мобильных ресурсов FXG в веб-приложении добавьте мобильную тему в исходный путь. После этого компилятор сможет находить файлы FXG.

Следующий пример *desktop* отображает различные ресурсы FXG компонента CheckBox при его использовании в мобильном приложении. При компиляции этого примера добавьте каталог frameworks\projects\mobiletheme\src\ к аргументу `source-path` компилятора.

```
<?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>
```

Использование текста в пользовательских мобильных темах оформления

Для визуализации текста в мобильных темах оформления используется класс StyleableStageText или StyleableTextField. Этот класс текста оптимизирован для использования в мобильных приложениях.

StyleableStageText предоставляет доступ к собственным функциям ввода текста для элементов управления TextInput и TextArea. Он расширяет класс UIComponent и реализует интерфейсы IEditableText и ISoftKeyboardHintClient.

StyleableTextField также используется в элементах управления TextInput и TextArea, если доступ к собственным функциям ввода не требуется. Кроме того, он используется в элементах управления без ввода текста, таких как ActionBar и Button. Он расширяет класс TextField и реализует интерфейсы ISimpleStyleClient и IEditableText.

Для получения подробной информации об использовании текстовых элементов управления в мобильных приложениях см. раздел «Использование текста в мобильном приложении» на странице 152.

TLF в мобильных темах оформления

Из соображений производительности не используйте классы, которые используют TLF в мобильных темах оформления. В некоторых случаях (например в компоненте Spark Label) можно применят классы, использующие FTE.

htmlText в мобильных темах оформления

Свойство `htmlText` невозможно использовать в мобильных приложениях.

Использование жестов с текстом

При использовании жестов касания и перетаскивания текст всегда выбирается, если его можно выбирать или изменять. Если текст расположен в элементе Scroller, то прокрутка Scroller возможна только в том случае, если жест выполнен за пределами текстового компонента. Эти жесты используются только для текста, который можно выбирать или изменять.

Разрешение выбора и изменения текста

Чтобы разрешить выбор и изменение текста, установите для свойств `editable` и `selectable` значение `true`:

```
textDisplay.editable = true;
textDisplay.selectable = true;
```

Двунаправленность

Двунаправленность текста не поддерживается в классе StyleableStageText или StyleableTextField.

Применение пользовательской мобильной темы оформления

Пользовательские темы оформления применяются к мобильным компонентам так же, как и к компонентам настольного приложения.

Применение темы оформления в ActionScript

```
// Call the setStyle() method:
myButton.setStyle("skinClass", "MyButtonSkin");
```

Применение темы оформления в MXML

```
<!-- Set the skinClass property: -->
<s:Button skinClass="MyButtonSkin"/>
```

Применение темы оформления в CSS

```
// Use type selectors for mobile skins, but only in the root document:
s|Button {
    skinClass: ClassReference("MyButtonSkin");
}
```

или

```
// Use class selectors for mobile skins in any document:
.myStyleClass {
    skinClass: ClassReference("MyButtonSkin");
}
```

Пример применения пользовательской мобильной темы оформления

В примере ниже представлены три метода применения пользовательских мобильных тем оформления к мобильным компонентам:

```
<?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>
```

Если для применения пользовательской темы оформления используется селектор *type* CSS, его необходимо установить в корневом файле мобильного приложения. Селекторы type невозможно установить в мобильном представлении, если оно совпадает с пользовательским компонентом. Однако селектор класса можно использовать для установки стилей в ActionScript, MXML или CSS в любом представлении или документе мобильного приложения.

Глава 7. Тестирование и отладка

Управление конфигурациями запуска

Конфигурации запуска Flash Builder используются при запуске и отладке мобильных приложений. В конфигурациях запуска указывается, следует ли запускать приложение на настольном компьютере или на устройстве, подключенном к компьютеру.

Для создания конфигурации запуска выполните следующие действия:

1 Чтобы открыть диалоговое окно выполнения конфигураций, выберите пункты меню «Выполнить» > «Выполнить конфигурации».

Чтобы открыть диалоговое окно отладки конфигураций, выберите пункты «Выполнить» > «Отладка конфигураций». «Тестирование и отладка мобильного приложения на устройстве» на странице 195

Меню выполнения или отладки конфигураций можно также выбрать в раскрывающемся списке кнопки «Выполнить» или «Отладка» в панели инструментов Flash Builder.

2 Разверните узел «Мобильное приложение». Нажмите кнопку «Создать конфигурацию запуска» в диалоговом окне панели инструментов.

3 Выберите целевую платформу в раскрывающемся списке.

4 Выберите способ запуска:

- **На настольном компьютере**

  Запускает приложение или выполняет отладку на настольном компьютере с помощью AIR Debug Launcher (ADL) в соответствии с указанной конфигурацией устройства. Этот метод запуска не является полнофункциональной эмуляцией выполнения приложения на устройстве, но позволяет просматривать макет приложения и взаимодействовать с приложением. См. раздел «Предварительный просмотр приложений с помощью ADL» на странице 195.

  Для редактирования конфигураций устройств нажмите кнопку «Настроить». См. раздел «Настройка данных устройства для просмотра на настольном компьютере» на странице 195.

- **На устройстве**

  Устанавливает и запускает приложение на устройстве.

  Если используется платформа Google Android, Flash Builder устанавливает и запускает приложение на устройстве. Flash Builder выполняет доступ к устройству, подключенному к компьютеру через порт USB. Подробную информацию см . в разделе «Тестирование и отладка мобильного приложения на устройстве» на странице 195.

  Чтобы подключить устройство Android к компьютеру, системе Windows требуется драйвер USB. Для получения подробной информации см. раздел «Установка драйверов устройств USB для устройств Android (Windows)» на странице 21.

5 Укажите, следует ли удалять данные приложения при каждом запуске.

Тестирование и отладка мобильного приложения на настольном компьютере

В случае отсутствия мобильного устройства Flash Builder позволяет выполнять первичное тестирование и отладку приложения на настольном компьютере с помощью AIR Debug Launcher (ADL).

Перед первым тестированием или отладкой мобильного приложения необходимо определить конфигурацию запуска. Укажите целевую платформу и в качестве метода запуска выберите «On Desktop». См. раздел «Управление конфигурациями запуска» на странице 194.

Настройка данных устройства для просмотра на настольном компьютере

Свойства, указанные в конфигурации устройства, определяют способ отображения приложения в ADL и в режиме Flash Builder Design.

В разделе «Настройка конфигурации устройства» на странице 16 перечислены поддерживаемые конфигурации. Конфигурации устройств не влияют на отображение приложения в устройстве.

Плотность экрана

Для просмотра приложения используется настольный компьютер разработки и для просмотра макета приложения - режим «Дизайн» Flash Builder. В Flash Builder используется плотность экрана 240 DPI. Отображение приложения во время предварительного просмотра может отличаться от отображения на устройстве, которое поддерживает другую плотность экрана.

Предварительный просмотр приложений с помощью ADL

Для предварительного просмотра приложений на рабочем столе Flash Builder загружает приложение с помощью ADL. ADL содержит меню устройства с соответствующими ярлыками, которые имитируют кнопки устройства.

Например, для имитации функции кнопки возврата на устройстве выберите пункты Device > Back. Для имитации поворота устройства выберите пункты Device > Rotate Left или Device > Rotate Right. Параметры поворота неактивны, если не была выбрана автоматическая ориентация.

Перетащите на экран список и используйте его для имитации прокрутки списка на устройстве.


В этом видеоруководстве сертифицированный специалист Adobe по продуктам Flex Brent Arnold рассказывает об использовании ADL для просмотра мобильного приложения на настольном компьютере.

Тестирование и отладка мобильного приложения на устройстве

Flash Builder используется для тестирования и отладки мобильных приложений на настольном компьютере, где выполнялась разработка, или другом устройстве.

При тестировании и отладке приложений используются определенные пользователем конфигурации запуска. Flash Builder использует аналогичную конфигурацию запуска для выполнения и отладки приложения. При отладке приложения на устройстве Flash Builder устанавливает отладочную версию приложения на этом устройстве.

Примечание. *Если на устройство экспортируется сборка выпуска, необходимо установить неотладочную версию приложения. Неотладочная версия не может использоваться для отладки.*

Для получения подробной информации см. раздел Управление конфигурациями запуска.

Отладка приложения на устройстве Google Android

Отладка приложений выполняется на устройстве Android с установленным приложением Android 2.2 или более новой версии.

Для отладки можно использовать любой из нижеперечисленных способов.

Отладка через USB. Для отладки приложения через USB подключите устройство к компьютеру хоста через порт USB. При этом способе отладки Flash Builder всегда упаковывает, устанавливает и запускает приложение на устройстве перед началом отладки. Убедитесь, что в течение всего сеанса отладки устройство подключено к компьютеру хоста через порт USB.

Отладка по сети. Для отладки приложения по сети устройство и компьютер хоста должны быть подключены к одной и той же сети. Для подключения компьютера хоста и устройства к сети можно использовать Wi-Fi, Ethernet или Bluetooth.

При отладке по сети Flash Builder выполняет отладку уже установленного на устройстве приложения, поэтому повторная установка приложения не требуется. Подключение устройства к компьютеру хоста через USB требуется только на этапе упаковки и установки приложения на устройстве. Во время отладки устройство можно отсоединить от порта USB. Убедитесь, что устройство и компьютер хоста подключены к сети во время всего сеанса отладки.

Подготовка к отладке приложения

Перед выполнением отладки через USB или по сети выполните следующие шаги:

1 (Windows) Убедитесь, что в системе установлен необходимый драйвер USB.

Установите драйвер USB Android, если используется ОС Windows. Для получения подробной информации см. документацию, поставляемую с Android SDK. Для получения подробной информации см. раздел «Установка драйверов устройств USB для устройств Android (Windows)» на странице 21.

2 Убедитесь, что на устройстве включена функция отладки через USB.

В меню Settings устройства выберите пункты Applications > Development и включите отладку через USB.

Проверка подключенных устройств

При выполнении или отладке мобильных приложений на устройстве Flash Builder проверяет подключенные устройства. Если в сети найдено одно подключенное устройство, Flash разворачивает и запускает приложение. Для указанных ниже случаев Flash Builder выводит диалоговое окно выбора устройства:

- подключенные устройства не обнаружены;
- найдено одно подключенное устройство, которое работает в автономном режиме, или используется неподдерживаемая версия ОС;
- найдено несколько подключенных устройств.

Если обнаружено несколько устройств, в диалоговом окне выбора устройства отображаются устройства и их состояния (в сети или автономно). Выберите устройство для запуска.

В диалоговом окне выбора устройства перечислены версии операционных систем и AIR. Если на устройстве не установлено приложение Adobe AIR, Flash Builder автоматически установит эту среду.

Настройка параметров сетевой отладки

Выполнение следующих шагов требуется только при отладке по сети.

Подготовка к отладке по сети

Перед отладкой приложения по сети выполните следующие шаги:

1 Windows: откройте порт 7935 (порт отладчика Flash Player) и порт 7 (порт эхо и проверки связи).

Подробные инструкции см. в этой статье на портале Microsoft TechNet.

Windows Vista: снимите флажок «Беспроводное сетевое соединение» в меню «Брандмауэр Windows» > «Изменить настройки» > «Дополнительно».

2 Настройте беспроводное подключение на устройстве, выбрав пункты меню Settings > Wireless and Network.

Выбор основного сетевого интерфейса

Компьютер хоста можно одновременно подключить к нескольким сетевым интерфейсам. При этом можно выбрать основной сетевой интерфейс для отладки. Для выбора этого интерфейса добавьте адрес хоста в файл пакета Android APK.

1 В Flash Builder откройте диалоговое окно «Параметры».

2 Выберите пункты «Flash Builder» > «Целевые платформы».

В диалоговом окне представлен список всех доступных сетевых интерфейсов на компьютере хоста.

3 Выберите сетевой интерфейс, который требуется добавить в пакет Android APK.

Убедитесь, что выбранный сетевой интерфейс доступен с устройства. Если устройству не удается выполнить доступ к выбранному сетевому интерфейсу по время подключения, Flash Builder выводит диалоговое окно с запросом IP-адреса компьютера хоста.

Отладка приложения

1 Подключите устройство через порт USB или сеть.

2 Чтобы настроить конфигурацию запуска для отладки, выберите пункты «Запуск» > «Отладка конфигураций».

- В качестве способа запуска выберите пункт «На устройстве».
- Выберите пункт «Отладка с USB» или «Отладка через сеть».

  При первой отладке приложения по сети можно установить приложение на устройстве через USB. Для этого выберите пункт «Установить приложение на устройстве через USB» и подключите устройство к компьютеру хоста через порт USB.

  Если после установки приложения не требуется подключение через USB для последующих сеансов отладки, отмените выбор параметра «Установить приложение на устройстве через USB».

- Укажите, следует ли удалять данные приложения при каждом запуске (необязательно).

Выберите этот параметр, если требуется сохранить состояние приложения для каждого сеанса отладки. Этот параметр применяется, только если для функции sessionCachingEnabled в приложении установлено значение True.

3 Чтобы начать сеанс отладки, выберите пункт «Отладка».

Отладчик запускается и ожидает запуска приложения. Сеанс отладки начнется после того, как отладчик установит соединение с устройством.

В некоторых случаях при отладке приложения на устройстве по сети может отобразиться запрос на указание IP-адреса. Вывод этого диалогового окна обозначает, что отладчику не удалось выполнить подключение. Убедитесь, что устройство использует сетевое подключение и компьютер, на котором запущена программа Flash Builder, доступен в этой сети.

Примечание. *В корпоративной, гостиничной или иной гостевой сети иногда не удается соединить устройство и компьютер, даже если они находятся в одной сети.*

Если для отладки используется сеть и приложение уже установлено на устройстве, введите IP-адрес компьютера хоста для запуска сеанса отладки.

Сертифицированный специалист Adobe по продуктам Flex Brent Arnold предлагает видеоруководство по отладке приложения с использованием USB на устройстве Android.

Дополнительные разделы справки

Отладка и упаковка приложений для устройств (видеоролик)

Отладка приложений на устройстве Apple iOS

Для отладки приложения на устройстве Apple iOS вручную разверните и установите отладочный пакет iOS (файл IPA) на устройстве iOS. Платформа Apple iOS не поддерживает автоматическое развертывание.

Важная информация. *Прежде чем приступить к отладке приложения на устройстве iOS, убедитесь, что были выполнены шаги, описанные в разделе «Подготовка к созданию, отладке или развертыванию приложения для iOS» на странице 24.*

1 Подключите устройство Apple iOS к компьютеру разработки.

2 Загрузите iTunes на устройстве iOS.

Примечание. *iTunes требуется для установки приложения и определения идентификатора используемого устройства iOS.*

3 В меню Flash Builder выберите пункты «Запуск» > «Отладка конфигураций».

4 В диалоговом окне «Отладка конфигураций» выполните следующие шаги:

a Выберите приложение для отладки.

b В качестве целевой платформы укажите Apple iOS.

c В качестве способа запуска выберите пункт «На устройстве».

d Выберите один из следующих способов упаковки:

Стандартный: этот способ используется для упаковки коммерческой версии приложения, которое может запускаться на устройствах Apple iOS. При этом приложение обеспечивает такую же производительность, что и пакет коммерческой версии, и может быть отправлено в Apple App Store.

Учтите, что процесс создания отладочного файла iOS (IPA) может занять несколько минут.

Быстрый: этот способ используется для быстрого создания файла IPA и последующего выполнения и отладки файла на устройстве. Этот способ рекомендуется для тестирования приложения. Производительность приложения при этом способе ниже коммерческой версии, поэтому не рекомендуется отправлять это приложение в Apple App Store.

e Нажмите кнопку «Настроить» и выберите соответствующий сертификат подписания кода, файл обеспечения и содержимое пакета.

f Откройте диалоговое окно настройки параметров сетевой отладки и выберите сетевой интерфейс для добавления в отладочный пакет iOS.

Примечание. *Компьютер хоста можно одновременно подключить к нескольким сетевым интерфейсам. При этом можно выбрать основной сетевой интерфейс для отладки.*

g Нажмите кнопку «Отладка». Flash Builder выводит диалоговое окно с запросом пароля. Введите пароль для сертификата P12.

Flash Builder создает отладочный файл IPA в папке bin-debug.

5 На устройстве iOS выполните следующие шаги:

1 (Дополнительно) В iTunes выберите пункты File > Add To Library и перейдите к файлу профиля мобильного обеспечения (с расширением .mobileprovision), полученному от компании Apple.

2 В iTunes выберите пункты File > Add To Library и перейдите к отладочному файлу IPA, созданному в шаге 4.

3 Синхронизируйте используемое устройство iOS с iTunes, выбрав пункты File > Sync.

4 Flash Builder подключается к адресу хоста, указанному в отладочном файле IPA. Если приложению не удается подключиться к адресу хоста, Flash Builder выводит диалоговое окно с запросом IP-адреса компьютера хоста.

Примечание. *Если код или ресурсы не изменялись со времени создания последнего отладочного пакета IPA, Flash Builder пропускает этап упаковки и отлаживает приложение. Таким образом, можно запустить установленное приложение на устройстве и нажать кнопку отладки для подключения к отладчику Flash Builder. Этот способ позволяет повторно отлаживать приложение без необходимости его упаковки.*

Глава 8. Установка на устройствах

Установка приложения на устройстве Google Android

Во время разработки, тестирования и развертывания проекта созданное приложение можно установить непосредственно на устройстве.

С помощью Flash Builder установить приложение непосредственно на устройстве Android. Если устройство, на которое устанавливается пакет, не содержит среду Adobe AIR, Flash Builder установит среду AIR автоматически.

1 Подключите устройство Google Android к компьютеру разработки.

   Flash Builder выполняет доступ к устройству, подключенному к компьютеру через порт USB. Убедитесь, что настроены правильные драйверы USB устройства. См. раздел «Подключение устройств Google Android» на странице 20

2 В меню Flash Builder выберите пункты «Выполнить» > «Выполнить конфигурации». В диалоговом окне «Выполнить конфигурации» выберите мобильное приложение, которое требуется развернуть.

3 В качестве способа конфигурации запуска выберите пункт «На устройстве».

4 (Дополнительно) Укажите, следует ли удалять данные приложения при каждом запуске.

5 Нажмите кнопку «Применить».

Flash Builder устанавливает и запускает приложение на устройстве Android. Если устройство, на которое устанавливается пакет, не содержит среду Adobe AIR, Flash Builder установит среду AIR автоматически.

В этом видеоруководстве сертифицированный специалист Adobe по продуктам Flex Brent Arnold рассказывает об установке и запуске приложения на устройстве Android.

Установка приложения на устройстве Apple iOS

На устройствах iOS необходимо вручную установить приложение (файл IPA), поскольку платформа Apple iOS не поддерживает автоматическое развертывание.

Важная информация. *Перед установкой приложения на устройстве iOS необходимо получить сертификат разработки Apple iOS (в формате P12) и профиль обеспечения версии разработки. Для этого выполните шаги, описанные в разделе «Подготовка к созданию, отладке или развертыванию приложения для iOS» на странице 24.*

1 Подключите устройство Apple iOS к компьютеру разработки.

2 Загрузите iTunes на компьютер разработки.

   Примечание. *iTunes требуется для установки приложения и получения уникального идентификатора устройства (UDID) iOS.*

3 В меню Flash Builder выберите пункты «Выполнить» > «Выполнить конфигурации».

4 В диалоговом окне «Выполнить конфигурации» выполните следующие шаги:

   a Выберите приложение для установки.

b В качестве целевой платформы укажите Apple iOS.

c В качестве способа запуска выберите пункт «На устройстве».

d Выберите один из следующих способов упаковки:

Стандартный: этот способ используется для упаковки коммерческой версии приложения, которое может запускаться на устройствах Apple iOS.

При стандартном способе упаковки байт-код файла SWF приложения преобразуется в инструкции ARM перед упаковкой. Поскольку перед началом упаковки требуется дополнительное преобразование, процесс создания файла приложения (IPA) может занять несколько минут. Стандартный способ упаковки является более медленным, чем быстрый способ. Однако производительность приложения при этом способе соответствует коммерческой версии и приложение готово к отправке в Apple App Store.

Быстрый: это способ быстрого создания файла IPA.

При быстром способе упаковки не требуется преобразование байт-кода и файл приложения SWF объединяется вместе с ресурсами в пакет в предварительно откомпилированной среде выполнения AIR. При быстром способе для упаковки требуется меньше времени, чем при стандартном способе. Производительность приложения при быстром способе ниже коммерческой версии, поэтому не рекомендуется отправлять это приложение в Apple App Store.

Примечание. *Между быстрым и стандартным способами отсутствуют различия по времени выполнения или функциональности.*

e Нажмите кнопку «Настроить» и выберите соответствующий сертификат подписания кода, файл обеспечения и содержимое пакета.

f Нажмите кнопку «Выполнить». Flash Builder выводит диалоговое окно с запросом пароля. Введите пароль для сертификата P12.

Flash Builder создает файл IPA в папке bin-debug.

5 Выполните следующие шаги на компьютере разработки:

1 В iTunes выберите пункты File > Add To Library и перейдите к файлу профиля мобильного обеспечения (с расширением .mobileprovision), полученному от компании Apple.

Файл профиля мобильного обеспечения можно также перетащить в iTunes.

2 В iTunes выберите пункты File > Add To Library и перейдите к файлу IPA, созданному в шаге 4.

Файл IPA можно также перетащить в iTunes.

3 Синхронизируйте используемое устройство iOS с iTunes, выбрав пункты File > Sync.

Приложение развернуто на устройстве и готово к запуску.

Глава 9. Упаковка и экспорт

Упаковка и экспорт мобильного приложения в онлайн-магазин

Для упаковки и экспорта коммерческих версий мобильных приложений используется функция экспорта сборки выпуска Flash Builder. Как правило, сборка выпуска представляет собой окончательную версию приложения для загрузки в онлайн-магазин, например в Android Market, Amazon Appstore или Apple App store.

Экспортируемое приложение можно установить на устройстве. Если устройство подключено к компьютеру во время экспорта, Flash Builder установит приложение на устройстве. Пакет приложения, созданный для определенной платформы, также можно экспортировать для последующей установки на устройстве. Созданный пакет устанавливается и развертывается таким же образом, как и приложение собственного формата.

Подробную информацию об экспорте приложения Android в магазины Android Market или Amazon App Store см. в разделе «Экспорт пакетов Android APK для выпуска» на странице 202.

Для получения подробной информации об экспорте приложения iOS в Apple App store см. раздел «Экспорт пакетов Apple iOS для выпуска» на странице 203.

Экспорт приложения со связанной средой выполнения AIR

Если функция «Экспорт сборки выпуска» используется для экспорта мобильного приложения, среду AIR можно встроить в пакет приложения.

После этого пользователи могут запускать приложение даже на устройстве, на котором не установлена среда AIR. В зависимости от платформы, на которую экспортируется пакет, можно выбрать связанную или общую среду.

Экспорт пакетов Android APK для выпуска

Перед экспортом мобильного приложения можно настроить разрешения Android. Для настройки разрешений вручную отредактируйте файл дескриптора приложения. Эти установки находятся в блоке `<android>` в файле bin-debug/*app_name*-app.xml. Для получения подробной информации см. раздел Установка свойств приложения AIR.

Если приложение экспортируется для последующей установки на устройстве, установите пакет приложения с помощью инструментов, предоставленных поставщиком ОС устройства.

1. В меню Flash Builder выберите пункты «Проект» > «Экспорт сборки выпуска».
2. Выберите проект и приложение для экспорта.
3. Выберите целевые платформы и местоположение для экспорта проекта.
4. Экспортируйте и подпишите пакет приложения с учетом платформы.

   Приложение можно упаковать с цифровой подписью для каждой целевой платформы или как приложение AIR с цифровой подписью для настольного компьютера.

Также приложение можно экспортировать как промежуточный файл AIRI, который будет подписан позже. При выборе этого способа используйте инструмент командной строки AIR adt для упаковки AIRI как файл APK. Затем установите файл APK на устройстве с помощью инструментов, зависящих от платформы, например с Android SDK используйте adb. Для получения информации об использовании инструментов командной строки для упаковки приложений см. раздел «Создание, тестирование и развертывание с помощью командной строки» на странице 204.

5 На странице «Параметры упаковки» укажите цифровой сертификат, содержимое пакета и любые собственные расширения.

Развертывание. Если требуется установить приложение на устройство, на странице развертывания выберите пункт «Установить и запустить приложение на любых подключенных устройствах». Убедитесь, что одно или несколько устройств подключены к компьютеру через порты USB.

- **Экспорт приложения со связанной средой выполнения**

  Выберите этот параметр, если необходимо встроить среду AIR в файл APK во время экспорта пакета приложения. После этого пользователи могут запускать приложение даже на устройстве, на котором не установлена среда AIR.

- **Экспорт приложения, использующего общую среду выполнения**

  Выберите этот параметр, если не требуется встраивать среду AIR в файл APK во время экспорта пакета приложения. Если среда AIR не установлена на устройстве, можно указать URL-адрес загрузки Adobe AIR для пакета приложения.

  URL-адрес по умолчанию указывает местоположение в Android Market. URL-адрес по умолчанию можно изменить, выбрать другое местоположение в Amazon Appstore или указать пользовательский URL-адрес.

Цифровая подпись. Откройте вкладку «Цифровая подпись» и создайте или выберите цифровой сертификат, удостоверяющий подлинность издателя приложения. Кроме того, можно указать пароль для выбранного сертификата.

Создаваемый сертификат должен быть *самозаверяющим*. Сертификат с коммерческой подписью можно получить в центре сертификации. См. раздел Цифровая подпись для приложений AIR.

Содержимое пакета. На вкладке «Содержимое пакета» выберите файлы для добавления в пакет (необязательно).

Собственные расширения. (Дополнительно) Выберите собственные расширения, которые необходимо включить в пакет приложения.

Для получения подробной информации о собственных расширениях см. раздел «Использование собственных расширений» на странице 14.

6 Нажмите кнопку «Готово».

Flash Builder создает `ApplicationName.apk` в каталоге, указанном на первой панели, которая по умолчанию находится на верхнем уровне проекта. Если устройство подключено к компьютеру во время экспорта, Flash Builder устанавливает приложение на устройство.

Экспорт пакетов Apple iOS для выпуска

После создания и экспорта пакет iOS может предназначаться для специализированного распространения или отправки в Apple App Store.

Важная информация. *Прежде чем приступать к экспорту пакета iOS, необходимо получить необходимые сертификаты и профиль обеспечения распространения от компании Apple. Для этого выполните шаги, описанные в разделе «Подготовка к созданию, отладке или развертыванию приложения для iOS» на странице 24.*

1 В меню Flash Builder выберите пункты «Проект» > «Экспорт сборки выпуска».

2 Чтобы экспортировать и подписать пакет IPA, в качестве целевой платформы выберите Apple iOS.

Нажмите кнопку «Далее».

3 Выберите сертификат в формате P12 и полученный от компании Apple профиль обеспечения распространения.

4 На странице «Параметры упаковки» укажите сертификат обеспечения, цифровой сертификат, содержимое пакета и любые собственные расширения.

Развертывание. Во время экспорта пакета iOS среда выполнения AIR по умолчанию встраивается в файл IPA.

Цифровая подпись. Выберите сертификат в формате P12 и полученный от компании Apple профиль обеспечения распространения.

Выберите один из следующих типов пакетов:

- **Ad Hoc Distribution For Limited Distribution** для ограниченного распространения приложения
- **Final Release Package For Apple App Store** для отправки приложения в Apple App Store

Содержимое пакета. На вкладке «Содержимое пакета» выберите файлы для добавления в пакет (необязательно).

Собственные расширения. (Дополнительно) Выберите собственные расширения, которые необходимо включить в пакет приложения.

Если в собственном расширении используются функции iOS5 SDK, выберите местоположение iOS SDK. Подробную информацию см. в разделе «Поддержка собственных расширений iOS5» на странице 16.

5 Нажмите кнопку «Готово».

Flash Builder проверяет конфигурацию настроек пакета и компилирует приложение. После завершения упаковки файл IPA можно установить на подключенное устройство Apple iOS или отправить в Apple App Store.

Для получения информации об упаковке файла IPA с помощью AIR Developer Tool (ADT) см. раздел Пакеты iOS в документе *Создание приложений AIR*.

Создание, тестирование и развертывание с помощью командной строки

Для создания мобильных приложений вне программы Flash Builder используйте инструменты командной строки mxmlc, adl и adt.

Ниже представлен общий процесс создания и установки мобильного приложения на устройстве из командной строки. Каждый из этих шагов будет рассмотрен более подробно в дальнейшем:

1 Скомпилируйте приложение с помощью инструмента mxmlc.

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

Для выполнения этого действия необходимо передать параметр `configname` со значением airmobile.

2 Протестируйте приложение в AIR Debug Launcher (ADL) с помощью инструмента adl:

```
adl MyMobileApp-app.xml -profile mobileDevice
```

Для выполнения этого действия необходимо создать файл дескриптора приложения и передать его в качестве аргумента в инструмент adl. Также требуется указать профиль `mobileDevice`.

3 Упакуйте приложение с помощью инструмента adt.

```
adt -package -target apk SIGN_OPTIONS MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

Для выполнения этого действия необходимо предварительно создать сертификат.

4 Установите приложение на мобильное устройство. Для установки приложения на мобильном устройстве используется инструмент adb.

```
adb install -r MyMobileApp.apk
```

Для выполнения этого действия необходимо предварительно подключить мобильное устройство к компьютеру через порт USB.

5 Разверните мобильное приложение в онлайн-магазинах.

Компиляция мобильного приложения с помощью mxmlc

Для компиляции мобильных приложений можно использовать компилятор командной строки mxmlc. Чтобы использовать mxmlc, присвойте параметру `configname` значение `airmobile`, например:

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

Передача `+configname=airmobile` означает, что компилятор должен использовать файл airmobile-config.xml. Этот файл находится в каталоге sdk/frameworks Этот файл выполняет следующие задачи:

- Применяет тему mobile.swc.
- Вносит следующие изменения в путь к библиотеке:
  - Удаляет libs/air из пути к библиотеке. Мобильные приложения не поддерживают классы Window и WindowedApplication.
  - Удаляет libs/mx из пути к библиотеке. Мобильные приложения не поддерживают компоненты MX (кроме диаграмм).
  - Добавляет libs/mobile в путь к библиотеке.
- Удаляет пространства имен ns.adobe.com/flex/mx и www.adobe.com/2006/mxml. Мобильные приложения не поддерживают компоненты MX (кроме диаграмм).
- Выключает специальные возможности.
- Удаляет записи RSL, так как мобильные приложения не поддерживают RSL.

Компилятор mxmlc создает SWF-файл.

Тестирование мобильного приложения с помощью adl

Тестирование мобильных приложений выполняется с помощью AIR Debug Launcher (ADL). При выполнении и тестировании приложения с помощью ADL устраняется необходимость предварительной упаковки и установки приложения на устройстве.

Отладка с инструментом adl

ADL печатает трассировочные инструкции и ошибки рабочей среды в виде стандартного вывода, но не поддерживает контрольные точки и иные функции отладки. Для устранения сложных ошибок отладки можно использовать интегрированную среду разработки, например Flash Builder.

Запуск инструмента adl

Чтобы запустить инструмент adl из командной строки, передайте файл дескриптора мобильного приложения и укажите для параметра `profile` значение `mobileDevice`, как показано в примере ниже:

```
adl MyMobileApp-app.xml -profile mobileDevice
```

Профиль `mobileDevice` определяет набор возможностей для приложений, установленных на мобильных устройствах. Для получения подробной информации о профиле `mobileDevice` см. раздел Возможности различных профилей.

Создание дескриптора приложения

Если для компиляции приложения не использовалась программа Flash Builder, создайте файл дескриптора приложения вручную. В качестве основы используйте файл /sdk/samples/descriptor-sample.xml. Как правило, необходимо внести как минимум перечисленные ниже изменения:

- Свяжите элемент `<initialWindow><content>` с именем файла SWF мобильного приложения:

```
<initialWindow>
    <content>MyMobileApp.swf</content>
    ...
</initialWindow>
```

- Измените заголовок приложения, который будет отображаться под значком приложения на мобильном устройстве. Чтобы изменить заголовок, отредактируйте элемент `<name><text>`:

```
<name>
    <text xml:lang="en">MyMobileApp by Nick Danger</text>
</name>
```

- Для установки специальных разрешений Android для приложения добавьте блок `<android>`. В зависимости от используемых на устройстве служб можно использовать следующее разрешение:

```
<application>
    ...
    <android>
        <manifestAdditions>
            <![CDATA[<manifest>
                <uses-permission android:name="android.permission.INTERNET"/>
            </manifest>]]>
        </manifestAdditions>
    </android>
</application>
```

В файле дескриптора можно также указать высоту и ширину приложения, местоположение файлов значков, информацию о версиях и другие сведения о местоположении установки.

Для получения подробной информации о создании и редактировании файлов дескриптора см. раздел Файлы дескриптора приложения AIR.

Упаковка мобильного приложения с помощью adt

Инструмент AIR Developer Tool (ADT) используется для упаковки мобильных приложений из командной строки. Инструмент adt может создать файл APK для развертывания на мобильном устройстве.

Создание сертификата

Перед созданием файла APK необходимо создать сертификат. В разработке можно использовать самозаверяющий сертификат. В примере ниже представлен процесс создания самозаверяющего сертификата с помощью инструмента adt:

```
adt -certificate -cn SelfSign -ou QE -o "Example" -c US 2048-RSA newcert.p12 password
```

Инструмент adt создает файл newcert.p12 в текущей директории. При упаковке приложения этот сертификат передается в adt. Не используйте самозаверяющие сертификаты для производственных приложений. Такие сертификаты обеспечивают лишь ограниченную защиту для пользователей. Для получения информации о подписании файлов установки AIR с помощью сертификата, выпущенного признанной сертифицирующей организацией, см. раздел Подписание приложений AIR.

Создание файла пакета

Чтобы создать файл APK для Android, передайте в инструмент adt подробные сведения о приложении, в том числе сертификат, как в примере ниже:

```
adt -package -target apk -storetype pkcs12 -keystore newcert.p12 -keypass password
MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

Инструмент adt выводит данные в файле *appname*.apk.

Пакет для iOS

Чтобы упаковать мобильное приложение для iOS, необходимо получить сертификат разработчика от компании Apple, а также файл обеспечения. Для этого необходимо участие в программе разработчиков компании Apple. Дополнительные сведения см. в разделе «Подготовка к созданию, отладке или развертыванию приложения для iOS» на странице 24.

Специалист по продуктам Flex Piotr Walczyszyn объясняет способы упаковки приложений с помощью ADT и Ant для устройств iOS.

Автор блога Valentin Simonov предоставляет дополнительную информацию о способах публикации приложения на iOS.

Установка мобильного приложения на устройстве с помощью adb

Для установки приложения (файл APK) на мобильном устройстве с установленным приложением Android используется Android Debug Bridge (adb). Инструмент adb является компонентом Android SDK.

Подключение устройства к компьютеру

Перед запуском adb для установки файла APK на мобильном устройстве необходимо подключить устройство к компьютеру. В системах Windows и Linux для этого требуются драйверы USB.

Для получения информации об установке драйверов USB на устройстве см. веб-страницу Using Hardware Devices.

Установка приложения на подключенном устройстве

После подключения устройства к компьютеру приложение можно установить на устройстве. Если для установки приложения используется инструмент adb, выберите параметр `install` и передайте имя файла APK, как в примере ниже:

```
adb install -r MyMobileApp.apk
```

Если требуется перезаписать установленное приложение, используйте параметр `-r`. В противном случае потребуется удалять приложение при каждой установке новой версии на мобильном устройстве.

Дополнительные разделы справки

Android Debug Bridge

Развертывание приложения в онлайн-магазинах

Созданное приложение можно развернуть в таких онлайн-магазинах приложений, как Android Market, Amazon Appstore или Apple App store.

Lee Brimlow объясняет способ развертывания нового AIR для приложения Android на портале Android Market.

Christian Cantrell объясняет способ развертывания приложения на портале Amazon Appstore for Android.