Руководство для разработчиков насыщенных интернет-приложений

Flex™ 3

Сборник рецептов

Символ®
O'REILLY®
Adobe
Adobe Developer Library

Джошуа Ноубл,
Тодд Андерсон

Flex 3 Cookbook

Joshua Noubl and Todd Anderson

O'REILLY®

Flex 3
Сборник рецептов

Джошуа Ноубл и Тодд Андерсон

Санкт-Петербург — Москва
2009

Джошуа Ноубл и Тодд Андерсон

Flex 3. Сборник рецептов

Перевод Е. Матвеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*А. Пасечник*
Редактор	*Ю. Сергиенко*
Корректор	*В. Листова*
Вёрстка	*И. Смарышева*

Ноубл Дж., Андерсон Т.

Flex 3. Сборник рецептов. – Пер. с англ. – СПб: Символ-Плюс, 2009. – 736 с., ил.

ISBN: 978-5-93286-146-2

Широкие возможности новых технологий лучше всего раскрываются на практических примерах. Именно такой подход используется в книге для представления технологии Adobe Flex 3.

Рассчитанная на широкий круг читателей и весьма практичная книга «Flex 3. Сборник рецептов» содержит более 300 решений, используемых при построении интерактивных RIA-приложений и сайтов Web 2.0. Авторы рассматривают широкий круг вопросов: от основ Flex до использования визуальных компонентов, от работы с базами данных до рекомендаций по разработке приложений, от модульного тестирования до Adobe AIR.

Каждый рецепт содержит решение стандартной проблемы, объясняет, почему и как это решение работает, а также содержит примеры готового кода, которые читатель сможет сразу использовать в своих программах, что позволит быстро добиться практических результатов как опытным разработчикам Flex, так и новичкам, знакомящимся с этой технологией. Книга идеально подходит для тех, кто желает повысить эффективность разработки своих веб-приложений.

ISBN: 978-5-93286-146-2
ISBN: 978-0-596-52985-7 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 25.02.2009. Формат 70×100[1]/16. Печать офсетная.
Объем 46 печ. л. Тираж 1000 экз. Заказ №

Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Flex 3 – мощная инфраструктура, предоставляющая в распоряжение разработчика компоненты коммерческого уровня для платформы Flash Player в формате языка разметки, понятном для любого пользователя с опытом работы на HTML или XML. Компоненты Flex Framework используются для визуального оформления, создания визуальных эффектов, построения таблиц данных, обмена данными с сервером, построения диаграмм и для других целей.

Бесспорно, инфраструктура Flex Framework чрезвычайно велика, и любые попытки сколько-нибудь глубокого ее описания окажутся несовершенными в том или ином отношении. Хорошо понимая это, мы решили рассмотреть темы, которые чаще всего досаждают программистам, работающим с Flex 3, – так, чтобы и дать некоторое представление о структуре Framework и одновременно помочь с решением типичных проблем. В официальной документации Flex достаточно подробно описано, как работают конкретные методы и классы, поэтому мы сосредоточились на решении типичных задач, на совместном использовании разных компонентов, на взаимодействии Flex с другими технологиями для построения RIA-приложений и т. д. Например, Adobe AIR позволяет использовать инструментарий Flex и Flash Player для создания самостоятельных настольных приложений. Распространение бесплатных и коммерческих средств программирования (для Java, .NET, PHP и т. д.) расширяет возможности Flex и позволяет применять эту технологию для решения новых задач.

Дополнительные главы в Сети

Тема Flex Framework настолько обширна, а нам хотелось представить столько рецептов и изложить столько информации, что все это просто не могло поместиться в бумажной книге. В Интернете доступны четыре дополнительные главы общим объемом 76 страниц; в них рассматривается работа с данными XML, компоненты для построения диаграмм, работа с SharedObjects и стратегии разработки для создания приложений Flex. Дополнительный материал размещен по адресу *www.oreilly.com/catalog/9780596529857*.

Для кого написана эта книга

Книга написана для программистов, которые хотят лучше разобраться в принципах работы Flex Framework или же нуждаются в практическом

руководстве для решения конкретных задач. Предполагается, что читатель уже обладает некоторым опытом работы с Flex и ActionScript 3. Примеры кода и объяснения ориентированы на программиста среднего уровня, понимающего связь между MXML и ActionScript, знакомого хотя бы с некоторыми компонентами Flex Framework и общей стратегией Flex-программирования.

Мы сознательно включили в каждый рецепт работоспособные компоненты и функциональные, протестированные реализации этих компонентов. Это делалось не для того, чтобы увеличить объем книги; мы постарались сделать книгу доступной как для программистов среднего и высокого уровня, которым достаточно увидеть небольшой фрагмент кода для понимания сути того или иного приема, так и для читателей, изучающих возможности Flex Framework и основные приемы практического использования этой среды.

Для кого эта книга не предназначена

Если вы изучаете Flex Framework «с нуля», обратитесь к книге «Programming Flex 3» Джои Лотта (Joey Lott) и Чафика Казуна (Chafic Kazoun) (O'Reilly, 2008), чтобы получить представление об основополагающих концепциях Flex-программирования. Понимание основ Flex и ActionScript повысит эффективность усвоения материала этой книги. Если вам нужно освежить в памяти программирование на ActionScript, в книге «ActionScript 3.0 Cookbook»[1] описаны многие полезные приемы из области базового программирования Flash ActionScript. Хотя в книге, которую вы сейчас держите, рассматриваются некоторые области пересечения Flex Framework с базовыми классами Flash ActionScript, основное внимание в ней уделяется разработке Flex-приложений.

Структура материала

Книга, как следует из ее названия, состоит из «рецептов» с описаниями приемов, которые помогут в полной мере использовать возможности Flex в своих приложениях. Чтобы читатель мог быстрее найти нужное решение, рецепты группируются по темам. Как правило, в каждой главе изложение ведется от простых тем к более сложным.

Книга не предназначена для чтения «от корки до корки». Скорее, ее следует использовать как справочник для решения конкретных задач или для получения общей информации о конкретных аспектах Flex Framework. В рецептах приводятся полные реализации компонентов, демонстрирующие практическое применение описываемых концепций. Читатель сможет использовать примеры кода в своих приложениях либо сразу, либо после минимальной адаптации.

---

[1] Дж. Лотт, Д. Шалл и К. Питерс «ActionScript 3.0. Сборник рецептов». – Пер. с англ. – СПб: Символ-Плюс, 2008.

Условные обозначения

В книге используются следующие условные обозначения:

Курсив

Новые **термины, URL-адреса, адреса электронной почты, имена и расширения файлов.**

`Моноширинный шрифт`

Листинги программ, а также различные программные элементы в тексте: имена переменных и функций, базы данных, типы данных, переменные среды, команды и ключевые слова.

`Моноширинный полужирный шрифт`

Команды и другой текст, который должен вводиться пользователем в точности так, как показано в книге.

`Моноширинный курсив`

Текст, который должен заменяться пользовательскими значениями (или значениями, определяемыми по контексту).

Использование примеров кода

Эта книга написана для того, чтобы помочь в решении конкретных задач. В общем случае вы можете использовать приводимые примеры кода в своих программах и документации. Связываться с авторами для получения разрешения не нужно, если только вы не воспроизводите значительный объем кода. Например, если ваша программа использует несколько фрагментов кода из книги, обращаться за разрешением не нужно. С другой стороны, для продажи или распространения дисков CD-ROM с примерами из книг O'Reilly потребуется разрешение. Если вы отвечаете на вопрос на форуме, приводя цитату из книги с примерами кода, обращаться за разрешением не нужно. Если значительный объем кода из примеров книги включается в документацию по вашему продукту, разрешение необходимо.

Мы будем признательны за ссылку на источник информации, хотя и не требуем ее. Обычно в ссылке указывается название, автор, издательство и код ISBN, например: «Flex 3 Cookbook by Joshua Noble and Todd Anderson. Copyright 2008 Joshua Noble and Todd Anderson, 978-0-596-5298-57».

Если вы полагаете, что ваши потребности выходит за рамки оправданного использования примеров кода или разрешений, приведенных выше, свяжитесь с нами по адресу *permissions@oreilly.com*.

Как пользоваться книгой

Относитесь к этой книге как к другу и советчику. Не ставьте ее на полку. Держите ее на столе, чтобы обращаться к ней как можно чаще. Если вы

не уверены в том, как работает какая-либо функция, или не знаете, как подойти к решению конкретной проблемы, возьмите книгу и откройте соответствующие рецепты. Мы постарались написать книгу в таком формате, чтобы читатель мог быстро найти ответ на конкретный вопрос. Не забывайте, что книга – не человек, так что не стесняйтесь обращаться к ней с вопросами. Слишком больших или слишком мелких вопросов вообще не бывает.

Конечно, ничто не мешает вам прочитать книгу от начала до конца, но мы рекомендуем обращаться к ней за ответами. Эта книга не излагает отвлеченную теорию, а помогает решать конкретные задачи. Она написана для практиков, а не для теоретиков.

«Сборники рецептов» O'Reilly

Ищете правильные ингредиенты для решения задачи из области программирования? «Сборники рецептов» O'Reilly к вашим услугам. В каждой книге приводятся сотни рецептов по программированию, сотни сценариев, программ и последовательностей команд, используемых для решения конкретных задач. Рецепты в этой серии книг строятся по простой схеме:

Задача

Четкая, конкретная и практичная формулировка каждой задачи, рассматриваемой в серии «книг рецептов» O'Reilly.

Решение

Понятное и легко реализуемое решение.

Обсуждение

Описание контекста проблемы и решения. В этом разделе также приводятся примеры реального кода, показывающие, как добиться желаемой цели. Все примеры кода можно загрузить с сайта книги по адресу *http://examples.oreilly.com/9780596529857/Flx3_Ckbk_code.zip*.

См. также

Ссылки раздела «См. также» отсылают вас к дополнительной информации, связанной с темой рецепта. Здесь приводятся ссылки на другие рецепты книги, другие книги (в том числе книги других издательств), веб-сайты и т. д.

Если вы захотите больше узнать о серии «Сборники рецептов» O'Reilly (а также найти другие «сборники рецептов» по вашей теме), обращайтесь на сайт *http://cookbooks.oreilly.com*.

Safari® Enabled

Если на обложке вашей любимой технической книги стоит значок Safari® Enabled, это означает, что книга доступна через O'Reilly Network Safari Bookshelf.

Система Safari лучше обычных электронных книг. Это целая виртуальная библиотека с возможностью поиска по тысячам лучших технических книг, копирования/вставки примеров кода, загрузки глав и быстрого поиска ответов, когда вам потребуется самая точная и актуальная информация. Safari можно бесплатно опробовать на сайте *http://safari.oreilly.com*.

Как с нами связаться

С замечаниями и вопросами, относящимися к книге, обращайтесь к издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (США или Канада)
707-829-0515 (международные или местные звонки)
707 829-0104 (факс)

На сайте издательства имеется веб-страница книги со списками обнаруженных опечаток, примерами и всей дополнительной информацией. Она доступна по адресу:

http://www.oreilly.com/catalog/9780596529857

С комментариями и техническими вопросами по поводу книги обращайтесь по электронной почте:

bookquestions@oreilly.com

За дополнительной информацией о книгах, конференциях, ресурсных центрах и сети O'Reilly Network обращайтесь на наш сайт:

http://www.oreilly.com

Благодарности

Эта книга создавалась совместными усилиями сообщества Flex. В работе над ней участвовали многие программисты и специалисты по связям с сообществом из фирмы Adobe – прежде всего Эми Вонг (Amy Wong), Мэтт Чотин (Matt Chotin), Эли Гринфилд (Ely Greenfeld) и Алекс Харуи (Alex Harui). Также следует поблагодарить разработчиков, которые пользовались продуктами Adobe и делились полезной информацией на сайте книги в своих блогах. Без их вклада эта книга была бы невозможна.

Огромное спасибо сотрудникам издательства O'Reilly. Хочу особо поблагодарить Стива Вайсса (Steve Weiss), Линду Лафламм (Linda Laflamme) и Мишель Филши (Michele Filshie) за усердную работу, гибкость и терпение в ходе подготовки материала и редактирования книги.

Высокое качество технической информации в книге обусловлено не только высокой квалификацией ее многочисленных авторов. Технические рецензенты книги – Марк Уолтерс (Mark Walters) (*http://www.*

digitalflipbook.com), Альфио Рэймонд (Alfio Raymond) и Джен Блэкледж (Jen Blackledge) – не только помогли с отладкой, исправлением и уточнением кода, но и поделились полезнейшими замечаниями по поводу изложения материала, структуры глав и отдельных рецептов.

От Джошуа

Прежде всего я благодарю Джои Лотта (Joey Lott), который любезно предоставил мне возможность написать эту книгу (и предыдущую тоже). Если бы не его вера в мои способности и содействие, я бы сейчас не писал эти слова. Не могу в достаточной мере выразить свою благодарность за то, что он порекомендовал меня для работы над книгой. То же относится к Стиву Вайссу, который доверился относительно неизвестному автору; благодаря ему эта книга достигла своих окончательных размеров. Спасибо моим соавторам, Тодду Андерсону (Todd Anderson) и Эби Джорджу (Abey George), а также сотрудникам Cynergy Systems: Эндрю Трайсу (Andrew Trice), Крейгу Драбнику (Craig Drabnik), Кьюну Ли (Keun Lee) и Райану Миллеру (Ryan Miller) – они помогали мне, когда я нуждался в помощи, и я им многим обязан. Благодарю Дэниела Райнхарта (Daniel Rinehart), который замечательно справился с написанием рецептов главы «FlexUnit и модульное тестирование», без всяких просьб, а просто для того, чтобы поделиться полезной информацией с сообществом Flex. То же можно сказать обо всех участниках обсуждения на сайте Adobe Cookbook и таких форумах, как FlexCoders; это энергичное, бескорыстное сообщество помогает всем нам.

Также спасибо всем моим друзьям и коллегам за помощь, советы, поддержку и хорошее настроение. Наконец, хочу поблагодарить свою семью (и особенно мать) за ободрение и мудрость.

От Тодда

Прежде всего благодарю Джоша Ноубла (Josh Noble), который пригласил меня участвовать в работе над книгой и постоянно делился своими знаниями, терпением и хорошим настроением. Спасибо Джои Лотту за содействие и веру в человеческие способности. Я благодарен моим друзьям и всему сообществу Flash за советы, шутки и полезный опыт. Спасибо моей семье – я не могу в должной мере поблагодарить вас всех за огромную любовь и поддержку.

Авторы

Джошуа Ноубл (Joshua Noble), программист и консультант из Нью-Йорка, соавтор книги «ActionScript 3.0 Bible» (Wiley, 2007). Использовал Flex и Flash при разработке многих веб-приложений на разных платформах в течение шести лет, также обладает опытом работы с PHP, Ruby, Erlang и C#. В свободное время развлекается с C++ и OpenCV, а также экспериментирует с микроконтроллерами и сенсорами для создания «умного окружения». Сайт: *http://thefactoryfactory.com*.

Тодд Андерсон (Todd Anderson) – ведущий программист Infrared5. За пять лет программирования на платформе Flash в областях RIA и компьютерных игр Тодд создал ряд настольных и веб-приложений для компаний, работающих в области издательского дела и индустрии развлечений, включая McGraw-Hill, Thomson, Motorola и Conde Nast Publications. В настоящее время живет неподалеку от Бостона; в свободное от программирования время занимается рисованием. Андерсон ведет блог *www.custardbelly.com/blog*, посвященный программированию для платформы Flash.

Об участниках

Эби Джордж (Abey George) – программист с опытом работы в проектировании и разработке RIA (Rich Internet Application). Реализовал множество RIA-решений для веб- и корпоративных приложений на базе Flex, Flash и C#. Обладатель ученой степени магистра наук Техасского университета A&M, более шести лет профессионального опыта в области программирования. Серьезно занимается эффективным применением RIA в корпоративных средах. В настоящее время Эби работает ведущим программистом в Fidelity Investments; ранее работал в Yahoo!, Keane и Mindseye.

Дэниел Райнхарт (Daniel Rinehart) – разработчик архитектуры программного обеспечения в Allurent, где занимается построением следующего поколения интернет-магазинов на базе Flex. Последние восемь лет работал в области разработки программного обеспечения. До прихода в Allurent Дэниел работал на Ruckus Network, Towers Perrin и Bit Group, а в число его клиентов входили Cisco и Dell Financial Services. С ним можно связаться на сайте *http://danielr.neophi.com/*.

Эндрю Трайс (Andrew Trice) – ведущий разработчик архитектуры программного обеспечения для Flex и AIR в Cynergy Systems. Специализируется на визуализации данных, архитектурах «клиент-сервер», принципах объектно-ориентированного программирования и разработке RIA. Занимается программированием веб-приложений свыше 10 лет, более 8 лет работает с платформой Flash. Привлеченный гибкостью и широтой возможностей Flex/Flash, Эндрю использует Flex, начиная с версии 1.5. Также обладает 7-летним опытом использования ColdFusion, является сертифицированным разработчиком Microsoft, разбирается в реляционных базах данных, Ajax/JavaScript, программировании .NET и веб-приложений Java.

Кьюн Ли (Keun Lee) – ведущий технический специалист Cynergy Systems. Специализируется на таких технологиях, как Adobe Flex и Microsoft Windows Presentation Foundation. Обладает разносторонним опытом в области бизнес-аналитики, архитектуры приложений B2B и разработки RIA. В свободное время занимается музыкой и построением всевозможных классных штук из имеющихся в его распоряжении технологических ресурсов.

Крейг Драбник (Craig Drabnik) с 2000 года занимается веб-программированием на базе DHTML, ColdFusion, Flash и Flex. В настоящее время работает в Cynergy Systems, где руководит реализацией проектов Flex.

Райан Тейлор (Ryan Taylor) – художник и программист, специализирующийся на объектно-ориентированном Flash-программировании и дизайне статической/динамической графики. В настоящее время работает старшим программистом в группе мультимедийных платформ в фирме Schematic. Райан часто выступает на отраслевых мероприятиях и делится своими мыслями и опытом, а также вносит свой вклад в движение Open Source, в блоге по адресу *www.boostworthy.com/blog*.

Марко Касарио (Marco Casario) основал Comtaste (*www.comtaste.com*) – компанию, занимающуюся передовыми разработками в области RIA и адаптацией веб-технологий для мобильных устройств. Является автором книги «Flex Solutions: Essential Techniques for Flex 2 and Flex 3 Developes» (Friends of ED, 2007) и «Advanced AIR Applications» (Friends of ED, 2008). Марко часто выступает на Adobe MAX, O'Reilly Web 2.0 Summit, FITC, AJAXWorld Conference & Expo, 360Flex, From A to Web, AdobeLive и других конференциях. Подробности можно узнать в его блоге по адресу *http://casario.blogs.com*.

Андрей Ионеску (Andrei Ionescu) – румынский веб-программист, любитель новых технологий. Увлекается созданием приложений RIA и построением всевозможных веб-приложений. Ведет блог *www.flexer.info*, посвященный программированию Flex. Сайт его компании – *www.designit.ro*.

Райан Миллер (Ryan Miller) занимается веб-программированием более 7 лет. Работая на многие крупные и мелкие компании, приобрел разносторонний практический опыт. В настоящее время работает на Cynergy Systems у себя дома в Бивертоне, штат Орегон, тратит на это все свободное время и получает от этого удовольствие.

1

Основы Flex и ActionScript

Приложение Flex состоит в основном из кода, написанного на двух разных языках: ActionScript и MXML. В последней на момент написания книги версии 3.0 *ActionScript* из сценарного языка на базе прототипов превратился в полностью объектно-ориентированный язык с жесткой типизацией, соответствующий стандарту ECMAScript. *MXML* – язык разметки, с которым легко освоится любой, кто имел дело с HTML (Hypertext Markup Language), XML (eXtensible Markup Language) или более новыми языками разметки.

Как MXML и ActionScript связаны друг с другом? Компилятор после разбора разнородных идиом преобразует их в сходные объекты, так что для фрагментов

```
<mx:Button id="btn" label="My Button" height="100"/>
```

и

```
var btn:Button = new Button();
btn.label = "My Button";
btn.height = 100;
```

будут созданы одинаковые объекты. Главное отличие заключается в том, что при создании объекта в ActionScript (второй пример) создается только кнопка и ничего более, а при создании объекта в MXML кнопка включается в компонент, содержащий код MXML. Flex Framework обеспечивает вызов конструктора объекта, описанного в MXML, с последующим включением его в родительский компонент или назначением в качестве свойства родителя.

Файлы MXML могут содержать вставки ActionScript в тегах `<mx:Script>`, однако файлы ActionScript не могут содержать код MXML. Возникает заманчивая мысль рассматривать MXML как описание внешнего вида и компонентов, входящих в состав приложения, а ActionScript – как

описание обработчиков событий и пользовательской логики, необходимой для работы приложения, однако это не всегда так. Намного правильнее понять, что оба языка в конечном счете описывают одни и те же объекты, но с разным синтаксисом. Некоторые возможности платформы Flex недоступны без применения ActionScript для создания циклов и условных команд, объявления функций и т. д. По этой причине код ActionScript и интеграция MXML/ActionScript необходимы всегда, за исключением самых простейших приложений.

В этой главе рассматриваются многие аспекты интеграции MXML с ActionScript: создание компонентов в MXML, создание классов в ActionScript, добавление слушателей (listeners) событий, создание файлов программной логики с использованием ActionScript и MXML, объявление функций. Хотя вы не найдете здесь ответов на все вопросы, материал этой главы поможет освоить азы ActionScript и MXML.

1.1. Создание проекта Flex в Flex Builder

Задача

Требуется создать проект в Flex Builder.

Решение

Воспользуйтесь мастером создания нового проекта (Create New Project).

Обсуждение

Flex Builder построен на базе Eclipse – заслуженной и уважаемой интегрированной среды разработки (IDE), наиболее тесно связанной с программированием на Java. Конечно, использование Flex Builder не является строго обязательным для Flex-программирования, однако эта программа является наиболее совершенным инструментом для создания Flex-приложений, а ее разнообразные функции повышают эффективность проектирования и разработки приложений. Flex Builder может использоваться как в форме автономного приложения, так и в форме подключаемого модуля для существующей установки Eclipse.

Работа над любым приложением Flex начинается с создания нового проекта Flex (рис. 1.1). Проекты Flex отличаются от всех остальных типов проектов, создаваемых в Flex Builder, – в них включается библиотека Flex SWC (в отличие от проектов ActionScript), и они компилируются в файл SWF, который может быть просмотрен в Flash Player (в отличие от проекта Flash Library). Чтобы создать проект Flex, откройте контекстное меню щелчком правой кнопкой мыши (или с нажатой клавишей Control на Mac) в навигаторе проекта Flex Builder, или откройте меню File у верхнего края окна приложения. Выберите в открытом меню команду New→Flex Project. На экране появляется диалоговое окно, руководящее созданием проекта.

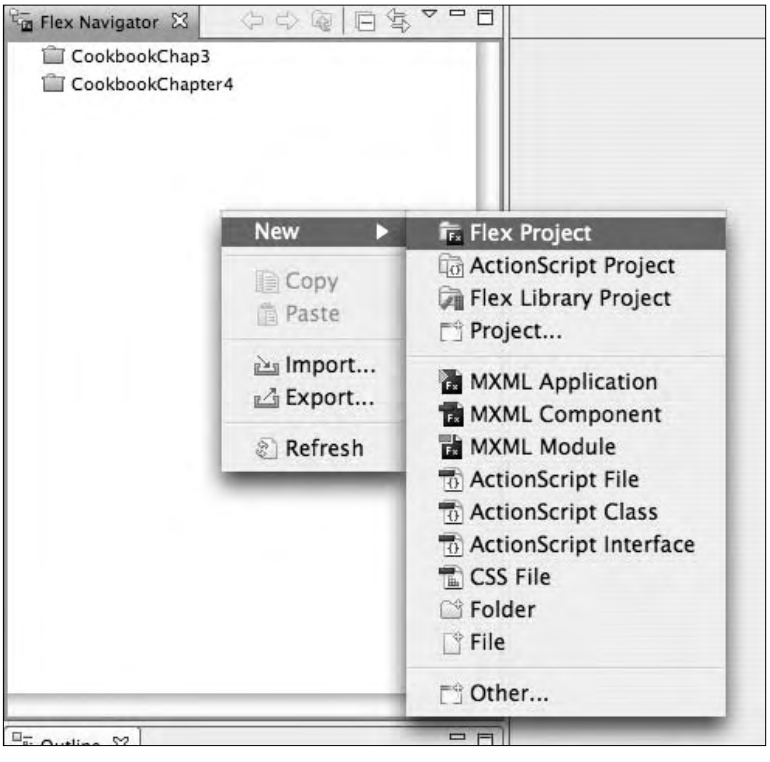

Рис. 1.1. *Создание проекта Flex*

Когда мастер предложит выбрать способ получения данных, выберите вариант Basic. Открывается диалоговое окно New Flex Project (рис. 1.2).

Введите название приложения и папку для хранения файлов. По умолчанию на компьютерах Windows используется папка C:/Documents and Settings/Username/Documents/workspace/Projectname, а на Mac – папка Users/Username/Documents/workspace/Projectname. Конечно, вы можете снять флажок Use default location и сохранить файлы там, где сочтете нужным. Имя проекта должно быть уникальным. В секции Application type выбирается тип создаваемого приложения – приложение AIR (Adobe Integrated Runtime) или приложение, выполняемое в броузере с использованием подключаемого модуля Flash Player. Наконец, настройки в секции Server Technology позволяют указать, будет ли приложение подключаться к серверу, и если будет – выбрать тип сервера и конфигурацию.

Когда вся необходимая информация будет введена, щелкните на кнопке Finish. Чтобы сменить папку, в которой будет размещен откомпилированный файл SWF, щелкните на кнопке Next; откроется окно, показанное на рис. 1.3.

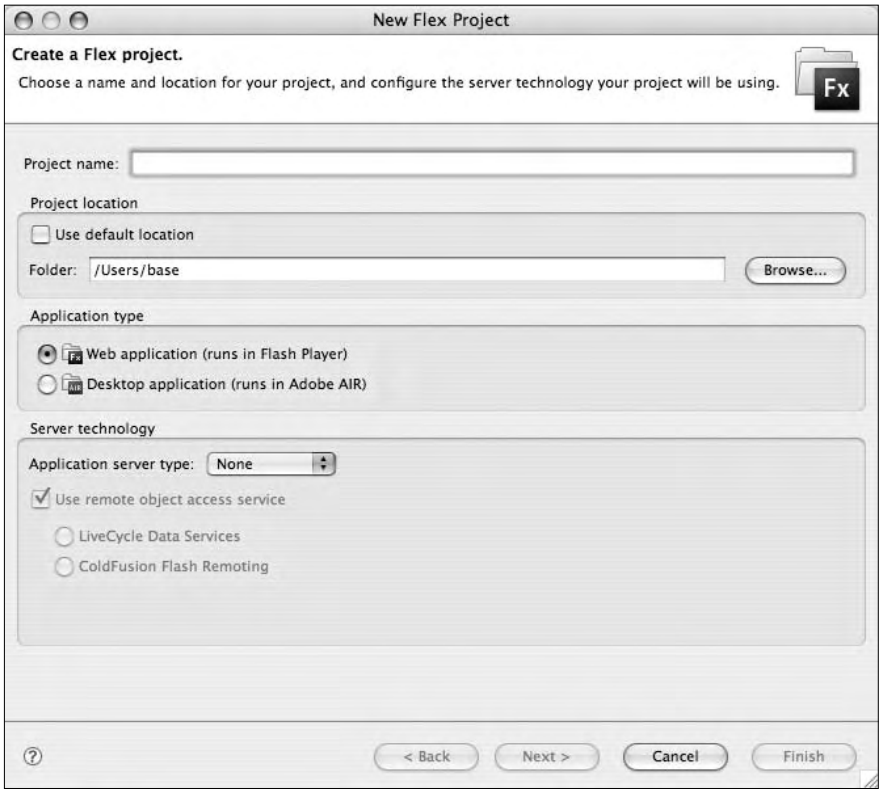

Рис. 1.2. *Создание нового проекта в Flex Builder*

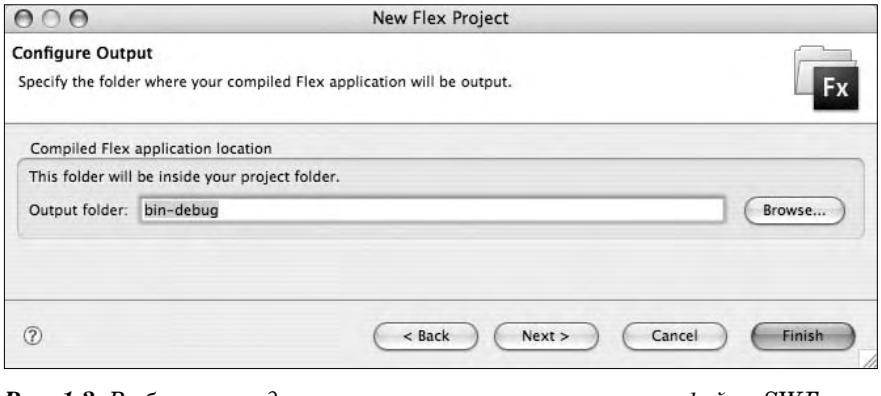

Рис. 1.3. *Выбор папки для хранения откомпилированного файла SWF*

После выбора папки для сгенерированного файла SWF можно либо завершить создание проекта, либо добавить в него другие папки и файлы SWC. Чтобы добавить новую папку или группу папок, щелкните на корешке вкладки Source path (рис. 1.4). Чтобы включить в проект файлы SWC, перейдите на вкладку Library path (рис. 1.5). В этом окне также можно выбрать главный файл приложения MXML; по умолчанию используется файл, имя которого совпадает с именем проекта.

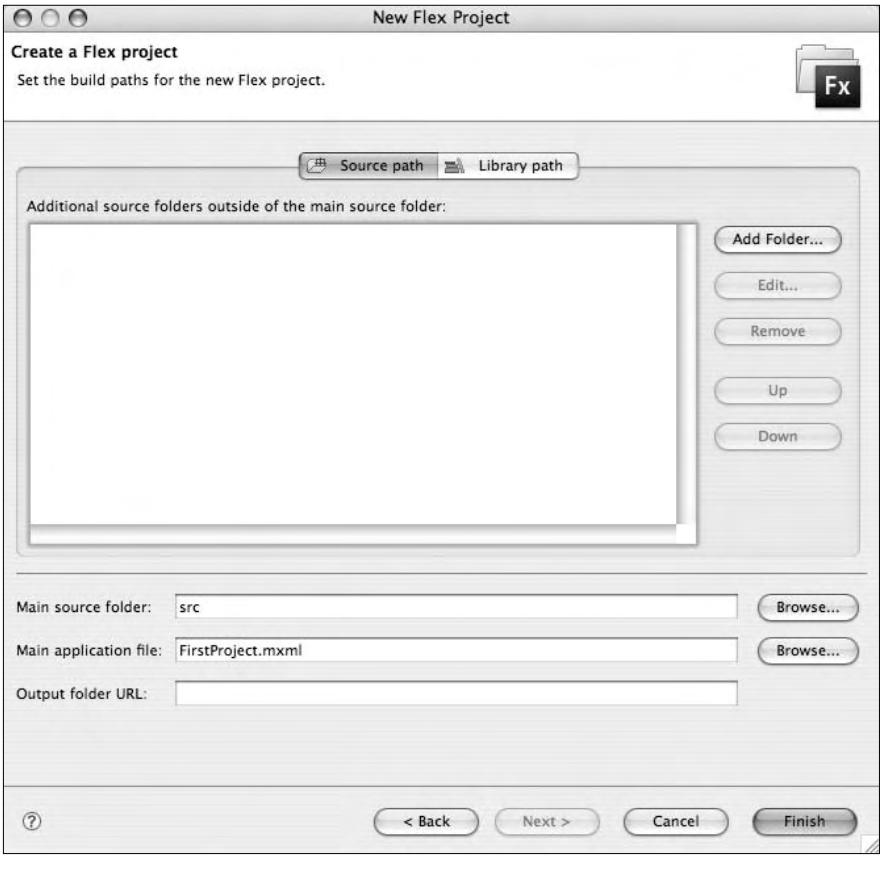

Рис. 1.4. *Назначение папок с исходным кодом и главного файла приложения*

Когда все пути и имена будут заданы, щелкните на кнопке Finish. Настройка проекта завершена, можно переходить к разработке.

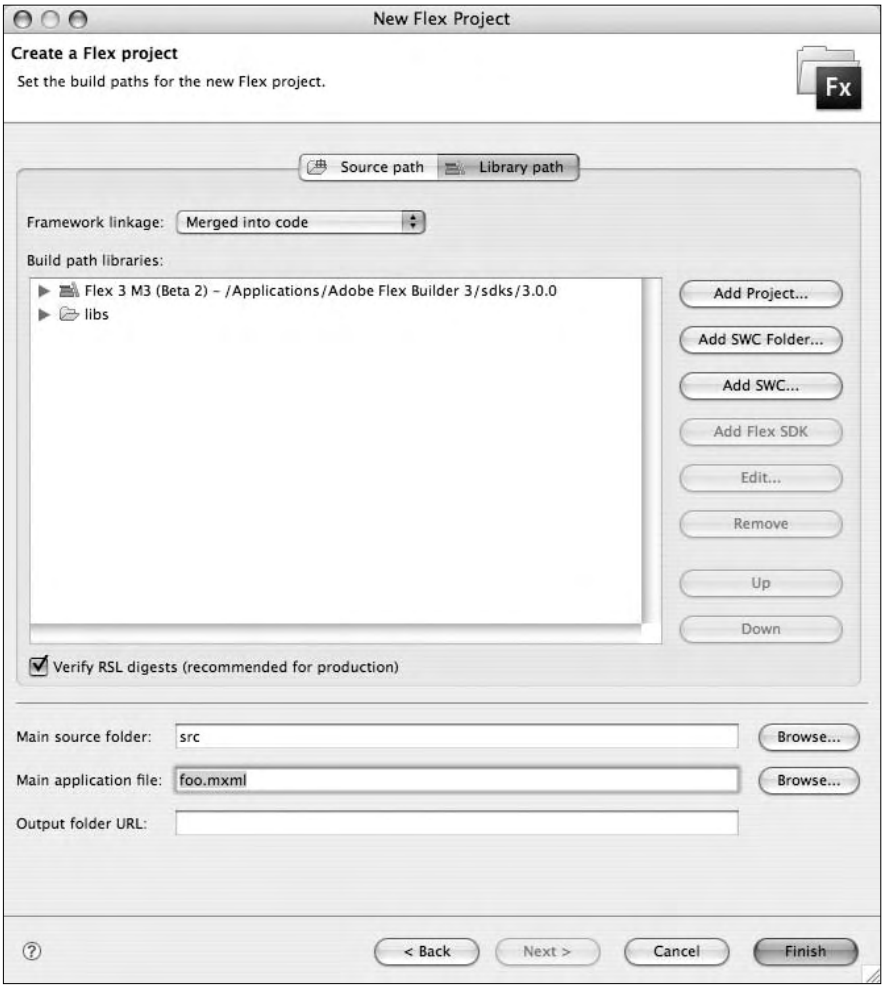

Рис. 1.5. *Включение дополнительных библиотек в проект Flex*

1.2. Создание проекта библиотеки Flex в Flex Builder

Задача

Требуется создать проект библиотеки Flex.

Решение

Выполните в Flex Navigator команду New→Flex Library Project. Команда запускает мастер создания нового проекта.

Обсуждение

Проект библиотеки Flex не имеет главного файла MXML, компилируемого в формат SWF. Вместо этого файлы компилируются в файл SWC, который может использоваться в других приложениях или в качестве источника для общей библиотеки времени выполнения (обычно сокращенно называемой RSL от «Runtime Shared Library»). Классы в библиотеке используются для создания группы ресурсов, доступных в нескольких проектах (на стадии компиляции или выполнения). Чтобы создать проект библиотеки Flex, откройте контекстное меню щелчком правой кнопкой мыши (или с нажатой клавишей Control на Mac) в навигаторе проекта Flex Builder (рис. 1.6) или откройте меню File. Выберите в открытом меню команду New→Flex Library Project.

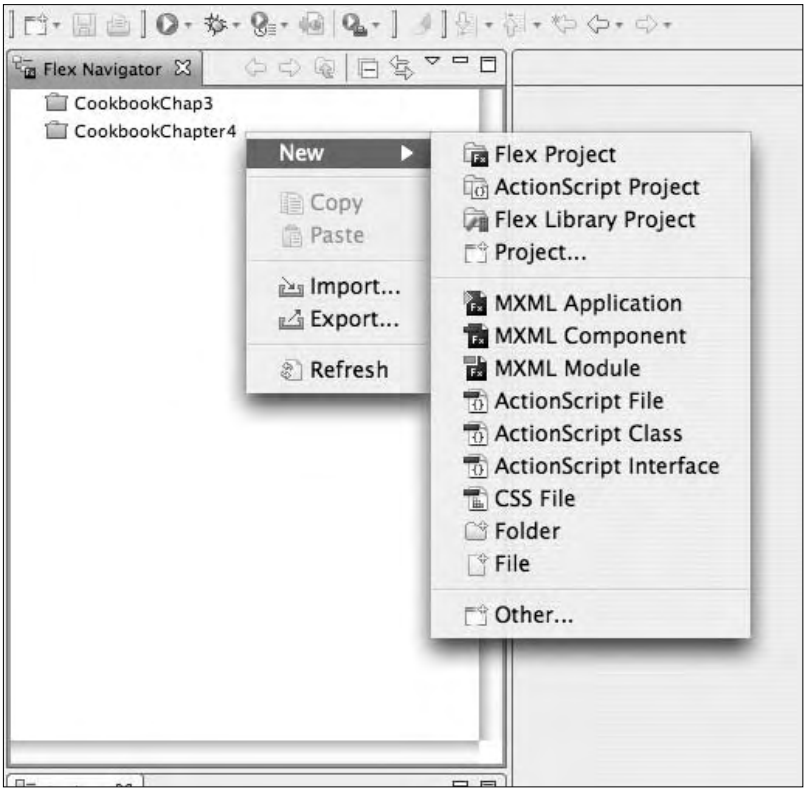

Рис. 1.6. *Создание проекта библиотеки Flex*

В открывшемся диалоговом окне (рис. 1.7) укажите имя проекта, а также его местонахождение.

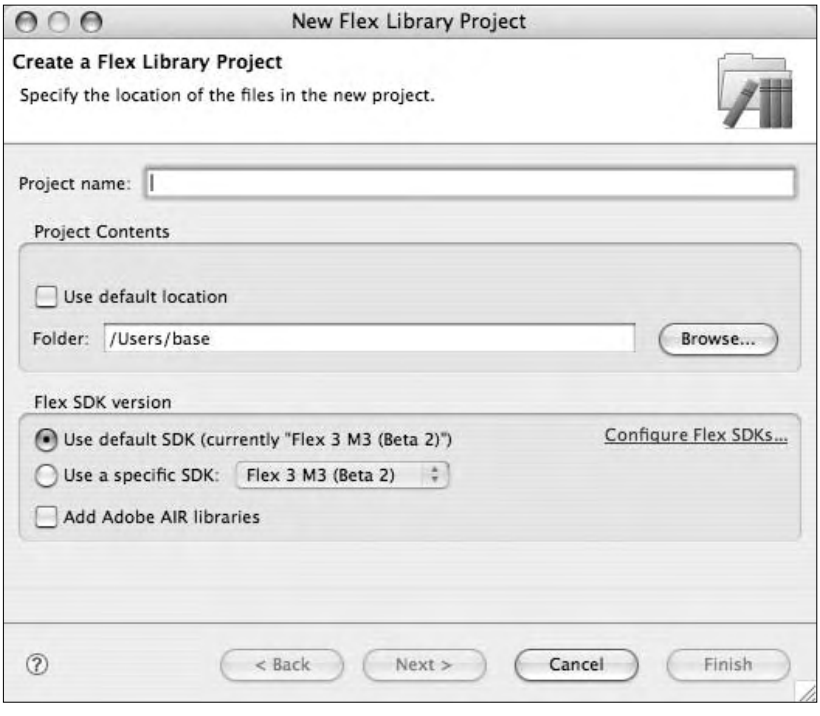

Рис. 1.7. *Ввод информации о местонахождении проекта и SDK для компилятора*

Когда вся необходимая информация будет введена, щелкните на кнопке Finish. Чтобы включить файлы, ресурсы или другие файлы SWC, включая библиотеки Adobe AIR, щелкните на кнопке Next и выберите их в открывшемся окне. Чтобы выбрать и включить в проект классы, сначала откройте включаемую папку, а затем выберите классы или графические ресурсы, которые будут включены в библиотеку при компиляции. Создание проекта завершается кнопкой Finish.

1.3. Создание проекта ActionScript

Задача

Требуется создать проект ActionScript, не использующий библиотеки Flex 3.

Решение

Воспользуйтесь мастером создания нового проекта и выберите тип проекта ActionScript Project.

Обсуждение

Проект ActionScript отличается от проекта Flex прежде всего тем, что Flex Framework в него вообще не включается. Проекты ActionScript зависят только от основных классов ActionScript в кодовой базе Flash, а компоненты **Flex Framework для них недоступны. Проекты Action**-Script создаются командой File→New→ActionScript Project (рис. 1.8).

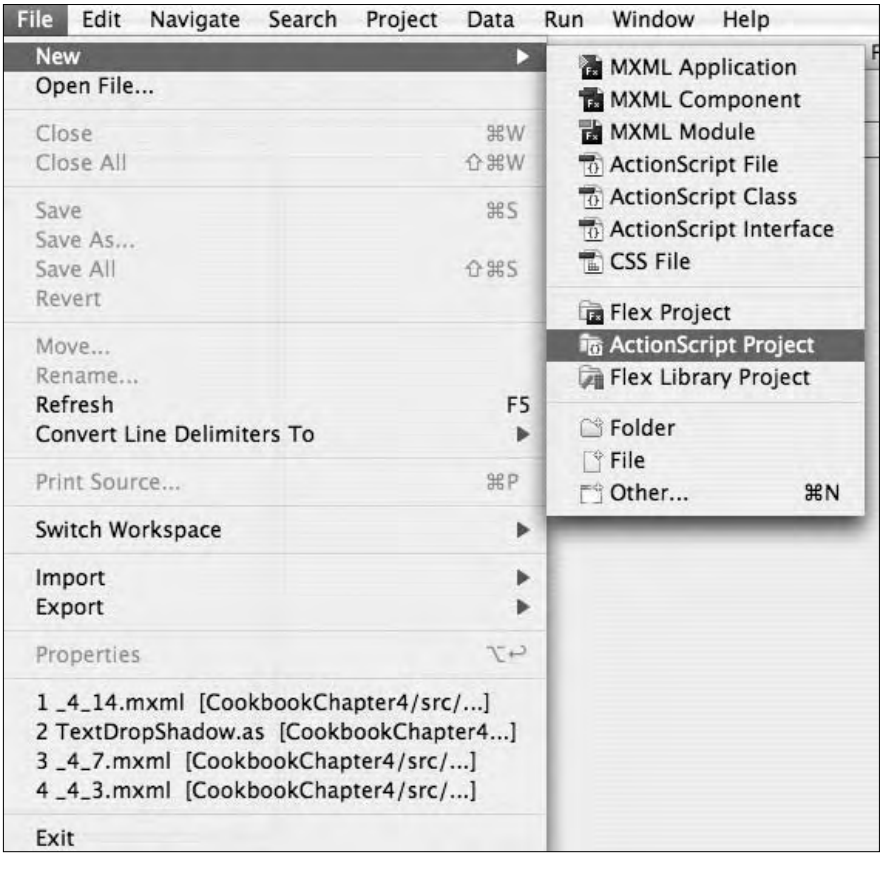

Рис. 1.8. *Создание проекта ActionScript*

В открывшемся диалоговом окне укажите имя проекта и папку для хранения файлов, включая откомпилированные SWF-файлы. Далее либо щелкните на кнопке Finish, чтобы создать проект в конфигурации по умолчанию, либо на кнопке Next, чтобы включить в проект дополнительные библиотеки или папки с исходным кодом, сменить главный файл приложения, добавить файлы SWC или изменить папку с выходным SWF-файлом. По умолчанию имя главного файла ActionScript совпадает с именем проекта, а файл SWF размещается в папке bin-debug.

1.4. Настройка параметров компилятора MXML в Flex Builder

Задача

Требуется задать нестандартные параметры компилятора MXML.

Решение

Задайте параметры компилятора в окне Flex Compiler диалогового окна свойств проекта.

Обсуждение

Компилятор MXML, также называемый *mxmlc*, – программа, компилирующая файлы ActionScript и MXML в файл SWF, который может быть просмотрен в Flash Player. При запуске или отладке приложения Flex в Flex Builder запускается компилятор MXML, а имена компилируемых файлов передаются ему в аргументах команды. В ходе отладки компилятору MXML передается аргумент, включающий режим создания отладочного файла SWF. Flex Builder также позволяет передавать компилятору MXML другие аргументы; например, в этих аргументах можно указать местонахождение внешней библиотеки, разрешить доступ к локальным файлам из SWF или выбрать цвет фона.

Чтобы изменить настройки компиляции для проекта, щелкните правой кнопкой мыши (или с нажатой клавишей Control на Mac) внутри проекта и выберите команду Properties в контекстном меню (рис. 1.9) или выберите команду Project→Properties в главном меню.

В открывшемся диалоговом окне свойств проекта (рис. 1.10) выберите категорию Flex Compiler. Эта категория содержит несколько параметров, управляющих процессом компиляции. В текстовом поле Additional compiler arguments можно ввести сразу несколько параметров; ставьте дефис (-) перед каждым параметром и разделяйте их пробелами.

Некоторые часто используемые параметры:

`verbose-stacktraces`

Определяет, будут ли выдаваться номера строк и имена файлов при возникновении ошибки времени выполнения в файле SWF. Включение этой информации увеличивает размер SWF, причем файл SWF с включенным режимом `verbose-stacktraces` отличается от отладочного файла SWF.

`source-path path-element`

Включение в проект файлов MXML или ActionScript. Допускается использование метасимволов для включения всех файлов и подкаталогов. Специальное обозначение `+=` присоединяет новый аргумент

к параметрам по умолчанию или параметрам, заданным в конфигурационном файле, например:

```
-source-path+=/Users/base/Project
```

`include-libraries`

Задает файлы SWC, включаемые в компилируемое приложение, и обеспечивает компоновку всех классов и ресурсов в SWF. Параметр полезен в том случае, если приложение загружает другие модули, которые обращаются к классам в файле SWC, не используемом файлом SWF.

`library-path`

Аналог `include-libraries`, но с включением только тех классов и ресурсов, которые используются в SWF. Позволяет избежать нежелательного разрастания файлов SWF.

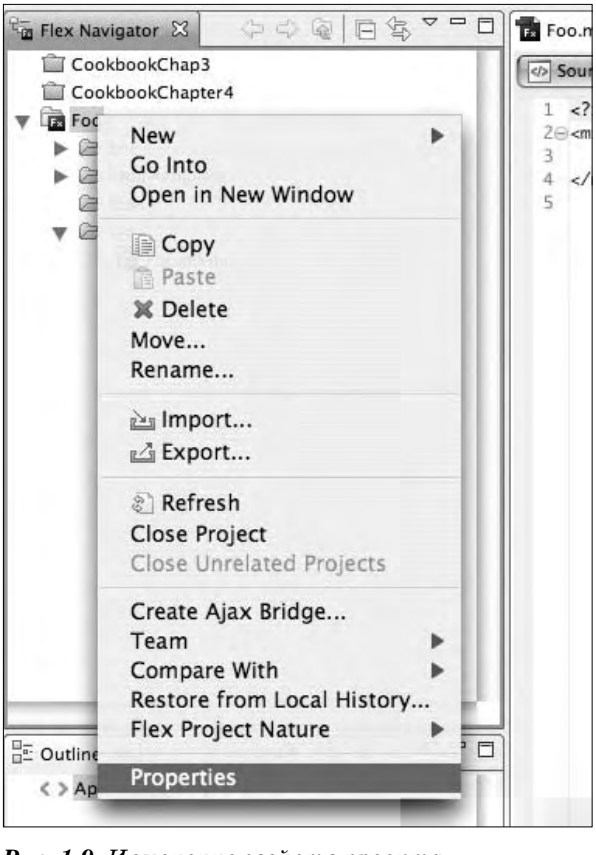

Рис. 1.9. *Изменение свойств проекта*

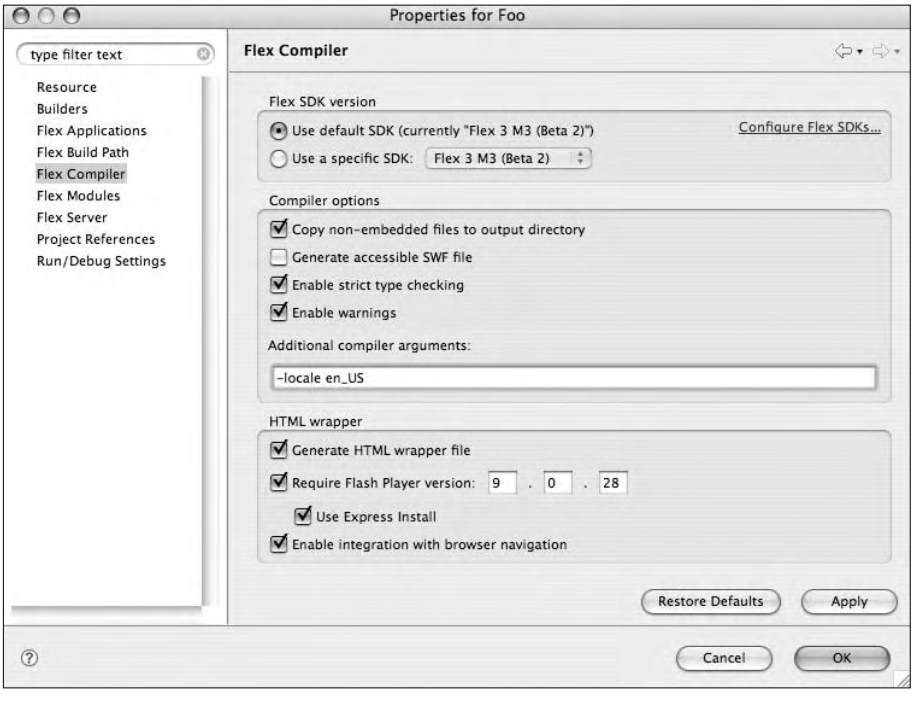

Рис. 1.10. *Параметры компилятора*

`locale`

Задает локальный контекст, связываемый с файлом SWF. Например, параметр `-locale=es_ES` назначает файлу SWF испанский локальный контекст.

`use-network`

Указывает, обладает ли файл SWF доступом к локальной файловой системе, или же для него должна использоваться стандартная модель безопасности Flash Player. Например, значение `-use-network=false` указывает, что файл SWF будет обладать доступом к локальной файловой системе, но не сможет пользоваться сетевым сервисом. По умолчанию используется значение `true`.

`frames.frame`

Позволяет назначить «фабрики ресурсов», подключаемые после запуска приложения и публикующие свой интерфейс при помощи класса `ModuleManager`. Преимущество такой схемы состоит в том, что она ускоряет запуск приложения по сравнению с включением ресурсов в программный код, но не требует перемещения ресурсов во внешний файл SWF. Данный параметр не столь очевиден, как предыдущие, но польза от него несомненна.

`keep-all-type-selectors`

Гарантирует включение всей стилевой информации (даже не используемой в приложении) в файл SWF. Это особенно важно при загрузке приложением других компонентов, для которых стилевая информация необходима. По умолчанию используется значение `false`, означающее, что неиспользуемая стилевая информация не компилируется в SWF.

Завершив настройку параметров компилятора, щелкните на кнопке Apply. Внесенные изменения сохраняются в проекте.

1.5. Компиляция проекта Flex за пределами Flex Builder

Задача

Вы не используете Flex Builder для разработки приложения Flex. Требуется откомпилировать проект.

Решение

Запустите компилятор MXML в окне терминала или командной строки.

Обсуждение

Flex Builder – мощный инструмент для разработки приложений Flex, однако его применение не является обязательным. Компилятор MXML (mxmlc) распространяется свободно, его можно бесплатно загрузить с сайта Adobe. Чтобы откомпилировать приложение Flex за пределами Flex Builder, откройте окно командной строки (Windows) или окно терминала (Mac OS X), запустите компилятор MXML с указанием имени файла приложения. Команда выглядит примерно так:

```
home:base$ . /Users/base/Flex SDK 3/bin/mxmlc ~/Documents/FlexTest/FlexTest.
mxml
```

Команда компилирует файл MXML в файл SWF; по умолчанию откомпилированный файл размещается в той же папке, в которой находился файл MXML. Все предупреждения и ошибки компилятора выводятся в окне терминала или командной строки. Чтобы задать дополнительные параметры при вызове компилятора MXML, присоедините аргументы к строке запуска. Например, команда

```
home:base$ ./mxmlc ~/Documents/FlexTest/FlexTest.mxml -output=/Users/base/
test/genera ted/Index.swf -library-path+=/Users/lib/MyLib.swc
```

генерирует файл SWF с именем Index.swf в каталоге /Users/base/test/generated/ и включает в него библиотеку SWC /Users/lib/MyLib.swc.

Чтобы компилятор MXML запускался из командной строки без указания полного пути к установке SDK (в приведенном примере C:\flex_sdk_3),

следует добавить каталог /bin, **в котором находится компилятор, в системную переменную** Path.

В системе Windows:

1. Запустите приложение Система из панели управления.
2. Перейдите на вкладку Дополнительно.
3. Щелкните на кнопке Переменные среды.
4. В поле Системные переменные найдите строку Path и дважды щелкните на ней.
5. Если содержимое поля Значение переменной не завершается символом «точка с запятой» (f;), введите этот символ, а затем путь к папке /bin в установочном каталоге Flex SDK (рис. 1.11).

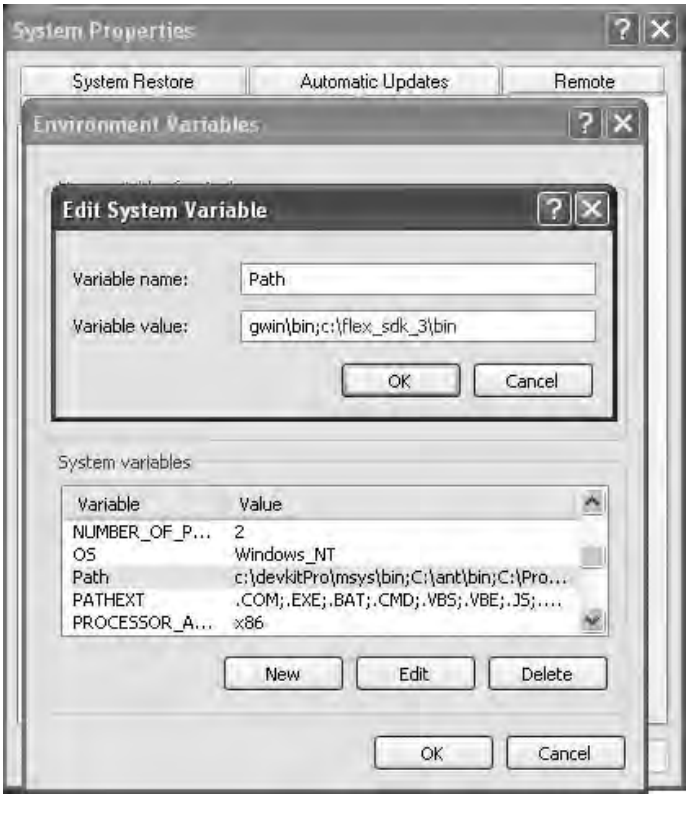

Рис. 1.11. *Включение каталога Flex SDK 3 в переменную Path*

6. После включения пути к каталогу компилятора MXML откройте окно командной строки, перейдите в каталог проекта и введите следующую команду:

```
C:\Documents\FlexTest> mxmlc FlexTest.mxml
```

Команда создает файл FlexTest.swf в каталоге C:\Documents\FlexTest, как и первая команда, представленная в этом разделе. Включение пути к каталогу /bin установки Flex 3 SDK позволяет запустить компилятор из любого каталога, в том числе (в данном примере) из каталога текущего проекта.

7. Если на шаге 6 будет получено сообщение об ошибке:

```
Error: could not find JVM
```

вы должны вручную ввести путь к каталогу, в котором среда JRE (Java Runtime Environment) установлена на вашем компьютере. Перейдите в установочный каталог Flex 3 SDK, откройте файл jvm.config в текстовом редакторе и присоедините путь к установочному каталогу JRE к переменной `java.home`. Если предположить, что среда Java установлена в корневом каталоге жесткого диска, соответствующая строка будет выглядеть так:

```
java.home=C:/Java/jre
```

В Mac OS X или Linux:

1. Откройте файл .bash_profile (если вы используете Bash) и присоедините к переменной PATH путь к компилятору MXML. Файл .bash_profile должен выглядеть примерно так:

```
PATH="${PATH}:~/flex3SDK/bin"
export PATH
```

Файл .bash_profile находится в домашнем каталоге пользователя (который всегда можно открыть командой `cd ~`). Если вы используете tsch, путь к компилятору MXML добавляется в файл .profile.

2. Если среда Java не настроена, введите следующие команды в окне терминала:

```
PATH="${PATH}:~/flex3SDK/bin"
export PATH
```

1.6. Добавление слушателей событий в коде MXML

Задача

Требуется определить в коде MXML слушателя событий, передаваемых дочерними компонентами.

Решение

Включите имя метода в тег компонента (с объектом события или без).

Обсуждение

Компоненты Flex отправляют события при выполнении различных действий – щелчке на кнопке, выборе в списке новой строки, загрузке дан-

ных и т. д. Чтобы принимать рассылаемые события, включите в тег компонента ссылку на функцию, обрабатывающую событие. Например:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">

    <mx:Script>
        <![CDATA[

            private function buttonClick():void
            {
                trace(" Button has been clicked ");
            }

        ]]>
    </mx:Script>

    <mx:Button click="buttonClick()" label="Click Me"/>
</mx:Canvas>
```

Конструкция `click="buttonClick()"` обеспечивает вызов функции `buttonClick` каждый раз, когда кнопка выдает событие щелчка.

Функции также может передаваться объект события. При отправке события компонент прилагает к нему объект типа `Event`, который может быть получен любым объектом, принимающим данное событие. Пример:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">

    <mx:Script>
        <![CDATA[

            private function buttonClick(event:Event):void
            {
                trace(event.target.id);
                if(event.target.id == "buttonOne")
                {
                    trace(" button one was clicked")
                } else {
                    trace(" button two was clicked")
}
            }

        ]]>
    </mx:Script>

    <mx:Button click="buttonClick(event)"
label="Click Me One" id="buttonOne"/>
    <mx:Button click="buttonClick(event)" label="Click Me Two"
id="buttonTwo"/>
</mx:HBox>
```

Слушатель события может по-разному реагировать на полученный объект события в зависимости от заданных критериев. В приведенном примере реакция зависит от того, какой компонент отправил событие.

Объекты событий и система диспетчеризации событий Flex принадлежат к числу важнейших понятий, в которых необходимо разобраться при изучении языка. Все события имеют тип, указываемый при прослушивании; например, если событие относится к типу `click`, то метод прослушивания события добавляется в событие click дочернего компонента:

```
<mx:Button click="trace('I was clicked')"/>
```

Информация о действиях пользователя, о сообщениях, полученных приложением от сервера или происходящих по таймеру, передается в виде событий. Объект события определяет несколько свойств, доступных в любой функции:

`bubbles`

Определяет, является ли событие «каскадным» (т. е. должен ли объект, получивший событие, передать его слушателям далее по цепочке обработки событий).

`cancelable`

Указывает, возможна ли отмена обработки, связанной с событием.

`currentTarget`

Объект, активно обрабатывающий объект события.

`eventPhase`

Текущая фаза цикла обработки события.

`Target`

Приемник события (объект, передавший событие обработчику).

`Type`

Тип события.

Обработчики событий можно определять прямо в коде MXML; фигурные скобки {} означают, что содержащийся в них код должен выполняться при возникновении события. Пример:

```
 <mx:Button click="{textComponent.text = 'You clicked the button}"
     label="Click Me"/
>
    <mx:Text id="textComponent"/>
```

При обработке этого кода компилятор Flex создает функцию и назначает ей в качестве тела фрагмент `textComponent.text = 'You clicked the button'`. На первый взгляд такая конструкция отличается от предыдущего метода, но конечный результат одинаков: прослушивание события и выполнение заданного кода. В таком способе нет ничего нежелательного, но для любых задач, менее тривиальных, чем задание одного свойства,

лучше определить отдельную функцию – это сделает программу более понятной и удобочитаемой.

1.7. Задание свойств дочернего компонента, определенного в MXML, в коде ActionScript

Задача

Требуется задать свойства дочернего компонента, определенного в теге `script` MXML, в контексте метода.

Решение

Используйте свойство `id` для задания свойств и вызова методов дочерних компонентов.

Обсуждение

Может показаться, что сценарная часть компонента отделена от определений MXML, но в действительности это совершенно не так. Возьмем следующий пример:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
        height="300">
    <mx:Script>
        <![CDATA[

            private function changeAppearance():void
            {
                this.width = Number(widthInputField.text);
                this.height = Number(heightInputField.text);
            }

        ]]>
    </mx:Script>
    <mx:Image id="imageDisplay"/>
    <mx:Text text="Enter a width"/>
    <mx:TextInput id="widthInputField"/>
    <mx:Text text="Enter an height"/>
    <mx:TextInput id="heightInputField"/>
    <mx:Button click="changeAppearance()" label="Change Size"/>
</mx:HBox>
```

Как видно из метода `changeAppearance`, ключевое слово `this` обозначает сам компонент `HBox` с дочерними компонентами; функция изменяет ширину и высоту компонента. Ссылки на текстовые поля `widthInputField` и `heightInputField` используются для получения текстовых данных из этих полей. Для обращения к текстовым полям используются их свойства `id`, по аналогии с тем, как мы обращаемся по свойству `id` к элементам модели DOM (Document Object Model). Значение `id` представляет собой уни-

кальное имя, используемое в приложении для обращения к дочерним компонентам. В пределах одного компонента иерархия всегда состоит из одного уровня; свойство id ссылается на дочерний компонент независимо от его фактического уровня вложенности (например, если он вложен в другой дочерний компонент). Рассмотрим похожий пример активного задания свойства дочернего компонента:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="520"
        height="650">
    <mx:Script>
        <![CDATA[

            private var fileName:String = "";

            private function saveResume():void
            {
                //...Вызов службы для отправки данных и задания имени файла
                fileNameDisplay.text = "Your resume has been saved
                    as "+fileName;
            }

        ]]>
    </mx:Script>
    <mx:Text id="fileNameDisplay" text="" width="500"/>
    <mx:RichTextEditor id="richTextControl" width="500" height="400"/>
    <mx:Button id="labelButton" label="Submit Resume" click="saveResume()"/>
</mx:VBox>
```

1.8. Определение массивов и объектов

Задача

Требуется определить объект Array или объект хеш-таблицы для хранения примитивных значений или объектов.

Решение

Используйте синтаксис ActionScript для создания новых объектов и массивов посредством конструктора или определите их в MXML.

Обсуждение

Как это обычно бывает в Flex, массивы и объекты – две самых распространенных разновидности контейнеров для хранения данных – могут определяться как в ActionScript, так и в MXML. Для определения массивов в MXML используется тег <mx:Array>:

```
<mx:Array>
        <mx:String>Flex</mx:String>
        <mx:String>Flash</mx:String>
        <mx:String>Flash Media Server</mx:String>
```

```
    <mx:String>Flash Lite</mx:String>
    <mx:String>AIR</mx:String>
</mx:Array>
```

При обращении к элементу массива указывается его индекс (начиная с нуля). Чтобы создать вложенные массивы в MXML, включите в массив дополнительные теги `Array`:

```
<mx:Array>
    <mx:Array>
       <mx:String>Flex</mx:String>
        <mx:String>Flash</mx:String>
    <mx:Array>
</mx:Array>
```

Чтобы создать объект в MXML, используйте тег `<mx:Object>`; свойства объекта с их значениями перечисляются в атрибутах тега. Пример:

```
<mx:Object id="person" firstName="John" lastName="Smith" age="50"
    socialSecurity="123-45-6789"/>
```

При создании объектов в MXML действует ограничение: не допускается создание вложенных объектов. Однако при создании объектов в сценарных тегах можно создать объект, содержащий сложные вложенные объекты. Чтобы создать объект, создайте переменную типа `Object`, вызовите конструктор, а затем присоедините к ней свойства:

```
var object:Object = new Object();
var otherObject:Object = new Object();
object.other = otherObject;
```

Также существует упрощенный синтаксис создания объектов: список свойств каждого объекта заключается в фигурные скобки. Пример:

```
var person:Object = {name:"John Smith", age:22, position:{department:
    "Accounting", salary:50000, title:"Junior Accountant"}, id:303};
```

Свойство `position` объекта `Person` ссылается на другой объект, обладающий собственным набором свойств. Обратите внимание: вам даже не нужно объявлять имя переменной для объекта, на который ссылается свойство `position`.

Чтобы создать массив в ActionScript, создайте переменную и вызовите для нее конструктор `Array`:

```
var arr:Array = new Array("red", "blue", "white", "black", "green",
    "yellow");
```

Также можно создать массив без вызова конструктора – объекты, сохраняемые в элементах массива, перечисляются в квадратных скобках:

```
var noConstructorArray:Array = [2, 4, 6, 8, 10, 12, 14, 16];
```

Результат будет таким же, как при вызове конструктора `Array` с передачей ему всех перечисленных объектов.

1.9. Ограничение доступа к переменным в ActionScript

Задача

Требуется сделать одни переменные общедоступными, а другие оградить от внешнего доступа.

Решение

Используйте модификаторы доступа ActionScript для переменных и методов.

Обсуждение

Переменные в файлах ActionScript и MXML обладают разными уровнями доступа. *Приватные* переменные и функции (`private`) доступны только в пределах самого компонента; никакие другие компоненты к ним обратиться не могут. Этот уровень доступа хорошо подходит для переменных и методов, используемых исключительно во внутренних операциях компонента; содержащиеся в них данные не должны изменяться за пределами класса. При проектировании сложных классов рекомендуется назначать всем свойствам, недоступным напрямую для внешних компонентов, приватный уровень доступа. *Открытые* (`public`) переменные доступны для всех объектов, располагающих ссылкой на объект класса, в котором определяется свойство. Проектировщик должен тщательно продумать, какие переменные могут понадобиться во внешних классах, и установить для них минимальный необходимый уровень доступа; это улучшает структуру классов и часто помогает программисту описать потребности некоторой части приложения. Приватные свойства недоступны даже в производных классах – только в том классе или компоненте, в котором они определяются. Наконец, *защищенные* (`protected`) переменные доступны только для любых объектов класса, производного от того, в котором свойство определяется, но не для внешних объектов.

Пример класса с двумя свойствами, защищенным и приватным:

```
package oreilly.cookbook
{
    public class Transport
    {
        protected var info:Object;
        private var speed:Object;
    }
}
```

Ключевое слово `extends` позволяет включить свойства и методы одного класса в другой класс. В следующем примере класс `Car` расширяет класс `Transport`, т. е. наследует все его свойства и методы (кроме приватных).

Однако приватные свойства класса Transport недоступны для Car, а при попытке обратиться к ним выдается ошибка с сообщением о том, что свойство не найдено.

```
package oreilly.cookbook
{
    public class Car extends Transport
    {
        public function set data(value:Object):void
        {
            info = value;
            speed = value.speed; /* ошибка - доступ к приватной переменной
                speed из других классов запрещен */
        }
    }
}
```

Защищенные свойства класса Transport доступны для класса Car, но не для других классов, содержащих экземпляр одного из этих классов в значениях свойств.

С другой стороны, все открытые свойства обоих классов доступны для любого класса, содержащего их экземпляры. Ключевое слово static означает, что обращение к свойству возможно без создания объекта этого класса. Другими словами, ключевое слово static определяет переменную, общую для всех экземпляров класса. Включите следующую строку в определение класса Car:

```
public static const NUMBER_OF_WHEELS:int = 4;
```

Теперь к свойству NUMBER_OF_WHEELS можно обращаться по ссылке на класс без создания экземпляра этого класса, как в следующем примере:

```
import oreilly.cookbook.Car;
public class CompositionTest
{
    private var car:Car;
    public function CompositionTest()
    {
        trace(" a Car object has "+Car.NUMBER_OF_WHEELS+" wheels");
    }
}
```

До настоящего момента речь шла об уровнях доступа переменных, связанных с классом. Однако переменные не обязаны быть свойствами класса; они могут создаваться и уничтожаться внутри функций. Такие переменные имеют смысл только в теле функции. Пример:

```
private function processSpeedData():void
{
    var speed:int;
    var measurement:String = "kilometers";
}
```

Вне тела этой функции переменные `speed` и `measurement` не имеют смысла. В ActionScript используется механизм *распространения видимости переменных*; переменная, определенная в любой точке внутри функции, видна в пределах всей функции. В отличие от языков с блочным ограничением видимости, ссылка на объект `newCar` в следующем фрагменте кода не приводит к ошибке:

```
private function makeCars():void
{
    for(var i:int = 0; i<10; i++)
    {
        var newCar:Car = new Car();
        carArray.push(newCar);
    }
    trace(newCar);
}
```

Ссылка на `newCar` становится недействительной только в результате ее выхода из области видимости после возврата из функции.

ActionScript также определяет модификатор `final`, который сообщает компилятору, что метод не может переопределяться производными классами. При помощи этого модификатора можно определять методы, которые не являются приватными, но не могут изменяться или переопределяться. Например, определение следующего метода в классе `Transport`

```
public final function drive(speed:Number):void{ /* Метод с модификатором
final не может переопределяться другими классами*/ }
```

сообщает, что любой класс, расширяющий `Transport`, будет обладать именно этой версией метода. Уровень доступа `public` означает, что все свойства будут доступны для любого объекта, обладающего доступом к этому объекту `Transport`, но субклассы не смогут переопределить этот метод.

1.10. Создание компонента в ActionScript

Задача

Требуется создать компонент в ActionScript без использования MXML.

Решение

Создайте файл ActionScript и воспользуйтесь наследованием от библиотечного компонента Flex.

Обсуждение

Компоненты могут создаваться не только в MXML, но и в ActionScript; в последнем случае код MXML вообще не задействован. Конечно, в про-

цедуру создания вносятся кое-какие изменения. Прежде всего необходимо позаботиться о правильной «упаковке» класса относительно главного файла приложения. В следующем примере компонент хранится в папке уровня приложения oreilly/cookbook; это обстоятельство отражено в имени пакета:

```
package oreilly.cookbook
{
```

Следующее различие состоит в том, что все классы, упоминаемые в компоненте, должны импортироваться по полному имени пакета. В частности, это относится и к классу, расширяемому компонентом, – в данном примере `mx.containers.Canvas`:

```
import mx.containers.Canvas;
import mx.controls.Text;
import mx.controls.Image;
import oreilly.cookbook.Person;
public class PersonRenderer extends Canvas
{
```

Все константы и переменные обычно перечисляются после объявления класса. В следующем примере перечислены приватные свойства класса. Эти свойства доступны для данного компонента, но скрыты от всех остальных компонентов. Для их чтения и записи определяются специальные методы `get` и `set`; если обращение к свойствам должно сопровождаться какими-либо дополнительными действиями (скажем, проверкой корректности присваиваемого значения), такие действия программируются в этих методах. Использование методов `get/set` – стандартный способ обращения к приватным переменным в функциях

```
private var _data:Object;
private var nameText:Text;
private var ageText:Text;
private var positionText:Text;
private var image:Image;
```

В ActionScript конструктор всегда является открытой функцией, не имеет возвращаемого значения, а его имя совпадает с именем самого класса. Пример:

```
public function PersonRenderer ()
{
    super();
```

Для каждого компонента, включаемого в создаваемый компонент, также необходимо вызвать конструктор, а затем передать созданный объект методу `addChild`. Последнее необходимо для того, чтобы дочерний компонент был включен в список вывода, а его свойства могли изменяться создаваемым компонентом:

```
    nameText = new Text();
    addChild(nameText);
```

```
        ageText = new Text();
        addChild(ageText);
```

В следующем примере компонент `ageText` типа `Text` позиционируется вручную; это необходимо из-за того, что его родительский компонент `PersonRenderer` относится к типу `Canvas` и не имеет собственных средств управления раскладкой (в отличие от компонентов `VBox` или `HBox`):

```
        ageText.y = 20;
        positionText = new Text();
        addChild(positionText);
        positionText.y = 40;
        image = new Image();
        addChild(image);
        image.y = 60;
    }
```

Если в компоненте уже определен метод инициализации, как в `mx.containers.Canvas` из нашего примера, то для выполнения компонентом каких-либо нестандартных действий этот метод необходимо переопределить. Ключевое слово `override` сообщает компилятору о переопределении метода суперкласса. Пример:

```
override public function set data(value:Object):void
{
    _data = value;
    nameText.text = value.name;
    ageText.text = String(value.age);
    positionText.text = value.position;
    image.source = value.image;
}
override public function get data():Object
{
    return _data;
}
```

Последний метод компонента определяется в классе и имеет открытый уровень доступа:

```
        public function retrievePerson():Person
        {
            /* Обработка данных */
            return null;
        }
    }
}
```

Включение класса в компонент может осуществляться в коде ActionScript:

```
var renderer:PersonRenderer = new PersonRenderer();
addChild(renderer);
```

или в коде MXML:

```
<renderers:PersonRenderer id="renderer"/>
```

В ActionScript используется явный вызов конструктора, а в версии MXML он вызывается при вызове конструктора компонента, в который вложен объект `PersonRenderer`.

1.11. Каскадная передача события

Задача

Требуется прослушивать события, передаваемые от дочерних компонентов родительским компонентам, без создания длинной цепочки слушателей.

Решение

Воспользуйтесь механизмом каскадной передачи событий для прослушивания событий, передаваемых от дочерних компонентов.

Обсуждение

Чтобы понять суть каскадной передачи событий, необходимо рассмотреть пример с несколькими классами. Каскадная передача возможна для нескольких типов событий: событий нажатия кнопок мыши, событий щелчков и событий клавиатуры. Когда пользователь щелкает на любом компоненте, это событие передается вверх по иерархии. Таким образом, родитель компонента может прослушивать события щелчка от дочернего компонента. Чтобы прослушивать все события некоторого типа от дочернего компонента, родитель просто добавляет к этому дочернему компоненту слушателя каскадного события.

Возьмем следующее определение класса из файла BubblingComponent.mxml:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="200">
    <mx:Script>
        <![CDATA[
            private function sendClick():void
            {
                trace(" BubblingComponent:: click ");
            }
        ]]>
    </mx:Script>
    <mx:Button click="sendClick()"/>
</mx:HBox>
```

Компонент содержит кнопку, которая передает событие вверх по списку отображения до любого компонента, содержащего экземпляр `BubblingComponent`. Чтобы прослушивать это событие, воспользуйтесь обработчиком click в компоненте, содержащем `BubblingComponent`:

```
<cookbook:BubblingComponent click="handleClick()" id="bubbler"/>
```

Определение компонента BubblingHolder, содержащего BubblingComponent, может выглядеть примерно так:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300"
xmlns:cookbook="oreilly.cookbook.*"
  creationComplete="complete()">
    <mx:Script>
        <![CDATA[
            private function handleClick():void
            {
                trace(" BubblingComponentHolder:: click ");
            }
        ]]>
    </mx:Script>
    <cookbook:BubblingComponent click="handleClick()" id="bubbler"/>
</mx:Canvas>
```

Компонент будет передавать событие любому прослушивающему компоненту, даже на уровне приложения. При включении BubblingHolder в главный файл приложения:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[
            public function createName():void
            {
                name = "Flex Cookbook";
            }
        ]]>
    </mx:Script>
    <cookbook:BubblingComponentHolder click="handleClick()"/>
</mx:Application>
```

событие click из файла BubblingComponent.mxml будет передаваться наверх вплоть до уровня приложения.

В серии событий MouseEvent информация о типе и позиции щелчка передается по списку отображения через все дочерние компоненты до того, который примет событие, а затем обратно вниз по списку вплоть до Stage.

Stage обнаруживает MouseEvent и передает его вниз по списку отображения, пока не будет найден приемник события (т. е. последний компонент, с которым пользователь взаимодействовал при помощи мыши) – так называемая *фаза захвата*. Далее срабатывают обработчики события в приемнике события. Это называется *фазой приема*: у события появляется конкретный приемник. В завершение наступает *фаза каскадной передачи*: событие снова передается вверх по списку отображения всем слушателям, заинтересованным в его получении, вплоть до Stage.

1.12. Использование модели отделенного кода

Задача

Требуется использовать модель отделенного кода (code-behind model) для разделения ActionScript от MXML.

Решение

Создайте в коде ActionScript компонент, который расширяет библиотечный класс Flex, предоставляющий всю необходимую функциональность; добавьте в него все необходимые свойства и методы. Создайте файл MXML, расширяющий созданный вами класс.

Обсуждение

Если вы знакомы с программированием ASP.NET, вам наверняка доводилось слышать термин «отделенный код», как, впрочем, и любому разработчику, знакомому с концепцией отделения контроллера от представления в любом приложении, смешивающем разметку с другим языком (Ruby on Rails, JSP (Java Server Pages), PHP и т. д.) Стратегия отделения элементов макета от управляющего ими программного кода способствует логической изоляции разных аспектов представления для улучшения удобочитаемости и четкости. Впрочем, в некоторых случаях увеличение количества файлов, необходимых для реализации этой схемы в приложении, затрудняет навигацию в проекте, так как для каждого компонента создаются два файла. Более того, отделение бизнес-логики от логики представления нередко создает проблемы и может привести к весьма запутанному разделению кода внутри компонента. И все же многие разработчики предпочитают эту схему за то, что она помогает прояснить логику приложения и его внутреннее устройство.

Начнем с рассмотрения компонента, который расширяет класс (в данном примере `mx.containers.Canvas`) и содержит методы прослушивания компонентов, добавляемых на сцену (stage), а также метод, который может обработать любое событие, но специализируется на обработке щелчков на кнопках.

```
package oreilly.cookbook
{
    import mx.containers.Canvas;
    import flash.events.Event;

    public class CodeBehindComponent extends Canvas
    {
        public function CodeBehindComponent()
        {
            super();
            addEventListener(Event.ADDED_TO_STAGE,
                addedToStageListener);
        }
```

```
        protected function addedToStageListener(event:Event):void
        {
            trace(" Added to Stage from Code Behind ");
        }
        protected function clickHandler(event:Event):void
        {
            trace(" Click handled from component "+event.target);
        }
    }
}
```

В этом примере методам, которые обычно помечаются как приватные, присваивается защищенный уровень доступа. Это объясняется тем, что кодовая часть модели отделенного кода, MXML, будет использовать наследование от класса `CodeBehindComponent`, и для нормального функционирования ей понадобится доступ к этим методам. Компонент MXML для этого компонента выглядит так:

```
<cookbook:CodeBehindComponent xmlns:mx="http://www.adobe.com/2006/mxml"
    width="200" height="400" xmlns:cookbook="oreilly.cookbook.*">
    <mx:Button click="clickHandler(event)"/>
</cookbook:CodeBehindComponent>
```

1.13. Включение привязки для свойств компонента

Задача

Вы создаете компонент и хотите разрешить привязку для свойств этого компонента, чтобы они могли связываться с другими компонентами.

Решение

Создайте методы `get/set` и пометьте их тегом метаданных `Bindable` с указанием имени события, которое будет передаваться методами при задании свойства.

Обсуждение

Любой объект может определять привязываемые (bindable) свойства; для этого изменение свойства сопровождается передачей события, а свойство помечается тегом метаданных `Bindable`. Пример:

```
package oreilly.cookbook
{
    import flash.events.EventDispatcher;
    import flash.events.Event;
    public class Person extends EventDispatcher
    {
        public static var NAME_CHANGE:String = "nameChange";
        private var _name:String;
```

```
        [Bindable(event=NAME_CHANGE)]
        public function get name():String
        {
            return _name;
        }
        public function set name(value:String):void
        {
            dispatchEvent(new Event(NAME_CHANGE));
            _name = value;
        }
    }
}
```

В теге `Bindable` должно указываться имя события, передаваемого при задании свойства `name`. Это гарантирует, что любой компонент, связавший свойство со свойством `name` объекта `Person`, будет оповещен об изменении значения.

После того как в объекте `Person` будет создано привязываемое свойство, все экземпляры `Person` могут использоваться в выражениях привязки:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[

        [Bindable]
        private var _person:Person;
        public function set person(person:Person:void
        {
            _person = person;
        }

        ]]>
    </mx:Script>
    <mx:Label text="{_person.name}"/>
</mx:Canvas>
```

1.14. Пользовательские события и передача данных с событиями

Задача

Требуется передать данные вместе с событием, представленным пользовательским классом события.

Решение

Создайте класс события, расширяющий `flash.events.Event`. Создайте свойство для хранения данных, которые должны передаваться вместе с событием.

Обсуждение

В некоторых ситуациях событие должно сопровождаться объектом данных, чтобы слушатели могли работать с этими данными без обращения к объекту, отправившему событие. Например, для объектов с большим уровнем вложенности, отправляющих события вверх через несколько промежуточных компонентов, часто бывает желательно передать данные, не требуя от компонента-слушателя, чтобы он искал объект-источник и обращался к его свойству. Проблема решается определением пользовательского типа события и включением всех данных, передаваемых вместе с событием, при вызове его конструктора. Не забудьте вызвать метод суперкласса `Event` для правильного создания экземпляра `Event`. Пример:

```
package oreilly.cookbook
{
    import flash.events.Event;

    public class CustomPersonEvent extends Event
    {
        public var person:Person;
        public var timeChanged:String;
        public function CustomPersonEvent(type:String,
             bubbles:Boolean=false, cancelable:Boolean=false,
             personValue:Person=null, timeValue:String="")
        {
            super(type, bubbles, cancelable);
            person = personValue;
            timeChanged = timeValue;
        }

        override public function clone():Event
        {
            return new CustomPersonEvent(type, bubbles, cancelable,
                personValue, timeValue);
        }
    }
}
```

В этом пользовательском классе события также переопределяется унаследованный метод `Event.clone`, обеспечивающий корректное создание дубликатов `CustomPersonEvent`. Если слушатель события попытается передать дальше пользовательское событие:

```
private function customPersonHandler(event:CustomPersonEvent):void {
    dispatchEvent(event);
}
```

то передаваться будет не полученный ранее объект события, а копия `CustomPersonEvent`, созданная методом `clone`. Копирование осуществляется внутри класса `flash.events.EventDispatcher`. Без переопределения метода `clone`, обеспечивающего перенос всех свойств `CustomPersonEvent`

в дубликат, возвращаемое событие будет относиться к типу `flash.events.Event` и не будет содержать новых свойств `CustomPersonEvent`.

1.15. Прослушивание событий клавиатуры

Задача

Требуется обнаружить нажатие клавиши пользователем, определить, какая клавиша была нажата, и обработать это событие должным образом.

Решение

Включите слушателя события `keyDown` в компонент или сцену приложения. Проверьте значение свойства `keyCode` класса `KeyboardEvent`.

Обсуждение

Чтобы прослушивать события `KeyboardEvent`, воспользуйтесь обработчиком события `keyDown`, который имеется у всех классов, расширяющих `UIComponent`. Класс `KeyboardEvent` определяет свойство `keyCode`, в котором содержится код нажатой клавиши. Пример:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
keyDown="keyHandler(event)" backgroundColor="#0000ff">
    <mx:Script>
        <![CDATA[

            import flash.events.KeyboardEvent;

            private function keyHandler(event:KeyboardEvent):void
            {
                switch(event.keyCode)
                {
                    case 13:
                        trace(" Enter pressed ");
                    break;
                    case 32:
                        trace(" Space Bar pressed ");
                    break;
                    case 16:
                        trace(" Shift Key pressed ");
                    break;
                    case 112:
                        trace(" F1 pressed ");
                    break;
                    case 8:
                        trace(" Delete pressed ");
                    break;
                }
            }
```

```
        ]]>
    </mx:Script>
    <mx:Button label="One"/>
</mx:HBox>
```

Небольшое замечание по поводу этого компонента: он будет прослушивать только те события, которые происходят во время наличия у кнопки фокуса ввода. Если удалить кнопку с компонента, функция `keyHandler` вызываться не будет. Чтобы перехватывать все события `KeyEvent`, происходящие в приложении (независимо от того, обладает компонент фокусом ввода или нет), включите следующую строку в открывающий тег компонента:

```
addedToStage="stage.addEventListener(KeyboardEvent.KEY_DOWN, keyHandler)"
```

Она гарантирует, что метод `keyHandler` будет обрабатывать события `KeyEvent`, перехватываемые сценой, т. е. все.

1.16. Определение необязательных параметров методов

Задача

Требуется определить метод с параметром, имеющим значение по умолчанию, вследствие чего метод может вызываться и без указания этого параметра.

Решение

Укажите значение по умолчанию в объявлении метода.

Обсуждение

Чтобы определить метод с необязательным параметром (или несколькими необязательными параметрами), просто задайте значение по умолчанию или `null` в сигнатуре метода. Однако следует учитывать, что примитивы ActionScript `String`, `Number`, `int` и `Boolean` не могут принимать псевдозначение `null`; им необходимо задать значение по умолчанию. Пример:

```
public function optionalArgumentFunction(value:Object, string:String,
count:int = 0, otherValue:Object = null):void
{
    if(count != 0)
    {
        /* Если значение count отлично от значения по умолчанию,
           обработать значение, переданное при вызове */
    }
    if(otherValue != null)
    {
```

```
        /* Если значение otherValue отлично от null,
           обработать значение, переданное при вызове */
    }
}
```

Другой способ определения методов с неопределенным количеством аргументов основан на использовании маркера `...` перед именем переменной. В этом случае в переменную заносится массив аргументов, элементы которого перебираются и обрабатываются в программе.

```
public function unlimitedArgumentsFunction(...arguments):void
{
    for each(var arg:Object in arguments)
    {
        /* Обработка каждого аргумента */
    }
}
```

1.17. Проверка типа объекта

Задача

Требуется проверить фактический тип объекта, переданного методу.

Решение

Воспользуйтесь оператором `is` для проверки типа объекта или его суперкласса.

Обсуждение

Для проверки типа объекта в ActionScript предусмотрен оператор `is`, который проверяет тип объекта и возвращает логический результат. Если проверяемый объект относится к указанному типу или если объект расширяет указанный тип, оператор `is` возвращает `true`. Например, поскольку объект `Canvas` расширяет `UIComponent`, оператор `is` вернет `true` при проверке объекта `Canvas` по типу `UIComponent`. С другой стороны, при проверке `UIComponent` по типу `Canvas` оператор вернет `false`, т. к. `UIComponent` не является производным от `Canvas`. При выполнении фрагмента

```
public function TypeTest()
{
    var uiComponent:UIComponent = new UIComponent();
    var canvas:Canvas = new Canvas();
    trace(" uiComponent is UIComponent " + (uiComponent is UIComponent));
    trace(" uiComponent is Canvas " + (uiComponent is Canvas));
    trace(" canvas is UIComponent " + (canvas is UIComponent));
}
```

будет получен следующий результат:

```
uiComponent is UIComponent true
```

```
uiComponent is Canvas false
canvas is UIComponent true
```

Проверка типа часто используется для определения компонента, инициировавшего событие. Это позволяет написать один метод для обработки события, в котором дальнейшие действия определяются в зависимости от проверки типа объекта.

```
private function eventListener(mouseEvent:MouseEvent):void
{
    if(mouseEvent.target Button)
    {
        /* Действия для Button */
    }
    else if(mouseEvent.target is ComboBox)
    {
        /* Действия для ComboBox */
    }
    else
    {
        /* Действия для всех остальных случаев */
    }
}
```

1.18. Определение и реализация интерфейса

Задача

Требуется определить интерфейс, а затем создать компонент, реализующий этот интерфейс.

Решение

Создайте файл ActionScript, объявите его как интерфейс и укажите все методы, обязательные для данного интерфейса. Чтобы реализовать интерфейс, включите ключевое слово `implements` в объявление класса компонента, использующего интерфейс.

Обсуждение

Интерфейсы – мощное средство для описания контрактов, которым должен соответствовать объект. Интерфейс содержит заранее заданный набор методов с определенным уровнем доступа, именем, параметрами и типом возвращаемого значения. Компоненты, использующие объект, в свою очередь рассчитывают на присутствие всех методов из набора. Это позволяет создавать упрощенные описания классов без фактического создания нового класса, загромождающего иерархию наследования. Классы, реализующие интерфейс, считаются относящимися к типу интерфейса; это позволяет использовать интерфейсы для объявления

типов параметров методов или типа возвращаемого значения, как в следующем примере:

```
public function pay(payment:IPaymentType):IReceipt
```

Этот метод получает любой объект, реализующий интерфейс `IPaymentType`, и возвращает объект, реализующий интерфейс `IReceipt`.

Интерфейс не может определять ни тело метода, ни какие-либо переменные. В следующем фрагменте объявляется интерфейс `IDataInterface` с пятью методами, которые должны определяться любым объектом, реализующим этот интерфейс:

```
package oreilly.cookbook
{
    public interface IDataInterface
    {
        function set dataType(value:Object):void;
        function get dataType():Object;
        function update():Boolean;
        function write():Boolean;
        function readData():Object;
    }
}
```

Чтобы реализовать интерфейс, объявите класс и включите в объявление маркер `implements`. Класс должен реализовать все методы, объявленные в интерфейсе. В следующем фрагменте каждому методу приведенного ранее интерфейса назначается тело функции:

```
package oreilly.cookbook
{
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;

    public class ClientData extends EventDispatcher
        implements IDataInterface
    {

        private var _dataType:Object;

        public function ClientData(target:IEventDispatcher=null)
        {
            super(target);
        }
        public function set dataType(value:Object):void
        {
            _dataType = value;
        }
        public function get dataType():Object
        {
            return _dataType;
        }
```

```
        public function update():Boolean
        {
            // Выполнение обновления
            var updateSuccessful:Boolean;
            if(updateSuccessful)
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        public function write():Boolean
        {
            var writeSuccess:Boolean;
            if(writeSuccess)
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        public function readData():Object
        {
            var data:Object;
            // Получение всех необходимых данных
            return data;
        }
    }
}
```

Чтобы реализовать интерфейс в коде MXML, добавьте ключевое слово `implements` **в тег верхнего уровня компонента. Пример:**

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
implements= "IDataInterface">
```

2

Меню и компоненты

Пакет разработчика Flex 3 (SDK) включает обширную библиотеку готовых компонентов пользовательского интерфейса (UI), заметно упрощающих разработку приложений. Поведение этих компонентов легко настраивается на уровне свойств в ActionScript или MXML, а для изменения их оформления используются каскадные таблицы стилей (CSS). Кроме того, поскольку ActionScript 3 является полноценным языком объектно-ориентированного программирования (ООП), стандартные компоненты можно расширять для реализации нестандартной функциональности; при этом используются классические конструкции ООП.

Элементы управления Flex 3 SDK находятся в пакете `mx.controls`. На момент написания книги пакет `mx.controls` содержал более 50 компонентов, используемых в качестве структурных блоков для построения полнофункциональных пользовательских интерфейсов.

2.1. Прослушивание события щелчка на кнопке

Задача

Требуется выполнить фрагмент кода в ответ на действие пользователя, например, вывести на консоль список имен, когда пользователь щелкнет на кнопке.

Решение

Включите в тег `<mx:Button>` атрибут события `click`, чтобы назначить обработчик события в MXML. В ActionScript для этой цели используется метод `addEventListener` экземпляра кнопки.

Обсуждение

Следующий фрагмент демонстрирует прослушивание щелчков на кнопке в коде MXML, с назначением обработчика в атрибуте `click` тега `<mx_Button>`:

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

    <mx:Button id="btn" label="Show Names" click="showNames(event)"/>

    <mx:Script>
        <![CDATA[

            private function showNames(evt:MouseEvent):void
            {
                var temp:Array = new Array("George","Tim","Alex","Dean");
                trace(temp.toString());
            }
        ]]>
    </mx:Script>

</mx:Application>
```

Код создает приложение с экземпляром кнопки `btn`. Чтобы при щелчке на экземпляре `btn` приложение выводило на консоль список имен, атрибут события `click` в экземпляре `btn` связывается с методом `showNames`:

```
<mx:Button id="btn" label="Show Names" click="showNames(event)"/>
```

Каждый раз, когда пользователь щелкает на кнопке, Flex Framework передает событие типа `MouseEvent.CLICK`. Приведенная строка кода указывает, что при каждой передаче кнопкой события щелчка должен вызываться метод `showNames`. В теле метода `showNames` создается и выводится на консоль список имен. Обратите внимание: объект события типа `MouseEvent` автоматически передается функции-обработчику. В зависимости от вида передаваемого события из этого объекта можно получить дополнительную информацию о самом событии. Запустите приложение в отладочном режиме (F11 в Eclipse); в окне консоли будет выведен следующий текст:

```
George,Tim,Alex,Dean
```

Слушатели событий также могут назначаться в коде ActionScript. В следующем примере слушатели `showNames` и `showTitles` связываются с экземпляром `btn` в ActionScript:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initApp(event);">
```

```
    <mx:Button id="btn" label="Show Names"/>

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

            private function initApp(evt:FlexEvent):void
            {
                btn.addEventListener(MouseEvent.CLICK,showNames);
                btn.addEventListener(MouseEvent.CLICK,showTitles);
            }

            private function showNames(evt:MouseEvent):void
            {
                var temp:Array = new Array("George","Tim","Alex","Dean");
                trace(temp.toString());
            }

            private function showTitles(evt:MouseEvent):void
            {
                var temp:Array = new Array("Director",
                "Vice-President","President","CEO");
                trace(temp.toString());
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Событие щелчка на кнопке связывается со слушателями showNames **и** showTitles **в обработчике события** creationComplete **уровня приложения:**

```
private function initApp(evt:FlexEvent):void
{
    btn.addEventListener(MouseEvent.CLICK,showNames);
    btn.addEventListener(MouseEvent.CLICK,showTitles);
}
```

Если снова запустить приложение в отладочном режиме, в окне консоли выводится следующий текст:

```
George,Tim,Alex,Dean
Director,Vice-President,President,CEO
```

Слушатели вызываются в порядке их регистрации. Так как слушатель showNames **был зарегистрирован раньше** showTitles**, список имен выводится перед списком должностей. Чтобы изменить порядок выполнения, либо измените порядок регистрации слушателей, либо (рекомендуемый способ) задайте приоритет при регистрации слушателей, как сделано в следующем примере:**

```
private function initApp(evt:FlexEvent):void
{
    btn.addEventListener(MouseEvent.CLICK,showNames,false,0);
```

```
        btn.addEventListener(MouseEvent.CLICK, showTitles, false, 1);
    }
```

На этот раз результат запуска приложения в отладочном режиме будет выглядеть так:

```
Director,Vice-President,President,CEO
George,Tim,Alex,Dean
```

Слушатели, которым при регистрации был назначен более высокий приоритет, будут вызваны до слушателей с низким приоритетом. Если несколько слушателей имеют одинаковые приоритеты, порядок выполнения определяется порядком регистрации.

2.2. Создание группы кнопок-переключателей

Задача

Требуется предоставить пользователю группу кнопок для выбора вариантов.

Решение

Создайте группу кнопок при помощи компонента `ToggleButtonBar` и массива `ArrayCollection`.

Обсуждение

Чтобы создать группу кнопок-переключателей, создайте приложение с экземпляром компонента `ToggleButtonBar`. Этот компонент определяет горизонтальную или вертикальную группу кнопок, сохраняющих свое состояние (кнопка выбрана или не выбрана). Пример возможной реализации:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    initialize="initApp(event)">

    <mx:ToggleButtonBar id="toggle"
        dataProvider="{dataProvider}"
        itemClick="setMode(event)"/>

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.events.FlexEvent;
            import mx.events.ItemClickEvent;

            [Bindable]
            private var dataProvider:ArrayCollection;
```

```
            private function initApp(evt:FlexEvent):void
            {
                var temp:Array = new Array(
                                    {label:"Show Labels",mode:"labels"},
                                    {label:"Show Titles",mode:"titles"});
                dataProvider = new ArrayCollection(temp);
            }

            private function setMode(evt:ItemClickEvent):void
            {
                switch (evt.item.mode) {
                    case "labels":
                        trace("George, Tim, Dean");
                        break;
                    case "titles":
                        trace("Vice President, President, Director");
                        break;
                    default:
                        break;
                }
            }
        ]]>
    </mx:Script>
</mx:Application>
```

В ходе инициализации приложения событие `initialize` **вызывает метод** `initApp`**. Метод** `initApp` **заполняет** `ArrayCollection` `DataProvider` **исходными данными** `ToggleButtonBar`**. Так как свойство** `dataProvider` **экземпляра** `ToggleButtonBar` **привязано к** `ArrayCollection`**, изображение обновляется для отображения новых кнопок.**

По умолчанию свойство `label` **компонентов** `ArrayCollection` **отображается в виде надписи на соответствующей кнопке экземпляра** `ToggleButtonBar`**. Чтобы в надписи выводилось значение другого свойства (например,** `mode`**), воспользуйтесь свойством** `labelField` **класса** `ToggleButtonBar`**:**

```
<mx:ToggleButtonBar id="toggle"
    dataProvider="{dataProvider}" labelField="mode"
    itemClick="setMode(event)"/>
```

Событие `itemClick` **экземпляра** `ToggleButtonBar` **связывается с методом** `setMode`**. Обратите внимание на передачу объекта события** `ItemClickEvent` **методу** `setMode`**. Свойство** `item` **экземпляра** `ItemClickEvent` **содержит ссылку на соответствующий компонент** `ArrayCollection` `dataProvider`**.**

Одна из стандартных задач при использовании `ToggleButtonBar` **– снятие выделения со всех кнопок при запуске приложения, до того момента, когда пользователь явно выберет одну из кнопок. По умолчанию при задании** `dataProvider` **выбирается первая кнопка. К счастью, Flex 3 предоставляет доступ к исходному коду SDK, и вы можете воспользоваться полученной информацией для расширения** `ToggleButtonBar`

в соответствии со своими потребностями. Исходный код ToggleButtonBar находится в файле

```
<Установочный каталог Flex 3 >/sdks/3.0.0/frameworks/
projects/framework/src/mx/controls/ToggleButtonBar.as
```

Метод highlightSelectedNavItem подсказывает, как снять выделение с кнопок. Он получает ссылку на текущую выделенную кнопку и задает ее свойству selected значение false:

```
child = Button(getChildAt(selectedIndex));
child.selected = false;
```

На основании информации, полученной из кода Framework, создается собственная версия ToggleButtonBar, отвечающая вашим потребностям – в данном случае автоматически переводящая все кнопки в «отжатое» состояние при запуске.

Далее приводится код класса CustomToggleButtonBar, который расширяет ToggleButtonBar функциональностью, необходимой для снятия выбора кнопок при каждом изменении dataProvider. Обратите внимание на переопределение set-метода dataProvider и использование флага dataReset для отслеживания изменений свойства dataProvider. Метод updateDisplayList компонента переопределяется так, чтобы при каждом сбросе dataProvider с текущей кнопки снималось выделение. После обновления изображения флаг dataReset возвращается в состояние по умолчанию.

```
package {
    import mx.controls.Button;
    import mx.controls.ToggleButtonBar;

    public class CustomToggleButtonBar extends ToggleButtonBar
    {
        public function CustomToggleButtonBar()
        {
            super();
        }

        private var dataReset:Boolean = false;
        override public function set dataProvider(value:Object):void {
            super.dataProvider = value;
            this.dataReset = true;
        }

        override protected function updateDisplayList(unscaledWidth:Number,
            unscaledHeight:Number):void
        {
            super.updateDisplayList(unscaledWidth,unscaledHeight);

            if(this.dataReset) {
                if(selectedIndex != -1) {
                    var child:Button;
                    child = Button(getChildAt(selectedIndex));
```

```
                    if(child)
                    {
                        child.selected = false;
                        this.dataReset = false;
                    }
                }
            }
        }
    }
```

Чтобы воспользоваться новым компонентом, просто замените `ToggleButtonBar` на `CustomToggleButtonBar` в приложении:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:local="*"
    layout="vertical"
    initialize="initApp(event)">

    <local:CustomToggleButtonBar id="toggle" selectedIndex="-1"
        dataProvider="{dataProvider}"
        itemClick="setMode(event)"/>
```

Как видите, приспособить стандартные компоненты к специфическим особенностям вашего приложения относительно несложно. Просмотр исходного кода Flex SDK не только принесет бесценную информацию о внутреннем строении Framework, но и откроет много интересных возможностей для расширения стандартного набора компонентов. Впрочем, будьте осторожны при использовании недокументированных возможностей или свойств/методов пространства `mx_internal`, потому что они с большой вероятностью изменятся в будущих версиях SDK.

2.3. Использование ColorPicker для выбора цвета

Задача

Требуется дать пользователю возможность выбрать цвет компонента `Canvas`.

Решение

Воспользуйтесь компонентом `ColorPicker`, предоставляющим цветовую палитру, и событием `change` компонента `ColorPicker` для изменения цвета фона компонента.

Обсуждение

Чтобы предоставить пользователю палитру для выбора цвета, задайте стиль `backgroundColor` компонента `Canvas` при помощи компонента `ColorPicker`. Компонент `ColorPicker` позволяет выбрать цвет из палитры с об-

разцами. Событие change компонента ColorPicker связывается с методом setColor. Метод setColor получает аргумент ColorPickerEvent с информацией о текущем выделенном цвете в палитре компонента ColorPicker. Программный код выглядит примерно так:

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

    <mx:Canvas id="cnv" width="450" height="450"
            backgroundColor="#eeaeaea">
        <mx:ColorPicker id="pckr" right="10" top="10"
            change="setColor(event)"/>
    </mx:Canvas>

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

            private function setColor(evt:ColorPickerEvent):void
            {
                cnv.setStyle("backgroundColor",evt.color); }
        ]]>
    </mx:Script>
</mx:Application>
```

При выборе нового цвета обновляется стиль backgroundColor компонента Canvas. Обратите внимание: так как backgroundColor является стилевым атрибутом, а не свойством компонента Canvas, для обновления стиля используется метод setStyle:

```
private function setColor(evt:ColorPickerEvent):void
{
    cnv.setStyle("backgroundColor",evt.color); }
```

2.4. Загрузка внешнего файла SWF

Задача

Требуется загрузить внешний файл SWF, созданный в Flex 3 или Flash CS3, в текущем приложении Flex во время выполнения.

Решение

Воспользуйтесь компонентом SWFLoader.

Обсуждение

Компонент SWFLoader предназначен для загрузки внешних файлов SWF во время выполнения. В следующем примере внешние файлы SWF загружаются в контейнеры Canvas, являющиеся дочерними компонентами

для TabNavigator. Атрибут source компонента SWFLoader указывает путь к внешним файлам SWF, загружаемым во время выполнения. Sub1.swf – приложение Flex 3; Sub2.swf – файл SWF, созданный в Flash CS3.

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

    <mx:TabNavigator resizeToContent="true"
        paddingTop="0">
        <mx:Canvas>
            <mx:SWFLoader source="assets/Sub1.swf"/>
        </mx:Canvas>
        <mx:Canvas>
            <mx:SWFLoader source="assets/Sub2.swf"/>
        </mx:Canvas>
    </mx:TabNavigator>

</mx:Application>
```

Компонент SWFLoader также способен загружать файлы SWF, встроенные в приложение Flex. Для этой цели используется директива Embed. В следующем примере файл Sub2.swf компилируется в главное приложение:

```
<mx:SWFLoader source="@Embed('assets/Sub2.swf')"/>
```

2.5. Назначение Tab-индексов компонентам

Задача

Требуется изменить принятый по умолчанию порядок перебора (Tab-порядок) компонентов в приложении Flex.

Решение

Используйте свойство tabIndex компонента Flex для задания нестандартного Tab-порядка.

Обсуждение

По умолчанию порядок перебора всех компонентов Flex, задействованных в Tab-переборе, определяется экранным макетом. Чтобы изменить порядок перебора, принятый по умолчанию, и явно задать пользовательский порядок перебора, используйте свойство tabIndex компонентов. В следующем примере свойства tabIndex компонентов TextInput задаются таким образом, чтобы перебор производился слева направо:

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

```
        <mx:VBox>
            <mx:Label text="First Name : "/>
            <mx:TextInput tabIndex="1"/>
            <mx:Label text="Home # : "/>
            <mx:TextInput tabIndex="3"/>
        </mx:VBox>

        <mx:VBox>
            <mx:Label text="Last Name : "/>
            <mx:TextInput tabIndex="2"/>
            <mx:Label text="Work # : "/>
            <mx:TextInput tabIndex="4"
                text="978-111-2345"/>
            <mx:Button label="Submit" tabIndex="5"/>
        </mx:VBox>

    </mx:Application>
```

Если Tab-индексы не указаны, по умолчанию используется порядок, определяемый расположением компонентов на экране (сверху вниз). Свойство `tabIndex` компонентов также может задаваться в коде ActionScript; данная возможность особенно полезна при необходимости управления Tab-порядком из пользовательских компонентов, динамически создающих дочерние компоненты во время выполнения.

2.6. Задание свойства labelFunction

Задача

Надпись, отображаемая в компоненте `ComboBox`, должна формироваться в результате объединения нескольких различных полей.

Решение

Задайте свойству `labelFunction` компонента `ComboBox` ссылку на пользовательскую функцию, которая будет формировать выводимую надпись.

Обсуждение

По умолчанию списковые компоненты Flex берут отображаемое значение из свойства `label` элементов `dataProvider`. Однако в некоторых случаях `dataProvider` может не иметь свойства `label`, или отображаемый текст должен формироваться из нескольких информационных полей `dataProvider`. Свойство `labelFunction` позволяет назначить пользовательский метод, который вызывается для каждого элемента в свойстве `dataProvider` списка и возвращает текст надписи для этого элемента. В следующем примере свойство `labelFunction` компонента `ComboBox` содержит ссылку на функцию `getFullName`, которая объединяет значения полей `fName` и `lName` записи из `dataProvider`, возвращая строку с полным именем:

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

    <mx:ComboBox dataProvider="{myDP}"
     labelFunction="getFullName"/>

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            private var myDP:ArrayCollection =
                new ArrayCollection([{id:1,fName:"Lucky", lName:"Luke"},
                                     {id:2, fName:"Bart", lName:"Simpson"}]);
            private function getFullName(item:Object):String
            {
                return item.fName + " " + item.lName;
            }
        ]]>

    </mx:Script>

</mx:Application>
```

2.7. Получение данных для создания меню

Задача

Требуется создать строку меню по данным, полученным от провайдера данных.

Решение

Задайте объект `Collection` (например, `ArrayCollection` или `XMLListCollection`) свойству `dataProvider` компонента `MenuBar`, используя код MXML.

Обсуждение

Простейший способ заполнения данными компонентов `MenuBar` основан на создании в компоненте экземпляра `XMLList`:

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

    <mx:MenuBar labelField="@label">
        <mx:XMLList>
            <menuitem label="File">
                <menuitem label="New"/>
                <menuitem label="Open"/>
```

```
                <menuitem label="Close" enabled="false"/>
            </menuitem>
            <menuitem label="Edit"/>
            <menuitem label="Source"/>
            <menuitem label="View">
                <menuitem label="50%"
                    type="radio" groupName="one"/>
                <menuitem label="100%"
                    type="radio" groupName="one"
                    selected="true"/>
                <menuitem label="150%"
                    type="radio" groupName="one"/>
            </menuitem>
        </mx:XMLList>
    </mx:MenuBar>

</mx:Application>
```

Так как `dataProvider` является свойством по умолчанию компонента `MenuBar`, объект `XMLList` может быть назначен прямым потомком тега `<mx:MenuBar>`. Узлы верхнего уровня `XMLList` соответствуют кнопкам `MenuBar`, а узлы `menuitem` – иерархии команд меню под кнопками верхнего уровня. Сами узлы могут называться как угодно – скажем, вместо `menuitem` их можно с таким же успехом назвать `subnode`. Однако атрибуты узлов обладают фиксированным смыслом и влияют на отображение меню и взаимодействие с ними:

`enabled`

Определяет, может ли пользовать выбрать эту команду меню.

`groupName`

Применяется для команд меню типа `radio`; задает общее имя для группы команд меню.

`icon`

Задает идентификатор класса графического ресурса.

`label`

Задает отображаемый текст для команды меню. Если `dataProvider` имеет формат ECMAScript for XML (E4X), как в приведенном ранее примере, поле `labelField` компонента `MenuBar` должно задаваться явно, несмотря на существующий атрибут с именем `label`.

`toggled`

Определяет состояние выбора команд меню типов `check` и `radio`.

`type`

Задает один из трех возможных типов команд меню: `check`, `radio` или `separator`.

2.8. Динамическое заполнение меню

Задача

Требуется заполнить строку меню и изменять ее на основе динамических данных.

Решение

Присвойте объект `Collection` (например, `ArrayCollection` или `XMLListCollection`) свойству `dataProvider` компонента `MenuBar`, используя код ActionScript.

Обсуждение

Компонент `MenuBar` в Flex 3 позволяет динамически изменять структуру меню во время выполнения. В следующем примере создается приложение с компонентом `MenuBar`, который заполняется при инициализации приложения с использованием `ArrayCollection`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initApp(event)">

    <mx:MenuBar id="menu" dataProvider="{menu_dp}"/>

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

            [Bindable]
            private var menu_dp:ArrayCollection;

            private function initApp(evt:FlexEvent):void
            {
                var temp:Array = new Array();

                var subNodes:ArrayCollection = new ArrayCollection( [
                            {label:"New"},
                            {label:"Open"},
                            {label:"Close",
                            enabled:false}
                        ]);

                temp.push(
                {label:"File",children:subNodes});
                temp.push({label:"Edit"});
                temp.push({label:"Source"});
                subNodes = new ArrayCollection( [
                           {label:"50%", type:"radio",
```

```
                                groupName:"one"},
                               {label:"100%", type:"radio",
                                groupName:"one",selected:true},
                               {label:"150%", type:"radio",
                                groupName:"one"}
                                                    ]);
                    temp.push({label:"View",children:subNodes});
                    menu_dp = new ArrayCollection(temp);
                }
            ]]>
        </mx:Script>

    </mx:Application>
```

В этом фрагменте переменная `ArrayCollection menu_dp` **привязывается к свойству** `dataProvider` **компонента** `MenuBar`**. В ходе обработки события** `creationComplete` **уровня приложения** `ArrayCollection menu_dp` **инициализируется и заполняется описанием меню. Как и при работе с другими компонентами Flex, управляемыми данными, использование коллекций (**`ArrayCollection`**,** `XMLListCollection` **и т. д.) гарантирует, что любые изменения в данных приведут к соответствующей модификации изображения.**

Классы коллекций предоставляют удобные методы для редактирования, добавления и удаления компонентов меню. В демонстрационных целях в этом примере под `MenuBar` **добавляется простой компонент** `Form` **для редактирования команд меню, определяемых индексами элементов** `ArrayCollection`**:**

```
<mx:Form>
    <mx:FormHeading label="Menu Editor"/>
    <mx:FormItem label="Menu Index">
        <mx:TextInput id="menuIdx" restrict="0-9" text="0" width="20"/>
    </mx:FormItem>
    <mx:FormItem label="Sub-Menu Index">
        <mx:TextInput id="subMenuIdx" restrict="0-9" width="20"/>
    </mx:FormItem>
    <mx:FormItem label="Menu Label">
        <mx:TextInput id="label_ti"/>
    </mx:FormItem>
    <mx:FormItem>
        <mx:Button label="Edit" click="editMenu()"/>
    </mx:FormItem>
</mx:Form>
```

В примере используется простейшая форма с компонентами ввода; чтобы получить доступ к команде меню, следует ввести индекс соответствующего элемента коллекции. Так, если ввести 0 в поле `menuIdx` **без заполнения поля** `subMenuIdx`**, содержимое формы будет относиться к меню File верхнего уровня; 0 в поле** `menuIdx` **и 0 в поле** `subMenuIdx` **обозначают команду New в подменю и т. д.**

Щелчок на кнопке Edit вызывает метод editMenu. В коде этого метода введенные индексы используются для обращения к команде меню и изменения текста ее надписи:

```
private function editMenu():void {
    var itemToEdit:Object;
    try {
        itemToEdit = menu_dp.getItemAt(int(menuIdx.text));
        if(subMenuIdx.text) {
            itemToEdit = itemToEdit.children.getItemAt(
                int(subMenuIdx.text));
        }
        itemToEdit.label = label_ti.text;
        menu_dp.itemUpdated(itemToEdit);
    }
    catch(ex:Error){
        trace("could not retrieve menu item");
    }
}
```

Код editMenu получает информацию о команде меню по содержимому полей menuIdx и subMenuIdx, а затем использует содержимое label_ti для обновления текста. Чтобы изображение меню изменялось должным образом, метод изменяет нижележащее значение dataProvider, связанное с MenuBar, а затем выдает запрос на обновление представления при помощи itemUpdated класса ArrayCollection. Во вложенных структурах данных такого рода завершающий вызов itemUpdated очень важен; без него нижележащие данные изменятся, а изображение будет соответствовать старому значению. Блок try…catch в приведенном примере обеспечивает простейшую проверку границ массива.

2.9. Определение обработчиков событий для компонентов на базе меню

Задача

Требуется обработать взаимодействие пользователя с меню приложения.

Решение

Добавьте обработчики для события itemClick компонента MenuBar.

Обсуждение

Чтобы обработать действие пользователя со строкой меню приложения, свяжите функцию-обработчик handleMenuClick с событием itemClick компонента MenuBar. Каждый раз, когда пользователь выбирает команду меню, компонент передает событие itemClick. Функция-обработчик получает аргумент с объектом MenuEvent. Объект содержит информацию

о команде меню, от которой поступило событие. Свойство item объекта MenuEvent ссылается на элемент dataProvider, связанный с этой конкретной командой меню. Программная реализация выглядит примерно так:

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

    <mx:MenuBar
        labelField="@label"
        itemClick="handleMenuClick(event)">
        <mx:XMLList>
            <menuitem label="File">
                <menuitem label="New"/>
                <menuitem label="Open"/>
                <menuitem label="Close" enabled="false"/>
            </menuitem>
            <menuitem label="Edit"/>
                <menuitem label="Source"/>
                <menuitem label="View">
                <menuitem label="50%"
                    type="radio" groupName="one"/>
                <menuitem label="100%"
                    type="radio" groupName="one"
                    selected="true"/>
                <menuitem label="150%"
                    type="radio" groupName="one"/>
            </menuitem>
        </mx:XMLList>
    </mx:MenuBar>

    <mx:Label id="disp0_lbl"/>

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

            private function handleMenuClick(evt:MenuEvent):void
            {
                this.disp0_lbl.text = evt.item.@label + " was selected";
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Поскольку содержимое dataProvider хранится в формате E4X, в этом примере для выборки атрибута используется запись E4X @label. Компонент MenuBar также поддерживает другие типы событий, в том числе change, itemRollOut, itemRollOver, menuHide и menuShow.

2.10. Вывод предупреждений

Задача

Требуется вывести сообщение в модальном режиме, возможно (но не обязательно) с вариантами дальнейших действий.

Решение

Воспользуйтесь компонентом `Alert`.

Обсуждение

Компонент `Alert` выводит модальное диалоговое окно с кнопками, с помощью которых пользователь может отреагировать на сообщение в диалоговом окне. Экземпляры `Alert` не могут создаваться в коде MXML; для этого необходимо использовать ActionScript. Пример:

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

    <mx:Button id="btn" click="showAlert(event)" label="Alert"/>

    <mx:Label id="lbl"/>

    <mx:Script>
        <![CDATA[
            import mx.events.CloseEvent;
            import mx.controls.Alert;
            import mx.events.MenuEvent;

            private function showAlert(evt:MouseEvent):void
            {
                var alert:Alert = Alert.show("Button was clicked",
                    "Alert Window Title",Alert.OK|Alert.CANCEL|Alert.NO|
                    Alert.YES,this,onAlertClose);
            }

            private function onAlertClose(evt:CloseEvent):void
            {
                switch(evt.detail)
                {
                    case Alert.OK:
                        lbl.text = "OK Clicked";
                        break;
                    case Alert.CANCEL:
                        lbl.text = "CANCEL Clicked";
                        break;
                    case Alert.NO:
                        lbl.text = "NO Clicked";
                        break;
```

```
                    case Alert.YES:
                        lbl.text = "YES Clicked";
                        break;
                }
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Когда пользователь щелкает на кнопке `btn`, приложение создает компонент `Alert` при помощи статического метода `show` класса `Alert`. Метод `show` получает следующие аргументы, определяющие конфигурацию предупреждения:

`text`

Сообщение, выводимое в окне.

`title`

Заголовок окна `Alert`.

`flags`

Кнопки в окне `Alert`. Допустимые значения: `Alert.OK`, `Alert.CANCEL`, `Alert.NO` и `Alert.YES`. Чтобы разместить в окне несколько кнопок, воспользуйтесь поразрядным оператором `OR` (например, `Alert.OK | Alert.CANCEL`).

`parent`

Экранный объект, по центру которого выравнивается окно предупреждения.

`closeHandler`

Обработчик события, вызываемый при нажатии любой кнопки в окне предупреждения.

`iconClass`

Класс ресурса для графического значка, размещаемого слева от сообщения в окне `Alert`.

`defaultButtonFlag`

Кнопка, используемая по умолчанию компонентом `Alert`; активизируется нажатием клавиши Enter. Допустимые значения: `Alert.OK`, `Alert.CANCEL`, `Alert.NO` и `Alert.YES`.

Метод `onAlertClose` назначается обработчиком события `closeHandler` компонента `Alert`. В аргументе метода передается объект `CloseEvent`; свойство `detail` объекта `CloseEvent` указывает, какой кнопкой было закрыто окно `Alert`.

2.11. Работа с датами и компонент Calendar

Задача

Требуется предоставить возможность выбора даты в календаре.

Решение

Воспользуйтесь компонентом `DateField` или компонентом `DateChooser` для отображения удобного календаря, в котором пользователь может выбрать дату.

Обсуждение

В Flex Framework включены два компонента для реализации календарных функций: `DateField` и `DateChooser`. Компонент `DateField` отображает компонент `TextInput` со значком календаря; если щелкнуть на значке, календарь открывается в приложении. Напротив, календарь `DateChooser` постоянно присутствует на экране. В следующем примере представлен простейший калькулятор для вычисления продолжительности поездки, демонстрирующий обе разновидности компонентов. Пользователь вводит начальную дату в `DateField`, а конечную – в `DateChooser`. Программа вычисляет разность между датами в обработчике события `change`. Свойство `selectedDate` обоих компонентов возвращает объект `Date`, представляющий выбранную дату.

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

    <mx:Form>
        <mx:FormHeading label="Trip Calculator"/>

        <mx:FormItem label="Start Date">
            <mx:DateField id="startDate" change="update(event)"/>
        </mx:FormItem>

        <mx:FormItem label="End Date">
            <mx:DateChooser id="endDate" change="update(event)"/>
        </mx:FormItem>

        <mx:FormItem label="Trip Duration (days)">
            <mx:Label id="display"/>
        </mx:FormItem>

    </mx:Form>

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

            private static const MILLISECONDS:int = 1000;
            private static const SECONDS:int = 60;
```

```
            private static const MINUTES:int = 60;
            private static const HOURS:int = 24;

            private function update(evt:CalendarLayoutChangeEvent):void
            {
                try
                {
                    var diff:Number = endDate.selectedDate.getTime() -
                    startDate.selectedDate.getTime();
                    // Преобразование миллисекунд в дни
                    var days:int = int(diff/(MILLISECONDS*
                    SECONDS*MINUTES*HOURS));
                    display.text = days.toString();
                }
                catch(ex:Error)
                {
                }
            }

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

При вычислениях с датами важно использовать метод `getTime` объекта `Date`, чтобы правильно учитывать високосные годы. Метод `getTime` возвращает количество миллисекунд, прошедших с 1 января 1970 года.

2.12. Отображение и позиционирование нескольких всплывающих окон

Задача

Требуется вывести дополнительные сообщения в нескольких всплывающих окнах.

Решение

Используйте класс `PopUpManager` для создания экземпляров компонента `TitleWindow` при взаимодействии с пользователем.

Обсуждение

В Flex Framework включен класс `PopUpManager`, содержащий статические методы для управления созданием, размещением и удалением окон верхнего уровня в приложениях Flex. Пример кода:

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

    <mx:Canvas horizontalCenter="0" verticalCenter="0">
```

```
        <mx:LinkButton label="Top" x="100" y="10"
             click="showDetail(event)"/>
        <mx:LinkButton label="Left" x="10" y="100"
             click="showDetail(event)"/>
        <mx:LinkButton label="Bottom" x="100" y="200"
             click="showDetail(event)"/>
        <mx:LinkButton label="Right" x="200" y="100"
             click="showDetail(event)"/>
        <mx:Canvas width="100" height="100" x="125" y="40"
             backgroundColor="#ff0000" rotation="45">
        </mx:Canvas>
    </mx:Canvas>

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;

            private const POPUP_OFFSET:int = 10;

            private function showDetail(evt:MouseEvent):void
            {
                // Создание всплывающего окна
                var popup:CustomPopUp =
                CustomPopUp(PopUpManager.createPopUp(this,
                CustomPopUp,false));
                popup.message = "This is the detail for " +
                      evt.target.label;
                // Позиционирование окна
                var pt:Point = new Point(0, 0);
                pt = evt.target.localToGlobal(pt);
                popup.x = pt.x + POPUP_OFFSET;
                popup.y = pt.y + evt.target.height + POPUP_OFFSET;
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Приложение создает и размещает группу компонентов `LinkButton` в `Canvas` с применением абсолютного позиционирования. Когда пользователь щелкает на компоненте `LinkButton`, под ним появляется всплывающее окно с уточняющей информацией. Для этого событие щелчка компонентов `LinkButton` связывается с методом `showDetail`, использующим метод `createPopUp` класса `PopUpManager` для создания экземпляра пользовательского компонента `CustomPopUp`. Далее свойству `message` всплывающего окна задается отображаемый текст. Наконец, всплывающее окно позиционируется относительно компонента `LinkButton`, инициировавшего запрос. Для этого координаты левого верхнего угла `LinkButton` (x = 0, y = 0 в пространстве координат `LinkButton`) преобразуются из локальной системы координат компонента в глобальную систему координат мето-

дом localToGlobal (вспомогательный метод, доступный для всех классов DisplayObject и их потомков). Класс CustomPopUp выглядит так:

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    width="300" height="50"
    styleName="customPopUp"
    showCloseButton="true"
    close="handleClose(event)">

    <mx:Style>
        .customPopUp
       {
            header-height:2;
            padding-left:5;
            padding-right:5;
            padding-top:5;
            padding-bottom:5;
            border-color:#000000;
            border-alpha:.5;
            border-thickness-left:5;
            border-thickness-right:5;
            border-thickness-bottom:5;
            border-thickness-top:5;
            background-color:#666666;
            color:#ffffff;
        }
    </mx:Style>

    <mx:Text width="100%" height="100%" text="{message}"/>

    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.events.CloseEvent;
            [Bindable]
            public var message:String;

            private function handleClose(evt:CloseEvent):void
            {
                PopUpManager.removePopUp(this);
            }
        ]]>
    </mx:Script>
</mx:TitleWindow>
```

Класс CustomPopUp расширяет TitleWindow, добавляя в него компонент Text для отображения сообщений. Событие close класса TitleWindow связывается с методом handleClose, закрывающим всплывающее окно методом removePopUp класса PopUpManager. Также он содержит стили CSS для настройки внешнего вида CustomPopUp.

2.13. Создание пользовательской рамки у всплывающего окна

Задача

Требуется изменить рамку всплывающего окна таким образом, чтобы окно было визуально связано с открывшим его компонентом.

Решение

Преобразуйте класс `PanelSkin` в субкласс и переопределите метод `updateDisplayList` так, чтобы в нем рисовалась стрелка. Задайте новый класс стилю `borderSkin` всплывающего окна.

Обсуждение

Решение строится на основе рецепта 2.12 с модификацией компонента `CustomPopUp`. Чтобы изменить рамку окна, задайте стилю `borderSkin` пользовательский класс `CustomPanelSkin`:

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    width="300" height="50"
    styleName="customPopUp"
    showCloseButton="true"
    close="handleClose(event)"
    borderSkin="CustomPanelSkin"
    initialize="initPopUp()">

    <mx:Style>
        .customPopUp {
            header-height:2;
            padding-left:5;
            padding-right:5;
            padding-top:5;
            padding-bottom:5;
            border-color:#000000;
            border-alpha:.5;
            border-thickness-left:5;
            border-thickness-right:5;
            border-thickness-bottom:5;
            border-thickness-top:5;
            background-color:#666666;
            color:#ffffff;
        }
    </mx:Style>

    <mx:Text width="100%" height="100%" text="{message}"/>

    <mx:Script>
        <![CDATA[
```

```
            import mx.managers.PopUpManager;
            import mx.events.CloseEvent;
            [Bindable]
            public var message:String;

            private function handleClose(evt:CloseEvent):void
            {
                PopUpManager.removePopUp(this);
            }
            private function initPopUp():void {
               this.isPopUp = false;
            }
        ]]>
    </mx:Script>
</mx:TitleWindow>
```

Далее приведен код класса `CustomPanelSkin`. Обратите внимание на задание свойству `isPopUp` значения `false`, это сделано для того, чтобы пользователь не мог перетаскивать всплывающее окно.

```
package {
    import flash.display.Graphics;

    import mx.skins.halo.PanelSkin;

    public class CustomPanelSkin extends PanelSkin
    {
        override protected function
            updateDisplayList(w:Number, h:Number):void {
            super.updateDisplayList(w,h);

            var gfx:Graphics = this.graphics;
            gfx.beginFill(this.getStyle("borderColor"),
                this.getStyle("borderAlpha"));
            gfx.moveTo(this.getStyle("cornerRadius"),0);
            gfx.lineTo(15,-10);
            gfx.lineTo(25,0);
        }
    }
}
```

Этот простой класс расширяет `PanelSkin`, по умолчанию определяющий прорисовку рамки `TitleWindow`. Метод `updateDisplayList` переопределяется, и в него включается логика рисования выносной стрелки к левому верхнему углу компонента `CustomPopUp`.

2.14. Обработка событий focusIn и focusOut

Задача

Требуется отобразить всплывающее окно, когда пользователь передает фокус надписи, и закрыть окно при потере фокуса.

Решение

Используйте события `focusIn` и `focusOut` (доступные для всех экземпляров классов, производных от `InteractiveObject`) для вызова соответствующих методов `PopUpManager`.

Обсуждение

Чтобы окно открывалось на основании передачи фокуса, воспользуемся адаптированной версией кода из двух предыдущих рецептов. Но вместо того, чтобы открывать окно по щелчку на `LinkButton`, мы будем открывать его по событию `focusIn`. Событие `focusIn` передается при получении компонентом фокуса (например, в результате перехода к нему клавишей Tab или щелчка). По сравнению с предыдущим рецептом в код обработчика события `focusIn` вносится дополнительная строка:

```
systemManager.removeFocusManager(IFocusManagerContainer(popup))
```

А вот как она выглядит в контексте решения:

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

    <mx:Canvas horizontalCenter="0" verticalCenter="0">
        <mx:LinkButton id="lbl" label="Top" x="100" y="10"
            focusIn="showDetail(event)"focusOut="closePopUp()"/>
        <mx:LinkButton label="Left" x="10" y="100"
            focusIn="showDetail(event)"focusOut="closePopUp()"/>
        <mx:LinkButton label="Bottom" x="100" y="200"
            focusIn="showDetail(event)"focusOut="closePopUp()"/>
        <mx:LinkButton label="Right" x="200" y="100"
            focusIn="showDetail(event)"focusOut="closePopUp()"/>
        <mx:Canvas width="100" height="100" x="125" y="40"
            backgroundColor="#ff0000" rotation="45">
        </mx:Canvas>
    </mx:Canvas>

    <mx:Script>
        <![CDATA[
            import mx.managers.IFocusManagerContainer;

            import mx.managers.PopUpManager;

            private const POPUP_OFFSET:int = 10;

            private var popup:CustomPopUp;

            private function showDetail(evt:FocusEvent):void {
                // Создание всплывающего окна
                popup = CustomPopUp(PopUpManager.createPopUp(
                    this,CustomPopUp,false));
                popup.message = "This is the detail for " +
                    evt.target.label;
```

```
                // Позиционирование окна
                var pt:Point = new Point(0, 0);
                pt = evt.target.localToGlobal(pt);
                popup.x = pt.x + POPUP_OFFSET;
                popup.y = pt.y + evt.target.height + POPUP_OFFSET;
                systemManager.removeFocusManager(
                    IFocusManagerContainer(popup))
            }

            private function closePopUp():void {
                PopUpManager.removePopUp(popup);
            }

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

При создании любого всплывающего окна SystemManager по умолчанию активизирует объект FocusManager, связанный с этим окном. Это делается для того, чтобы цикл передачи фокуса (управляющий Tab-перебором) базировался на только что созданном всплывающем окне. В данном случае требуется другое поведение: при потере фокуса LinkButton всплывающее окно должно закрываться. Задача решается удалением объекта FocusManager всплывающего окна из SystemManager, приводящим к повторной активизации объекта FocusManager приложения. Обработчик события focusOut – closePopUp – содержит логику закрытия окна. Если запустить приложение и несколько раз нажать клавишу Tab, фокус будет передаваться между экземплярами LinkButton, с автоматическим созданием и удалением всплывающих окон.

3

Контейнеры

Термином *контейнеры* обычно обозначаются все классы, входящие в пакет `mx.containers` Flex Framework. Контейнеры расширяют класс `UIComponent` и добавляют в него функциональность управления раскладкой, методы управления созданием дочерних компонентов и автоматическую прокрутку. Реализации контейнеров заметно отличаются друг от друга, но все контейнеры обладают общими возможностями позиционирования дочерних компонентов, размещения их с применением ограничений или стилей, управления прокруткой и реакцией дочерних компонентов на события прокрутки.

Ограничения (constraints) относятся к числу нововведений Flex 3. Они позволяют разработчикам определять правила позиционирования как для расположения, так и для размеров компонентов; эти правила назначаются дочерним компонентам контейнеров. Ограничения работают только с контейнерами, поддерживающими абсолютное позиционирование (например, `Canvas`) в том же смысле, в котором этот термин употребляется в CSS. Контейнеры `Box` и `Tile` обеспечивают автоматическое размещение дочерних компонентов и методы, управляющие включением дочерних компонентов в автоматическое управление раскладкой.

3.1. Позиционирование дочерних компонентов при управлении раскладкой

Задача

Требуется выстроить несколько дочерних компонентов по вертикали или горизонтали, управляя их размещением.

Решение

Воспользуйтесь контейнером HBox или VBox. Задайте свойство horizontalGap или verticalGap для HBox или VBox соответственно, чтобы установить расстояние между компонентами.

Обсуждение

Компоненты HBox и VBox, расширяющие общий базовый класс mx.containers.Box, размещают свои дочерние компоненты по горизонтали или вертикали соответственно. Количество дочерних компонентов не ограничивается. Если размер дочернего компонента больше высоты или ширины компонента HBox/VBox, последний по умолчанию добавляет полосы прокрутки. Для определения расстояния между дочерними компонентами контейнеры VBox используют свойство verticalGap, а контейнеры HBox — свойство horizontalGap. Пример:

```
<mx:VBox width="400" height="300" verticalGap="20">
    <mx:Button label="Button"/>
    <mx:LinkButton label="Link Button"/>
</mx:VBox>
```

Однако контейнеры HBox и VBox не соблюдают ограничения в свойствах bottom, left, right и top. Чтобы увеличить интервал между дочерними компонентами в контейнере Box, воспользуйтесь компонентом Spacer:

```
<mx:VBox width="400" height="300" verticalGap="20">
    <mx:Button label="Button"/>
    <mx:ComboBox top="60"/>
    <mx:Spacer height="20"/>
    <mx:LinkButton label="Link Button"/>
</mx:VBox>
```

Для изменения способа формирования отступа или расстояния между границей компонента и его дочерними компонентами используются стили paddingTop, paddingLeft, paddingRight или paddingBottom. Их действие распространяется на все дочерние компоненты, включенные в контейнер. Чтобы сместить один дочерний компонент влево/вправо в VBox или вверх/вниз в HBox, добавьте внутренний контейнер и воспользуйтесь им для позиционирования:

```
<mx:HBox x="400" horizontalGap="10" top="15">

    <mx:Canvas>
        <mx:Button top="50" label="Button" y="20"/>
    </mx:Canvas>

    <mx:Panel height="40" width="40"/>
    <mx:Spacer width="25"/>
    <mx:LinkButton label="Label"/>
    <mx:ComboBox/>
</mx:HBox>
```

В этом примере в контейнер Canvas включаются контейнеры HBox и VBox, демонстрирующие обе разновидности типа раскладки:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" >
    <mx:VBox width="400" height="300" verticalGap="20">
        <mx:Button label="Button"/>
        <mx:ComboBox/>
        <mx:Spacer height="20"/>
        <mx:LinkButton label="Link Button"/>
    </mx:VBox>
    <mx:HBox x="400" horizontalGap="10" top="15">
        <mx:Canvas>
            <mx:Button top="50" label="Button" y="20"/>
        </mx:Canvas>
        <mx:Panel height="40" width="40"/>
        <mx:Spacer width="25"/>
        <mx:LinkButton label="Label"/>
        <mx:ComboBox/>
    </mx:HBox>
</mx:Canvas>
```

3.2. Процентное позиционирование и изменение размеров дочерних компонентов

Задача

Требуется задать размеры дочерних компонентов в соответствии с размерами их родительского компонента.

Решение

Воспользуйтесь процентным позиционированием, чтобы при изменении размеров компонента размеры его дочерних компонентов были автоматически изменены Flex Framework.

Обсуждение

Процентное позиционирование – мощный инструмент для простого определения размеров и местоположения дочернего компонента по отношению к его родителю. Например, приведенный далее компонент RelativePositioningChild.mxml размещается таким образом, чтобы он занимал 40% от вычисляемой ширины и 70% от вычисляемой высоты своего родительского компонента:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="40%" height="70%"
    background Color="#0033ff">
    <mx:Image source="@Embed('../../assets/image.png')"/>
    <mx:Image source="@Embed('../../assets/image.png')"/>
</mx:VBox>
```

В следующем примере несколько экземпляров RelativePositioningChild выстраиваются в родительском контейнере, который тоже использует процентное позиционирование. Родитель, в который будет включен этот компонент, определит его ширину и высоту, а соответственно и ширину/высоту его дочерних компонентов.

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="75%" height="50%"
    background Color="#0099ff" alpha="0.3" xmlns:cookbook="oreilly.
    cookbook.*">
    <cookbook:RelativePositioningChild/>
    <cookbook:RelativePositioningChild/>
</mx:HBox>
```

Для демонстрации процентного изменения размеров предыдущий фрагмент, сохраненный в файле RelativePositioningParent.mxml, используется в следующем примере:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[

            private function changeWidth():void
            {
                this.width = slider.value*150;
            }

        ]]>
    </mx:Script>
    <cookbook:RelativePositioningParent/>
    <mx:HSlider id="slider" change="changeWidth()"/>
</mx:Application>
```

По мере того как ползунок изменяет ширину окна приложения, происходит соответствующее изменение позиции и размеров RelativePositioningParent и RelativePositioningChild.

3.3. Отслеживание позиции мыши в разных системах координат

Задача

Требуется отслеживать позицию мыши относительно родительского контейнера, а также относительно дочерних компонентов этого контейнера.

Решение

Используйте позиционные свойства класса MouseEvent и свойства mouseX/mouseY класса UIComponent, расширяемого всеми контейнерами.

Обсуждение

Класс MouseEvent содержит четыре свойства для определения позиции мыши. Свойства localX и localY содержат координаты относительно компонента, инициировавшего событие мыши, тогда как свойства stageX и stageY содержат координаты относительно самой сцены.

В следующем примере при наведении указателя мыши на компонент LinkButton свойства localX и localY будут отражать позицию мыши относительно компонента LinkButton. Если указатель мыши не находится над LinkButton, свойства отражают позицию мыши над VBox:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    mouseMove="traceMousePosition (event)">
    <mx:LinkButton label="MyButton"/>
</mx:VBox>
```

Чтобы определить позицию мыши относительно некоторого компонента независимо от того, находится указатель мыши над компонентом или нет, используйте свойства mouseX и mouseY объекта DisplayObject. Эти свойства возвращают позицию мыши относительно точки (0,0) контейнера или компонента. Наконец, класс Container определяет свойства contentMouseX и contentMouseY, описывающие позицию мыши относительно всего содержимого контейнера. Следующий пример возвращает позицию мыши относительно двух дочерних компонентов Panel, включенных в HBox (вместо левого верхнего угла HBox):

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="500" height="300"
    mouseMove="trace(this.contentMouseX+' : '+this.contentMouseY);">
    <mx:Panel width="400">
        <mx:Label text="Center" horizontalCenter="200"/>
    </mx:Panel>
<mx:Panel width="400">
<mx:Label text="Center" horizontalCenter="200"/>
</mx:Panel>
</mx:HBox>
```

Поскольку сумма ширин двух дочерних компонентов HBox превышает 812 пикселов, при прокрутке вправо и наведении указателя мыши на HBox будут отображаться значения x, превышающие установленный размер HBox. Контейнер определяет свойство contentPane — приватный объект DisplayObject, в который включаются все дочерние компоненты. Контейнер маскирует дочерние компоненты, если их ширина или высота превышает заданную ширину или высоту контейнера. Свойства contentMouseX и contentMouseY определяют позицию мыши в защищенном содержимом DisplayObject.

Последний пример демонстрирует практическое применение этих свойств:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="300" height="500"
paddingTop="10" paddingLeft="10" verticalGap="15"
```

```
        mouseMove="traceMousePosition(event)">
        <mx:Script>
            <![CDATA[

                private function traceMousePosition(event:MouseEvent):void
                {
                    trace(" MouseEvent local position "+
                        event.localX+" "+event.localY);
                    trace(" MousePosition stage position "+
                        event.stageX+" "+event.stageY);
                    trace(" MouseEvent position from w/in
                        component " +this.mouseX+" "+this.mouseY);
                    trace(" Content Mouse Position "+
                        this.contentMouseX+" "+this.contentMouseY);
                }

            ]]>
        </mx:Script>
        <mx:Image source="@Embed('../../assets/image.png')"/>
        <mx:Image source="@Embed('../../assets/image.png')"/>
        <mx:Image source="@Embed('../../assets/image.png')"/>
        <mx:Image source="@Embed('../../assets/image.png')"/>
    </mx:VBox>
```

Функция вызывается при перемещении мыши над компонентом `VBox`, поэтому она выводит только положительные значения `mouseX` и `mouseY`.

3.4. Динамическое добавление и удаление дочерних компонентов в контейнере

Задача

Требуется добавлять и удалять дочерние компоненты из контейнера во время выполнения без использования компонентов `Repeater` или `DataProvider`.

Решение

Используйте методы `addChild`/`addChildAt` для добавления дочерних компонентов и методы `removeChildAt`/`removeAllChildren` для их удаления.

Обсуждение

Методы Flex, предназначенные для добавления и удаления дочерних компонентов, просты и удобны, но `UIComponent` и контейнеры работают по слегка различающимся правилам.

Метод `addChild` добавляет в компонент, для которого был вызван метод, произвольный дочерний компонент, расширяющий `UIComponent`. Пример:

```
var component:UIComponent = new UIComponent();
addChild(component);
```

Методу addChildAt при вызове передается индекс позиции, в которую будет добавлен дочерний компонент. Для контейнеров, не обладающих средствами управления раскладкой (как, например, объекты Canvas), это означает, что добавленный дочерний компонент будет отображаться с заданным z-индексом глубины. Для контейнеров с управлением раскладкой (например, HBox и VBox) добавленный дочерний компонент будет находиться в заданной позиции. Пример:

```
var component:UIComponent = new UIComponent();
addChildAt(component, 3);
```

Удаление произвольного дочернего компонента осуществляется методом removeChildAt, который удаляет компонент с заданным индексом:

```
removeChildAt(2);
```

Контейнеры и UIComponent также содержат метод removeChild, которому передается ссылка на дочерний компонент, удаляемый из списка отображения.

Flex предоставляет разные методы для обращения к дочерним компонентам, добавленным в контейнер. Если вы можете обратиться к дочернему компоненту, то эта ссылка может использоваться и для его удаления. Например, к дочерним компонентам, добавленным в контейнер MXML, можно обращаться по свойству id. Для обращения к произвольному дочернему компоненту контейнера можно воспользоваться методом getChildAt:

```
getChildAt(index:int);
```

Также возможно обращение и по имени (если оно было присвоено дочернему компоненту). Для определения количества дочерних компонентов, содержащихся в произвольном компоненте, используется свойство numChildren:

```
var i:int = 0;
while(i<this.numChildren)
{
    trace(getChildAt(i));
    i++;
}
```

При удалении дочернего компонента указывается его индекс или свойство id:

```
removeChild(getChildAt(2);
```

или

```
removeChild(this.childId);
```

Наконец, метод removeAllChildren удаляет все дочерние компоненты, включенные в компонент.

3.5. Раскладки с ограничениями

Задача

Размеры дочерних компонентов должны определяться внутренними границами родительского компонента.

Решение

Используйте свойства ограничений: `left`, `right`, `top` и `bottom`.

Обсуждение

Свойства ограничений класса `UIComponent` позволяют определять ширину и высоту компонента по отношению к внутренней области родителя. Скажем, если компонент имеет ширину в 200 пикселов, а оба свойства `left` и `right` дочернего компонента равны 20 пикселам, ширина дочернего компонента составит 160 пикселов. Дочерний компонент с ограничением `bottom` в 40 пикселов внутри компонента высотой 200 пикселов будет размещен в верхних 160 пикселах родителя. Ограничения не могут использоваться с компонентами, не поддерживающими абсолютного позиционирования, например `HBox` и `VBox`. К категории контейнеров, у которых дочерним компонентам могут назначаться ограничения, принадлежат `Canvas`, `Panel` и `Application`.

В следующем примере компонент `innerCanvas` смещается на 20 пикселов от левого, верхнего и правого краев и на 50 пикселов от нижнего края; таким образом, `innerCanvas` имеет ширину 360 пикселов и высоту 230 пикселов. При расширении или уменьшении внешнего компонента `innerCanvas` сохраняет неизменные отступы со всех четырех сторон. Размеры компонентов `Button` и `TextInput` в компоненте будут изменяться в соответствии с размерами компонента `innerCanvas`; при этом будут сохраняться отступы, заданные свойствами ограничений.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
     height="300">
    <mx:Canvas left="20" right="20" top="20" bottom="50" id="innerCanvas ">
        <mx:Button label="Button Label" left="20" right="20" top="10"/>
        <mx:TextInput left="30" right="30" bottom="10"/>
    </mx:Canvas>
</mx:Canvas>
```

В следующем фрагменте кода два компонента `Panel` отображаются с шириной 320 пикселов из-за назначенных им ограничений. По высоте компонент `topPanel` начинается с 40 пикселов (ограничение `top`, заданное для `Panel`), а `bottomPanel` доходит до 260 пикселов (ограничение `bottom`).

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Panel left="40" right="40" top="40" id="topPanel ">
        <mx:Label text="Label text"/>
```

```
    </mx:Panel>
    <mx:Panel left="40" right="40" bottom="40" id="bottomPanel">
        <mx:Label text="Label text"/>
    </mx:Panel>
</mx:Canvas>
```

3.6. Задание максимального и минимального размера дочерних компонентов в контейнере

Задача

Требуется включить в компонент несколько дочерних компонентов таким образом, чтобы в случае превышения порога общей ширины компоненты переходили в следующий ряд.

Решение

Используйте свойства `maxWidth` и `maxHeight` при размещении дочерних компонентов в компоненте.

Обсуждение

Свойства `maxWidth` и `maxHeight` определяют максимальную ширину и высоту, с которыми родитель разрешит отображать данный компонент. В следующем фрагменте кода проверка `maxWidth` гарантирует, что добавление следующего компонента не приведет к превышению максимальной ширины, разрешенной родительским контейнером. Если максимальная ширина будет превышена, приложение создает дополнительный компонент `HBox` и помещает изображение в него. Контейнер `VBox` обеспечит правильную раскладку добавленных дочерних компонентов.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" height="400"
xmlns:cookbook="oreilly.cookbook.*" backgroundColor="#0000ff">
    <cookbook:AddConstraintChildren maxHeight="400" maxWidth="800"
        horizontalAlign="center" verticalScrollPolicy="off"/>
</mx:Canvas>

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.controls.Image;

            [Embed(source="../../assets/image.png")]
            public var image:Class;
            private var childCount:int = 0;
            private var currentHolder:HBox;

            private function addMoreChildren():void
            {
                var image:Image = new Image();
                image.source = image;
```

```
                if((currentHolder.width + image.width) >=
                    this.maxWidth)
                {
                    var holder:HBox = new HBox();
                    addChild(holder);
                    currentHolder = holder;
                }
                currentHolder.addChild(image);
            }
        ]]>
    </mx:Script>
    <mx:Button label="addMoreChildren()" click="addMoreChildren()"/>
    <mx:HBox id="topHolder" creationComplete="currentHolder = topHolder"/>
</mx:VBox>
```

См. также

Рецепт 3.5.

3.7. Задание ограничений для строк и столбцов компонентов в контейнере

Задача

Требуется задать ограничения для отдельных строк и столбцов дочерних компонентов без определения ограничений для каждого дочернего компонента.

Решение

Используйте свойства `ConstraintRow` и `ConstraintColumn` для установления ограничений, относящихся к отдельным областям контейнера.

Обсуждение

Объекты `ConstraintRow` и `ConstraintColumn` позволяют определить сетку ограничений, которая может использоваться для позиционирования компонентов. Такие ограничения используются почти так же, как ограничения, задаваемые относительно границ контейнера. Например, строка

```
left="columnName:10"
```

размещает компонент на 10 пикселов левее столбца, определяемого следующим фрагментом:

```
<mx:Canvas>
    <mx:constraintColumns>
        <mx:ConstraintColumn id="leftCol" width="200"/>
        <mx:ConstraintColumn id="rightCol" width="60%"/>
    </mx:constraintColumns>
```

```
    <mx:constraintRows>
        <mx:ConstraintRow id="topRow" height="80%"/>
        <mx:ConstraintRow id="bottomRow" height="20%"/>
    </mx:constraintRows>
<mx:Button label="Click Me" left="leftCol:0" right="leftCol:0"
    top="row2:0" bottom="row2:0"/>
</mx:Canvas>
```

Также можно добавить компоненты для увеличения или уменьшения места, выделяемого для каждого столбца, и проследить за тем, как изменение размеров контейнера влияет на изменение раскладки. Пример:

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

<mx:Script>
    <![CDATA[
        [Bindable]
        public var txt:String = "Cortázar is highly regarded
            as a master of short stories of a fantastic bent,
            with the collections Bestiario (1951)" +
            " and Final de Juego (1956) containing many of
            his best examples in the genre, including the
            remarkable \"Continuidad de los Parques\"" +
            " and \"Axolotl.\" These collections received
            early praise from Álvaro Cepeda Samudio, and
            selections from the two volumes were published" +
            " in 1967 in English translations by Paul Blackburn,
            under the title End of the Game and
            Other Stories (in later editions, Blow-Up and " +
            "Other Stories, in deference to the English
            title of Antonioni's celebrated film of 1966 of
            Cortázar's story Las babas del diablo).";

            private function changeColumnProportion():void
            {
                this.leftCol.percentWidth = (10*c_slider.value);
                this.rightCol.percentWidth = 100 - (10*c_slider.value);
            }

            private function changeRowProportion():void
            {
                this.row1.percentHeight = (10*r_slider.value);
                this.row2.percentHeight = 100 - (10*r_slider.value);
            }
    ]]>
</mx:Script>

<mx:Canvas width="100%" height="100%" horizontalScrollPolicy="off"
verticalScrollPolicy="off">
    <mx:constraintColumns>
        <mx:ConstraintColumn id="leftCol" width="50%" />
        <mx:ConstraintColumn id="rightCol" width="50%" />
```

```
    </mx:constraintColumns>
    <mx:constraintRows>
        <mx:ConstraintRow id="row1" height="20%"/>
        <mx:ConstraintRow id="row2" height="80%" />
    </mx:constraintRows>

    <mx:HSlider id="c_slider" change="changeColumnProportion()" value="5"/>
    <mx:HSlider id="r_slider" x="200" change="changeRowProportion()" val-
ue="5"/>

    <mx:Text text="{txt}" left="leftCol:0" right="leftCol:0" top="row2:0"
        bottom="row2:0"/>

    <mx:Text text="{txt}" left="rightCol:0" right="rightCol:0" top="row2:0"
        bottom="row2:0"/>
</mx:Canvas>
</mx:Application>
```

3.8. Использование ограничений при форматировании текста

Задача

Требуется сформировать макет для нескольких текстовых разделов.

Решение

Создайте и включите в компонент `Canvas` объекты `ConstraintRow` и `ConstraintColumn`; используйте их для задания ограничений для дочерних компонентов.

Обсуждение

Во всех контейнерах с поддержкой ограничений добавляемые ограничения строк и столбцов сохраняются в двух массивах. После добавления ограничения в соответствующий массив любой дочерний компонент сможет обращаться к этому ограничению за информацией. В примере этого рецепта ограничения используются для управления форматированием текста. Приложение создает несколько объектов `ConstraintRow` и `ConstraintColumn` и добавляет их в свойства `constraintRows` и `constraintColumns` объекта `Canvas`.

Чтобы использовать ограничение для динамически сгенерированного объекта `UIComponent`, установите ограничение методом `setStyle`. В приведенном примере ограничение динамически загружается из массива объектов `ConstraintColumn` командой

```
text.setStyle("left", constraintColumn.id+":10");
```

или с чуть более хитроумным синтаксисом, обеспечивающим обращение к нужному столбцу:

```
child.setStyle("left", (constraintColumns[i-(constraintColumns.length/2)-2]
as ConstraintColumn).id+":10");
```

В приведенном далее полном листинге приложение генерирует объекты `ConstraintColumn` **и назначает новым дочерним компонентам стилевые свойства, базирующиеся на этих объектах. При добавлении новых строк позиции дочерних компонентов обновляются с учетом изменения раскладки, а объекты** `ConstraintColumn` **удаляются из массива** `Container.constraintColumns` **простым сокращением размера:**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1000"
height="800">
    <mx:Script>
        <![CDATA[
            import mx.core.UIComponent;
            import mx.controls.TextArea;
            import mx.containers.utilityClasses.ConstraintColumn;

            [Bindable]
            public var txt:String = "Cortázar is highly
            regarded as a master of short stories of a
            fantastic bent, with the collections Bestiario (1951)" +
            " and Final de Juego (1956) containing
            many of his best examples in the genre,
            including the remarkable \"Continuidad de los Parques\"" +
            " and \"Axolotl.\";

            private function addText(event:Event):void
            {
                var text:TextArea = new TextArea();
                addChild(text);
                text.text = txt;
                var constraintColumn:ConstraintColumn =
                new ConstraintColumn();
                constraintColumn.id =
                "column"+numChildren.toString();
                constraintColumns.push(constraintColumn);
                if(constraintColumns.length > 1)
                {
                    for each(var col:ConstraintColumn in
                    constraintColumns)
                    {
                        col.width = (width / (numChildren-2));
                    }
                }
                constraintColumn.width = (width / (numChildren-2));
                text.setStyle("top", "row:30");
                text.setStyle("bottom", "row:30");
                text.setStyle("left", constraintColumn.id+":10");
                text.setStyle("right", constraintColumn.id+":10");
            }
```

```
                private function addRow(event:Event):void
                {
                    var constraintRow:ConstraintRow = new ConstraintRow();
                    constraintRows.push(constraintRow);
                    constraintRow.id = "row"+constraintRows.length;
                    for each(var row:ConstraintRow in
                    constraintRows)
                    {
                        row.height = (height / (constraintRows.length-1));
                    }
                    var i:int = Math.round(numChildren -
                     (numChildren-2)/constraintRows.length);
                    while(i < numChildren)
                    {
                        var child:UIComponent = (getChildAt(i) as UIComponent);
                        child.setStyle("top",
                                       "row"+constraintRows.length+":30");
                        child.setStyle("bottom",
                                       "row"+constraintRows.length+":30");
                        child.setStyle("left",
                        (constraintColumns[i-(constraintColumns.length/2)-2]
                            as ConstraintColumn).id+":10");
                        child.setStyle("right",
                        (constraintColumns[i-(constraintColumns.length/2)-2]
                            as ConstraintColumn).id+":10");
                        i++;
                    }
                    constraintColumns.length =
                    constraintColumns.length / constraintRows.length;
                }
            ]]>
        </mx:Script>
        <mx:constraintRows>
            <mx:ConstraintRow id="row" height="100%"/>
        </mx:constraintRows>
        <mx:Button click="addText(event)" label="add text"/>
        <mx:Button click="addRow(event)" label="add row" x="150"/>
    </mx:Canvas>
```

3.9. Управление прокруткой и перетеканием в контейнерах

Задача

Требуется отключить вертикальные полосы прокрутки для контейнера. Прокрутка должна осуществляться перемещением указателя мыши над специально определяемыми областями.

Решение

Используйте свойства `horizontalScrollPolicy`**,** `verticalScrollPolicy` **и** `verticalScrollPosition`**.**

Обсуждение

Отображением полос прокрутки управляют свойства `horizontalScrollPolicy` **и** `verticalScrollPolicy`**. Используйте значение** `on` **для одной или обеих полос, чтобы они отображались всегда, или значение** `off`**, чтобы они не отображались никогда. В режиме** `auto` **полосы прокрутки появляются в том случае, если ширина содержимого контейнера превышает заданную ширину или высоту. Например, если задать свойство** `horizontalScrollPolicy` **равным** `auto`**, полоса прокрутки будет появляться в том случае, если ширина контейнера превысит заданное значение** `width`**.**

Прокрутка компонента осуществляется при помощи свойств `horizontalScrollPosition` **и** `verticalScrollPosition`**. Эти свойства определяют расстояние между левым или верхним краем области содержимого компонента и видимым левым или верхним краем. Пример:**

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="200" horizontal ScrollPolicy="auto" verticalScrollPolicy="off"
mouseMove="autoScroll(event)">
    <mx:Script>
        <![CDATA[
            private var hasAddedScroll:Boolean = false;
            private function autoScroll(event:MouseEvent):void
            {
                if(mouseX > width - 50 && !hasAddedScroll)
                {
                    addEventListener(Event.ENTER_FRAME, scrollRight);
                    hasAddedScroll = true;
                }
                else if(mouseX < 50 && !hasAddedScroll)
                {
                    addEventListener(Event.ENTER_FRAME, scrollLeft);
                    hasAddedScroll = true;
                }
                else
                {
                    removeEventListener(Event.ENTER_FRAME, scrollRight);
                    removeEventListener(Event.ENTER_FRAME, scrollLeft);
                    hasAddedScroll = false;
                }
            }

            private function scrollRight(event:Event):void
            {
                if(horizontalScrollPosition < maxHorizontalScrollPosition)
                {
                    horizontalScrollPosition+=4;
```

```
                }
                else
                {
                    removeEventListener(Event.ENTER_FRAME, scrollRight);
                    hasAddedScroll = false;
                }
            }

            private function scrollLeft(event:Event):void
            {
                if(horizontalScrollPosition > 0)
                {
                    horizontalScrollPosition-=4;
                }
                else
                {
                    removeEventListener(Event.ENTER_FRAME, scrollLeft);
                    hasAddedScroll = false;
                }
            }
        ]]>
    </mx:Script>
    <mx:Image source="@Embed('assets/image.png')"/>
    <mx:Image source="@Embed('assets/image.png')"/>
    <mx:Image source="@Embed('assets/image.png')"/>
    <mx:Image source="@Embed('assets/image.png')"/>
    <mx:Image source="@Embed('assets/image.png')"/>
</mx:HBox>
```

3.10. Управление раскладкой в компонентах Box

Задача

Требуется переключить раскладку компонента `Box` с вертикальной на горизонтальную, а также управлять вертикальными и горизонтальными отступами и центровкой дочерних компонентов.

Решение

Используйте свойства `verticalAlign` и `horizontalAlign`, задайте направление раскладки `Box` при помощи свойства `direction`.

Обсуждение

Класс `mx.containers.Box` определяет несколько свойств, управляющих раскладкой дочерних компонентов в `Box`:

`direction`

Направление раскладки дочерних компонентов контейнером. Допустимые значения: `horizontal` и `vertical`.

`horizontalAlign`

Режим горизонтального выравнивания дочерних компонентов. Допустимые значения: `left`, `right` и `center`.

`horizontalGap`

Расстояние между дочерними компонентами, если свойству `direction` задано значение `horizontal`. Если значение `direction` отлично от `horizontal`, свойство `horizontalGap` игнорируется.

`verticalAlign`

Режим горизонтального выравнивания дочерних компонентов. Допустимые значения: `top`, `bottom` и `center`.

`verticalGap`

Расстояние между дочерними компонентами, если свойству `direction` задано значение `vertical`. Если значение `direction` отлично от `vertical`, свойство `verticalGap` игнорируется.

В следующем примере все свойства связываются с компонентами, что позволяет динамически изменять их в демонстрационных целях:

```
<mx:HSlider change="{vbox.verticalGap = vSlider.value*10}" id="vSlider"/>
    <mx:VBox y="100" direction="{_direction}" id="vbox">
        <mx:Panel width="90" height="60">
            <mx:Label text="Some Text" />
        </mx:Panel>
        <mx:Panel width="90" height="60">
            <mx:Label text="Some Text" />
        </mx:Panel>
        <mx:Panel width="90" height="60">
            <mx:Label text="Some Text" />
        </mx:Panel>
        <mx:Panel width="90" height="60">
            <mx:Label text="Some Text" />
        </mx:Panel>
        <mx:Panel width="90" height="60">
            <mx:Label text="Some Text" />
        </mx:Panel>
</mx:Box>
```

Поскольку `verticalGap` является привязываемым, а не стилевым свойством, оно может задаваться динамически во время выполнения без вызова `setStyle()`.

3.11. Инициализация контейнеров

Задача

Для улучшения скорости отклика приложения необходимо сделать так, чтобы все дочерние компоненты контейнера создавались сразу же после инициализации приложения.

Решение

Используйте свойства `verticalAlign` и `horizontalAlign`, задайте направление раскладки `Box` при помощи свойства `direction`.

Обсуждение

У всех контейнеров (а фактически всех `UIComponent`) процесс создания экземпляра, создания свойств, создания дочерних компонентов и формирования раскладки состоит из трех шагов. На первом шаге Framework вызывает конструктор контейнера и передает событие `preinitialize`. На втором шаге происходит предварительная инициализация всех дочерних компонентов вплоть до максимального уровня глубины. На третьем шаге инициализируется дочерний компонент на максимальном уровне глубины, и процесс проходит в обратном направлении, повторяется вплоть до родительского контейнера. Например, следующая иерархия компонентов:

```
<mx:HBox>
    <mx:VBox>
        <mx:Panel/>
        <mx:Panel/>
    </mx:VBox>
</mx:HBox>
```

инициализируется в следующем порядке:

```
HBox preinitialize
    VBox preinitialize
        FirstPanel preinitialize
        SecondPanel preinitialize
        FirstPanel initialize
        SecondPanel initialize
    VBox initialize
    HBox initialize
        FirstPanel creationComplete
        SecondPanel creationComplete
    VBox creationComplete
HBox creationComplete
```

На момент отправки компонентом событий `preinitialize` и `initialize` его дочерние компоненты еще не были созданы. Следовательно, для обращения ко всем дочерним компонентам необходимо прослушивать событие `creationComplete`. После отправки события `initialize` размеры компонента были вычислены, компонент был включен в раскладку и прорисован, но может оказаться, что создание его дочерних компонентов еще не завершено. О завершении создания дочерних компонентов сигнализирует событие `creationComplete`.

3.12. Создание TitleWindow

Задача

Требуется создать компонент TitleWindow для отображения диалогового окна и использовать PopUpManager для его удаления при выполнении некоторых условий.

Решение

Используйте компонент TitleWindow. Этот компонент расширяет Panel и дополняет его функциональностью назначения заголовка окна и стилевой информации для прорисовки рамки.

Обсуждение

В следующем примере компонент TitleWindow используется для создания окна заставки приложения. Класс PopUpManager предоставляет механизм удаления компонента с экрана вызовом метода PopUpManager.removePopUp:

```
PopUpManager.removePopUp(this);
```

Предполагается, что компонент TitleWindow был добавлен через PopManager:

```
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
borderColor="#0000ff"
backgroundAlpha="0.6" title="Title Window" x="168" y="86">
    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.controls.Text;

            // Компонент TextInput для размещения результата.
            public var loginName:Text;
            public var loggedIn:Boolean;

            // Обработчик события для кнопки OK.
            private function returnName():void
            {
                loginName.text="Name entered: " + userName.text;
                PopUpManager.removePopUp(this);
            }

            private function checkUserNameAndPass():void
            {
                /* Обработка */
            }

            /* Обработка события при регистрации */
            private function returnValueFromLogin(event:Event):void
```

```
            {
                if(loggedIn)
                {
                    PopUpManager.removePopUp(this);
                }
                else
                {
                    successText.text = "User/Pass not recognized";
                }
            }
        ]]>
    </mx:Script>
    <mx:HBox>
        <mx:Label text="Username "/>
        <mx:TextInput id="userName" width="100%"/>
    </mx:HBox>
    <mx:HBox>
        <mx:Label text="Password"/>
        <mx:TextInput id="password" width="100%"/>
    </mx:HBox>
    <mx:HBox>
        <mx:Button label="Enter" click="checkUserNameAndPass();"/>
        <mx:Button label="Cancel" click="PopUpManager.removePopUp(this);"/>
    </mx:HBox>
<mx:Text id="successText" color="#ff0000"/>
</mx:TitleWindow>
```

3.13. Управление контейнером ViewStack через LinkBar

Задача

Требуется управлять контейнером `ViewStack` при помощи компонента `LinkBar`.

Решение

Используйте свойство `selectedIndex` или `selectedItem` компонента `LinkBar` для определения отображаемого элемента `ViewStack`.

Обсуждение

У компонента `LinkBar` в качестве провайдера данных может использоваться массив; также возможно указать контейнер с несколькими дочерними компонентами (например, `ViewStack`) – `LinkBar` интерпретирует его как провайдера данных. Второй способ в данном примере особенно удобен. При передаче контейнера выделенный элемент, который будет отображаться в контейнере, автоматически привязывается к выделенному элементу компонента `LinkBar`. А это означает, что `LinkBar` можно передать контейнер с несколькими дочерними компонентами, которые

будут использоваться для заполнения LinkBar. Компонент LinkBar автоматически добавляет необходимое количество кнопок для дочерних компонентов из ViewStack и связывает с каждой кнопкой код назначения свойства selectedChild компонента ViewStack.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800"
height="600">
    <mx:LinkBar dataProvider="{viewStack}" direction="horizontal"
        labelField="name"/>
    <mx:ViewStack id="viewStack" y="60">
        <mx:Panel width="150" height="150"
        name="first" label="First Panel" title=
        "First Panel">
            <mx:Label text="First label"/>
        </mx:Panel>
        <mx:Panel width="150" height="150" name="second"
        label="Second Panel" title=
        "Second Panel">
            <mx:Label text="Second label"/>
        </mx:Panel>
        <mx:Panel width="150" height="150" name="third"
        label="Third Panel" title=
        "Third Panel">
            <mx:Label text="Third label"/>
        </mx:Panel>
    </mx:ViewStack>
</mx:Canvas>
```

См. также

Рецепт 3.16.

3.14. Привязка свойства selectedIndex компонента ViewStack к переменной

Задача

Требуется привязать свойство selectedIndex компонента ViewStack к целочисленной переменной, которая может быть изменена в любой точке кода компонента.

Решение

Создайте переменную с пометкой bindable и привяжите к ней свойство selectedIndex компонента ViewStack.

Обсуждение

В случае компонента LinkBar выделенный элемент компонента ViewStack автоматически привязывается к выделенному элементу компонента

LinkBar. При использовании других компонентов индекс или выделенный элемент ViewStack или другого компонента, имеющего несколько дочерних компонентов, но отображающего их по одному, должен быть привязан к переменной или задаваться по событию. Также существует другой способ управления ViewStack: свяжите свойство selectedIndex компонента ViewStack с переменной и изменяйте переменную во время выполнения, как показано в следующем примере:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" click="changeViewStack()">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var selectedIndexInt:int = 0;

            private function changeViewStack():void
            {
                if(selectedIndexInt == 2)
                {
                    selectedIndexInt = 0;
                }
                else
                {
                    selectedIndexInt++;
                }
            }

        ]]>
    </mx:Script>
    <mx:ViewStack selectedIndex="{selectedIndexInt}">
        <mx:HBox height="{this.height}" width="{this.width}">
            <mx:Label text="First View Item"/>
            <mx:Label text="First View Item"/>
        </mx:HBox>
        <mx:VBox height="{this.height}" width="{this.width}">
            <mx:Label text="Second View Item"/>
            <mx:Label text="Second View Item"/>
        </mx:VBox>
        <mx:Canvas height="{this.height}" width="{this.width}">
            <mx:Label text="Third View Item"/>
            <mx:Label text="Third View Item" y="40"/>
        </mx:Canvas>
    </mx:ViewStack>
</mx:Canvas>
```

См. также

Рецепт 3.17.

3.15. Отложенное создание компонентов для ускорения запуска

Задача

Компоненты должны создаваться только тогда, когда возникнет необходимость в их отображении на экране.

Решение

Назначьте классу `Container` политику создания `queued`. Задайте свойство `creationIndex` для тех дочерних компонентов, для которых это необходимо.

Обсуждение

По умолчанию класс `Container` создает компоненты только перед их отображением, поскольку свойство `creationPolicy` по умолчанию задается равным `auto`. Другие возможные значения `creationPolicy`: `none` (компоненты не создаются), `all` (создаются все компоненты) и `queued` (компоненты создаются в соответствии со значением `creationIndex`). Массив `creationIndex` с нумерацией элементов, начинающейся с нуля, определяет порядок создания дочерних компонентов заданного компонента.

В следующем примере продемонстрировано использование `creationPolicy` для компонента `ComboBox`:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" height="600"
width="600">
    <mx:Script>
        <![CDATA[

            private function changeViewStackCreation():void
            {
                viewStack.creationPolicy =
                (comboBox.selectedItem as String);
                viewStack.createComponentsFromDescriptors(true);
            }

            private function changeViewStack():void
            {
                viewStack.selectedIndex = comboBoxChangeIndex.selectedIndex;
            }

        ]]>
    </mx:Script>
    <mx:Fade alphaFrom="0" alphaTo="1" duration="4000" id="fadeIn"/>
    <mx:ComboBox dataProvider="{['none', 'all', 'queued', 'auto']}"
        change="changeViewStackCreation()" id="comboBox"/>
    <mx:ComboBox dataProvider="{[1, 2, 3, 4]}"
        change="changeViewStack()" id="comboBoxChangeIndex" x="150"/>
```

```
    <mx:ViewStack id="viewStack" width="400" height="300"
        creationPolicy="none"y="100">
        <mx:Canvas creationCompleteEffect="{fadeIn}"
            creationIndex="0" backgroundColor="#0000ff" id="canvas1">
            <mx:LinkButton label="Link Button Number One"/>
        </mx:Canvas>
        <mx:Canvas creationCompleteEffect="{fadeIn}"
            creationIndex="1" backgroundColor="#0000ff" id="canvas2">
            <mx:LinkButton label="Link Button Number Two"/>
        </mx:Canvas>
        <mx:Canvas creationIndex="2"
        creationCompleteEffect="{fadeIn}" backgroundColor="#0000ff"
            id="canvas3">
            <mx:LinkButton label="Link Button Number Three"/>
        </mx:Canvas>
        <mx:Canvas creationIndex="3" creationCompleteEffect="{fadeIn}"
            backgroundColor="#0000ff" id="canvas4">
            <mx:LinkButton label="Link Button Number Four"/>
        </mx:Canvas>
    </mx:ViewStack>
</mx:Canvas>
```

3.16. Создание контейнеров переменного размера и управление ими

Задача

Требуется создать контейнер, размеры которого можно изменить перетаскиванием углового маркера.

Решение

Используйте классы `MouseEvent` для прослушивания событий `MouseDown`, `mouseMove` и `mouseUp` на маркере перетаскивания. Измените размеры контейнера при завершении перетаскивания.

Обсуждение

Добавление слушателей одновременно в MXML и ActionScript обеспечивает прослушивание события `mouseDown` для объекта `Icon`. При обнаружении события `mouseDown` все дальнейшие перемещения мыши по сцене отслеживаются как операции по изменению размеров контейнера. Методы `startDrag` и `stopDrag` класса `UIComponent` позволяют указать, находится ли экземпляр `UIComponent`, представляющий маркер перетаскивания, в состоянии перетаскивания или нет. В следующем примере позиция маркера используется для задания новой ширины и высоты `Canvas` при помощи методов `explicitWidth` и `explicitHeight`:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" backgroundColor="#ccccff" verticalScrollPolicy="off"
```

```
    horizontalScrollPolicy="off">
    <mx:Script>
        <![CDATA[

            private function startResize():void
            {
                stage.addEventListener(MouseEvent.MOUSE_MOVE,
                    resize);
                resizeIcon.addEventListener(MouseEvent.MOUSE_UP,
                    stopResize);
                resizeIcon.addEventListener(MouseEvent.ROLL_OUT,
                    stopResize);
                resizeIcon.startDrag();
            }
```

Когда пользователь нажимает кнопку мыши, все дальнейшие события `mouseMove` на сцене захватываются. При отпускании кнопки мыши обработчики событий удаляются, и для `resizeIcon` вызывается метод `stopDrag`.

```
private function stopResize(mouseEvent:MouseEvent):void
{
    resizeIcon.removeEventListener(MouseEvent.MOUSE_UP, stopResize);
    resizeIcon.removeEventListener(MouseEvent.ROLL_OUT, stopResize);
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, resize);
    resizeIcon.stopDrag();
}
```

Свойства `explicitHeight` и `explicitWidth` компонента позволяют задать размер компонента в пикселах независимо от логики вычисления и изменения размеров Flex Framework:

```
            private function resize(mouseEvent:MouseEvent):void
            {
                this.explicitHeight = resizeIcon.y + resizeIcon.height + 10;
                this.explicitWidth = resizeIcon.x + resizeIcon.width + 10;
            }

        ]]>
    </mx:Script>
    <mx:Panel width="60%" height="60%" top="20" left="20"/>
    <mx:Image id="resizeIcon" source="@Embed('../../assets/Resize.png')"
         mouseDown="startResize()" x="360" y="260"/>
</mx:Canvas>
```

См. также

Рецепт 3.9.

3.17. Создание и управление блокировкой TabControl в TabNavigator

Задача

Требуется динамически добавлять и удалять вкладки в компоненте TabNavigator, а также устанавливать и снимать блокировку с некоторых вкладок.

Решение

Используйте методы addChild и removeChild компонента TabNavigator для добавления и удаления дочерних компонентов, а свойство enabled — для установления и снятия блокировки вкладок в TabNavigator.

Обсуждение

Для любого дочернего компонента, включенного в TabNavigator, на навигационной панели TabNavigator создается новая вкладка. При удалении дочернего компонента вкладка автоматически уничтожается. Обе операции выполняются с использованием привязки dataProvider компонента TabNavigator. Любые изменения в dataProvider приводят к обновлению TabNavigator с динамическим добавлением или удалением дочерних компонентов с верхней навигационной панели. В следующем примере определяются методы доступа для работы со свойствами TabNavigator из компонента:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Script>
        <![CDATA[

            import mx.core.UIComponent;

            public function addChildToNavigator(value:UIComponent):void
            {
                navigator.addChild(value);
            }

            public function removeNavigatorChild(value:int = 0):void
            {
                if(value == 0)
                {
                    navigator.removeChildAt(navigator.numChildren-1);
                }
                else
                {
                    navigator.removeChildAt(value);
                }
            }
```

```
            public function disableTab(value:int = 0):void{
                if(value == 0)
                {
                    (navigator.getChildAt(navigator.numChildren - 1)
                        as UIComponent). enabled = false;
                }
                else
                {
                    (navigator.getChildAt(value) as UIComponent).
                        enabled = false;
                }
            }

            public function get tabNumChildren():Number
            {
                return navigator.numChildren;
            }
        ]]>
    </mx:Script>
    <mx:TabNavigator id="navigator">

    </mx:TabNavigator>
</mx:Canvas>
```

Чтобы использовать компонент, передайте ему любой экземпляр объекта `Container` **(в данном случае** `Canvas`**), вызывая метод по ссылке на свойство** `id` **компонента. Свойство** `label` **экземпляра** `Container`**, переданного** `TabNavigator`**, будет использоваться для задания заголовка добавляемой вкладки. Компоненты** `Button` **предназначены для добавления, удаления и блокировки вкладок.**

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[

            import mx.controls.Label;
            import mx.containers.Canvas;

            private function addCanvas():void
            {
                var canvas:Canvas = new Canvas();
                canvas.title = "Tab " +
                this.tabNavigator.tabNumChildrentoString();
                var label:Label = new Label();
                label.text = "Hello Label " +
                this.tabNavigator.tabNumChildren.toString();
                canvas.addChild(label);
                canvas.minHeight = 200;
                canvas.minWidth = 200;
```

```
                this.tabNavigator.addChildToNavigator(canvas);
            }

        ]]>
    </mx:Script>
    <cookbook:TabNavigatorDemo id="tabNavigator"/>
    <mx:HBox y="300">
        <mx:Button click="addCanvas()" label="add child"/>
        <mx:Button click="tabNavigator.removeNavigatorChild()" y="300"
            label="removechild"/>
        <mx:Button click="tabNavigator.disableTab()" y="300"
            label="disable child"/>
    </mx:HBox>
</mx:Application>
```

3.18. Создание компонента TabNavigator с функцией закрытия вкладок

Задача

Требуется создать экземпляр `TabNavigator`, вкладки которого снабжаются кнопкой закрытия. Щелчок на кнопке удаляет как вкладку, так и дочерний компонент, находящийся в `TabNavigator` по соответствующему индексу.

Решение

Используйте компонент `SuperTabNavigator` из библиотеки flexlib.

Обсуждение

Библиотека flexlib содержит компоненты с открытым кодом, созданные разными разработчиками для использования в любых Flex-проектах. Компонент `SuperTabNavigator`, написанный Дугом Мак-Кьюном (Doug McCune), поддерживает закрытие и перестановку вкладок, а также раскрывающийся список в правом верхнем углу компонента.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="600"
xmlns:containers="flexlib.containers.*">
    <containers:SuperTabNavigator>
        <mx:VBox width="500" label="First Label">
            <mx:Label text="Label"/>
            <mx:TextInput/>
        </mx:VBox>
        <mx:VBox height="500" label="Second Label">
            <mx:Label text="Label"/>
            <mx:TextInput/>
        </mx:VBox>
        <mx:VBox label="Third Label">
```

```
            <mx:Label text="Label"/>
            <mx:TextInput/>
        </mx:VBox>
    </containers:SuperTabNavigator>
</mx:Canvas>
```

Библиотека flexlib доступна по адресу *http://code.google.com/p/flexlib/*. Загрузите библиотеку и добавьте распакованную папку в проект, созданный в Flex Builder, или укажите ее в параметрах компилятора MXML.

3.19. Создание и управление Alert

Задача

Требуется создать компонент `Alert` и задать параметры его отображения.

Решение

Используйте метод `show` класса `mx.controls.Alert` с регистрацией функции обратного вызова для закрытия.

Обсуждение

Все свойства компонента `Alert` могут задаваться по статической ссылке на класс `mx.controls.Alert`. Заданные значения продолжают действовать до тех пор, пока не будут заменены другими. Например, после установки свойства `yesLabel` все компоненты `Alert` с надписями `yes` будут использовать заданное значение вплоть до его замены.

При вызове метода `show` класса `Alert` передается сообщение, заголовок, признаки включения кнопок `Yes/No/Cancel`, родительский компонент, функция обратного вызова, активизируемая при закрытии, и отображаемый графический значок. Пример:

```
mx.controls.Alert.show("This is an alert", "title of the alert", 1|2|8,
this, alertClosed, iconclass);
```

Все параметры, кроме первых двух, не являются обязательными. По умолчанию отображается кнопка `OK`, щелчок на которой закрывает окно. Пример приложения с полноценным использованием `Alert`:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Script>
        <![CDATA[

            import mx.events.CloseEvent;
            import mx.controls.Alert;

            [Embed(source="../../assets/4.png")]
            private var iconclass:Class
```

```
            private function createAlert():void
            {
                mx.controls.Alert.buttonHeight = 20;
                mx.controls.Alert.buttonWidth = 150;
                mx.controls.Alert.yesLabel = "Click Yes?"
                mx.controls.Alert.noLabel = "Click No?"
                mx.controls.Alert.cancelLabel = "Click Cancel?"
                mx.controls.Alert.show("This is an alert", "title
                    of the alert", 1|2|8, this, alertClosed, iconclass);
            }

            private function alertClosed(event:CloseEvent):void
            {
                trace(" alert closed ");
                if(event.detail == Alert.YES)
                {
                    trace(" user clicked yes ");
                }
                else if(event.detail == Alert.CANCEL)
                {
                    trace(" user clicked cancle ");
                }
                else
                {
                    trace(" user clicked no ");
                }
            }

        ]]>
    </mx:Script>
    <mx:Button label="Create Simple Alert" click="createAlert()"/>
</mx:Canvas>
```

3.20. Определение размеров и позиции диалогового окна в зависимости от вызывающего компонента

Задача

Требуется создать диалоговое окно с таким же размером и позицией, как у вызывающего компонента.

Решение

Используйте свойство `target` объекта `MouseEvent` для получения информации о компоненте, вызвавшем метод. Для определения фактической ширины, высоты и позиции вызывающего компонента на сцене используется класс `mx.geometry.Rectangle`.

Обсуждение

Чтобы диалоговое окно включалось в приложение и размещалось в правильном месте независимо от выбранного режима раскладки (`absolute`, `horizontal` или `vertical`), вам потребуется компонент `TabNavigator` с закрываемыми вкладками, удаляющий вкладку вместе с соответствующим элементом компонента `TabNavigator`. Задайте свойству `includeInLayout` диалогового окна значение `false`, чтобы приложение не изменяло позицию диалогового окна:

```
dialogue = new Dialogue();
mx.core.Application.application.rawChildren.addChild(dialogue);
dialogue.includeInLayout = false;
```

Метод `getBounds` возвращает прямоугольник с координатами (x,y) и высотой/шириной объекта граничной области объекта `DisplayObject`, переданного методу. В следующем примере при вызове `getBounds` в качестве `DisplayObject` передается вся сцена, так что позиция компонента возвращается в контексте всего приложения.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="700" height="500"
left="50" top="50" verticalGap="50">
    <mx:Script>
        <![CDATA[

            import mx.core.Application;
            import mx.core.UIComponent;

            private var dialogue:Dialogue;

            private function placeLabel(event:MouseEvent):void
            {
                if(dialogue == null)
                {
                    dialogue = new Dialogue();
                    mx.core.Application.application.rawChildren.
                        addChild(dialogue);
                    dialogue.includeInLayout = false;
                }
                var rect:Rectangle = (event.target as UIComponent).
                    getBounds(this.stage);
                dialogue.x = rect.x;
                dialogue.y = rect.y + (event.target
                    as UIComponent).height;
            }

        ]]>
    </mx:Script>
    <mx:HBox horizontalGap="50">
        <mx:Label text="First label."/>
        <mx:Button label="Place First" click="placeLabel(event)"/>
    </mx:HBox>
```

```
    <mx:HBox horizontalGap="50">
        <mx:Label text="Second label."/>
        <mx:Button label="Place Second" click="placeLabel(event)"/>
    </mx:HBox>
    <mx:HBox horizontalGap="50">
        <mx:Label text="Third label."/>
        <mx:Button label="Place Third" click="placeLabel(event)"/>
    </mx:HBox>
    <mx:HBox horizontalGap="50">
        <mx:Label text="Fourth label."/>
        <mx:Button label="Place Fourth" click="placeLabel(event)"/>
    </mx:HBox>
</mx:VBox>
```

3.21. Управление несколькими всплывающими диалоговыми окнами

Задача

Требуется выполнять операции с несколькими диалоговыми окнами, в том числе изменять их.

Решение

Используйте метод `createPopup` класса `PopUpManager`.

Обсуждение

Для выполнения операций с несколькими диалоговыми окнами необходима ссылка на компонент `PopUp`, которую метод `PopUpManager.addPopUp` не предоставляет. Вместо него придется использовать метод `createPopUp` класса `PopUpManager`. Метод возвращает ссылку на созданный объект, которую можно сохранить в массиве. В крупном приложении массив должен быть доступен глобально через открытые статические методы `get/set`, чтобы любой компонент мог при необходимости обратиться к сгенерированным данным. Пример:

```
var pop:Panel = (PopUpManager.createPopUp(this, mx.containers.Panel,
false, PopUpManagerChildList.POPUP) as Panel);
```

При вызове `createPopUp` передается ссылка на родительский контейнер, класс для создания всплывающего окна и флаг модальности; метод возвращает ссылку на созданный объект. Далее при необходимости можно задать свойства объекта.

В следующем примере все всплывающие окна, создаваемые `PopUpManager`, сохраняются в массиве для последующего обращения:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600" height="500"
creationComplete="addDialog()">
    <mx:Script>
```

```
    <![CDATA[

        import mx.managers.PopUpManagerChildList;
        import mx.controls.LinkButton;
        import mx.containers.Panel;
        import mx.managers.PopUpManager;

        public var popUpArray:Array = new Array();

        private function addDialog():void
        {
            var pop:Panel = (PopUpManager.createPopUp(this,
                mx.containers.Panel, false,
                PopUpManagerChildList.POPUP) as Panel);
            pop.title = "First Pop Up";
            pop.y = 100;
            popUpArray.push(pop);
            pop = (PopUpManager.createPopUp(this, mx.containers.Panel,
                false, PopUpManagerChildList.POPUP) as Panel);
            pop.title = "Second Pop Up";
            pop.y = 200;
            popUpArray.push(pop);
            pop = (PopUpManager.createPopUp(this, mx.containers.Panel,
                false, PopUpManagerChildList.POPUP) as Panel);
            pop.title = "Third Pop Up";
            pop.y = 300;
            popUpArray.push(pop);
        }

        private function returnDialog():void
        {
            var link:LinkButton = new LinkButton();
            link.label = "Hello";
            (popUpArray[selectDialog.selectedIndex] as Panel).
                addChild(link);
        }

    ]]>
</mx:Script>
<mx:ComboBox id="selectDialog" change="returnDialog()">
    <mx:dataProvider>
        <mx:Array>
            <mx:Number>0</mx:Number>
            <mx:Number>1</mx:Number>
            <mx:Number>2</mx:Number>
        </mx:Array>
    </mx:dataProvider>
</mx:ComboBox>
<mx:Panel>
    <mx:LinkButton label="Button"/>
</mx:Panel>
```

```
    <mx:Panel>
        <mx:LinkButton label="Button"/>
    </mx:Panel>
</mx:HBox>
```

3.22. Прокрутка контейнера до определенного дочернего компонента

Задача

Требуется изменить стандартный механизм прокрутки контейнера и управлять прокруткой при помощи отдельного элемента.

Решение

Используйте метод `getChildAt` для получения всех дочерних компонентов до индекса того компонента, к которому необходимо прокрутить контейнер; просуммируйте их высоты. Полученное значение задается свойству `verticalScrollPosition` контейнера.

Обсуждение

В следующем примере компонент `VBox` содержит дочерние компоненты, у которых свойству `verticalScrollPolicy` задано значение `off`, а слушатель событий привязывается к свойству `change` компонента `ComboBox`. При срабатывании события `change` функция перебирает все дочерние компоненты `VBox`, суммируя их высоты, пока не будет найден нужный компонент. Полученное значение задается свойству `verticalScrollPosition` компонента `VBox`.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800"
height="600">
    <mx:Script>
        <![CDATA[

            private function showScrollValue():void
            {
                trace(this.verticalScrollPosition+"
                    "+this.horizontalScrollPosition);
            }

            private function changeScrollToShowChild():void
            {
                vbox.verticalScrollPosition =
                    (returnChildrenHeights((comboBox.
                    selectedItem as Number)+1)) - vbox.height;
            }
```

Вычисляемое значение высоты всех дочерних компонентов `VBox` **определяет значение, которое должно быть присвоено** `verticalScrollPosition` **для прокрутки к нужному дочернему компоненту.**

```
            private function returnChildrenHeights(index:int):Number
            {
                var i:int = 0;
                var sumHeight:Number = 0;
                while(i<index)
                {
                    sumHeight+=vbox.getChildAt(i).height;
                    i++;
                }
                return sumHeight;
            }

        ]]>
    </mx:Script>

    <mx:ComboBox id="comboBox" change="changeScrollToShowChild()">
        <mx:dataProvider>
            <mx:Array>
                <mx:Number>1</mx:Number>
                <mx:Number>2</mx:Number>
                <mx:Number>3</mx:Number>
                <mx:Number>4</mx:Number>
                <mx:Number>5</mx:Number>
            </mx:Array>
        </mx:dataProvider>
    </mx:ComboBox>

    <mx:VBox width="650" height="300" id="vbox"
        backgroundColor="#00ffff" y="50" verticalScrollPolicy="off"
        scroll="showScrollValue()" paddingLeft="50">
        <mx:Panel height="150" width="550">
            <mx:LinkButton label="First"/>
        </mx:Panel>
        <mx:Panel height="160" width="550">
            <mx:LinkButton label="Second"/>
        </mx:Panel>
        <mx:Panel height="110" width="550">
            <mx:LinkButton label="Third"/>
        </mx:Panel>
        <mx:Panel height="150" width="550">
            <mx:LinkButton label="Fourth"/>
        </mx:Panel>
        <mx:Panel height="130" width="550">
            <mx:LinkButton label="Fifth"/>
        </mx:Panel>
    </mx:VBox>
</mx:Canvas>
```

3.23. Создание шаблона с использованием IDeferredInstance

Задача

Требуется создать шаблон компонента, которому могут передаваться разные типы компонентов, а также отложить создание дочерних компонентов для ускорения загрузки.

Решение

Используйте маркер `IDeferredInstance`, который указывает, что массив или свойство будет обрабатывать любой переданный ему тип компонента, а дочерние компоненты будут создаваться при создании родителя.

Обсуждение

Интерфейс `IDeferredInstance` реализуется всеми компонентами `UIComponent`; он позволяет включить дочерний компонент в массив и отложить создание его экземпляра на более поздний срок, до создания родительского компонента. Все компоненты, передаваемые массиву с пометкой `IDeferredInstance`, будут храниться в массиве до момента их включения в список отображения, как показано в следующем примере:

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

            import mx.containers.HBox;
            import mx.containers.ViewStack;
            import mx.core.UIComponent;

            // Компонент верхнего уровня
            // реализует интерфейс IDeferredInstance
            public var header:IDeferredInstance;
            // Определение массива для строки компонентов
            [ArrayElementType("mx.core.IDeferredInstance")]
            public var leftDataRow:Array;

            [ArrayElementType("mx.core.IDeferredInstance")]
            public var centerDataRow:Array;

            [ArrayElementType("mx.core.IDeferredInstance")]
            public var rightDataRow:Array;

            public var layoutHBox:HBox;

            public var layoutWidth:int = 0;

            public function createDeferredComponents():void {
                addChild(UIComponent(header.getInstance()));
```

```
                layoutHBox = new HBox();
                if(layoutWidth != 0){
                    layoutHBox.setStyle("horizontalGap", layoutWidth);
                }

                if(leftDataRow.length > 0){
                    var leftVBox:VBox = new VBox();
                    layoutHBox.addChild(leftVBox);
                    for (var i:int = 0; i < leftDataRow.length; i++){
                        leftVBox.addChild(UIComponent(leftDataRow[i].
                        getInstance()));
                    }
                }
                if(centerDataRow.length > 0){
                    var centerVBox:VBox = new VBox();
                    layoutHBox.addChild(centerVBox);
                    for (var i:int = 0; i < centerDataRow.length; i++){
                        centerVBox.addChild(UIComponent(centerDataRow[i].
                        getInstance()));
                    }
                }
                if(rightDataRow.length > 0){
                    var rightVBox:VBox = new VBox();
                    layoutHBox.addChild(rightVBox);
                    for (var i:int = 0; i < rightDataRow.length; i++){
                        rightVBox.addChild(UIComponent(rightDataRow[i].
                        getInstance()));
                    }
                }
                // Включение контейнера HBox в VBox.
                addChild(layoutHBox);
            }
        ]]>
    </mx:Script>
</mx:VBox>
```

Как показывает следующий пример, компоненту можно передать любой массив объектов, реализующих `IDeferredInstance`, а затем вызвать метод для добавления этих дочерних компонентов в `displayList`. В данном примере метод вызывается явно, хотя он также может быть связан с асинхронным или любым другим событием.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="700"
height="500"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[
            private function createDeferredInstance():void
            {
                this.deferredInstance.createDeferredComponents();
            }
        ]]>
```

```
    </mx:Script>
    <mx:Button click="createDeferredInstance()" label="make components"/>
    <cookbook:DeferredInstance layoutWidth="30" id="deferredInstance">
        <cookbook:header>
            <mx:Label text="This will be the header
                of my templated component"/>
        </cookbook:header>
        <cookbook:leftDataRow>
            <mx:Array>
                <mx:Label text="data"/>
                <mx:Label text="data"/>
                <mx:Label text="data"/>
            </mx:Array>
        </cookbook:leftDataRow>
        <cookbook:centerDataRow>
            <mx:Array>
                <mx:Button label="press me"/>
                <mx:Label text="data"/>
            </mx:Array>
        </cookbook:centerDataRow>
        <cookbook:rightDataRow>
            <mx:Array>
                <mx:CheckBox label="click"/>
                <mx:Button label="press me"/>
                <mx:Label text="data"/>
            </mx:Array>
        </cookbook:rightDataRow>
    </cookbook:DeferredInstance>
</mx:Canvas>
```

3.24. Ручное формирование раскладки контейнера

Задача

Требуется сформировать раскладку дочерних компонентов контейнера в соответствии с их типом и свойствами.

Решение

Переопределите метод `updateDisplayList` класса `UIComponent` для перемещения дочерних компонентов.

Обсуждение

Чтобы переопределить логику раскладки или определения размеров для любого объекта `Container` или `UIComponent`, переопределите метод `updateDisplayList` и вставьте свою логику после вызова `super.updateDisplayList`. В этом рецепте основное внимание будет направлено на раскладку до-

черних компонентов HBox, определяемую значениями пользовательских свойств.

Пример состоит из четырех файлов, трех классов и одного интерфейса, решающих разные задачи. Упрощенная диаграмма UML (Universal Modeling Language) на рис. 3.1 описывает связь между ними.

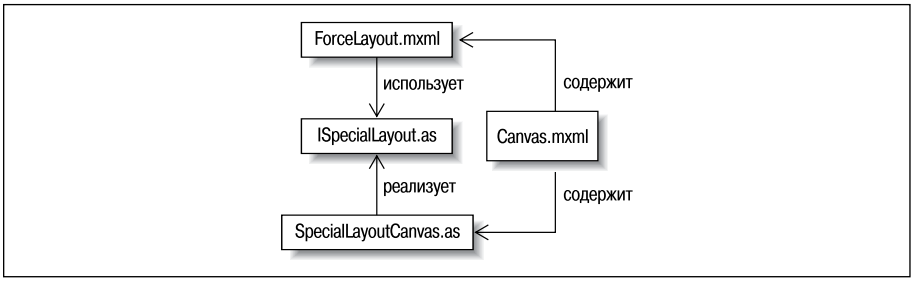

***Рис. 3.1.** Отношения между контейнерами и интерфейсами*

Файл Canvas.mxml добавляет компонент ForceLayout и передает ему дочерние компоненты. Компонент ForceLayout может иметь произвольное количество дочерних компонентов; но если дочерний компонент реализует интерфейс ISpecialLayout, компонент ForceLayout формирует его раскладку в зависимости от того, является ли компонент выделенным. SpecialLayoutCanvas просто определяет методы для проверки того, является ли компонент выделенным.

Начнем с файла SpecialLayoutCanvas.as:

```
package oreilly.cookbook
{
    import mx.containers.Canvas;
    import mx.controls.Label;
    import mx.core.UIComponent;
    import flash.events.MouseEvent;
    public class SpecialLayoutCanvas extends
        Canvas implements ISpecialLayout
    {
        private var titlelabel:Label;
        private var selectedlabel:Label;
        private var _isSelected:Boolean = false;

        public function SpecialLayoutCanvas()
        {
            super();
            titlelabel = new Label();
            addChild(titlelabel);
            titlelabel.text = "Label";
            this.addEventListener(MouseEvent.MOUSE_DOWN, setIsSelected);
            minHeight = 45;
```

```
            minWidth = 80;
            selectedlabel = new Label();
            //addChild(selectedlabel);
            selectedlabel.text = "Selected";
        }

        private function setIsSelected(mouseEvent:MouseEvent):void
        {
            _isSelected ? isSelected = false : isSelected = true;
        }

        public function set isSelected(value:Boolean):void
        {
            _isSelected = value;
            if(isSelected)
            {
                addChild(selectedlabel);
                selectedlabel.y = 30;
            } else {
                try{
                removeChild(selectedlabel);
                }catch(err:Error){}
            }
            if(parent != null)
            {
                (parent as UIComponent).invalidateDisplayList();
            }
        }

        public function get isSelected():Boolean
        {
            return _isSelected;
        }
    }
}
```

Класс просто определяет свойство `selected` **с методами get/set; выделенный объект снабжается надписью, а если объект не выделен – надпись удаляется.**

Компонент `ForceLayout` **проверяет все свои дочерние компоненты, определяя, реализуют ли они интерфейс** `ISpecialLayout`**, и если интерфейс реализуется? находятся ли они в выделенном состоянии:**

```
package oreilly.cookbook
{
    import mx.core.EdgeMetrics;
    import mx.core.UIComponent;
    import mx.containers.VBox;
    import mx.containers.Panel;
    import mx.containers.Canvas;
    import flash.display.DisplayObject;
```

```
public class ForceLayout extends VBox {
    public var gap:Number;
    public function ForceLayout() {
        super();
    }
```

Метод `updateDisplayList` вызывается Flex Framework каждый раз, когда возникает необходимость в перерисовке компонента. Поскольку одной из задач, которую должен решить компонент при перерисовке, является повторное позиционирование всех его дочерних компонентов, вся логика позиционирования выполняется именно здесь:

```
    override protected function updateDisplayList(unscaledWidth:Number,
                                unscaledHeight:Number):void {
        super.updateDisplayList(unscaledWidth, unscaledHeight);
        var yPos:Number = unscaledHeight;
        // Временная переменная для дочернего компонента
        var child:UIComponent;
        var i:int = 0;
        while(i<this.numChildren)
        {
            // Получение первого дочернего компонента для контейнера
            child = UIComponent(getChildAt(i));
            // Определение координаты y дочернего компонента
            yPos = yPos - child.height;
            // Задание координат x и y дочернего компонента. Обратите
            // внимание: координата x остается неизменной.
            if(child is ISpecialLayout)
            {
                if((child as ISpecialLayout).isSelected)
                {
                    yPos -= 20;
                    child.move(child.x, yPos);
                    yPos -= 20;
                }
                else
                {
                    child.move(child.x, yPos);
                }
            }
            else
            {
                child.move(child.x, yPos);
            }
            // Сохранение координаты y дочернего компонента
            // с вертикальным отступом между дочерними компонентами.
            // Значение используется для вычисления координаты
            // следующего дочернего компонента.
            yPos = yPos - gap;
            i++;
        }
```

```
                i = 0;
                var amountToCenter:Number = yPos / 2;
                while(i<this.numChildren)
                {
                    getChildAt(i).y -= amountToCenter;
                    i++;
                }
            }
        }
    }
```

В последнем листинге используются оба компонента: контейнер `ForceLayout` **включается в** `Canvas`**, и в него добавляются дочерние компоненты** `SpecialLayoutCanvas`**. Так как специальные свойства не являются обязательными, а раскладка изменяется при их наличии, в** `ForceLayoutCanvas` **могут добавляться любые типы дочерних компонентов; также могут использоваться любые дочерние компоненты, реализующие интерфейс** `ISpecialLayout`**.**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800"
height="600"
xmlns:cookbook="oreilly.cookbook.*">
    <cookbook:ForceLayout width="400" height="500"
        backgroundColor="#ffffff">
        <mx:HBox>
            <mx:Button label="button"/>
            <mx:LinkButton label="link"/>
        </mx:HBox>
        <cookbook:SpecialLayoutCanvas isSelected="false"
             backgroundColor="#c0c0cc"/>
        <mx:HBox>
            <mx:Button label="button"/>
            <mx:LinkButton label="link"/>
        </mx:HBox>
        <cookbook:SpecialLayoutCanvas isSelected="false"
             backgroundColor="#ccc0c0"/>
        <cookbook:SpecialLayoutCanvas isSelected="true"
             backgroundColor="#cc00cc"/>
        <cookbook:SpecialLayoutCanvas isSelected="false"
             backgroundColor="#ccc0c0"/>
    </cookbook:ForceLayout>
</mx:Canvas>
```

3.25. Вычисление и изменение размеров контейнера

Задача

Требуется изменить размер контейнера на основании области, занимаемой его дочерними компонентами.

Решение

Переопределите метод `measure` контейнера, используемый при вызове метода `updateDisplay` из Flex Framework.

Обсуждение

Размер контейнера определяется методом `measure`. Этот метод вызывается каждый раз, когда Flex Framework требуется определить размер дочернего компонента с учетом всех стилей и ограничений. По аналогии с тем, как мы переопределяли метод `updateDisplayList` в рецепте 3.24, метод `measure` тоже можно переопределить для выполнения любых нестандартных действий, задействованных при определении размеров.

Сначала в файле MXML определяется класс `ExtendableCanvas`:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.core.Container;
            import mx.core.UIComponent;

            override protected function measure():void
            {
                super.measure();
                var childrenWidth:int = 0;
                var childrenHeight:int = 0;
                // Перебор всех дочерних компонентов с определением
                // ширины и высоты каждого компонента.
                for(var i:int = 0; i<this.numChildren; i++)
                {
                    var obj:UIComponent = (getChildAt(i) as UIComponent);
                    if(obj is Container)
                    {
                        // Использование viewMetricsAndPadding
                        // позволяет получить всю стилевую информацию,
                        // связанную с дочерним компонентом,
                        // вместе с его фактической шириной.
                        childrenWidth += Container(obj).
                            viewMetricsAndPadding.left+Container(obj).
                            viewMetricsAndPadding.right+obj.width;
                        childrenHeight += Container(obj).
                            viewMetricsAndPadding.top+Container(obj).
                            viewMetricsAndPadding.bottom+obj.height;
                    }
                    else
                    {
                        childrenWidth += obj.width;
                        childrenHeight += obj.height;
                    }
                }
                // Компоненту задается высота и ширина,
```

```
                // полученные на основании вычислений.
                measuredHeight = childrenHeight;
                measuredWidth = childrenWidth;
            }

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

Затем файл ExtendableCanvas.mxml используется в Canvas, и в него включаются дочерние компоненты:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1400"
height="500"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[

            import mx.containers.Panel;

            private function addChildToExtendableCanvas():void
            {
                var panel:Panel = new Panel();
                panel.height = 100 + Math.random()*200;
                panel.width = 100 + Math.random()*200;
                extCanvas.addChild(panel);
            }
        ]]>
    </mx:Script>
    <mx:Button click="addChildToExtendableCanvas()" label="add child"/>
    <cookbook:ExtendableCanvas id="extCanvas" y="50"
        verticalScrollPolicy="off" horizontalScrollPolicy="off"/>
</mx:Canvas>
```

По мере включения в Container новых дочерних компонентов экземпляр ExtendableCanvas перерисовывается и изменяет размеры по ширине и высоте всех дочерних компонентов с добавлением отступов, указанных в стилевой информации.

3.26. Управление видимостью и раскладкой дочерних компонентов

Задача

Требуется исключить дочерние компоненты из раскладки контейнера без их уничтожения.

Решение

Используйте свойство includeInLayout класса UIComponent и сделайте компонент невидимым.

Обсуждение

Свойство `includeInLayout` дочернего компонента контейнера указывает, должен ли этот компонент включаться в схему раскладки, используемую родителем: `VBox`, `HBox` или `Canvas`. Если просто отключить видимость дочернего компонента, он все равно будет включаться в раскладку родителя, и для него будет резервироваться место, определяемое его шириной и высотой. Вместо этого следует задать свойству `includeInLayout` значение `false`, чтобы компонент `HBox` не выделял место для дочернего компонента:

```
(event.target as UIComponent).includeInLayout = false;
```

Чтобы снова включить дочерний компонент в раскладку, задайте свойству `includeInLayout` значение `true`. В следующем примере свойство `includeInLayout` используется при перетаскивании дочернего элемента. При включении дочернего компонента в раскладку компонент `HBox` изменит его позицию так, чтобы она соответствовала раскладке. Поскольку свойство `includeInLayout` ложно, дочерний компонент будет проигнорирован при задании компонентом `HBox` координаты x всех его дочерних компонентов:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="400">
    <mx:Script>
        <![CDATA[

            import mx.core.UIComponent;

            private function removeFromLayout(event:MouseEvent):void
            {
                (event.target as UIComponent).includeInLayout = false;
                (event.target as UIComponent).startDrag();
            }

            private function reincludeInLayout(event:MouseEvent):void
            {
                (event.target as UIComponent).stopDrag();
                (event.target as UIComponent).includeInLayout = true;
            }
            private function hideAndRemoveFromLayout(event:Event):void
            {
                (event.target as UIComponent).visible = false;
                (event.target as UIComponent).includeInLayout = false;
            }
        ]]>
    </mx:Script>

    <mx:VBox>
        <mx:LinkButton id="firstLabel"
            label="this is the first label"
```

```
            mouseDown="removeFromLayout(event)"
            mouseUp="reincludeInLayout(event)"/>
        <mx:LinkButton id="secondLabel"
            label="this is the second label"
            mouseDown="removeFromLayout(event)"
            mouseUp="reincludeInLayout(event)"/>
        <mx:Button id="button"
            label="My First Button"
            mouseDown="removeFromLayout(event)"
            mouseUp="reincludeInLayout(event)"/>
    </mx:VBox>

    <mx:VBox>
        <mx:Button label="Remove from Layout and Hide"
            click="hideAndRemoveFromLayout(event)"
            borderColor="#0000ff"/>
        <mx:Button label="Remove from Layout and Hide"
            click="hideAndRemoveFromLayout(event)"
            borderColor="#00ff00"/>
        <mx:Button label="Remove from Layout and Hide"
            click="hideAndRemoveFromLayout(event)"
            borderColor="#ff0000"/>
    </mx:VBox>
</mx:HBox>
```

3.27. Создание контейнера Tile с простой реструктуризацией

Задача

Требуется предоставить пользователям возможность перетаскивания плиток в контейнере `Tile`, чтобы контейнер автоматически изменял структуру содержимого при отпускании плитки.

Решение

Используйте метод `swapChildren` класса `DisplayObjectContainer`, расширяемого `Tile`, для изменения позиций дочерних компонентов внутри `Tile`.

Обсуждение

Контейнер `Tile` формирует раскладку плиток практически так же, как контейнер `Box` (не считая возможности добавления нескольких `itemRenderer`). Скажем, если свойству `direction` контейнера `Tile` задано значение `horizontal`, ширина `Tile` составляет 400 пикселов, а добавляемые плитки имеют ширину 150 пикселов, контейнер размещает по два компонента в строке, а при необходимости включит вертикальные полосы прокрутки. Плитки всегда образуют структуру в виде решетки (рис. 3.2).

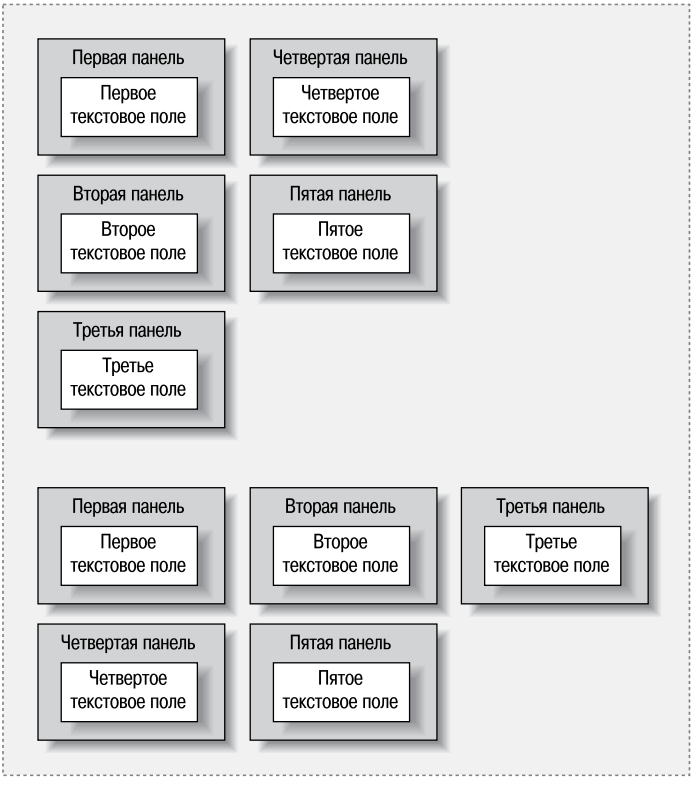

Рис. 3.2. *Вертикальное и горизонтальное размещение плиток*

В следующем примере компонент `Tile` используется для автоматического размещения дочерних компонентов.

При отпускании дочернего компонента после перетаскивания для `Tile` устанавливается флаг перерисовки, для чего вызывается метод `invalidateDisplayList`:

```
<mx:Tile xmlns:mx="http://www.adobe.com/2006/mxml" width="300" height="600"
direction="horizontal">
    <mx:Script>
        <![CDATA[

            import mx.core.UIComponent;
            private function childStartDrag(event:Event):void
            {
                (event.currentTarget as UIComponent).startDrag(false,
                    this.getBounds(stage));
                (event.currentTarget as UIComponent).
                    addEventListener(MouseEvent.MOUSE_UP,
                    childStopDrag);
                (event.currentTarget as UIComponent).
```

```
                addEventListener(MouseEvent.ROLL_OUT,
                childStopDrag);
            swapChildren((event.currentTarget as UIComponent),
                getChildAt(numChildren-1));
        }

        private function childStopDrag(event:Event):void
        {
            swapChildren((event.currentTarget as UIComponent),
                hitTestChild((event.currentTarget as UIComponent)));
            (event.currentTarget as UIComponent).stopDrag();
            this.invalidateDisplayList();
            this.invalidateProperties();
        }

        private function hitTestChild(obj:UIComponent):DisplayObject
        {
            for(var i:int = 0; i<this.numChildren; i++)
            {
                if(this.getChildAt(i).hitTestObject(obj))
                {
                    return getChildAt(i);
                }
            }
            return getChildAt(0)
        }

    ]]>
</mx:Script>
<mx:Panel title="First Panel" mouseDown="childStartDrag(event)">
    <mx:Text text="First Text"/>
</mx:Panel>
<mx:Panel title="Second Panel" mouseDown="childStartDrag(event)">
    <mx:Text text="Second Text"/>
</mx:Panel>
<mx:Panel title="Third Panel" mouseDown="childStartDrag(event)">
    <mx:Text text="Third Text"/>
</mx:Panel>
<mx:Panel title="Fourth Panel" mouseDown="childStartDrag(event)">
    <mx:Text text="Fourth Text"/>
</mx:Panel>
</mx:Tile>
```

3.28. Фоновый рисунок и закругленные углы в HBox

Задача

Требуется создать контейнер `HBox` с закругленными углами и графическим изображением, назначенным в качестве фонового рисунка.

Решение

Загрузите объект графического изображения и создайте заливку методом `beginBitmapFill`.

Обсуждение

Если фоновый рисунок `HBox` не является графическим изображением, для создания закругленных углов у контейнера `HBox` достаточно задать свойство `cornerRadius`. Но если в качестве фона `HBox` выбрано графическое изображение, как в следующем примере:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
backgroundImage="../../assets/beach.jpg" borderStyle="solid"
borderThickness="0" cornerRadius="20">
```

углы `HBox` будут прямоугольными, как углы изображения. Чтобы закруглить их, следует использовать графическое изображение как заливку. Все компоненты `UIComponent` обладают свойство `graphics`; его значение представляет собой экземпляр объекта `flash.display.Graphic` с низкоуровневыми графическими примитивами. Метод `beginBitmapFill` позволяет создать заливку для вызова сложного метода `drawRoundRect`, заполняющего закругленный прямоугольник битами загруженного изображения. Вызов метода `endFill` завершает построение заливки:

```
this.graphics.beginBitmapFill(bm, m, true, true);
    this.graphics.drawRoundRectComplex(0, 0, this.width, this.height, 20,
        20, 20, 20);
    this.graphics.endFill();
```

После загрузки изображения графические данные прорисовываются на объекте `BitmapData`:

```
var bm:BitmapData = new BitmapData(loader.width, loader.height,
    true, 0x000000);
bm.draw(this.loader);
```

Затем остается лишь использовать объект `BitmapData` для создания заливки. Полный код примера выглядит так:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="createFill()" cornerRadius="20">
    <mx:Script>
        <![CDATA[

            import flash.net.URLRequest;

            private var loader:Loader;

            private function createFill():void
            {
                loader = new Loader();
                loader.contentLoaderInfo.addEventListener(
```

```
                Event.COMPLETE, completeLoad);
            loader.load(new URLRequest("../../assets/beach.jpg"));
        }

        private function completeLoad(event:Event):void
        {
            var bm:BitmapData = new BitmapData(loader.width,
                loader.height, true,0x000000);
            bm.draw(this.loader);
            var m:Matrix = new Matrix();
            m.createBox(this.width/loader.width,
                this.height/loader.height);
            this.graphics.beginBitmapFill(bm, m, true, true);
            this.graphics.drawRoundRectComplex(0, 0, this.width,
                this.height, 20, 20, 20, 20);
            this.graphics.endFill();
        }

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

3.29. Управление позиционированием и прокруткой дочерних компонентов

Задача

Требуется прокрутить родительский компонент и переместить все дочерние компоненты, кроме одного.

Решение

Измените позицию дочернего компонента на основании `verticalScrollPosition` в коде метода `scrollChidren`, определяемого контейнером.

Обсуждение

Метод `scrollChidren` контейнера вычисляет размеры `contentPane` контейнера, вычисляет размеры `DisplayObject` со всеми дочерними компонентами, добавленными в контейнер, и определяет, какая часть вычисленной области будет отображаться контейнером в ходе прокрутки. Перемещение `contentPane` осуществляется с учетом `horizontalScrollPosition` и `verticalScrollPosition`. Сам контейнер выполняет функции «маски» для `contentPane`, а позиция `contentPane` определяется позицией соответствующей полосы прокрутки и свойствами `ViewMetrics` контейнера.

Следующий фрагмент кода сохраняет координату y компонента `top`; это позволяет пользователю перетащить компонент так, чтобы компонент `top` остался в положенном месте, а положение всех остальных компонентов определялось стандартным образом:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="500"
height="500">
    <mx:Script>
        <![CDATA[

        private var top:TopComponent;
        // Сохранение позиции y
        private var topY:Number;

        private function showTop():void
        {
            top = new TopComponent();
            addChild(top);
            top.verticalScrollPolicy = "none";
            top.x = 200;
            top.y = 100;
            topY = top.y;
            top.addEventListener(MouseEvent.MOUSE_DOWN, dragTop);
        }

        private function dragTop(event:Event):void
        {
            top.startDrag(false, this.getBounds(stage));
            top.addEventListener(MouseEvent.MOUSE_UP, stopDragTop);
        }

        private function stopDragTop(event:Event):void
        {
            topY = top.y;
            top.stopDrag();
            top.removeEventListener(MouseEvent.MOUSE_UP, stopDragTop);
        }

        override protected function scrollChildren():void
        {
            super.scrollChildren();
            if(top){
                top.y = verticalScrollPosition+topY;
                //
                top.verticalScrollPosition = this.verticalScrollPosition/
                    height *top.height;
            }
        }
    ]]>
    </mx:Script>
    <mx:Panel>
        <mx:Label text="LABEL BABEL"/>
        <mx:Label text="LABEL BABEL"/>
        <mx:Label text="LABEL BABEL"/>
    </mx:Panel>
    <mx:Panel y="500" height="200">
```

```
            <mx:Label text="LABEL BABEL"/>
            <mx:Label text="LABEL BABEL"/>
            <mx:Label text="LABEL BABEL"/>
        </mx:Panel>
        <mx:Button click="showTop()"/>
    </mx:Canvas>
```

Пример получился немного упрощенным, однако аналогичная логика может использоваться для реализации пользовательской прокрутки всех дочерних компонентов контейнера. Если известно, что прокручиваться будут все дочерние компоненты в контейнере, метод `scrollChildren` можно переопределить следующим образом:

```
private var dir:String;
private function handleScroll(event:ScrollEvent):void {
    dir = event.direction;
}

override protected function scrollChildren():void {
    var i:int = 0;
    do{
        var comp:DisplayObject = getChildAt(i);
        if(comp is Container){
            trace( Container(comp).maxVerticalScrollPosition );
        dir == "horizontal" ?
        Container(comp).horizontalScrollPosition =
            horizontalScrollPosition *
            (Container(comp).maxHorizontalScrollPosition/
            maxHorizontalScrollPosition) :
        Container(comp).verticalScrollPosition = verticalScrollPosition *
            (Container(comp).maxVerticalScrollPosition/
            axVerticalScrollPosition);
        }
        i++;
    } while (i < numChildren)
}
```

В этом примере свойства `maxVerticalScrollPosition` и `maxHorizontalScrollPosition` используются для вычисления правильной величины прокрутки. Контейнер определяет эти свойства, вычитая фактическую высоту и ширину контейнера из высоты и ширины всех содержащихся в нем дочерних компонентов. В предыдущем фрагменте кода значения `maxVerticalScrollPosition` и `maxHorizontalScrollPosition` каждого дочернего компонента делятся на родительские значения, чтобы определить величину прокрутки для дочернего компонента. Это гарантирует, что даже если родитель меньше или больше дочернего компонента, последнему будет передана правильная величина.

4

Текст

Для работы с текстом в приложениях Flex используются следующие компоненты: `mx.text.Text`, `mx.text.RichTextEditor`, `mx.text.Label`, `mx.text.TextArea`, `mx.text.TextInput` и `flash.text.TextField`. Все они выполняют разные функции в контексте приложения Flex. Так, элементы `TextInput`, `TextArea` и `RichTextEditor` обеспечивают взаимодействие с пользователем и редактирование. `TextArea`, `RichTextEditor` и `Text` поддерживают работу с многострочным текстом. Наконец, низкоуровневый класс `flash.text.TextField` предоставляет средства для точного управления форматированием и выполнения операций с текстом в `TextField`, но требует, чтобы в приложении Flex использовался содержащий его экземпляр `UIComponent`. На практике все компоненты Flex в пакете `mx.text` используют `flash.text.TextField`, дополняя его разной функциональностью.

Flex поддерживает вывод простого текста и подмножества HTML; для управления внешним видом текста используется как текстовое форматирование, так и стили CSS. При использовании подмножества HTML, поддерживаемого Flash Player, для загрузки контента могут быть задействованы графические файлы и другие файлы SWF. Текстовое форматирование (т. е. управление размером и цветом шрифта) осуществляется заданием свойств компонентов `mx.text` средствами CSS, или для компонента `flash.text.TextField` – использованием объекта `flash.text.TextFormat`. Выделение текста осуществляется пользователем или задается на программном уровне методом `setSelection`. В рецептах этой главы представлены все шесть текстовых компонентов.

4.1. Правильное задание значения объекта Text

Задача

Требуется обеспечить правильное отображение HTML и простых строк, передаваемых объекту `Text`.

Решение

Используйте свойства `htmlText` и `text` (в зависимости от типа входных данных) для правильного отображения текста. Проанализируйте строку, передаваемую объекту `Text`, при помощи регулярного выражения.

Обсуждение

Чтобы компоненты `Text` и `TextArea` корректно отображали форматирование HTML, код HTML должен быть передан свойству `htmlText` компонента `Text` или `TextArea`. Обычно передача компоненту `Text` или `TextArea` в формате, отличном от HTML, не создает проблем, если только этот текст не содержит специальных символов HTML.

Регулярные выражения – мощный инструмент для быстрого и эффективного разбора текста по шаблону без утомительных манипуляций со строками. Следующее выражение ищет символ <, за которым следует произвольное количество алфавитных символов, завершаемое символом >:

```
var regexp:RegExp = /<.+\w.>/;
```

В следующем примере регулярное выражение проверяет, содержит ли строка, передаваемая компоненту `Text`, код HTML или XML:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[

            private var htmlStr1:String =
            '<b>Header</b><br/>Hello.<i>Hello.</i>
                <font color="#ff0000" size="15">RED</font>';
            private var htmlStr2:String =
            "<ul><li>Item 1</li><li>Item 2</li><li>Item 3</li></ul>";
            private var textStr1:String = "It is a long established
                fact that a reader will be distracted
                by the readable content of a page when looking at
                its layout, if say the amount of text > 100.";
            private var textStr2:String =
                " We can use <<String>> to indicate
                in Erlang that the values being passed are Strings";

            private function setNewText():void
            {
                determineTextType(
                changeText.selectedItem.value.toString());
            }

            private function determineTextType(str:String):void
            {
                var regexp:RegExp = /<.+\w.>/;
```

```
                if(regexp.test(str))
                {
                    textArea.htmlText = str;
                }
                else
                {
                    textArea.text = str;
                }
            }

        ]]>
    </mx:Script>
    <mx:ComboBox id="changeText" dataProvider="{[{label:'HTML1',
        value:htmlStr1}, {label:'HTML2', value:htmlStr2},
        {label:'Text1', value:textStr1}, {label:'Text2',
        value:textStr2}]}" change="setNewText()"/>
    <mx:TextArea id="textArea" height="100%"/>
</mx:VBox>
```

4.2. Привязка TextInput

Задача

Требуется привязать данные, введенные пользователем в компоненте `TextInput`, к другому компоненту.

Решение

Используйте теги привязки, чтобы связать текст компонента `TextInput` с компонентом `Text`, в котором будут отображаться введенные данные.

Обсуждение

Компонент `TextInput` в следующем примере используется для ввода текста, который будет отображаться в компоненте `TextArea`. С увеличением объема текста ширина `TextArea` увеличивается благодаря механизму привязки Flex Framework:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:TextInput id="input" width="200"/>
    <mx:TextArea text="{input.text}" width="200"
        id="area" backgroundAlpha="0" height="{(Math.round(input.text.
        length/40)+1) * 20}" wordWrap="true"/>
</mx:VBox>
```

Компонент `TextInput` можно связать как с переменной, так и с ее значением. Это не приведет к автоматическому изменению переменной, к которой привязано свойство `text` компонента `TextInput`, в случае ввода текста пользователем, но изменит свойство `text` любого компонента, привязанного к `TextInput`. Пример:

```
<mx:Script>
    <![CDATA[

        [Bindable]
        private var bindableText:String = "Zero Text";

        private function setText():void
        {
            bindableText = String(cb.selectedItem);
        }
    ]]>
</mx:Script>
<mx:ComboBox id="cb" dataProvider="{['First Text', 'Second Text',
    'Third Text', 'Fourth Text']}" change="setText()"/>
<mx:TextInput id="inputFromCB" width="200" text="{bindableText}"/>
<mx:Text id="textFromCB" width="200" text="{inputFromCB.text}"/>
```

4.3. Вывод рекомендаций при заполнении TextInput

Задача

Требуется создать компонент `TextInput`, который читает данные из словаря и предлагает пользователю возможные варианты в процессе ввода.

Решение

Используйте событие `change` компонента `TextInput` для прослушивания пользовательского ввода. Поиск совпадений в словаре осуществляется при помощи регулярного выражения.

Обсуждение

Компонент `TextInput` определяет событие `change`, передаваемое при каждом изменении значения компонента. Это событие может использоваться для проверки ввода и поиска в небольшом словаре. Пример:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Script>
        <![CDATA[

            private var allWords:Array = ["apple", "boy", "cat",
               "milk", "orange","pepper", "recipe", "truck"];
            private var regexp:RegExp;

            private function checkInput():void
            {
                var i:int = 0;
                var temp:Array = allWords.filter(filter);
```

```
                input.text = temp[0];
            }

            private function filter(element:*, index:int, arr:Array):Boolean
            {
                regexp = new RegExp(input.text);
                return (regexp.test(element as String));
            }

        ]]>
    </mx:Script>

    <mx:TextInput id="input" change="checkInput()"/>
</mx:Canvas>
```

Использованная в примере функция `filter` входит в класс `Array`. Она позволяет создать метод, который получает произвольный объект, и после некоторой обработки возвращает логический признак включения флага в отфильтрованный массив. Массив `temp`, созданный в методе `checkInput`, включает все элементы, прошедшие проверку по регулярному выражению.

4.4. Редактирование «на месте»

Задача

Требуется создать компонент, который предоставляет возможность редактирования, когда пользователь щелкнет на тексте.

Решение

Используйте прослушивание события `click` компонента `Text` для изменения состояния компонента с отображением `TextInput`. Используйте события `enter` и `focusOut` компонента `TextInput` для определения момента завершения редактирования и возвращения к компоненту `Text` посредством смены состояния.

Обсуждение

Состояния – мощный и удобный механизм создания нескольких представлений для одного компонента. В примере данного рецепта используются два состояния: `display` и `edit`. В состоянии `display` присутствует компонент `Label` с текущим текстом, а в состоянии `edit` – компонент `TextInput` для его редактирования.

Чтобы сменить состояние компонента, задайте свойству `currentState` строку с именем нужного состояния. Пример:

```
currentState = "display";
```

Чтобы введенные данные были сохранены после того, как пользователь нажмет клавишу Enter (или в стороне от `TextInput`**), компонент** `TextInput` **передает фокус самому себе при создании и прослушивает события** `enter` **и** `focusOut` **для вызова** `changeState`**.**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="250"
height="40" top="10" currentState="display">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var value:String;

            private function changeState(event:Event = null):void
            {
                if(this.currentState == "display")
                {
                    currentState = "edit";
                }
                else
                {
                    value = editInput.text;
                    currentState = "display";
                }
            }

        ]]>
    </mx:Script>
    <mx:states>
        <mx:State id="display" name="display">
            <mx:AddChild>
                <mx:Label text="{value}" id="text" x="{editorValue.x +
                editorValue.width}" click="changeState()" minWidth="100"
                minHeight="20"/>
            </mx:AddChild>
        </mx:State>
```

Когда свойство `currentState` **компонента переходит в состояние** `edit`**, свойству** `text` **компонента** `TextInput` **задается значение** `Label` **из состояния** `display`**. После того как пользователь нажмет клавишу Enter, компонент возвращается к состоянию** `display`**, а текущее содержимое** `TextInput` **используется для задания содержимого** `Label` **в состоянии** `display`**. Событие** `enter` **компонента** `TextInput` **означает, что пользователь нажал клавишу Enter или Return в то время, пока фокус ввода принадлежал компоненту.**

```
        <mx:State id="edit" name="edit">
            <mx:AddChild>
                <mx:TextInput creationComplete="editInput.setFocus()"
                    focusOut="changeState()" id="editInput"
                    x="{editorValue.x + editorValue.width}"
```

```
                    text="{value}" minWidth="100" minHeight="20"
                enter="changeState()"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>
    <mx:Label text="Value: " id="editorValue"/>
</mx:Canvas>
```

4.5. Получение списка всех шрифтов

Задача

Требуется получить список всех шрифтов, установленных на компьютере пользователя, и разрешить пользователю выбрать один из них для отображения текста в компоненте `Text`.

Решение

Используйте метод `enumerateFonts`, определяемый классом `Font`. Задайте стилю `fontFamily` компонента `Text` значение свойства `fontName` выбранного шрифта.

Обсуждение

Класс `Font` определяет статический метод `enumerateFonts`, который возвращает перечень всех системных шрифтов, обнаруженных на компьютере пользователя. Метод возвращает массив объектов `flash.text.Font`, определяющих три свойства:

`fontName`

Название шрифта, полученное от системы. В некоторых случаях (например, для знаков японского, корейского или арабского языка) возможно некорректное воспроизведение названия шрифта в Flash Player.

`fontStyle`

Начертание шрифта: обычный, полужирный, курсив, полужирный курсив.

`fontType`

Допустимые значения – `Device` (означает, что шрифт установлен на компьютере пользователя) и `Embedded` (шрифт встроен в файл SWF).

В следующем примере список шрифтов используется для заполнения компонента `ComboBox`, в котором пользователь выбирает тип шрифта для отображения `Text`. Вызов `setStyle`

```
text.setStyle("fontFamily", (cb.selectedItem as Font).fontName);
```

назначает выбранный шрифт для компонента `Text` по свойству `fontName` объекта `Font`, выбранного в `ComboBox`.

Полный код решения этой задачи:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="findAllFonts()">
    <mx:Script>
        <![CDATA[

            private var style:StyleSheet;

            [Bindable]
            private var arr:Array;

            private function findAllFonts():void
            {
                arr = Font.enumerateFonts(true);
                arr.sortOn("fontName", Array.CASEINSENSITIVE);
            }

            private function setFont():void
            {
                text.setStyle("fontFamily",
                    (cb.selectedItem as Font).fontName);
            }
        ]]>
    </mx:Script>
    <mx:ComboBox id="cb" dataProvider="{arr}" change="setFont()"
        labelField="fontName"
/>
    <mx:Text text="Sample Text" id="text" fontSize="16"/>
</mx:VBox>
```

4.6. Создание пользовательской версии TextInput

Задача

Требуется создать пользовательскую версию компонента `TextInput`.

Решение

Используйте `UIComponent` и добавьте к нему метод `flash.text.TextField`. Затем привяжите свойство `htmlText` компонента `Text` к свойству `text` нового компонента.

Обсуждение

Компонент `mx.controls.TextInput` ограничивает доступ к внутреннему компоненту `flash.text.TextField`. Чтобы иметь более полный доступ к `TextField`, просто добавьте его к `UIComponent`.

Все методы, необходимые для предоставления доступа к `TextField`, должны быть определены в компоненте. Если необходим неограниченный

доступ к `TextField`**, возможно, будет проще определить** `TextField` **как открытое свойство. Кроме того, компонент желательно оповещать об изменении свойств** `TextField` **или обращениях к ним. Рассмотрим возможную практическую реализацию:**

```
package oreilly.cookbook
{
    import flash.events.Event;
    import flash.events.TextEvent;
    import flash.text.TextField;
    import flash.text.TextFieldType;

    import mx.core.UIComponent;

    public class SpecialTextInput extends UIComponent
    {
        private var textInput:TextField;
        public static var TEXT_CHANGED:String = "textChanged";
        public function SpecialTextInput()
        {
            textInput = new TextField();
            textInput.multiline = true;
            textInput.wordWrap = true;
            textInput.type = flash.text.TextFieldType.INPUT;
            textInput.addEventListener(TextEvent.TEXT_INPUT, checkInput);
            addChild(textInput);
            super();
        }
        private function checkInput(textEvent:TextEvent):void
        {
            text = textInput.text;
        }

        override public function set height(value:Number):void
        {
            textInput.height = this.height;
            super.height = value;
        }

        override public function set width(value:Number):void
        {
            textInput.width = this.width;
            super.width = value;
        }

        [Bindable(event="textChanged")]
        public function get text():String
        {
            return textInput.text;
        }
        public function set text(value:String):void
```

```
            {
                dispatchEvent(new Event("textChanged"));
            }
        }
    }
```

Чтобы воспользоваться новым компонентом, привяжите свойство `htmlText` компонента `Text` к свойству `text` нового компонента:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
         xmlns:cookbook="oreilly.cookbook.*">
    <cookbook:SpecialTextInput id="htmlInput" height="200" width="300"/>
    <mx:TextArea htmlText="{htmlInput.text}"/>
</mx:VBox>
```

См. также

Рецепт 1.18.

4.7. Задание стилевого оформления для текстовых блоков

Задача

Требуется задать шрифтовые атрибуты для текстового блока без использования HTML.

Решение

Используйте класс `TextRange` для задания свойств диапазона символов.

Обсуждение

Класс `TextRange` получает компонент, содержащий `TextField`, параметр, который указывает, должны ли свойства, заданные в объекте `TextRange`, использоваться для модификации компонента, а также два целых числа, определяющих начало и конец текстового блока в `TextField`. Пример конструирования объекта `TextRange`:

```
var textRange:TextRange = new TextRange(component:UIComponent,
modify:Boolean,
startIndex:int, endIndex:int);
```

Свойства объекта `TextRange` отражаются на переданном ему компоненте. В следующем примере при установке флажка устанавливается цвет символов и межсимвольные интервалы в области, выбранной пользователем:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.controls.textClasses.TextRange;
```

```
            private function alterTextSnapshot():void
            {
                var textRange:TextRange = new TextRange(area, true,
                    area.selectionBeginIndex, area.selectionEndIndex);
                textRange.color = 0xff0000;
                textRange.letterSpacing = 3;
            }

        ]]>
    </mx:Script>
    <mx:CheckBox change="alterTextSnapshot()"/>
    <mx:TextArea id="area" text="Lorem ipsum dolor sit amet,
        consectetuer adipiscingelit." width="200" height="50"/>
</mx:VBox>
```

4.8. Отображение графики и SWF в HTML

Задача

Требуется использовать графические изображения и внешние файлы SWF в тексте HTML, отображаемом в компоненте Flex.

Решение

Используйте тег `<img>`, поддерживаемый ядром визуализации HTML Flash Player. Укажите в атрибуте `src` URL-адрес загружаемого файла SWF или графического файла.

Обсуждение

Тег `<img>` позволяет указать местоположение графического файла или файла SWF, загружаемого и отображаемого в компоненте `Text`. Тег `<img>` поддерживает следующие атрибуты:

`src`

URL-адрес файла в формате GIF, JPEG, PNG или SWF; только этот атрибут является обязательным. Остальные атрибуты управляют размещением изображения по отношению к окружающему его тексту.

`align`

Способ горизонтального выравнивания встроенного изображения в текстовом поле. Допустимые значения: `left` и `right`. По умолчанию используется значение `left`.

`height`

Высота изображения в пикселах.

`hspace`

Горизонтальные поля (не содержащие текста), окружающие изображение. По умолчанию значение `hspace` равно 8.

vspace

Вертикальные поля, окружающие изображение. По умолчанию значение vspace равно 8.

width

Ширина изображения в пикселах.

В следующем фрагменте кода файл SWF загружается в приложение при помощи тега HTML <img> с атрибутом vspace, равным 10; таким образом, у верхнего и нижнего края изображения будут отображаться поля высотой 10 пикселов.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
                width="400" height="300">
    <mx:TextArea width="300" height="300" backgroundAlpha="0">
        <mx:htmlText>
            <![CDATA[
                <img src='../assets/fries.jpg' width='100' height='100'
                    align='left' hspace='10' vspace='10'>
                <p>This is the text that is going to appear above
                    the swf.</p><p> <img src='../assets/test_swf.swf'
                    width='100' height='100' align='left' hspace='10'
                    vspace='10'>
                Here is text that is going to be below the image.</p>
            ]]>
        </mx:htmlText>
    </mx:TextArea>
</mx:Canvas>
```

4.9. Выделение текста, введенного пользователем в поле поиска

Задача

Требуется создать компонент TextArea для поиска, с визуальным выделением текста, введенного пользователем в TextInput.

Решение

Используйте объект flash.text.TextField и задайте свойство alwaysShowSelection=true. Затем воспользуйтесь методом setSelection для задания индекса и длины выделенного текста.

Обсуждение

Для отображения выделенного текста компонент mx.controls.TextArea должен обладать фокусом. Чтобы обойти это ограничение, можно создать субкласс компонента TextArea, предоставляющий доступ к содержащемуся в TextArea экземпляру flash.text.TextField:

```
public function createComp():void{
    textField.alwaysShowSelection = true;
}
```

Задание истинного значения свойству `alwaysShowSelection` **означает, что в объекте** `TextField` **выделенный текст будет отображаться независимо от наличия фокуса. Теперь при вызове метода** `setSelection` **компонент** `TextArea` **автоматически прокручивается к выделению:**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1000"
height="800"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var text_string:String = "Aenean quis nunc id purus
                pharetra pharetra. Cras a felis sit amet ipsum ornare
                luctus. Nullam scelerisque" + " placerat velit.
                Pellentesque ut arcu congue risus facilisis pellentesque.
                Duis in enim. Mauris eget est. Quisque tortor. ";
            private function searchText():void
            {
                var index:int = masterText.text.indexOf(input.text);
                masterText.verticalScrollPosition = 0;
                if(index != -1)
                {
                    masterText.setSelection(index, index+input.text.length);
                }
            }

        ]]>
    </mx:Script>
    <mx:TextInput id="input" change="searchText()"/>
    <cookbook:SpecialTextArea editable="false" id="masterText"
        text="{text_string}" fontSize="20" width="600"
        height="200" x="200"/>
</mx:Canvas>
```

4.10. Работа с отдельными символами

Задача

Требуется выполнять операции с отдельными символами как с графическими объектами для создания визуальных эффектов.

Решение

Используйте метод `getCharBoundaries` **для получения фактической высоты, ширины и координат символа в** `TextField`**. Создайте на базе** `TextField` **объект растрового изображения с нужным символом.**

Обсуждение

Метод getCharBoundaries возвращает с описанием координат, ширины и высоты символа с заданным индексом в TextField. Полученная информация может использоваться для создания растрового изображения символа с сохранением всей визуальной информации и его анимации. Ключевую роль в этом процессе занимает следующий фрагмент:

```
char_bounds = addCharacters.getTextField().getCharBoundaries(i);
bitmapData=new BitmapData(char_bounds.width, char_bounds.height, true, 0);
matrix = new Matrix(1, 0, 0, 1, -char_bounds.x, char_bounds.y);
bitmapData.draw(addCharacters.getTextField(), matrix, null, null,
    null, true);
bitmap = new Bitmap(bitmapData);
```

Объект char_bounds представляет прямоугольник с полной информацией о положении символа. Эта информация используется для создания объекта flash.display.BitmapData, данные которого находятся в объекте растрового изображения. В конструкторе объекта BitmapData передаются четыре параметра:

```
BitmapData(width:Number, height:Number, transparency:boolean,
fillColor:Number);
```

После создания объекта растрового изображения создается объект Matrix для хранения информации о конкретной сохраняемой части TextField, а именно области TextField в границах, возвращаемых методом getCharBoundaries. Объект Matrix передается методу прорисовки BitmapData, который берет непосредственные графические данные из передаваемого объекта DisplayObject. После прорисовки можно создать объект Bitmap, который является отображаемым объектом (DisplayObject) и может быть включен в список отображения с использованием недавно заполненного объекта BtmapData.

В остальном коде примера перебираются все символы TextField, с каждым символом выполняется описанная операция, и для созданного объекта растрового изображения включается анимация:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="600"
height="300"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[
            import flash.display.Sprite;

            import mx.core.UIComponent;
            import flash.text.TextField;
            import flash.text.TextFormat;

            private var characterArray:Array = new Array();
            // Главный индекс символа, который будет
            // использоваться при анимации
```

```
private var charIndex:int = 0;

// Итоговая позиция, в которой будут
// размещаться все символы
private var finalX:Number = 0;

private function addNewCharacters():void
{
    render();
    invalidateDisplayList();
    invalidateProperties();
}

public function render():void
{
    // Определение всех необходимых переменных
    var bitmapData:BitmapData;
    var bitmap:Bitmap;
    var char_bounds:Rectangle;
    var matrix:Matrix;
    var component:UIComponent;
    // Получение формата текста и заполнение tf
    // tf.defaultTextFormat =
    // addCharacters.getTextField().defaultTextFormat;
    // tf.text = addCharacters.text;
    for each(component in characterArray)
    {
        holder.removeChild(component);
    }
    characterArray.length = 0;
    for(var i:int=0; i<addCharacters.text.length; i++)
    {
        char_bounds = addCharacters.getTextField().
            getCharBoundaries(i);
        bitmapData = new BitmapData(char_bounds.width,
            char_bounds.height, true,0);
        matrix = new Matrix(1,0,0,1,
            -char_bounds.x,char_bounds.y);
        bitmapData.draw(
            addCharacters.getTextField(),
            matrix, null, null, null, true);
        bitmap = new Bitmap(bitmapData);
        component = new UIComponent();
        component.width = bitmapData.width;
        component.height = bitmapData.height;
        component.addChild(bitmap);
        holder.addChild(component);
        component.x=char_bounds.x;
        characterArray[i] = component;
    }
    holder.invalidateDisplayList();
```

```
                startEffect();
            }

            private function startEffect():void
            {
                addEventListener(Event.ENTER_FRAME, moveCharacter);
            }

            private function moveCharacter(event:Event):void
            {
                var comp:UIComponent = (characterArray[charIndex]
                as UIComponent);
                if(comp)
                {
                    if(comp.x < 200 - finalX)
                    {
                        (characterArray[charIndex] as Sprite).x+=2;
                    }
                    else
                    {
                        if(charIndex == characterArray.length - 1)
                        {
                            removeEventListener(
                              Event.ENTER_FRAME, moveCharacter);
                            return;
                        }
                        finalX += comp.width+2;
                        charIndex++;
                    }
                }
            }

        ]]>
    </mx:Script>
    <mx:HBox>
        <cookbook:AccessibleTextInput id="addCharacters" fontSize="18"/>
        <mx:Button label="add characters" click="addNewCharacters()"/>
    </mx:HBox>
    <mx:Canvas id="holder" y="200"/>
</mx:Canvas>
```

В этом примере используется элемент `AccessibleTextInput` **— простое расширение элемента** `TextInput`**, предоставляющее доступ к компоненту** `TextField` **в** `TextInput`**:**

```
<mx:TextInput xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            public function getTextField():IUITextField
            {
                return this.textField;
            }
```

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

Это упрощает передачу объекта `TextFormat` компоненту `TextField`, создаваемому для манипуляций с графикой, и прямое чтение из `TextField` для метода `getCharBoundaries`.

4.11. Назначение стилей для кода HTML в TextField

Задача

Требуется применить стили CSS к коду HTML, отображаемому в `TextField`. Стили либо загружаются из внешнего файла CSS, либо определяются в приложении.

Решение

Используйте метод `parseStyle` объекта `StyleSheet` для разбора строки и назначения таблицы стилей компоненту `TextArea`.

Обсуждение

Объект `StyleSheet` может разобрать любой синтаксически корректный код CSS в строковом виде, обработать полученную информацию и назначить ее компоненту (или использовать в тексте HTML). Для получения исходных данных следует либо загрузить внешний файл при помощи объекта `URLLoader` и передать загруженные данные методу `parseCSS` в виде строки, или сконструировать строку и передать ее объекту `StyleSheet` напрямую. В следующем примере стилевая информация записывается в строку, разбираемую при передаче события `creationComplete`.

Свойство `htmlText` компонента `TextArea` задается после применения стиля, чтобы обеспечить правильность применения стилей к HTML. Стиль `<span>` задается при помощи стилевого атрибута `class`:

```
"<span class='largered'>
```

Стиль задается в строке, передаваемой методу `StyleSheet.parseStyle`:

```
.largered { font-family:Arial, Helvetica; font-size:16; color: #ff0000; }
```

Полный код примера:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="1000"
height="800"
creationComplete="createStyle()">
    <mx:Script>
        <![CDATA[
            // Обратите внимание, что даже имена набраны строчными буквами
```

```
            private var styleText:String =
                '.largered { font-family:Arial, Helvetica; font-size:16;
                     color: #ff0000; }' +
                '.smallblue { font-size: 11; color: #0000ff;
                     font-family:Times NewRoman, Times; }' +
                'li { color:#00ff00; font-size:14; }' +
                'ul {margin-left:50px;}';

            [Bindable]
            private var lipsum:String =
                "<span class='largered'>Nulla metus.</span>Nam ut dolor
                   vitae risus condimentum auctor."+
                " <span class='smallblue'>Cras sem quam,</span> malesuada
                   eu, faucibusaliquam, dictum ac, dui. Nullam blandit"+
                " ligula sed arcu. Fusce nec est.<ul><li> Etiam
                   </li><li> aliquet,</li><li>nunc</li></ul>
                   eget pharetra dictum, magna"+
                " leo suscipit pede, in tempus erat arcu et velit.
                   Aenean condimentum.Nunc auctor"+
                " nulla vitae velit imperdiet gravida";

            [Bindable]
            private var style:StyleSheet;

            private function createStyle():void
            {
                style = new StyleSheet();
                style.parseCSS(styleText);
                text.styleSheet = style;
                text.htmlText = lipsum;
            }
        ]]>
    </mx:Script>
    <mx:TextArea id="text" width="200" height="300"/>
</mx:Application>
```

4.12. Использование RichTextEditor

Задача

Требуется создать компонент с поддержкой расширенного форматирования текста (в частности, с использованием всех шрифтов, установленных на компьютере пользователя), а затем использовать код HTML, созданный в отформатированном тексте.

Решение

Используйте компонент `RichTextEditor` **и прочитайте свойство** `htmlText` **из компонента. Результаты, полученные при вызове** `Font.emumerateFonts`**, задаются свойству** `fontFamilyCombo.dataProvider` **компонента** `RichTextEditor`**.**

Обсуждение

Компонент RichTextEditor чрезвычайно удобен; он позволяет создавать текст HTML, который позднее читается из компонента через свойство htmlText. RichTextEditor содержит кнопки для определения начертания текста (полужирный, курсив, подчеркивание), компонент ComboBox для выбора шрифта и компонент ColorPicker для задания цвета выделенного текста; все эти компоненты доступны из RichTextEditor. В следующем примере компонент fontFamilyCombo заполняется данными всех установленных шрифтов, чтобы пользователь мог выбрать любой шрифт для оформления текста.

Для получения текста со всеми атрибутами, назначенными пользователем, используйте свойство htmlText компонента RichTextEditor. Данное свойство доступно как для чтения, так и для записи; чтобы инициализировать компонент готовым содержимым, просто задайте htmlText строку с синтаксически корректным кодом HTML.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="900" height="500"
creationComplete="addFonts()">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            private function addFonts():void
            {
                 var arr:Array = Font.enumerateFonts(true);
                 richText.fontFamilyCombo.labelField = 'fontName';
                 richText.fontFamilyCombo.dataProvider =
                     Font.enumerateFonts(true);
            }
        ]]>
    </mx:Script>
    <mx:RichTextEditor id="richText" width="400" height="400"
        change="trace(richText.htmlText+' '+richText.text)"/>
    <mx:TextArea height="100%" width="400" htmlText="{richText.htmlText}"
        x="410"/>
</mx:Canvas>
```

См. также

Рецепт 4.5.

4.13. Применение встроенных шрифтов в HTML

Задача

Требуется использовать встроенный шрифт в тексте HTML.

Решение

Используйте тег `@font-face` для внедрения шрифта, затем включите в тег `<font>` атрибут `family` с указанием встроенного шрифта.

Обсуждение

Применение встроенного шрифта в тексте HTML – гораздо более сложная задача, чем использование системного шрифта. Стандартный метод применения шрифта сводится к заданию свойства `font-family` в стиле и его последующему применению к текстовому диапазону. Однако для применения встроенного шрифта в HTML должен использоваться тег `font`:

```
<font size="20" family="DIN">Using the new font</font>
```

Тег `font` использует свойство `fontFamily`, заданное в объявлении `font-face` в теге `<mx:Style>`:

```
@font-face{
    src:url("../assets/DIN-BLAC.ttf");
    fontFamily:DIN;
    advancedAntiAliasing: true;
}
```

Полный код примера:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Style>
        @font-face{
            src:url("../assets/DIN-BLAC.ttf");
            fontFamily:DIN;
            advancedAntiAliasing: true;
        }
        .embeddedFontStyle{
            fontFamily:DIN;
            fontSize:18px;
            color:#CCCCFF;
        }
        .white{
            color:#ffffff;
        }
    </mx:Style>
    <mx:VBox backgroundColor="#000000">
        <mx:Label text="This is some test
            text" styleName="embeddedFontStyle"/>
        <mx:TextArea id="ta" backgroundAlpha="0" width="250"
            height="150" styleName="white">
            <mx:htmlText>
                <![CDATA[
                    Not Using the New Font.<font size="20"
                        family="DIN">Using the newfont</font>
                ]]>
            </mx:htmlText>
```

```
        </mx:TextArea>
</mx:VBox>
</mx:Application>
```

4.14. Имитация тени в текстовых компонентах

Задача

Требуется создать эффект тени, отбрасываемой текстом в компоненте TextArea.

Решение

Используйте объект BitmapData для получения копии TextField. Тень имитируется включением полученного объекта в родительский компонент с небольшим смещением.

Обсуждение

Чтобы имитировать тень, отбрасываемую содержимым компонента TextArea или Text, получите растровое изображение текущего содержимого текстового компонента, а затем добавьте его в родительский компонент. Для достижения желаемого эффекта следует слегка сместить изображение и «приглушить» его посредством уменьшения альфа-канала. В данном случае стоит построить пользовательский компонент на базе класса UIComponent; прямое чтение и добавление низкоуровневых экранных компонентов ActionScript из пакета flash.display упростит разработку.

Метод перенесения bitmapData в объект растрового изображения описан в рецепте 4.10.

```
package oreilly.cookbook
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.events.TextEvent;
    import flash.text.TextField;
    import mx.core.UIComponent;

    public class TextDropShadow extends UIComponent
    {
        private var _useShadow:Boolean = false;
        private var _shadowHolder:Bitmap;
        private var _bitmapData:BitmapData;
        private var _textField:TextField;
```

Экземпляр Bitmap создается и размещается в родительском компоненте с небольшим смещением, имитирующим тень:

```
public function TextDropShadow()
{
    super();
```

```
        _shadowHolder = new Bitmap();
        addChild(_shadowHolder);
        _shadowHolder.x = 5;
        _shadowHolder.y = 5;
        _textField = new TextField();
        _textField.type = "input";
        _textField.addEventListener(TextEvent.TEXT_INPUT, inputListener);
        addChild(_textField);
    }
```

Переопределение метода `updateDisplayList` **обеспечивает вывод** `TextField` **и всей его визуальной информации, включая текст, в** `Bitmap`**:**

```
        override protected function updateDisplayList(unscaledWidth:Number,
            unscaledHeight:Number):void
                {
                    super.updateDisplayList(unscaledWidth, unscaledHeight);
                    if(_useShadow)
                    {
                        _bitmapData = new BitmapData(_textField.width,
                            _textField.height, true);
                        _bitmapData.draw(_textField);
                        _shadowHolder.bitmapData = _bitmapData;
                        _shadowHolder.alpha = 0.7;
                    }
                }
                private function inputListener(event:TextEvent):void
                {
                    invalidateDisplayList();
                }

                public function set useShadow(value:Boolean):void
                {
                    _useShadow = value;
                }

                public function get useShadow():Boolean
                {
                return _useShadow;
                }
            }
        }
```

См. также

Рецепт 4.10.

4.15. Поиск последнего символа в TextArea

Задача

Требуется найти последний выведенный символ в компоненте `TextArea`**.**

Решение

Используйте метод `bounds` компонента `TextField` для получения размеров `TextArea`; метод `GetLineMetrics` возвращает фактическую высоту строк. После определения последней видимой строки используйте методы `getLineOffset` и `getLineLength` для поиска последнего видимого символа в последней видимой строке.

Обсуждение

Каждая строка в компоненте `TextField` может обладать особыми свойствами. Чтобы получить эти свойства, вызовите метод `getLineMetrics` и воспользуйтесь полученным объектом `TextLineMetrics`. В объекте `TextLineMetrics` определяется набор свойств строки текста: высота, ширина, базовая линия и межстрочные интервалы. На рис. 4.1 изображены свойства строки текста, описываемой объектом `TextLineMetrics`.

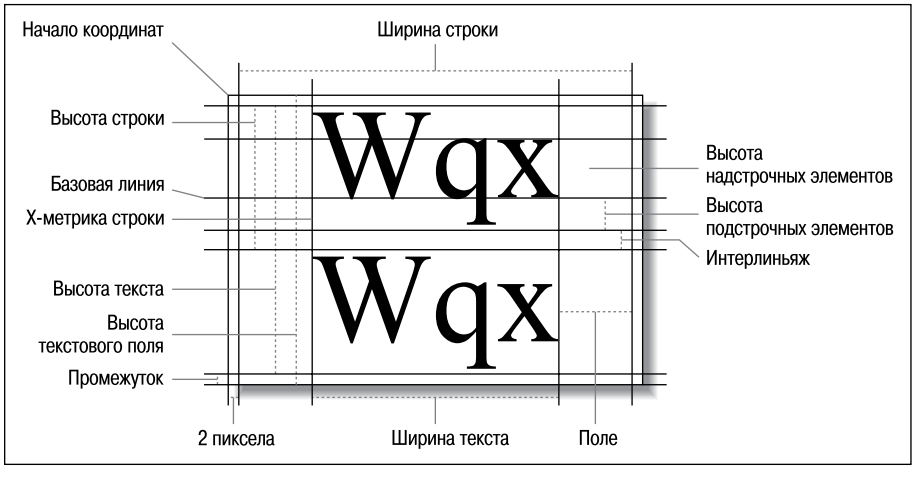

***Рис. 4.1.** Свойства объекта TextLineMetrics*

В следующем примере свойство `height` используется для поиска последней строки, выведенной в `TextArea`. Сначала метод `getBounds` возвращает прямоугольник, представляющий высоту, ширину и координаты видимой области `TextArea`. Затем высоты строк суммируются вплоть до последней видимой строки; информация берется из объектов `TextLineMetrics`. Наконец, последний символ строки определяется суммированием индекса первого символа строки, полученного методом `getLineOffset`, с длиной строки:

```
changeArea.text.charAt(changeArea.getLineOffset(i-1) +
changeArea.getLineLength(i-1))
```

Полный код примера:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[

            private function findLineMetrics():void
            {
                if(changeField.text.length > 1)
                {
                    var rect:Rectangle = changeArea.getBounds(this);
                    var visibleTextHeight:Number = 0;
                    var i:int = 0;
                    while(visibleTextHeight < rect.height &&
                        i < changeArea.numLines)
                    {
                        var metrics:TextLineMetrics =
                            changeArea.getLineMetrics(i);
                        visibleTextHeight +=
                            metrics.ascent+metrics.height;
                        i++;
                    }
                    trace(changeArea.text.charAt(
                        changeArea.getLineOffset(i-1) +
                        changeArea.getLineLength(i-1)));
                }
            }

        ]]>
        </mx:Script>
    <mx:TextInput id="changeField" width="200"
        textInput="findLineMetrics()"/>
    <cookbook:SpecialTextArea id="changeArea" text="{changeField.text}"
        wordWrap="true" width="150" height="30" y="100"/>
</mx:Canvas>
```

5

Компоненты List, Tile и Tree

Три компонента, рассматриваемые в этой главе, расширяют класс `mx.controls.listclasses.ListBase`, причем в реализации этих компонентов, управляемых данными, используются сходные методы. У каждого компонента дочерние компоненты генерируются провайдером данных; все они поддерживают сортировку, фильтрацию, упорядочение и назначение визуальных компонентов в качестве рендереров и редакторов. Кроме того, все компоненты поддерживают перетаскивание мышью.

В этой главе основное внимание уделяется работе с рендерерами элементов, управлению выделением, назначению стилей в `ListBase` и работе с провайдерами данных для разных типов компонентов. Следует помнить, что материал главы не дает сколько-нибудь исчерпывающего изложения темы работы с компонентами `ListBase`. Другие рецепты по использованию перетаскивания приведены в главе 10, а применение скинов для оформления рассматривается в главе 9.

5.1. Создание редактируемого списка

Задача

Требуется создать список с возможностью прямого редактирования любого элемента.

Решение

Задайте свойству `editable` компонента `List` значение `true` и прослушивайте свойства `itemEditBegin` и `itemEditEnd` или задайте `editedItemPosition` посредством передачи объекта со свойствами `columnIndex` и `rowIndex`.

Обсуждение

Чтобы включить возможность редактирования для любого компонента List, достаточно задать его свойству editable значение true. В этом случае каждый рендерер преобразуется в мини-редактор с компонентом TextInput, который заполняется текущим значением выбранным пользователем itemRenderer. Класс List также определяет ряд событий, оповещающих приложение о начале и завершении редактирования:

itemEditBegin

: Событие передается при задании свойства editedItemPosition, когда элемент готов к редактированию. При выдаче этого события List создает объект itemEditor методом createItemEditor и копирует данные из элемента в редактор.

itemEditBeginning

: Событие передается при отпускании кнопки мыши над элементом, передаче фокуса списку или внутри списка, и при других попытках редактирования списка.

itemEditEnd

: У списковых компонентов для этого события имеется обработчик по умолчанию, который копирует данные из редактора элемента в провайдера данных списка. По умолчанию List использует свойство editorDataField для определения свойства itemEditor, содержащего новые данные, и обновляет провайдера данных полученной информацией.

Использование этих событий и управление данными, возвращаемыми itemEditor, более подробно рассматривается в главе 7.

Редактируемый элемент задается либо щелчком пользователя на itemRenderer в списке, либо заданием свойству List.editemItemPosition объекта, обладающего свойствами columnIndex и rowIndex. В компоненте значение columnIndex всегда равно 0, а rowIndex содержит индекс строки, в которой создается редактор, например:

```
listImpl.editedItemPosition = {columnIndex:0, rowIndex:2};
```

Полный код примера выглядит так:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Script>
        <![CDATA[
            [Bindable]
            private var dp:Array = [{name:"John Smith", foo:"bar"},
                {name:"Ellen Smith", foo:"baz"}, {name:"James Smith",
                foo:"eggs"}, {name:"Jane Smith", foo:"spam"}]
```

Свойство selectedItem возвращает значение отредактированного элемента без изменений, внесенных пользователем:

```
            private function editEnd(event:Event):void {
                trace(listImpl.selectedItem.foo + '  ' +
                      listImpl.selectedItem.name);
            }
```

Установка свойства `editedItemPosition` создает редактор в позиции, заданной свойствами `rowIndex` и `columnIndex`:

```
            private function setEditor():void {
                listImpl.editedItemPosition = {columnIndex:0, rowIndex:2};
            }

        ]]>
    </mx:Script>
    <mx:Button click="setEditor()"/>
    <mx:List y="30" width="200" selectedIndex="6" id="listImpl"
        selectionColor="#CCCCFF" labelField="name" dataProvider="{dp}"
        editable="true" itemEditBegin="trace(listImpl.editedItemPosition)"
        itemEditEnd="editEnd(event)" editorXOffset="5" editorYOffset="2"/>
</mx:Canvas>
```

5.2. Значки для элементов List

Задача

Требуется задать значки для `itemRenderer` компонента `List` на основании данных из списка.

Решение

Используйте свойство `iconFunction` компонента `List` и создайте метод, возвращающий правильное изображение в виде встроенного класса.

Обсуждение

Свойство `iconFunction` задает метод, который возвращает объект `Class`, представляющий встроенное изображение для `itemRenderer` компонента `List`. Ссылка на `iconFunction` просто передается как ссылка на имя метода. Например, в следующем примере `setIcon` – имя метода, получающего произвольное значение:

```
iconFunction="setIcon"
```

Методу `iconFunction` всегда передается значение объекта, находящегося в `dataProvider` с соответствующим индексом:

```
private function setIcon(value:*):Class
```

В теле метода `iconFunction` должна обрабатываться все информация об объекте данных с указанным индексом в `dataProvider`. `iconFunction` возвращает класс объекта, который будет использоваться для создания значка рендерера. Полный код примера:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var dp:Array = [{name:"John Smith", position:
               "developer"}, {name:"Ellen Smith", position:"manager"},
               {name:"James Smith", position:"accountant"},
               {name:"Jane Smith", position:"designer"}];

            [Embed(source="../../assets/manager.png")]
            private var managerIcon:Class;

            [Embed(source="../../assets/designer.png")]
            private var designerIcon:Class;

            [Embed(source="../../assets/accountant.png")]
            private var accountantIcon:Class;

            [Embed(source="../../assets/developer.png")]
            private var developerIcon:Class;

            private function setIcon(value:*):Class
            {
                if(value.position != null)
                {
                    switch(value.position)
                    {
                        case "developer":
                            return developerIcon;
                        break;
                        case "designer":
                            return designerIcon;
                        break;
                        case "accountant":
                            return accountantIcon;
                        break;
                        case "manager":
                            return managerIcon;
                        break;
                    }
                }
                return null;
            }
        ]]>
    </mx:Script>
    <mx:List width="200" selectedIndex="6" id="listImpl"
         selectionColor="#CCCCFF" labelField="name" dataProvider="{dp}"
         editable="true" iconFunction="setIcon"/>
</mx:Canvas>
```

5.3. Эффекты для обозначения изменений

Задача

Требуется реализовать для списка эффекты, отображаемые при изменении данных.

Решение

Создайте серию эффектов и передайте их свойству `itemsChangeEffects` компонента `List`.

Обсуждение

Эффекты изменения данных – одно из мощных нововведений Flex 3. В предыдущих версиях разработчик мог записать изменения данных самостоятельно, а затем передать и зарегистрировать события и слушателей событий, но с появлением в Flex 3 свойства `itenChangeEffect` компонент `List` (и другие классы, расширяющие `ListBase`) может передавать события при изменении `dataProvider`; это событие инициирует любые эффекты и последовательности событий, переданные `List` в новом свойстве `itemsChangeEffect`.

Поскольку событие `dataChange` инициируется `dataProvider`, если назначить массив свойству `dataProvider` компонента `List`, события `itemsChangeEffect` не будут передаваться при изменении массива. Помните, что при изменении массивов события не передаются. Однако класс `ArrayCollection` передает события при изменении, как и любой класс, расширяющий `EventDispatcher` и настроенный на передачу событий при вызове метода `set` для изменения значения одного из объектов нижележащего массива.

В следующем примере `itemsChangeEffect` назначается экземпляр `DefaultListEffect` с 2-секундным растворением. Фактически это приводит к тому, что при изменении списка применяется эффект `mx.effects.Glow`:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="900" top="20" left="20">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            // Обратите внимание: для работы этого примера провайдер
            // данных должен быть оформлен в виде ArrayCollection
            [Bindable]
            private var dp:ArrayCollection = new ArrayCollection([
                {name:"John Smith", position:"developer"},
                {name:"Ellen Smith", position:"manager"},
                {name:"James Smith",position:"accountant"},
                {name:"Jane Smith", position:"designer"}]);

            private function addItem():void
```

```
            {
                dp.addItem({name:"Jim Smith", position:"Janitor"});
            }

        ]]>
    </mx:Script>
    <mx:DefaultListEffect color="0xccccff" fadeOutDuration="2000"
        id="glow"/>
    <mx:List width="300" itemsChangeEffect="{glow}" dataProvider="{dp}"
        editable="true" labelField="name"/>
    <mx:Button click="addItem()" label="add item"/>
    <mx:List width="300" itemsChangeEffect="{glow}" dataProvider="{dp}"
        editable="true" labelField="name"/>
</mx:VBox>
```

Рассмотрим другой пример практического применения `itemsChangeEffect`**. Тег** `Sequence` **используется для создания серии событий, инициируемых при изменении данных списка:**

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="900"
    top="20" left="20">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.effects.easing.*;

            [Bindable]
            private var dp:ArrayCollection = new ArrayCollection([
                {name:"John Smith", position:"developer"},
                {name:"Ellen Smith", position:"manager"},
                {name:"James Smith", position:"accountant"},
                {name:"Jane Smith", position:"designer"}]);

        ]]>
    </mx:Script>
    <mx:Sequence id="itemsChangeEffect">
        <mx:WipeDown duration="500"/>
        <mx:Parallel>
            <mx:Move
                duration="750"
                easingFunction="{Elastic.easeOut}"
                perElementOffset="20"/>
            <mx:RemoveItemAction
                startDelay="400"
                filter="removeItem"/>
            <mx:AddItemAction
                startDelay="400"
                filter="addItem">
            </mx:AddItemAction>
            <mx:Blur
                startDelay="410"
                blurXFrom="18" blurYFrom="18" blurXTo="0" blurYTo="0"
```

```
                duration="300"
                filter="addItem"/>
        </mx:Parallel>
    </mx:Sequence>
    <mx:List width="300" itemsChangeEffect="{itemsChangeEffect}"
        dataProvider="{dp}" editable="true" labelField="name"/>
    <mx:Button click="{dp.addItem({name:'Jim Smith', position:'Janitor'});}"
        label="add item"/>
    <mx:Button click="{dp.removeItemAt(3)}" label="remove item"/>
    <mx:Button click="{dp.setItemAt(dp.getItemAt(1), 2)}"
        label="change item"/>
</mx:VBox>
```

5.4. Назначение itemRenderer для класса TileList

Задача

Требуется назначить для класса `TileList` пользовательский рендерер элементов `itemRenderer`, который будет присоединять некоторое изображение в зависимости от кода в данных, полученных от `TileList`.

Решение

Создайте объект `VBox` и переопределите метод `set data`, включив в него чтение из хеш-таблицы, устанавливающей соответствие между изображениями и кодами, передаваемыми при задании данных для отображения.

Обсуждение

`itemRenderer` для списка передается объект данных, представляющий каждый элемент коллекции. Этот объект используется для создания рендерера для каждого конкретного столбца и строки. Вся нестандартная обработка данных, которая должна выполняться в `itemRenderer`, должна происходить в методе `set` свойства `data`; это обеспечивает синхронность рендерера с родительским списком.

В следующем примере список изображений хранится в виде статических открытых ссылок в отдельном файле. Хеш-таблица ставит им в соответствие значение `positionType`, которое, как предполагается, содержится в передаваемом параметре `value`:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var type:String;

            [Bindable]
            private var imageClass:Class;
```

```
            private var typeToPositionHash:Object = {1:"Manager",
                2:"Accountant", 3:"Designer", 4:"Developer"};
            private var typeToImageHash:Object = {1:Assets.managerIcon,
                2:Assets.accountantIcon, 3:Assets.designerIcon,
                4:Assets.developerIcon};

            override public function set data(value:Object):void {
                type = typeToPositionHash[value.positionType];
                imageClass = typeToImageHash[value.positionType];
                nameText.text = value.name;
            }

        ]]>
    </mx:Script>
    <mx:Text text="{type}"/>
    <mx:Image source="{imageClass}"/>
    <mx:Text id="nameText" fontSize="16"/>
</mx:VBox>
```

Чтобы использовать `itemRenderer` **в приведенном примере, достаточно передать компоненту** `List` **ссылку на полное имя класса. Учтите, что если объекты** `dataProvider` **не обладают свойством** `positionType`**,** `itemRenderer` **не будет работать так, как ожидается. В больших приложениях для предотвращения подобных проблем все типы данных должны быть представлены объектами данных с сильной типизацией.**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
    <mx:Script>
        <![CDATA[

            import oreilly.cookbook.SimpleRenderer;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var dp:ArrayCollection = new ArrayCollection([
            {name:"John Smith", positionType:1}, {name:"Ellen Smith",
            positionType:2}, {name:"James Smith", positionType:3},
            {name:"Jane Smith", positionType:4}]);

        ]]>
    </mx:Script>
    <mx:List itemRenderer="oreilly.cookbook.SimpleRenderer"
    dataProvider="{dp}"/>
</mx:Canvas>
```

5.5. Данные XML в компоненте Tree

Задача

Требуется использовать компонент `Tree` **для представления данных XML, загружаемых из внешнего источника.**

Решение

Задайте объекту HTTPService тип e4x и загрузите файл XML с назначением результата запроса провайдером данных Tree. Чтобы в дереве отображались правильные метки, воспользуйтесь синтаксисом ECMAScript for XML (E4X) для передачи свойства узлов, которое должно использоваться в качестве labelField узлов Tree.

Обсуждение

Компонент Tree отлично справляется с данными XML при наличии правильно заданного свойства labelField, обеспечивающего отображение нужных атрибутов. В следующем примере свойство label соответствует атрибуту с именем label:

```
<data label="2004">
    <result label="Jan-04">
        <product label="apple">81156</product>
        <product label="orange">58883</product>
        <product label="grape">49280</product>
    </result>
    <result label="Feb-04">
        <product label="apple">81156</product>
        <product label="orange">58883</product>
        <product label="grape">49280</product>
    </result>
</data>
```

Задайте свойство labelField в синтаксисе E4X так, чтобы оно обозначало атрибут или свойство, которые должны использоваться для меток:

```
labelField="@label"
```

Эта конструкция означает, что в качестве labelField должен использоваться атрибут label, существующий у каждого узла в XML. По умолчанию при отсутствии labelField компонент Tree выводит значение узла. Полный пример:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" left="20" top="10">
    <mx:HTTPService id="xmlLoader" url="assets/data.xml" resultFormat="e4x"
        contentType="application/xml" result="xmlReturned(event)"/>
    <mx:Script>
        <![CDATA[
            import mx.collections.XMLListCollection;
            [Bindable]
            private var xmlValue:XMLListCollection;
            private function xmlReturned(event:Event):void
            {
                xmlValue = new XMLListCollection(
                    new XMLList(xmlLoader.lastResult));
            }
        ]]>
```

```
        </mx:Script>
        <mx:Button click="xmlLoader.send()" label="get XML"/>
        <mx:Tree dataProvider="{xmlValue}" y="100" labelField="@label"
    width="300"
    showRoot="true"/>
    </mx:Canvas>
```

5.6. Создание рендерера для компонента Tree

Задача

Требуется использовать компонент `Tree` для представления данных XML, загружаемых из внешнего источника.

Решение

Задайте объекту `HTTPService` тип `e4x` и загрузите файл XML, с назначением результата запроса провайдером данных `Tree`. Чтобы в дереве отображались правильные метки, воспользуйтесь синтаксисом ECMAScript for XML (E4X) для передачи свойства узлов, которое должно использоваться в качестве `labelField` узлов `Tree`.

Обсуждение

Изменить режим работы компонента `Tree` несколько сложнее, чем у других списковых компонентов. В отличие от компонентов `DataGrid`, `TileList` и `List`, для `Tree` нельзя использовать `dropInItemRenderer`. Вместо этого в `itemRenderer` приходится расширять класс `TreeItemRenderer`. Класс `TreeItemRenderer` определяет рендерер по умолчанию для компонента `Tree`. По умолчанию `TreeItemRenderer` выводит текст, связанный с каждым элементом дерева, необязательный графический значок и необязательный значок сворачивания.

В этом рецепте используется объект `TreeListData`, получаемый `TreeItemRenderer` от родителя. Объект `TreeListData` определяет следующие свойства:

`depth : int`

Уровень элемента в дереве.

`disclosureIcon : Class`

Класс, представляющий значок свертки для элемента дерева.

`hasChildren : Boolean`

Содержит `true`, если у узла имеются потомки.

`icon : Class`

Класс, представляющий значок элемента дерева.

`indent : int`

Отступ узла по умолчанию в дереве.

`item : Object`

Данные элемента.

`label : String`

Текстовое представление данных элемента, основанное на методе `itemToLabel` класса `List`.

`open : Boolean`

Содержит `true` для открытых узлов.

В следующем примере этот метод используется для выделения текста ветвей фиолетовым цветом с полужирным начертанием. В каждую метку папки также включается текст с указанием количества объектов в данной ветви:

```
package oreilly.cookbook
{

    import mx.controls.treeClasses.*;
    import mx.collections.*;

    public class CustomTreeItemRenderer extends TreeItemRenderer
    {
        public function CustomTreeItemRenderer() {
        super();
        mouseEnabled = false;
    }
```

Свойство `listData` класса `TreeItemRenderer` ссылается на данные родительского объекта `Tree`. Здесь оно используется для проверки того, имеет ли объект данных, принадлежащий рендереру, дочерние узлы:

```
override public function set data(value:Object):void {
    if(value != null) {
        super.data = value;
        if(TreeListData(super.listData).hasChildren) {
            setStyle("color", 0x660099);
            setStyle("fontWeight", 'bold');
        } else {
            setStyle("color", 0x000000);
            setStyle("fontWeight", 'normal');
        }
    }
}
```

Для проверки наличия потомков у узла, переданного рендереру, переопределенная версия метода `updateDisplayList` использует данные `TreeListData` родительского объекта `Tree`:

```
        override protected function updateDisplayList(unscaledWidth:Number,
                unscaledHeight:Number):void {
            super.updateDisplayList(unscaledWidth, unscaledHeight);
            if(super.data)
```

```
                {
                    if(TreeListData(super.listData).hasChildren)
                    {
                        var tmp:XMLList =
                            new XMLList(TreeListData(super.listData).item);
                        var myStr:int = tmp[0].children().length();
                        super.label.text = TreeListData(super.listData).label +
                            "(" + myStr + " objects)";
                    }
                }
            }
        }
    }
```

5.7. Сложные объекты данных в компоненте Tree

Задача

Требуется передать компоненту `Tree` сложные объекты данных и обеспечить правильность их разбора.

Решение

Реализуйте интерфейс `ITreeDataDescriptor` в классе. Задайте экземпляр своего класса свойству `dataDecriptor` компонента `Tree`.

Обсуждение

Чтобы использовать сложный объект с компонентом `Tree`, необходимо передать `Tree` объект, реализующий интерфейс `ITreeDataDescriptor`; он обеспечит разбор данных и возвращение правильной информации об отношениях между объектами. Данные XML хорошо укладываются в эту схему, но объекты, определяющие отношения внутри дерева, также позволяют использовать сложные объекты.

Рассмотрим типичный сложный тип данных `ComplexDataType`:

```
package oreilly.cookbook
{
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;

    public class ComplexDataType extends EventDispatcher
    {
        public static const NAME_CHANGED:String = "nameChanged";
        public static const OFFICE_CHANGED:String = "officeChanged";
        public static const RECORD_CHANGED:String = "recordChanged";

        private var _name:Name;
        private var _office:Office;
```

```
        private var _label:String;

        public function ComplexDataType(target:IEventDispatcher=null)
        {
            super(target);
        }

        [Bindable(RECORD_CHANGED)]
        public function set label(value:String):void
        {
            _label = value;
            dispatchEvent(new Event(RECORD_CHANGED));
        }

        public function get label():String
        {
            return _label;
        }

        [Bindable(NAME_CHANGED)]
        public function set name(value:Name):void
        {
            _name = value;
            dispatchEvent(new Event(NAME_CHANGED));
        }

        public function get name():Name
        {
            return _name;
        }

        [Bindable(OFFICE_CHANGED)]
        public function set office(value:Office):void
        {
            _office = value;
            dispatchEvent(new Event(OFFICE_CHANGED));
        }

        public function get office():Office
        {
            return _office;
        }
    }
}
```

Объекты `Office` **и** `Name` **представляют собой простые пары свойств: название и адрес – у объекта** `Office`**,** `firstName` **и** `lastName` **–у объекта** `Name`**. Без объекта** `ITreeDataDescriptor`**, описывающего отношения между** `ComplexDataType` **и содержащимися в нем свойствами** `Office` **и** `Name`**, компонент** `Tree` **не сможет корректно отобразить** `Array` **или** `ArrayCollection` **таких объектов. Вместо этого он просто покажет два узла, которые не будут раскрываться.**

Интерфейс `ITreeDataDescriptor` **содержит следующие методы:**

```
addChildAt(parent:Object, newChild:Object, index:int,
model:Object = null):Boolean
```

Метод вставляет в структуру новый объект и определяет, в каком месте структуры данных должны вноситься изменения.

```
getChildren(node:Object, model:Object = null):ICollectionView
```

Для любого узла этот метод определяет, имеются ли у него потомки. При наличии дочерних узлов метод возвращает их в виде `CollectionView`**; если дочерних узлов нет, метод возвращает** `null`**.**

```
getData(node:Object, model:Object = null):Object
```

Метод возвращает все данные для указанного узла.

```
hasChildren(node:Object, model:Object = null):Boolean
```

Метод проверяет, имеются ли у узла потомки, и возвращает логическое значение.

```
isBranch(node:Object, model:Object = null):Boolean
```

Метод проверяет, является ли узел «не-листовым» (т. е. имеет ли потомков).

```
removeChildAt(parent:Object, child:Object, index:int,
model:Object = null):Boolean
```

Метод удаляет объект из структуры и определяет, что должно произойти со структурой данных после удаления узла.

Далее приводится правильная реализация дескриптора данных для `ComplexDataType`**:**

```
package oreilly.cookbook
{
    import mx.collections.ArrayCollection;
    import mx.collections.ICollectionView;
    import mx.controls.treeClasses.ITreeDataDescriptor;

    public class ComplexDataCollection implements ITreeDataDescriptor
    {
        public function getChildren(node:Object,
            model:Object=null):ICollectionView
        {
            var col:ArrayCollection;
            if(node is Office){
                col = new ArrayCollection([node.officeAddress,
                    node.officeName]);
                return col;
            }
            if(node is Name){
                col = new ArrayCollection([node.firstName, node.lastName]);
                return col;
            }
```

```
        if(node is ComplexDataType){
            col = new ArrayCollection([node.office, node.name]);
            return col;
        }
        return null;
    }

    public function hasChildren(node:Object, model:Object=null):Boolean
    {
        if(node is Office){
            if(node.officeAddress != "" && node.officeName != ""){
                return true;
            } else {
                return false;
            }
        }
        if(node is Name){
            if(node.firstName != "" && node.firstName != ""){
                return true;
            } else {
                return false;
            }
        }
        if(node is ComplexDataType){
            if(node.office != null && node.name != null) {
                return true;
            } else {
                return false;
            }
        }
        return false;
    }

    public function isBranch(node:Object, model:Object=null):Boolean
    {
        if(node is Office){
            if(node.officeAddress != "" && node.officeName != ""){
                return true;
            } else {
                return false;
            }
        }
        if(node is Name) {
            if(node.firstName != "" && node.firstName !=     ""){
                return true;
            } else {
                return false;
            }
        }
        if(node is ComplexDataType) {
            if(node.office != null && node.name != null) {
```

```
                    return true;
                } else {
                    return false;
                }
            }
            return true;
        }

        public function getData(node:Object, model:Object=null):Object {
            if(node is Office) {
                return {children:{label:node.officeName,
                    label:node.officeAddress}};
            }
            if(node is Name) {
                return {children:{label:node.firstName,
                    label:node.lastName}};
            }
            if(node is ComplexDataType){
                return {children:{label:node.office, label:node.name}};
            }
            return null;
        }

        public function addChildAt(parent:Object, newChild:Object,
            index:int, model:Object=null):Boolean
        {
            return false;
        }

        public function removeChildAt(parent:Object, child:Object,
            index:int, model:Object=null):Boolean
        {
            return false;
        }

    }
}
```

Чтобы компонент `Tree` **корректно отображал массив объектов** `ComplexDataType`**, достаточно задать экземпляр класса** `ComplexDataCollection`**, приведенного выше, свойству** `dataDescriptor` **компонента** `Tree`**:**

```
dataDescriptor="{new ComplexDataCollection()}"
```

Полный код примера:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
     height="300"
creationComplete="initCmp()">
    <mx:Script>
        <![CDATA[

            import mx.collections.ArrayCollection;
```

```
            [Bindable]
            private var dataProv:ArrayCollection;

            private function initCmp():void
            {
                var compData:ComplexDataType = new ComplexDataType();
                compData.label = "13";
                var _name:Name = new Name();
                _name.firstName = "Joe";
                _name.lastName = "Smith";
                compData.name = _name;
                var office:Office = new Office();
                office.officeAddress = "544 Happy St. Anytown NY 01092";
                office.officeName = "Happy Town Branch";
                compData.office = office;

                var compData2:ComplexDataType = new ComplexDataType();
                compData2.label = "328";
                var _name2:Name = new Name();
                _name2.firstName = "Jane";
                _name2.lastName = "Doe";
                compData2.name = _name2;
                var office2:Office = new Office();
                office2.officeAddress = "544 Happy St. Anytown NY 01092";
                office2.officeName = "Happy Town Branch";
                compData2.office = office2;

                dataProv = new ArrayCollection([compData, compData2]);
            }

        ]]>
    </mx:Script>
    <mx:Tree dataDescriptor="{new ComplexDataCollection()}"
dataProvider="{dataProv}"
labelField="label" width="300"/>
</mx:Canvas>
```

5.8. Ограничение выделения элементов списка

Задача

Требуется разобрать значение `dataProvider` так, чтобы некоторые элементы списка не могли выделяться пользователем.

Решение

Создайте свойство `filterFunction` в субклассе компонента `List`. Используйте методы `mouseEventToItemRenderer` и `finishKeySelection` для проверки выделения функцией `filter` и разрешения или запрета выделения в зависимости от полученного результата.

Обсуждение

Чтобы ограничить возможность выделения пользователем некоторых элементов списка, необходимо отслеживать выделение как при помощи мыши, так и с клавиатуры. С выделением мышью задача решается тривиально: переопределите метод `mouseEventToItemRenderer` так, чтобы он возвращал `null`, если содержащиеся в `itemRenderer` данные не должны выделяться. С обработкой событий клавиатуры дело обстоит чуть сложнее: при попытке перехода к элементу, для которого выделение запрещено, должен происходить переход к следующему элементу списка с разрешенным выделением.

Чтобы реализовать пользовательский фильтр для идентификации невыделяемых элементов в каждом экземпляре пользовательского класса списка, создайте открытое свойство `disabledFilterFunction`; задавая значение этого свойства, пользователь создает фильтр. Пример:

```
public var disabledFilterFunction:Function;
```

Далее переопределяется метод `mouseEventToItemRenderer`, возвращающий `itemRenderer` для выделяемого элемента по событию `MouseEvent`, и метод `finishKeySelection`, который выбирает `itemRenderer` по событию `keyboardEvent`:

```
<mx:List xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[

            import flash.events.MouseEvent;
            import flash.ui.Keyboard;
            import mx.controls.listClasses.IListItemRenderer;
            import mx.controls.List;

            public var disabledFilterFunction:Function;
            private var selectionIsAbove:Boolean;

            // Необходимо перехватывать все события мыши,
            // способные привести к изменению индекса
            override protected function
                mouseEventToItemRenderer(event:MouseEvent):
                IListItemRenderer {
                var listItem:IListItemRenderer =
                    super.mouseEventToItemRenderer(event);
                if (listItem){
                    if (listItem.data){
                        if (disabledFilterFunction(listItem.data)){
                            return null;
                        }
                    }
                }
                return listItem;
            }
```

Мы перехватываем все события клавиатуры, которые могут привести к изменению индекса:

```
override protected function finishKeySelection():void {
    super.finishKeySelection();
    var i:int;
    var uid:String;
    var rowCount:int = listItems.length;
    var partialRow:int = (rowInfo[rowCount-1].y + rowInfo[rowCount-1].
         height > listContent.height) ? 1 : 0;
    var item:IListItemRenderer = listItems[caretIndex -
         verticalScrollPosition][0];

    if (item) {
        if (item.data) {
        if (disabledFilterFunction(item.data)){
```

Свойство `disabledFilterFunction` определяет, может ли конкретный элемент выделяться в списке. Если текущий выделенный элемент выделяться не должен, приложение просто переходит к другому незаблокированному видимому элементу.

```
            rowCount = rowCount - partialRow;
            var currIndex:int = caretIndex - verticalScrollPosition;
            if (selectionIsAbove){
                // Поиск к началу:
                i = currIndex - 1;
                while(i>0){
                    item = listItems[i][0];
                    if (!disabledFilterFunction(item.data)){
                        selectedIndex = i - verticalScrollPosition;
                        return;
                    }
                    i--;
                }
                i = currIndex + 1
                while(i<this.rowCount){
                    item = listItems[i][0];
                    if (!disabledFilterFunction(item.data)){
                        selectedIndex = i - verticalScrollPosition;
                        return;
                    }
                    i++;
                }
            } else {
                // Поиск к концу:
                while(i<this.rowCount){
                    item = listItems[i][0];
                        if (!disabledFilterFunction(item.data)){
                            selectedIndex = i - verticalScrollPosition;
                            return;
                        }
```

```
                    i++;
                }
                while(i>0){
                    item = listItems[i][0];
                    if (!disabledFilterFunction(item.data)){
                        selectedIndex = i - verticalScrollPosition;
                        return;
                    }
                    i--;
                }
            }
        }
    }
}

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

5.9. Форматирование и проверка данных, введенных в редакторе элемента

Задача

Требуется проверить данные, введенные пользователем в редакторе элемента, до передачи данных списку.

Решение

В обработчике события `itemEditEnd` получите текст от редактора элемента (свойство `itemEditorInstance` класса `ListBase`) и обработайте результаты.

Обсуждение

Чтобы проверить формат, в котором были введены данные, необходимо прослушивать события редактирования элементов, передаваемые `List` в начале и завершении редактирования элемента списка. В процессе редактирования элемента компонент `List` передает три события:

`itemEditBegin`

Событие передается при задании свойства `editedItemPosition`, когда элемент готов к редактированию. При выдаче этого события `List` создает объект `itemEditor` методом `createItemEditor` и копирует данные из элемента в редактор. По умолчанию объект `itemEditor` является экземпляром компонента `TextInput`. Свойству `itemEditor` компонента `List` назначается пользовательский класс `itemEditor` и в конечном итоге задается значение `itemEditorInstance`. Чтобы предотвратить

создание редактора, можно вызвать `preventDefault` в процессе обработки события по умолчанию.

`itemEditBeginning`

Событие передается при отпускании кнопки мыши над элементом, передаче фокуса списку или внутри списка и при других попытках редактирования списка. Слушатель этого события по умолчанию задает свойству `List.editedItemPosition` ссылку на элемент, обладающий фокусом, что приводит к началу сеанса редактирования. Обычно разработчик пишет собственный обработчик этого события, чтобы запретить редактирование некоторого элемента (или элементов). Вызов метода `preventDefault` из пользовательского слушателя этого события предотвращает выполнение слушателя по умолчанию.

`itemEditEnd`

У списковых компонентов для этого события имеется обработчик по умолчанию, который копирует данные из редактора элемента в провайдера данных списка. По умолчанию `List` использует свойство `editorDataField` для определения свойства `itemEditor`, содержащего новые данные, и обновляет провайдера данных полученной информацией. Так как `itemEditor` по умолчанию представляет собой компонент `TextInput`, по умолчанию свойству `editorDataField` задается значение `text` – это означает, что свойство `text` компонента `TextInput` содержит новые данные элемента. Далее вызывается метод `destroyItemEditor`, уничтожающий `itemEditor`. В обработчике события можно изменить данные, возвращаемые редактором компоненту `List`. Пользовательский слушатель может проанализировать данные, введенные в `itemEditor`. Если данные некорректны, вызов метода `preventDefault` запретит Flex возвращать новые данные элементу `List` и закрывать редактор.

В следующем примере метод `preventDefault` используется для предотвращения закрытия редактора и сохранения данных в `dataProvider` в результате проверки по регулярному выражению. Если значение, полученное от `itemEditor`, является синтаксически корректным, строка перед сохранением проходит предварительное форматирование.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
    <mx:Script>
        <![CDATA[

            import mx.events.ListEvent;
            import mx.controls.TextInput;
            import mx.events.ListEvent;
            import mx.collections.ArrayCollection;

        [Bindable]
        private var dp:ArrayCollection = new ArrayCollection([
            {name:"John Smith", positionType:1},
```

```
{name:"Ellen Smith", positionType:2},
{name:"James Smith", positionType:3},
{name:"Jane Smith", positionType:4}]);
```

Далее приводится слушатель события `itemEditEnd`**. Обратите внимание: событие представлено объектом** `ListEvent`**, содержащим свойство** `reason`**. Это свойство используется для проверки того, было ли значение** `itemEditor` **изменено пользователем.**

```
public function formatData(event:ListEvent):void {
    // Проверка причины выдачи события.
    if (event.reason == "cancelled") {
        // Не обновлять ячейку
        return;
    }

    // Получение новых данных от редактора.
    var newData:String = TextInput(event.currentTarget.
        itemEditorInstance).text;
    // Проверяем, является ли новое значение
    // пустой строкой
    var reg:RegExp = /\d/;
    if(newData == "" || reg.test(newData)) {
        // Запрет на передачу фокуса ввода.
        // Редактор ячейки остается открытым.
        event.preventDefault();
        // При помощи errorString сообщаем
        // пользователю о возникших проблемах.
        TextInput(listInstance.itemEditorInstance).
            setStyle("borderColor", 0xff0000);
        TextInput(listInstance.itemEditorInstance).
            errorString = "Enter a valid string.";
        return void;
    }
    // Проверка формата "Имя Фамилия"
    reg = /\w+.\s.\w+/
    if(!reg.test(newData)) {
        event.preventDefault();
       TextInput( listInstance.itemEditorInstance ).
            setStyle( "borderColor", 0xff0000);
       TextInput(listInstance.itemEditorInstance).
            errorString = "Enter first name and last name";
    } else {

        // Обеспечение правильного
        // форматирования имени
        var firstName:String =
            newData.substring(0, newData.indexOf(" "));
        var lastName:String =
            newData.substring(newData.indexOf(" ")+1);
        firstName = firstName.charAt(0).toUpperCase() +
            firstName.substr(1);
```

```
                    lastName = lastName.charAt(0).toUpperCase() +
                        lastName.substr(1);
                    TextInput(listInstance.itemEditorInstance).text =
                        firstName+" "+lastName;
                    newData = newData.charAt(0).toLocaleUpperCase()+
                        newData.substring(1, newData.indexOf(" "))+
                        newData.charAt(newData.indexOf(" ")+1)+
                        newData.substring(newData.indexOf(" ")+2);
                }
            }

        ]]>
    </mx:Script>
    <mx:List id="listInstance" dataProvider="{dp}" editable="true"
         labelField="name" itemEditEnd="formatData(event)" width="300"/>
</mx:Canvas>
```

5.10. Отслеживание выделенных элементов в TileList

Задача

Требуется переключить рендерер для `TileList` без потери выделенных элементов.

Решение

Расширьте компонент `TileList` и создайте пользовательский рендерер для использования в качестве `itemRenderer`. По событию `toggle` рендерера передайте `TileList` событие с уникальным идентификатором `uid`, по которому все идентификаторы будут сохраняться в массиве.

Обсуждение

Все экземпляры `itemRenderer` получают доступ к своему родительскому компоненту `ListBase` при установке свойства `listData`. Рендерер, реализующий `IDropInListItemRenderer`, должен реализовать методы чтения и записи `listData`, т.е. объекта `BaseListData`. Класс `BaseListData` предоставляет доступ к объекту `ListBase`, который задает данные `itemRenderer` через свойство `owner`:

```
public function set listData( value:BaseListData ):void
    private var listParent:ListBase;
    listParent = _listData.owner as ListBase;
}
```

После получения ссылки на экземпляр `ListBase`, создавший рендерер, все открытые свойства `ListBase` становятся доступными для этого рендерера. В примере этого рецепта создается `ArrayCollection` всех включенных кнопок, доступных для `itemRenderer`. В данном случае это класс `Selected-`

ChildrenTileListRender. Класс SelectedChildrenTileListRender просматривает содержимое ArrayCollection и либо добавляет, либо удаляет себя из массива в зависимости от того, находится ли его кнопка во включенном или отключенном состоянии.

Класс SelectedChildrenTileListRender использует свойство uid, передаваемое в свойстве BaseListData, для определения того, предоставлен ли itemRenderer в массиве включенных рендереров. Уникальный идентификатор uid гарантирует, что даже при наличии в данных дубликатов itemRenderer всегда будет располагать уникальной ссылкой для самоидентификации.

SelectedChildrenTileListRender определяет используемый рендерер:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" implements="mx.controls.
    listClasses. IDropInListItemRenderer">
   <mx:Script>
      <![CDATA[

        import mx.controls.listClasses.BaseListData;
        import mx.controls.Button;
        import mx.controls.TileList;
        import mx.events.FlexEvent;
        import mx.controls.listClasses.IDropInListItemRenderer;

        private var _listData:BaseListData;
        [Bindable]
        public var key:String;
        // Родительский компонент рендерера
        private var tileList:SelectedChildrenTileList;

        override public function set data (value:Object):void {
           super.data = value;
           this.invalidateProperties();
           dispatchEvent(new FlexEvent(FlexEvent.DATA_CHANGE));
        }

        [Bindable("dataChange")]
        public function get listData():BaseListData {
           return _listData;
        }

        public function set listData(value:BaseListData ):void {
           // Получение данных списка
           _listData = value;
           // Получение доступа к TileList
           tileList = _listData.owner as SelectedChildrenTileList;
           // Отслеживание уникального идентификатора,
           // который назначается каждому
           // элементу BaseListData.
           key=_listData.uid;
        }
```

```
            private function onToggle(event:Event):void
            {
               var i:int = tileList.toggledButtons.getItemIndex(key);
               // Если ключ рендерера присутствует
               // в списке включенных кнопок, удалить его.
               if(i != -1 && (event.target as Button).selected==false){
                  tileList.toggledButtons.removeItemAt(i);
               }

               // Если ключ рендерера не встречается
               // в списке включенных кнопок, добавить его
               if(i == -1 && (event.target as Button).selected == true){
                  tileList.toggledButtons.addItem(key)
               }
               tileList.invalidateList();
            }

            override protected function updateDisplayList(
            unscaledWidth:Number,unscaledHeight:Number):void
            {
               super.updateDisplayList(unscaledWidth, unscaledHeight);
               if(key)
               {
                  var i:int = tileList.toggledButtons.getItemIndex(key);
                  if(i != -1){ toggleBtn.selected=true; }
                  else { toggleBtn.selected = false; }
               }
            }

         ]]>
      </mx:Script>
      <mx:Button id="toggleBtn" width="100%" label="{data.label}" toggle="true"
           change="onToggle(event)"/>
</mx:VBox>
```

Компонент `SelectedChildrenTileList`**, представленный ниже, использует** `SelectedChildrenTileListRender`**, расширяет** `TileList` **и, как упоминалось ранее, определяет** `ArrayCollection` **для сохранения** `uid` **всех выделенных элементов. Также необходимо определить метод** `returnAllSelected` **для получения** `itemRenderer` **всех выделенных элементов:**

```
<mx:TileList xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="toggledButtons = new
ArrayCollection()">
    <mx:Script>
        <![CDATA[

            import mx.collections.ArrayCollection;
            [Bindable]
            public var toggledButtons:ArrayCollection;
            public function removeAll():void{
                toggledButtons.removeAll();
```

```
                // Использование метода invalidatList
                // для немедленной перерисовки
                this.invalidateList();
            }
            public function returnAllSelected():Array{
                var arr:Array = new Array();
                for(var i:int = 0;
                    i < (dataProvider as ArrayCollection).length; i++)
                {
                    if(toggledButtons.contains((indexToItemRenderer(i)
                        as SelectedChildrenTileListRender).key)){
                        arr.push((indexToItemRenderer(i)
                        as SelectedChildrenTileListRender).data);
                    }
                }
                return arr;
            }

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

5.11. Пустые элементы в рендерере

Задача

Требуется обеспечить отображение пустых элементов в разреженном массиве.

Решение

Задайте свойство `nullItemRenderer` **для компонента** `List`**.**

Обсуждение

Используйте свойство `nullItemRenderer` **для пустых (null) объектов в** `dataProvider`**любого класса, расширяющего** `ListBase`**:**

```
<mx:TileList nullItemRenderer="oreilly.cookbook.NullItemRenderer"/>
```

В файле NullItemRenderer.mxml представлен полный листинг типичного класса `nullItemRenderer`**:**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="50" height="50">
    <mx:Image source="Assets.notAvailableImage"/>
    <mx:Text text="sorry, unavailable" y="30"/>
</mx:Canvas>
```

Класс `nullItemRenderer` **задается свойству** `nullItemRenderer` **компонента** `TileList`**, как показано в следующем фрагменте:**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
```

```
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            [Bindable]
            private var dp:ArrayCollection = new ArrayCollection([null,
                {name:"Ellen Smith", positionType:2}, null,
                {name:"Jane Smith", positionType:4}]);
        ]]>
    </mx:Script>
    <mx:TileList width="100%" columnWidth="150" rowHeight="150"
         dataProvider="{dp}" labelField="name" nullItemRenderer =
         "oreilly.cookbook.NullItemRenderer"/>
</mx:Canvas>
```

5.12. Создание контекстного меню

Задача

Требуется создать пользовательское контекстное меню, отображаемое при щелчке правой кнопкой мыши (или с нажатой клавишей Control) на конкретном элементе списка.

Решение

Создайте объекты `ContextMenu` и `ContextMenuItem`; назначьте их рендереру, который будет назначен списку в качестве `itemRenderer`.

Обсуждение

Контекстное меню в приложениях Flex вызывается щелчком правой кнопкой мыши или с нажатой клавишей Control. По умолчанию в него включаются команды Loop, Play, Print, Quality, Rewind, Save и Zoom, а также ссылка на общую информацию о Flash Player 9. Впрочем, это меню легко настраивается при помощи нового объекта `ContextMenu`. Вызовите конструктор класса `ContextMenu` и задайте свойству `contextMenu` любого отображаемого объекта только что созданный объект, как в следующем примере:

```
var menu:ContextMenu = new ContextMenu();
this.contextMenu = menu;
```

Этот код должен быть выполнен в `DisplayObject`, т.е. любом объекте, обладающем визуальным представлением. Созданное контекстное меню будет отображаться только при щелчке правой кнопкой (или с нажатой клавишей Control) на объекте `DisplayObject` или компоненте с заданным свойством `contextMenu`.

Чтобы добавить новые команды в контекстное меню, воспользуйтесь массивом `customItems`, определяемым в `ContextMenu`. Создайте объекты `ContextMenutem` и включите их в массив методом `push`.

Конструктор `ContextMenuItem` **обладает следующей сигнатурой:**

```
ContextMenuItem(caption:String, separatorBefore:Boolean = false,
enabled:Boolean = true, visible:Boolean = true)
```

Свойство `caption` **определяет текст команды меню. Свойство** `separatorBefore` **определяет, должна ли выводиться над командой, представленной** `ContextMenuItem`**, тонкая линия, отделяющая ее от других команд меню. Наконец, свойства** `visible` **и** `enabled` **управляют соответственно видимостью элемента и возможностью его выбора пользователем.**

При выборе команды меню пользователем `ContextMenuItem` **передает событие** `ContextMenuEvent` **типа** `SELECT`**.**

Следующий пример создает для компонента `List` **рендерер, генерирующий пользовательские контекстные меню на основании типа данных, полученного от** `List`**:**

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="150"
    height="80" paddingLeft="10">
    <mx:Script>
        <![CDATA[

            import flash.display.*;
            override public function set data(value:Object):void
            {
                if(value is Name)
                {
                    text1.text = value.firstName;
                    text2.text = value.lastName;
                    var personMenu:ContextMenu = new ContextMenu();
                    var lookupRecord:ContextMenuItem =
                        new ContextMenuItem("Look Up Record");
                    var lookupPicture:ContextMenuItem =
                        new ContextMenuItem("Look Up Picture");
                    personMenu.customItems.push(lookupRecord);
                    personMenu.customItems.push(lookupPicture);
                    this.contextMenu = personMenu;
                }
                else if(value is Office)
                {
                    text1.text = value.officeAddress;
                    text2.text = value.officeName;
                    var officeMenu:ContextMenu = new ContextMenu();
                    var lookupMap:ContextMenuItem =
                        new ContextMenuItem("Look Up Map");
                    lookupMap.addEventListener(
                        ContextMenuEvent.MENU_ITEM_SELECT, show Map);
                    var lookupEmployees:ContextMenuItem =
                        new ContextMenuItem("Look Up Employees");
                    lookupEmployees.addEventListener(
                        ContextMenuEvent.MENU_ITEM_SELECT, showEmployees);
                    officeMenu.customItems.push(lookupEmployees);
```

```
                officeMenu.customItems.push(lookupMap);
                this.contextMenu = officeMenu;
            }
        }

        private function showMap(event:ContextMenuEvent):void
        {
            // Операции с картой
        }
        private function showEmployees(event:ContextMenuEvent):void
        {
            // Операции поиска
        }
    ]]>
</mx:Script>
<mx:Text id="text1"/>
<mx:Text id="text2"/>
</mx:VBox>
```

5.13. Настройка внешнего вида выделения в компоненте List

Задача

Требуется присоединить графику к выделению в компоненте `List`.

Решение

Переопределите метод `drawSelectionIndicator` класса `ListBase` и измените спрайтовый индикатор, используемый этим методом.

Обсуждение

Компонент `List` создает оформление выделенного `itemRenderer` в списке посредством метода `drawSelectionIndicator`. Метод имеет следующую сигнатуру:

```
override protected function drawSelectionIndicator(indicator:Sprite,
    x:Number, y:Number, width:Number, height:Number, color:uint,
    itemRenderer:IListItemRenderer):void
```

Выделение представляется отдельным изображением, которое выводится каждый раз, когда пользователь задерживает указатель мыши над `itemRenderer` в списке.

Полный листинг с реализацией этого приема:

```
<mx:List xmlns:mx="http://www.adobe.com/2006/mxml" selectionColor="#ffcccc">
    <mx:Script>
        <![CDATA[
            import mx.controls.listClasses.IListItemRenderer;
            [Embed("../../assets/outline_arrow.gif")]
```

```
            private var img:Class;
            [Embed("../../assets/in_arrow.gif")]
            private var highlight_img:Class;
            override protected function drawHighlightIndicator(indicator:
            Sprite, x:Number, y:Number, width:Number, height:Number,
            color:uint, itemRenderer:IListItem Renderer):void
            {
                var this_img:Object = new highlight_img();
                indicator.addChild((this_img as DisplayObject));
                (this_img as DisplayObject).x = itemRenderer.width -
                (this_img as Disp layObject).width
                super.drawHighlightIndicator (indicator,
                x, y, width, height, 0xff0000,itemRenderer);
            }
            override protected function drawSelectionIndicator(indicator:
            Sprite, x:Number, y:Number, width:Number, height:Number,
            color:uint, itemRenderer: IListItemRenderer):void
            {
                var this_img:Object = new img();
                indicator.addChild((this_img as DisplayObject));
                (this_img as DisplayObject).x = itemRenderer.width -
                (this_img as DisplayObject).width
                super.drawSelectionIndicator (indicator,
                x, y, width, height, 0xffcccc, itemRenderer);
            }

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

6

Компоненты DataGrid и AdvancedDataGrid

Компонент DataGrid тоже относится к категории списковых компонентов, но он оптимизирован для отображения больших наборов данных в многостолбцовом макете. Среди прочего, компонент DataGrid поддерживает изменение ширины столбцов, настройку рендереров и сортировку. С выходом Flex 3 семейство DataGrid пополнилось двумя новыми компонентами: AdvancedDataGrid и OLAPDataGrid.

Компонент AdvancedDataGrid расширяет DataGrid, дополняя его новыми возможностями визуализации данных: агрегированием форматированием данных, многостолбцовой сортировкой и т. д. По своей функциональности он напоминает сводные таблицы Microsoft Excel. Компоненты AdvanceDataGrid и OLAPDataGrid распространяются с инфраструктурой визуализации данных, входящей в поставку Flex Builder 3 Professional Edition.

Два компонента DataGrid обычно используются для отображения массивов или коллекций однотипных объектов данных. Компонент AdvancedDataGrid также может отображать объекты HierarchicalData, показывать отношения «родитель-потомок» в сложных объектах данных и поддерживать специализированную группировку данных.

6.1. Создание пользовательских столбцов в DataGrid

Задача

Требуется создать пользовательские столбцы в компоненте DataGrid и управлять отображением данных.

Решение

Используйте тег DataGridColumn для создания пользовательских свойств в компоненте DataGrid.

Обсуждение

В рецепте к свойству columns компонента DataGrid добавляются три тега DataGridColumn. Приложение использует файл данных homesforsale.xml. Теги DataGridColumn задают порядок отображения свойств объектов из dataProvider и текст названия столбца. Свойство dataField тега DataGrid-Column задает свойство объекта, отображаемое в ячейках этого столбца. В следующем примере свойство range не отображается в компоненте DataGrid, поскольку не существует тега DataGridColumn со свойством data-Field, ассоциированным со свойством range. Необходимый код выглядит так:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:DataGrid id="grid"
        width="100%"
        height="100%"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"
                dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }
            private function onResult(evt:ResultEvent):void {
```

```
                this.homesForSale = evt.result.data.region;
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Тег `DataGridColumn` **поддерживает расширенную настройку изображения с использованием** `itemRenderer`**. Следующий фрагмент кода добавляет новый тег** `DataGridColumn`**, использующий пользовательский рендерер** `RangeRenderer` **для более содержательного отображения свойства** `range`**. Свойство** `range` **содержит три значения, описывающих процентное распределение продаваемых домов по ценовым диапазонам:** `range1` **содержит процент домов стоимостью ниже $350 000,** `range2` **– процент домов стоимостью от $350 000 до $600 000 и** `range3` **– процент домов стоимостью свыше $600 000.**

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:DataGrid id="grid"
        width="100%"
        height="100%"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"
                dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
            <mx:DataGridColumn headerText="Price Ranges"
                dataField="range"
                itemRenderer="RangeRenderer"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }
```

```
            private function onResult(evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Класс `RangeRenderer`**, показанный в следующем фрагменте, использует процентные значения** `range` **для отображения цветовой диаграммы, обозначающих распределение по диапазонам. Для этого переопределенная версия метода** `updateDisplayList` **рисует цветные полосы при помощи графического API. За дополнительной информацией об использовании** `itemRenderer` **обращайтесь к главе 7.**

```
package {
    import flash.display.Graphics;
    import mx.containers.Canvas;

    public class RangeRenderer extends Canvas {
        override public function set data(value:Object):void {
            super.data = value;
            if(value!= null && value.range != null) {
                this.invalidateDisplayList();
            }
        }

        override protected function updateDisplayList(unscaledWidth:Number,
            unscaledHeight:Number):void
        {
            var g:Graphics = this.graphics;
            if(this.data) {
                var w1:Number = (this.data.range.range1 *
                    unscaledWidth)/100;
                var w2:Number = (this.data.range.range2 *
                    unscaledWidth)/100;
                var w3:Number = (this.data.range.range3 *
                    unscaledWidth)/100;

                var x1:Number = 0;
                var x2:Number = w1;
                var x3:Number = w1 + w2;

                g.beginFill(0x0000ff);
                g.drawRect(x1,0,w1,unscaledHeight);
                g.beginFill(0x00ff00);
                g.drawRect(x2,0,w2,unscaledHeight);
                g.beginFill(0xff0000);
                g.drawRect(x3,0,w3,unscaledHeight);
            }
        }
    }
}
```

Обратите внимание: попытка сортировки по столбцу `range` компонента `DataGrid` по-прежнему приводит к ошибке времени выполнения. Проблема решается определением пользовательской функции сортировки, представленной в следующем рецепте.

См. также

Рецепт 6.2.

6.2. Определение функций сортировки для столбцов DataGrid

Задача

Требуется использовать пользовательскую логику сортировки сложных объектов в `DataGrid`.

Решение

Используйте свойство `sortCompareFunction` тега `DataGridColumn` для назначения ссылки на функцию, выполняющую пользовательскую логику сортировки.

Обсуждение

В компонент `DataGrid`, упоминаемый в предыдущем рецепте, можно включить пользовательскую функцию сортировки. В этом примере используется пользовательский `itermRenderer` с именем `RangeRengerer`, добавляющий функцию сортировки `sortRanges` в тег `DataGridColumn` для свойства `range`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:DataGrid id="grid" width="100%" height="100%"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"
                dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
            <mx:DataGridColumn headerText="Price Ranges [&lt;350K]
```

```
                [350K -600K] [&gt;600K]"
                dataField="range"
                itemRenderer="RangeRenderer"
                sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }
            private function onResult(evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }

           private function sortRanges(obj1:Object,
            obj2:Object):int{
              var value1:Number = obj1.range.range1;
               var value2:Number = obj2.range.range1;
               if(value1 < value2) {
                   return -1;
               }
               else if(value1 > value2){
                   return 1;
               }
               else {
                   return 0;
               }
           }

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

Свойству `sortCompareFunction` четвертого тега `DataGridColumn` назначается функция `sortRanges`, реализующая пользовательскую логику сортировки диапазонов. Эта функция должна иметь следующую сигнатуру:

```
sortCompareFunction(obj1:Object, obj2:Object):int
```

Функция получает два параметра, соответствующих двум сортируемым объектам `dataProvider`; она возвращает целочисленное значение –1, 1 или 0, определяющее порядок следования объектов после сортировки. Когда пользователь щелкает на заголовке `DataGridColumn`, компонент `DataGrid` выполняет `sortCompareFunction` для всех элементов `dataProvider` и использует возвращаемое значение для определения порядка элементов.

Функция sortRange определяет порядок сортировки по вложенному свойству range1 элемента dataProvider. Теперь, когда пользователь щелкает на заголовке столбца Price Ranges, элементы списка сортируются на основании значений range1.

6.3. Многостолбцовая сортировка в DataGrid

Задача

Требуется обеспечить поддержку многостолбцовой сортировки в компоненте DataGrid.

Решение

Используйте компонент AdvancedDataGrid с AdvancedDataGridColumn.

Обсуждение

Компонент AdvancedDataGrid, появившийся в Flex 3, обладает встроенной поддержкой сортировки по нескольким столбцам. Для демонстрации в следующем примере предыдущий пример слегка видоизменяется: DataGrid и DataGridColumn заменяются на AdvancedDataGrid и AdvancedDataGridColumn соответственно. AdvancedDataGrid поддерживает два режима многостолбцовой сортировки. В режиме по умолчанию (sortExpertMode=false) первичная сортировка осуществляется щелчком на заголовке столбца, а вторичная – щелчком в области многостолбцовой сортировки (справа от заголовка). В приведенном примере используется режим sortExpertMode=true, в котором первичная сортировка осуществляется щелчком на заголовке столбца, а вторичная – щелчком правой кнопки мыши (Windows) или с нажатой клавишей Control (Mac).

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:AdvancedDataGrid id="grid"
        width="100%"
        height="100%"
        sortExpertMode="true"
        dataProvider="{homesForSale}">
        <mx:columns>
           <mx:AdvancedDataGridColumn headerText="Total No."
                dataField="total"/>
           <mx:AdvancedDataGridColumn headerText="City"
                dataField="city"/>
```

```
            <mx:AdvancedDataGridColumn headerText="State"
                 dataField="state"/>
            <mx:AdvancedDataGridColumn headerText="Price Ranges [&lt;350K]
                 [350K -600K] [&gt;600K]"
                 dataField="range"
                 itemRenderer="RangeRenderer"
                 sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }
            private function onResult(evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }
            private function sortRanges(obj1:Object, obj2:Object):int{
                var value1:Number = obj1.range.range1;
                var value2:Number = obj2.range.range1;
                if(value1 < value2) { return -1; }
                else if(value1 > value2) { return 1;}
                else { return 0; }
            }
        ]]>
    </mx:Script>
</mx:Application>
```

6.4. Фильтрация данных в DataGrid

Задача

Требуется обеспечить «живую» фильтрацию набора данных, отображаемого в `DataGrid`, на стороне клиента.

Решение

Используйте свойство `filterFunction` класса `ArrayCollection` и задайте ссылку на пользовательскую функцию, проверяющую условие фильтрации.

Обсуждение

Для демонстрации фильтрации на стороне клиента в следующем примере предыдущий пример дополняется фильтрацией по городу. В поль-

зовательский интерфейс добавляется поле TextInput для ввода названия города; содержимое DataGrid отфильтровывается по введенному названию. Когда пользователь вводит данные в текстовом поле cityFilter, это приводит к передаче события change, обрабатываемого методом applyFilter. Метод applyFilter присваивает ссылку на функцию свойству filterFunction коллекции homesForSale, если это не было сделано ранее, и переходит к вызову метода refresh класса ArrayCollection. Метод filterCities реализует простую проверку строки введенного текста в нижнем регистре по свойству city элемента dataProvider.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
resultFormat="object"
result="onResult(event)"/>

    <mx:Form>
        <mx:FormItem label="City">
            <mx:TextInput id="cityFilter" change="applyFilter()"/>
        </mx:FormItem>
    </mx:Form>

    <mx:AdvancedDataGrid id="grid" width="100%"
        height="100%" sortExpertMode="true" dataProvider="{homesForSale}">
        <mx:columns>
            <mx:AdvancedDataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:AdvancedDataGridColumn headerText="City"
                dataField="city"/>
            <mx:AdvancedDataGridColumn headerText="State"
                dataField="state"/>
            <mx:AdvancedDataGridColumn
                headerText="Price Ranges [&lt;350K] [350K -600K] [&gt;600K]"
                    dataField="range" itemRenderer="RangeRenderer"
                    sortCompare Function="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }

            private function onResult(evt:ResultEvent):void
```

```
            {
                this.homesForSale = evt.result.data.region;
            }

            private function sortRanges(obj1:Object,
                obj2:Object):int
            {
                var value1:Number = obj1.range.range1;
                var value2:Number = obj2.range.range1;

                if(value1 < value2) {
                    return -1;
                }
                else if(value1 > value2){
                    return 1;
                }
                else {
                    return 0;
                }
            }
```

Функция фильтрации применяется к `dataProvider` компонента `DataGrid`, а вызов метода `refresh` гарантирует перерисовку всех рендереров:

```
        private function applyFilter():void {
            if(this.homesForSale.filterFunction == null) {
                this.homesForSale.filterFunction = this.filterCities;
        }
        this.homesForSale.refresh();
    }
```

Используемый метод фильтрации возвращает `true`, если элемент должен быть включен в отфильтрованный массив, и `false` в противном случае:

```
private function filterCities(item:Object):Boolean {
            var match:Boolean = true;

            if(cityFilter.text != "") {
                var city:String = item["city"];
                var filter:String = this.cityFilter.text;
                if(!city ||
                    city.toLowerCase().indexOf(filter.toLowerCase())
                        < 0) {
                    match = false;
                }
            }
            return match;
        }

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

6.5. Создание пользовательских заголовков для AdvancedDataGrid

Задача

Требуется видоизменить заголовок столбца в компоненте DataGrid, добавив в него компонент CheckBox.

Решение

Используйте класс AdvancedDataGridHeaderRenderer; переопределите методы createChildren и updateDisplayList так, чтобы в них добавлялся компонент CheckBox.

Обсуждение

Этот рецепт строится на основе предыдущего: в нем для DataGridColumn city добавляется пользовательский рендерер заголовка. Создание пользовательского заголовка напоминает создание пользовательского рендерера элементов или редактора элементов. Свойству headerRenderer тега DataGridColumn задается ссылка на класс, реализующий интерфейс IFactory; столбец берет на себя создание экземпляра объекта. В данном примере используется класс рендерера с именем CheckBoxHeaderRenderer, который создает заголовок с содержащимся в нем компонентом CheckBox:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
            resultFormat="object"
            result="onResult(event)"/>
    <mx:Form>
        <mx:FormItem label="City">
            <mx:TextInput id="cityFilter"
                change="applyFilter()"/>
        </mx:FormItem>
    </mx:Form>
    <mx:AdvancedDataGrid id="grid" width="100%" height="100%"
        sortExpertMode="true" dataProvider="{homesForSale}">
        <mx:columns>
            <mx:AdvancedDataGridColumn headerText="Total No."
                dataField="total"/>
```

Так как данный пользовательский рендерер должен задаваться только для этого конкретного столбца, но не для других, свойству headerRenderer столбца AdvancedDataGrid задается имя класса, который будет использоваться при создании заголовков:

```
            <mx:AdvancedDataGridColumn headerText="City"
               sortable="false"
               headerRenderer="CheckBoxHeaderRenderer"
```

```
                dataField="city"/>
            <mx:AdvancedDataGridColumn headerText="State dataField="state"/>
            <mx:AdvancedDataGridColumn headerText="Price Ranges [&lt;350K]
                [350K -600K] [&gt;600K]" dataField="range"
                itemRenderer="RangeRenderer"
                sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }

            private function onResult(
                evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }

            private function sortRanges(obj1:Object,
                obj2:Object):int{
                var value1:Number = obj1.range.range1;
                var value2:Number = obj2.range.range1;

                if(value1 < value2) {
                    return -1;
                }
                else if(value1 > value2){
                    return 1;
                }
                else {
                    return 0;
                }
            }

            private function applyFilter():void {
                if(this.homesForSale.filterFunction == null)
                {
                    this.homesForSale.filterFunction = this.filterCities;
                }
                this.homesForSale.refresh();
            }

            private function filterCities(
```

```
                item:Object):Boolean {
                var match:Boolean = true;

                if(cityFilter.text != "") {
                    var city:String = item["city"];
                    var filter:String =
                        this.cityFilter.text;
                    if(!city ||
                        city.toLowerCase().indexOf(
                            filter.toLowerCase()) < 0) {
                            match = false;
                    }
                }
                return match;
            }

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

Далее приводится код класса пользовательского рендерера заголовка `CheckBoxHeaderRenderer`**. Обратите внимание на переопределение метода** `createChildren` **класса** `AdvancedDataGridHeader` **для создания нового компонента** `CheckBox` **и его включения в список отображения. Метод** `updateDisplayList` **заставляет** `CheckBox` **изменить свои размеры в соответствии с размерами по умолчанию. О том, как передавать и обрабатывать события пользовательского рендерера, рассказано в следующем рецепте.**

```
package {

    import flash.events.Event;

    import mx.controls.AdvancedDataGrid;
    import mx.controls.CheckBox;
    import mx.controls.advancedDataGridClasses.
        AdvancedDataGridHeaderRenderer;
    import mx.events.AdvancedDataGridEvent;

    public class CheckBoxHeaderRenderer extends
        AdvancedDataGridHeaderRenderer {

        private var selector:CheckBox;

        override protected function createChildren():void {
            super.createChildren();
            this.selector = new CheckBox();
            this.selector.x = 5;
            this.addChild(this.selector);
        }

        override protected function updateDisplayList(unscaledWidth:Number,
            unscaledHeight:Number):void {
```

```
            super.updateDisplayList(unscaledWidth,
                unscaledHeight);
            this.selector.setActualSize(
              this.selector.getExplicitOrMeasuredWidth(),
              this.selector.getExplicitOrMeasuredHeight());
        }
    }
}
```

6.6. Обработка событий компонентов DataGrid/AdvancedDataGrid

Задача

Требуется организовать обработку событий, передаваемых `DataGrid` и его рендерерами элементов.

Решение

Используйте свойство `owner` в рендерерах элементов для передачи события от родительского компонента `DataGrid`.

Обсуждение

В предыдущем рецепте для `DataGridColumn` создавался пользовательский рендерер заголовка; для этого ссылка на класс задавалась свойству `headerRenderer` данного столбца. Здесь класс рендерера из предыдущего рецепта будет расширен. При щелчке на компоненте `CheckBox` в заголовке класс рендерера передает событие компоненту `DataGrid` — владельцу столбца с используемым `headerRenderer`:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:Form>
        <mx:FormItem label="City">
            <mx:TextInput id="cityFilter"
                change="applyFilter()"/>
        </mx:FormItem>
    </mx:Form>

    <mx:AdvancedDataGrid id="grid" width="100%"
        height="100%" sortExpertMode="true"
        dataProvider="{homesForSale}"
        creationComplete="assignListeners()">
        <mx:columns>
```

```
        <mx:AdvancedDataGridColumn
            headerText="Total No." dataField="total"/>
        <mx:AdvancedDataGridColumn headerText="City"
            sortable="false"
          headerRenderer="CheckBoxHeaderRenderer2"
            dataField="city"/>
        <mx:AdvancedDataGridColumn headerText="State"
            dataField="state"/>
        <mx:AdvancedDataGridColumn
            headerText="Price Ranges [&lt;350K] [350K-600K] [&gt;600K]"
            dataField="range" itemRenderer="RangeRenderer"
            sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }

            private function onResult(evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }

            private function sortRanges(obj1:Object, obj2:Object):int{
                var value1:Number = obj1.range.range1;
                var value2:Number = obj2.range.range1;

                if(value1 < value2) {
                    return -1;
                }
                else if(value1 > value2){
                    return 1;
                }
                else {
                    return 0;
                }
            }

            private function applyFilter():void {
                if(this.homesForSale.filterFunction == null) {
                    this.homesForSale.filterFunction =
                    this.filterCities;
                }
```

```
                this.homesForSale.refresh();
            }

            private function filterCities(item:Object):Boolean {
                var match:Boolean = true;

                if(cityFilter.text != "") {
                    var city:String = item["city"];
                    var filter:String = this.cityFilter.text;
                    if(!city ||
                        city.toLowerCase().indexOf(
                        filter.toLowerCase()) < 0) {
                        match = false;
                    }
                }

                return match;
            }
```

Так как событие «всплывает» от DataGridColumn к родительскому компоненту DataGrid, вы можете просто добавить к DataGrid слушателя события для его перехвата. Метод onColumnSelect получает пользовательское событие типа ColumnSelectedEvent с информацией о столбце, в котором используется рендерер заголовка.

```
            private function assignListeners():void {
                this.grid.addEventListener(ColumnSelectedEvent.
                    COLUMN_SELECTED, onColumnSelect);
            }

            private function onColumnSelect(evt:ColumnSelectedEvent):void {
                trace("column selected = " + evt.colIdx);
            }

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

Пример из предыдущего рецепта дополняется новым рендерером заголовка CheckBoxHeaderRenderer2 для столбца city компонента AdvancedDataGrid. Также добавляется слушатель события ColumnSelectedEvent — пользовательского события, передаваемого AdvancedDataGrid. Функция-слушатель onColumnSelected просто выводит на консоль индекс выделенного столбца (как показано в следующем рецепте, этот прием также может использоваться и в более полезных целях). Использование CheckBoxHeaderRenderer2 продемонстрировано в следующем коде:

```
package {

    import flash.events.MouseEvent;
    import mx.controls.AdvancedDataGrid;
    import mx.controls.CheckBox;
```

```
import mx.controls.advancedDataGridClasses.
   AdvancedDataGridHeaderRenderer;

public class CheckBoxHeaderRenderer2 extends
    AdvancedDataGridHeaderRenderer {

    private var selector:CheckBox;

    override protected function createChildren():void {
        super.createChildren();
        this.selector = new CheckBox();
        this.selector.x = 5;
        this.addChild(this.selector);
        this.selector.addEventListener(MouseEvent.CLICK,
            dispatchColumnSelected);
        }
        override protected function updateDisplayList(
            unscaledWidth:Number, unscaledHeight:Number):void {
            super.updateDisplayList(unscaledWidth, unscaledHeight);
            this.selector.setActualSize(
                this.selector.getExplicitOrMeasuredWidth(),
                this.selector.getExplicitOrMeasuredHeight());
        }

        private function dispatchColumnSelected(evt:MouseEvent):void {
            var event:ColumnSelectedEvent = new ColumnSelectedEvent(
                ColumnSelectedEvent.COLUMN_SELECTED,
                listData.columnIndex, selector.selected);
            AdvancedDataGrid(listData.owner).
                    dispatchEvent(event);
        }
    }
}
```

Хотя событие `ColumnSelectedEvent` в конечном итоге передается компонентом `AdvancedDataGrid`, оно исходит от экземпляра рендерера заголовка при выделении флажка. Метод `dispatchColumnSelected` класса `CheckBoxHeaderRenderer2` использует свойство `listData.owner` для получения ссылки на родительский компонент `AdvancedDataGrid` и дальнейшую передачу события «владельцем».

```
AdvancedDataGrid(listData.owner).dispatchEvent(event);
```

Наконец, взгляните на код пользовательского класса события `CustomSelectedEvent`. Он просто расширяет `Event` двумя свойствами: `colIdx` для хранения индекса столбца, и `isSelected` для признака выделения столбца.

```
package {
    import flash.events.Event;

    public class ColumnSelectedEvent extends Event {
        public var colIdx:int;
```

```
        public var isSelected:Boolean;

        public static const COLUMN_SELECTED:String =
            "columnSelected";

        public function ColumnSelectedEvent(
            type:String,colIdx:Int,
            isSelected:Boolean)
        {
            super(type);
            // Задание новых свойств.
            this.colIdx = colIdx;
            this.isSelected = isSelected;
        }

        override public function clone():Event {
            return new ColumnSelectedEvent(
                type, colIdx,isSelected);
        }
    }
}
```

6.7. Выделение элементов в AdvancedDataGrid

Задача

Требуется выделить несколько ячеек `AdvancedDataGrid` на программном уровне.

Решение

Задайте свойству `selectionMode` компонента `AdvancedDataGrid` значение `multipleCells`, а свойству `selectedCells` – массив объектов, содержащих значения `rowIndex` и `columnIndex` выделяемых столбцов.

Обсуждение

Компонент `AdvancedDataGrid` в Flex 3 поддерживает несколько режимов выделения элементов в таблицах. Режим выделения определяется свойством `selectionMode` компонента; доступны следующие варианты:

- Несколько ячеек
- Несколько строк
- Одна ячейка
- Одна строка
- Без выделения

Чтобы разрешить выделение нескольких ячеек, также необходимо задать свойство `allowMultipleSelection` равным `true`.

В следующем примере щелчок на флажке в столбце `city` **выделяет все элементы в этом столбце. Программный код строится на основе предыдущего рецепта и дополняет его выделением ячеек. При выделении** `CheckBox` **в заголовке столбца** `city` **компонент** `AdvancedDataGrid` **передает событие** `ColumnSelectEvent`**, обрабатываемое методом** `onColumnSelect`**. Метод** `onColumnSelect` **конструирует массив объектов, каждый из которых содержит** `rowIndex` **и** `cellIndex` **выделяемой ячейки. В нашем примере выделяются все ячейки столбца** `city`**. Далее массив объектов задается свойству** `selectedCells` **компонента. Учтите, что для нормальной передачи информации о внесенных изменениях необходимо создать новый массив и задать его свойству** `selectedCells`**. Скажем, простое добавление элементов в свойство** `selectedCells` **с использованием** `grid.selectedCells.push` **не сработает, потому что компонент не сможет обнаружить изменения в массиве.**

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
resultFormat="object"
result="onResult(event)"/>
    <mx:Form>
        <mx:FormItem label="City">
            <mx:TextInput id="cityFilter"
                change="applyFilter()"/>
        </mx:FormItem>
    </mx:Form>
    <mx:AdvancedDataGrid id="grid" width="100%"
        height="100%" sortExpertMode="true"
        dataProvider="{homesForSale}"
        selectionMode="multipleCells"
        creationComplete="assignListeners()">
        <mx:columns>
            <mx:AdvancedDataGridColumn
                headerText="Total No." dataField="total"/>
            <mx:AdvancedDataGridColumn headerText="City"
                sortable="false"
                headerRenderer="CheckBoxHeaderRenderer2"
                dataField="city"/>
            <mx:AdvancedDataGridColumn headerText="State"
                dataField="state"/>
            <mx:AdvancedDataGridColumn
                headerText="Price Ranges [&lt;350K]
                            [350K -600K] [&gt;600K]"
                dataField="range"
                itemRenderer="RangeRenderer"
                sortCompareFunction="sortRanges"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:Script>
        <![CDATA[
```

```
import mx.events.FlexEvent;
import mx.collections.ArrayCollection;
import mx.rpc.events.ResultEvent;

[Bindable]
private var homesForSale:ArrayCollection;

private function initApp():void {
    this.srv.send();
}

private function onResult(evt:ResultEvent):
    void {
    this.homesForSale = evt.result.data.region;
}

private function sortRanges(obj1:Object,
    obj2:Object):int{
    var value1:Number = obj1.range.range1;
    var value2:Number = obj2.range.range1;

    if(value1 < value2) {
        return -1;
    }
    else if(value1 > value2){
        return 1;
    }
    else {
        return 0;
    }
}

private function applyFilter():void {
  if(this.homesForSale.filterFunction == null) {
        this.homesForSale.filterFunction =
            this.filterCities;
    }
    this.homesForSale.refresh();
}

private function filterCities(
    item:Object):Boolean {
    var match:Boolean = true;
    if(cityFilter.text != "") {
        var city:String = item["city"];
        var filter:String =
            this.cityFilter.text;
        if(!city ||
            city.toLowerCase().
            indexOf(filter.toLowerCase()) < 0) {
                match = false;
```

```
                    }
                }
                return match;
            }

            private function assignListeners():void {
                this.grid.addEventListener(
                    ColumnSelectedEvent.COLUMN_SELECTED,
                    onColumnSelect);
            }

            private function onColumnSelect(
                evt:ColumnSelectedEvent):void {
                var selectedCells:Array = new Array();
                var colIdx:int = evt.colIdx;
                if(evt.isSelected) {
                    for(var i:int=0;
                        i<this.homesForSale.length;i++) {
                        selectedCells.push(
                        {rowIndex:i,columnIndex:colIdx});
                    }
                }
                this.grid.selectedCells = selectedCells;
            }
        ]]>
    </mx:Script>
</mx:Application>
```

6.8. Поддержка перетаскивания в DataGrid

Задача

Требуется включить для `DataGrid` поддержку перетаскивания, чтобы пользователи могли перетаскивать данные из одной таблицы в другую.

Решение

Задайте истинные значения свойству `dragEnabled` исходного компонента `DataGrid` и свойству `dropEnabled` приемного компонента `DataGrid`.

Обсуждение

Включение поддержки перетаскивания в списковых компонентах данных (таких, как `DataGrid`) часто сводится к простой установке свойств – Flex 3 Framework выполняет все низкоуровневые операции за разработчика. Например, в следующем примере свойство `dragEnabled` компонента `DataGrid` задается равным `true`; фактически этим вы разрешаете перетаскивание элементов данных за пределы компонента. Также обратите внимание на установку истинного значения свойства `dropEnabled` ком-

понента DataGrid, которая позволяет компоненту принимать элементы, перетаскиваемые снаружи. На поддержку перетаскивания также влияет свойство dragMoveEnabled компонента AdvancedDataGrid. Оно определяет, должны ли перетаскиваемые данные перемещаться за пределы исходного компонента, или они просто копируются в приемник. По умолчанию используется значение false (копирование данных).

Для других компонентов или для реализации нестандартного поведения перетаскивания в Flex 3 Framework был включен компонент DragManager, подробно рассматриваемый в главе 10.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" creationComplete="initApp()">
    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object" result="onResult(event)"/>
    <mx:AdvancedDataGrid id="grid" width="100%"
        height="100%" sortableColumns="false"
        dragEnabled="true"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:AdvancedDataGridColumn
                headerText="Total No." dataField="total"/>
            <mx:AdvancedDataGridColumn headerText="City"
                sortable="false" dataField="city"/>
            <mx:AdvancedDataGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:AdvancedDataGrid>

    <mx:DataGrid width="100%" height="100%"
        dropEnabled="true">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"
                sortable="false" dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:DataGrid>

    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;
            private function initApp():void {
                this.srv.send();
            }
```

```
            private function onResult(evt:ResultEvent):void
            {
                this.homesForSale = evt.result.data.region;
            }
        ]]>
    </mx:Script>
</mx:Application>
```

6.9. Редактирование данных в DataGrid

Задача

Требуется включить возможность редактирования данных в DataGrid.

Решение

Задайте свойству selectionMode компонента AdvancedDataGrid значение multipleCells, а свойству selectedCells — массив объектов, содержащих значения rowIndex и columnIndex выделяемых столбцов.

Обсуждение

В приведенном примере компоненты AdvancedDataGrid и DataGrid привязаны к одному набору данных dataProvider. Свойство editable каждого компонента задано равным true. Установка свойства editable дает возможность редактирования ячеек прямо в таблице. Поскольку оба компонента связаны с одним источником dataProvider, изменения содержимого ячейки в одной таблице автоматически отражается в другой таблице. Необходимый код выглядит так:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="initApp()">

    <mx:HTTPService id="srv"
        url="assets/homesforsale.xml"
        resultFormat="object"
        result="onResult(event)"/>

    <mx:AdvancedDataGrid id="grid" width="100%"
        height="100%" sortableColumns="false"
        editable="true"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:AdvancedDataGridColumn
                headerText="Total No." dataField="total"/>
            <mx:AdvancedDataGridColumn headerText="City"
                sortable="false" dataField="city"/>
            <mx:AdvancedDataGridColumn headerText="State"
                dataField="state"/>
```

```
        </mx:columns>
    </mx:AdvancedDataGrid>

    <mx:DataGrid width="100%" height="100%"
        editable="true"
        dataProvider="{homesForSale}">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"
                sortable="false" dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:DataGrid>

    <mx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;

            private function initApp():void {
                this.srv.send();
            }
            private function onResult(evt:ResultEvent):void {
                this.homesForSale = evt.result.data.region;
            }

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

6.10. Поиск в DataGrid и автоматическая прокрутка к результатам

Задача

Требуется найти элемент в `DataGrid` и прокрутить компонент к результату.

Решение

Для поиска следует использовать метод `findFirst` интерфейса `IViewCursor` класса `ArrayCollection`. Метод `scrollToIndex` компонента `DataGrid` обеспечит прокрутку к позиции найденного элемента.

Обсуждение

Центральное место в этом рецепте занимает компонент DataGrid с простой формой, на которой размещается компонент TextInput для ввода названия города и кнопка поиска. Пользователь щелкает на кнопке (search_btn), приложение ищет совпадение в данных dataProvider компонента DataGrid, выделяет соответствующую строку и прокручивает представление к ней (если она отсутствует в видимой области).

Два главных аспекта этого решения – поиск совпадений и позиционирование элемента DataGrid. Для поиска используется IViewCursor – интерфейс методов и свойств, обеспечивающих перебор в представлении коллекции. Все объекты коллекций Flex поддерживают метод createCursor, который возвращает экземпляр реализации IViewCursor для конкретной коллекции. В нашем примере следующий фрагмент создает курсор для экземпляра ArrayCollection, используемого в качестве dataProvider компонента DataGrid:

```
private function onResult(evt:ResultEvent):void {
    var sort:Sort = new Sort();
    sort.fields = [ new SortField("city",true) ];
    this.homesForSale = evt.result.data.region;
    this.homesForSale.sort = sort;
    this.homesForSale.refresh();
    this.cursor = this.homesForSale.createCursor();
}
```

Объекту sort также может быть присвоена коллекция ArrayCollection, в которой свойство city данных dataProvider определяется как сортируемое поле. Это объясняется тем, что findFirst и другие методы find интерфейса IViewCursor могут вызываться только для отсортированных представлений.

Созданный курсор используется для перемещений между записями и выдачи запросов к связанному с ним представлению. Приведенный далее метод searchCity вызывается при щелчке на кнопке поиска:

```
private function searchCity():void {
    if(search_ti.text != "") {
        if(this.cursor.findFirst({city:search_ti.text})){
            var idx:int = this.homesForSale.getItemIndex(
                this.cursor.current);
                this.grid.scrollToIndex(idx);
                this.grid.selectedItem = this.cursor.current;
        }
    }
}
```

Название города, введенное пользователем, передается в параметре метода findFirst интерфейса IViewCursor. Метод возвращает true для первого совпадения, найденного в ArrayCollection, а также обновляет свойство current объекта курсора ссылкой на совпадающий элемент.

После обнаружения совпадения метод `getItemIndex` **объекта** `ArrayCollection` **используется для получения индекса элемента в** `dataProvider`**. Наконец, компонент** `DataGrid` **перерисовывается методом** `scrollToIndex` **с прокруткой к индексу совпадения, а свойству** `selectedItem` **компонента задается соответствующий индекс.**

Полный код примера:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" creationComplete="initApp()">

    <mx:HTTPService id="srv" url="assets/homesforsale.xml"
        resultFormat="object" result="onResult(event)"/>

    <mx:Form>
        <mx:FormItem label="Search">
            <mx:TextInput id="search_ti"/>
        </mx:FormItem>
        <mx:FormItem>
            <mx:Button label="Search City"
                click="searchCity()"/>
        </mx:FormItem>
    </mx:Form>

    <mx:DataGrid id="grid" width="300" height="150"
        editable="true" dataProvider=" {homesForSale}">
        <mx:columns>
            <mx:DataGridColumn headerText="Total No."
                dataField="total"/>
            <mx:DataGridColumn headerText="City"="false"
                dataField="city"/>
            <mx:DataGridColumn headerText="State"
                dataField="state"/>
        </mx:columns>
    </mx:DataGrid>

    <mx:Script>
        <![CDATA[
            import mx.collections.SortField;
            import mx.collections.Sort;
            import mx.collections.IViewCursor;
            import mx.events.FlexEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var homesForSale:ArrayCollection;
            private var cursor:IViewCursor;

            private function initApp():void {
                this.srv.send();
            }
```

```
            private function onResult(evt:ResultEvent):void
            {
               var sort:Sort = new Sort();
               sort.fields = [ new SortField("city",true) ];
               this.homesForSale = evt.result.data.region;
               this.homesForSale.sort = sort;
               this.homesForSale.refresh();
               this.cursor = this.homesForSale.createCursor();
            }

            private function searchCity():void {
                if(search_ti.text != "") {
                if(this.cursor.findFirst(
                    {city:search_ti.text})){
                        var idx:int =
                        this.homesForSale.getItemIndex(this.cursor.current);
                        this.grid.scrollToIndex(idx);
                        this.grid.selectedItem = this.cursor.current;
                    }
                }
            }
        ]]>
    </mx:Script>
</mx:Application>
```

6.11. Построение сводки плоских данных

Материал предоставлен Шринивасом Рамасвами (**Sreenivas Ramaswamy**), *http://fxpearls.blogspot.com*.

Задача

Требуется сгенерировать сводку по плоским данным в таблице.

Решение

Используйте класс `GroupingCollection` для построения сводки плоских данных. Внесите соответствующие изменения в настройку компонента `AdvancedDataGrid`.

Обсуждение

Класс `GroupingCollection` используется для построения сводки плоских данных, а настройка `AdvancedDataGrid` обеспечивает нужный режим ее отображения.

При построении сводки сортировка и группировка по `dataField` нежелательны, поскольку сводка должна отображаться как полученная в результате обработки плоского набора данных. В следующем примере генерируется фиктивная группа с недействительным полем группиров-

ки; а именно, в свойстве dataField класса GroupingField указывается поле fieldNameNotPresent. После этого для получения нужной сводки можно использовать объекты SummaryRow и SummaryField.

Когда сводка будет готова, можно переходить ко второй задаче. При передаче ADG.dataProvider коллекции GroupingCollection провайдер данных пытается отобразить коллекцию в виде иерархического дерева, так как GroupingCollection реализует IHierachicalData. Во внутренней реализации GroupingCollection преобразуется в HierarchicalCollectionView, и ADG.dataProvider возвращает экземпляр HierarchicalCollectionView. (Происходит то же, что при назначении dataProvider массива, который во внутренней реализации преобразуется в ArrayCollection.) Оформление корневого узла определяется свойством showRoot объекта HierarchicalCollectionView. Если задать ему значение false, фиктивная группа отображаться не будет.

Компонент AdvancedDataGrid по умолчанию использует рендерер AdvancedDataGridGroupItemRenderer для отображения иерархических данных. Этот рендерер отображает значки папок и свертки у родительских элементов. Если задать AdvancedDataGridGroupItemRenderer значение AdvancedDataGrid.groupItemRenderer, значки групп отображаться не будут.

Полный код примера:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" width="460" height="428" >
<mx:Script>
    <![CDATA[
        import mx.controls.advancedDataGridClasses.
            AdvancedDataGridItemRenderer;
        import mx.collections.IGroupingCollection;
        import mx.controls.advancedDataGridClasses.
            AdvancedDataGridColumn;
        import mx.collections.GroupingField;
        import mx.collections.Grouping;
        import mx.collections.ArrayCollection;
        import mx.collections.GroupingCollection;
        var flatData:ArrayCollection = new ArrayCollection(
            [

            { Region:"Southwest", Territory:"Arizona",
              Territory_Rep:"Barbara Jennings",
              Estimate:40000 , Actual:38865 },
            { Region:"Southwest", Territory:"Arizona",
              Territory_Rep:"Dana Binn",
              Estimate:30000 , Actual:29885 },
            { Region:"Southwest", Territory:"Central California",
              Territory_Rep:"Joe Schmoe" ,
              Estimate:30000 , Actual:29134 },
            { Region:"Southwest", Territory:"Northern California" ,
              Territory_Rep: "Lauren Ipsum" ,
              Estimate:40000 , Actual:38805 },
```

```
            { Region:"Southwest", Territory:"Northern California" ,
              Territory_Rep: "T.R. Smith" ,
              Estimate:40000 , Actual:55498 },
            { Region:"Southwest", Territory:"Southern California" ,
              Territory_Rep: "Jane Grove" ,
              Estimate:45000 , Actual:44913 },
            { Region:"Southwest", Territory:"Southern California" ,
              Territory_Rep: "Alice Treu" ,
              Estimate:45000 , Actual:44985 },
            { Region:"Southwest", Territory:"Nevada",
              Territory_Rep:"Bethany Pittman",
              Estimate:45000 , Actual:52888 }

            ]);
```

Свойство `styleFunction` **компонента** `AdvancedDataGrid` **используется для форматирования объектов** `itemRenderer`**, в объекте данных которых присутствует свойство** `summary`**:**

```
        private function formatSummary(data:Object,
          col:AdvancedDataGridColumn): Object {
            if (data.hasOwnProperty("summary"))
            {
                return { color:0xFF0000, fontWeight:"bold", fontSize:12 };
            }

            return {};
        }

        private function flatSummaryObject():Object
        {
            return { Territory_Rep:"Total", summary:true };
        }

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

`AdvancedDataGridItemRenderer` **используется здесь как провайдер** `groupItemRenderer` **для предотвращения вывода значков в первом столбце. Свойство** `groupItemRenderer` **задает рендерер, используемый для оформления нелистовых узлов дерева для родительских узлов в** `dataProvider` **компонента** `AdvancedDataGrid`**. Компонент** `AdvancedDataGrid` **также определяет свойство** `groupLabelFunction`**, в котором задается метод, используемый компонентом для получения меток родительских узлов в** `dataProvider`**.**

```
<mx:AdvancedDataGrid id="adg" creationComplete="groupedData.refresh();
    adg.dataProvider .showRoot=false" groupItemRenderer=
    "mx.controls.advancedDataGridClasses. AdvancedDataGridItemRenderer"
    x="30" y="30" width="400" height="377" styleFunction="formatSummary">
    <mx:dataProvider>
        <mx:GroupingCollection id="groupedData"
            source="{flatData}" >
```

```
            <mx:Grouping>
                <!-- Использование фиктивного поля с заданием
                     showRoot=false для dataProvider ADG -->
                <mx:GroupingField name="fieldNameNotPresent" >
                    <mx:summaries>
                    <!-- summaryObjectFunction возвращает пользовательский
                         объект, который может использоваться в функции
                         форматирования для идентификации строки
                         со сводкой -->
                    <mx:SummaryRow summaryPlacement="last"
                         summaryObjectFunction=
                         "flatSummaryObject">
                        <mx:fields>
                            <mx:SummaryField dataField="Estimate" />
                            <mx:SummaryField dataField="Actual" />
                        </mx:fields>
                    </mx:SummaryRow>
                    </mx:summaries>
                </mx:GroupingField>
            </mx:Grouping>
        </mx:GroupingCollection>

    </mx:dataProvider>
    <mx:groupedColumns>
        <mx:AdvancedDataGridColumn
            headerText = "Territory Rep" dataField="Territory_Rep" />

        <mx:AdvancedDataGridColumnGroup
            headerText="Sales Figures" textAlign="center">

          <mx:AdvancedDataGridColumn headerText = "Estimate"
            textAlign="center" dataField="Estimate" width="100" />

          <mx:AdvancedDataGridColumn headerText = "Actual"
            textAlign="center" dataField="Actual" width="100" />
        </mx:AdvancedDataGridColumnGroup>
    </mx:groupedColumns>
</mx:AdvancedDataGrid>
</mx:Application>
```

6.12. Асинхронное обновление GroupingCollection

Материал предоставлен Шринивасом Рамасвами (Sreenivas Ramaswamy).

Задача

Требуется асинхронно обновлять содержимое очень большой таблицы данных с `GroupingCollection`**, чтобы перерисовка производилась только по требованию.**

Решение

Используйте метод `GroupingCollection.refresh(async:Boolean)` с флагом `async=true`.

Обсуждение

Метод `GroupingCollection.refresh` получает флаг, который указывает, как должна производиться группировка – синхронно или асинхронно. При большом количестве входных записей флагу `async` можно задать значение `true` в вызове, обновляющем более ранний результат группировки. Это также может быть сделано для предотвращения тайм-аута Flash Player, если выполнение `GroupingCollection.refresh` занимает слишком много времени.

Асинхронная группировка также пригодится в ситуациях с интерактивной группировкой данных. Метод `GroupCollection.cancelRefresh` может использоваться для прерывания текущей и запуска новой группировки для обновленных данных, полученных от пользователя.

В следующем примере кнопка `populateADGButton` генерирует случайные данные и отображает их в компоненте `AdvancedDataGrid`. Счетчик изменяет количество записей данных. Кнопка `Group` запускает асинхронное обновление, и компонент `AdvancedDataGrid` отображает результаты немедленно. Кнопка `Cancel Grouping` позволяет в любой момент отменить группировку.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" width="520" height="440">

    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.collections.IGroupingCollection;
            import mx.collections.GroupingField;
            import mx.collections.Grouping;
            import mx.collections.GroupingCollection;

            [Bindable]
            private var generatedData:Array = [];

            private var companyNames:Array = ["Adobe", "BEA", "Cosmos",
                "Dogma", "Enigma", "Fury", "Gama", "Hima", "Indian",
                "Jaadu", "Karish", "Linovo", "Micro", "Novice", "Oyster",
                "Puple", "Quag", "Rendi", "Scrup", "Tempt", "Ubiqut",
                "Verna", "Wision", "Xeno", "Yoga", "Zeal" ];

            private var products:Array = [ "Infuse", "MaxVis", "Fusion",
                "Horizon", "Apex", "Zeeta", "Maza", "Orion", "Omega",
                "Zoota", "Quata", "Morion" ];

            private var countries:Array = [ "India", "USA", "Canada",
```

```
            "China", "Japan", "France", "Germany", "UK", "Brazil",
            "Italy", "Chile", "Bhutan", "Sri Lanka" ];

        private var years:Array = ["2000", "2001", "2002", "2003",
            "2004", "2005", "2006", "2007", "2008", "2009", "2010",
            "2011", "2012", "2013", "2014", "2015", "2016","2017",
            "2018", "2019", "2020", "2021", "2022", "2023", "2024"];

        private var quarters:Array = ["Q1", "Q2", "Q3", "Q4"];

        private var months:Array = ["Jan", "Feb", "Mar", "Apr", "May",
            "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ];

        private var sales:Array = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ;

        private var costs:Array = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ;

        private var dimNameMatch:Object = {Company:companyNames,
            Product:products, Country:countries, Year:years,
            Quarter:quarters, Month:months, Sales:sales, Cost:costs};
```

Случайная выборка из этих массивов создает `dataProvider` **с правильным количеством записей:**

```
private function generateData():void{
    generatedData = [];
    var length:int = numRows.value;
    var dimNameMap:Object = dimNameMatch;

    for (var index:int = 0; index < length; ++index)
    {
        var newObj:Object = {};
        for (var prop:String in dimNameMap)
        {
            var input:Array = dimNameMap[prop];
            var inputIndex:int =
                Math.random()*input.length;
            newObj[prop] = input[inputIndex];
        }
        generatedData.push(newObj);
    }
}

private function populateADG():void
{
    if (generatedData.length != numRows.value)
        generateData();
    adg.dataProvider = generatedData;
}

[Bindable]
private var gc:GroupingCollection;
```

```
private function groupData():void
{
    var fields:Array = [];
    if (company.selected)
        fields.push(new GroupingField("Company"));
    if (product.selected)
        fields.push(new GroupingField("Product"));
    if (year.selected)
       fields.push(new GroupingField("Year"));
    if (fields.length == 0)
    {
        Alert.show("Select at least one of the items to group on");
        return;
    }

    gc = new GroupingCollection();

    gc.source = generatedData;

    gc.grouping = new Grouping();

    gc.grouping.fields = fields;
    // Включаем асинхронное обновление,
    // чтобы ускорить получение результатов
    gc.refresh(true);

    adg.dataProvider = gc;
}

private function handleOptionChange():void
{
    // Пользователь еще не запустил группировку
    if (!gc)
        return;

    // Прерывание текущего обновления
    gc.cancelRefresh();

    var fields:Array = [];
    if (company.selected)
        fields.push(new GroupingField("Company"));
    if (product.selected)
        fields.push(new GroupingField("Product"));
    if (year.selected)
        fields.push(new GroupingField("Year"));

    // Если пользователь отключил все варианты:
    if (fields.length == 0)
    {
        return;
    }
```

```
            gc.grouping.fields = fields;

            gc.refresh(true);
        }
    ]]>
</mx:Script>

    <mx:AdvancedDataGrid id="adg" width="100%" height="260">
        <mx:columns>
           <mx:AdvancedDataGridColumn dataField="Company" />
           <mx:AdvancedDataGridColumn dataField="Product" />
           <mx:AdvancedDataGridColumn dataField="Year" />
           <mx:AdvancedDataGridColumn dataField="Sales" />
        </mx:columns>
    </mx:AdvancedDataGrid>
    <mx:HBox>
        <mx:NumericStepper id="numRows" stepSize="1000"
            minimum="1000" maximum="10000" />
        <mx:Button label="Populate ADG"
            click="populateADG()" id="populateADGButton"/>
    </mx:HBox>
    <mx:VBox>
    <mx:HBox>
```

Этот фрагмент обеспечивает выбор полей группировки:

```
    <mx:Label text="Grouping fields:" />
    <!-- Можно использовать функцию API cancelRefresh
         для прекращения обновления и немедленного
         вызова refresh (true) в обработчике change.-->
    <mx:CheckBox id="company" label="Company" selected="true"
        click="handleOptionChange()"/>
    <mx:CheckBox id="product" label="Product" click="handleOptionChange()"/>
    <mx:CheckBox id="year" label="Year" click="handleOptionChange()"/>
</mx:HBox>
<mx:HBox>
    <mx:Button label="Group" click="groupData()" />
```

А здесь вызывается метод `cancelRefresh` класса `GroupingCollection`:

```
    <mx:Button label="Cancel grouping" click="gc.cancelRefresh()" enabled=
        "{gc != null}"/>
</mx:HBox>
</mx:VBox>
</mx:Application>
```

Три флажка позволяют выбрать разные комбинации условий группировки. Пользователь может изменить группировку во время обновления. Метод `cancelRefresh` класса `GroupCollection` приказывает `AdvancedDataGrid` прервать создание и отображение новой группировки.

7

Рендереры и редакторы

Рендереры (или *рендереры элементов*) – мощная, часто применяемая на практике концепция Flex Framework, которая позволяет разработчику определять компоненты, используемые компонентами данных для отображения массивов или коллекций данных. В частности, они широко используются компонентами `DataGrid`, `List`, `Tile` и `ComboBox`; данные каждого элемента из `dataProvider` передаются определенному рендереру, обеспечивающему отображение и обновление данных. В приложениях Flex широко используются табличные данные и списки, поэтому возможность выбора наиболее эффективного способа отображения данных и их редактирования после вывода является важным аспектом Flex-программирования.

Для эффективной работы с рендерерами и редакторами элементов необходимо хорошо понимать отношения между `itemRenderer` и родительским компонентом, определяемым в Flex Framework. Все рендереры элементов должны обладать свойством `data`, которое используется родительским компонентом для задания соответствующего элемента данных. Способ отображения данных находится на полном усмотрении разработчика; например, он может использовать простой стандартный (drop-in) рендерер или же написать собственный компонент. Рендереры элементов также могут поддерживать редактирование данных, которое приводит к автоматическому обновлению данных `dataProvider` родительского компонента.

Редакторы элементов работают по-разному, но базовая схема более или менее одинакова: `mx.controls.List` создает экземпляр редактора при щелчке на компоненте рендерера. При потере фокуса редактором компонент `List` пытается прочитать свойство `editorDataField`, чтобы сравнить его с оригиналом и узнать, изменились ли данные. Если данные изменились, `dataProvider` компонента `List` обновляется, после чего `itemEditor`

уничтожается и заменяется itemRenderer. Это означает, что для любого столбца List или DataGrid в качестве itemRenderer может быть назначен только один класс.

7.1. Создание собственного рендерера

Задача

Требуется создать рендерер элементов для компонента List или DataGrid.

Решение

Определите рендерер в коде MXML внутри компонента List, передав новый рендерер в свойстве itemRenderer компонента List. Также можно определить рендерер в отдельном файле и передать его itemRenderer с уточнением полного имени класса.

Обсуждение

Работать с рендерерами в их простейшем воплощении несложно; существует несколько способов их создания. Проще всего передать нужный рендерер компоненту List в свойстве itemRenderer:

```
<mx:List dataProvider="{simpleArray}">
    <mx:itemRenderer>
        <mx:Component>
            <mx:Label color="#007700"/>
        </mx:Component>
    </mx:itemRenderer>
</mx:List>
```

Для каждого элемента в массиве или экземпляре ArrayCollection, хранящемся в dataProvider, создается новый экземпляр компонента, назначенного itemRenderer. Информация из соответствующего элемента массива передается свойству data рендерера. Эта схема работает с компонентом Label, использованном в предыдущем фрагменте, потому что свойство Label.data автоматически заполняет Label текстом из передаваемых данных. Чтобы все работало так, как ожидается, данные, передаваемые рендереру, должны быть простыми строками. При передаче Label сложного объекта или другого типа данных будет выведена строка [Object object].

Для передачи сложных объектов или других типов данных, отличных от String, необходимо создать новый рендерер и переопределить метод set data для правильного отображения данных. Пример:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    height="800">
    <mx:Script>
        <![CDATA[
```

```
            [Bindable]
            private var simpleArray:Array = new Array(
                "one", "two", "three", "four", "five");
        ]]>
    </mx:Script>
    <!-- С этим списком можно использовать встроенный
         рендерер, потому что массив, используемый List
         в качестве провайдера данных, состоит
         из простых объектов -->
    <mx:List dataProvider="{simpleArray}">
        <mx:itemRenderer>
            <mx:Component>
                <mx:Label color="#007700"/>
            </mx:Component>
        </mx:itemRenderer>
    </mx:List>
    <!-- Для этого компонента необходим пользовательский
         рендерер, так как ему передается произвольный
         объект; в этом случае рендерер должен знать, как
         обрабатывать каждое поле элемента -->
    <mx:List
        itemRenderer="oreilly.cookbook.SevenOneRenderer"
        dataProvider="{DataHolder.genericCollectionOne}"/>
    <!-- Здесь можно использовать подходящий стандартный
         рендерер, так как DataGrid обрабатывает каждое поле
         Object отдельно от других -->
    <mx:DataGrid
        dataProvider="{DataHolder.genericCollectionOne}">
        <mx:columns>
            <mx:DataGridColumn dataField="name"/>
            <mx:DataGridColumn dataField="age"/>
            <mx:DataGridColumn dataField="appearance"
                width="200">
                <mx:itemRenderer>
                    <mx:Component>
                        <!-- Любой компонент, размещаемый
                             здесь, должен расширять
                             IDataRenderer и обладать
                             свойством data, которое
                             правильно отображает данные,
                             передаваемые методу set data-->
                        <mx:TextArea/>
                    </mx:Component>
                </mx:itemRenderer>
            </mx:DataGridColumn>
        </mx:columns>
    </mx:DataGrid>
</mx:VBox>
```

Конечно, для сколько-нибудь нетривиальных данных приходится создавать пользовательский рендерер. Ни один из компонентов `Flex`, которые могут включаться в тег `Component`, не поддерживает работу с данными,

состоящими из сложных объектов. Часто это не создает проблем – объекты данных, состоящие из нескольких полей, разбираются компонентом DataGrid, а каждое свойство объекта обрабатывается отдельным рендерером. Впрочем, существуют и исключения. Чтобы корректно обработать объект данных с несколькими полями, необходимо переопределить метод set data и обработать каждое свойство полученного объекта. В следующем примере рендерер разбирает объект данных, полученный от списка, и для каждого поля создается компонент Label. Такое решение хорошо работает в том случае, если поля данных могут отсутствовать.

```
package oreilly.cookbook
{
    import mx.containers.HBox;
    import mx.controls.Label;
    import mx.core.IDataRenderer;

    public class SevenOneRenderer extends HBox
    {
        private var nameLabel:Label;
        private var ageLabel:Label;
        private var appearanceLabel:Label;
        private var _data:Object;
        public function SevenOneRenderer() {
            super();
        }

        override public function get data():Object {
            if(_data != null) {
                return _data;
            }
            return null;
        }

        override public function set data(value:Object):void
        {
            _data = value;
            if(_data.name != null) {
                nameLabel = instantiateNewLabel(_data.name);
            }
            if(_data.age != null) {
                ageLabel = instantiateNewLabel(_data.age);
            }
            if(_data.appearance != null) {
                appearanceLabel =
                    instantiateNewLabel(_data.appearance);
            }
            setStyle("backgroundColor", 0xddddff);
        }

        private function instantiateNewLabel(value:*):Label
        {
```

```
            var label:Label = new Label();
            label.text = String(value);
            addChild(label);
            return label;
        }
    }
}
```

7.2. Использование ClassFactory для создания рендереров

Задача

Требуется изменить свойства рендереров, используемых компонентом List или DataGrid, во время выполнения.

Решение

Определите рендерер в коде MXML внутри компонента List, передав новый рендерер в свойстве itemRenderer компонента List. Также можно определить рендерер в отдельном файле и передать его itemRenderer с уточнением полного имени класса.

Обсуждение

Рецепт получился довольно длинным, и это понятно – в нем рассматриваются сразу две темы: архитектурный паттерн «Фабрика» и его реализация mx.core.ClassFactory, встроенная в Flex Framework. Паттерн «Фабрика» позволяет назначить класс, который должен использоваться для создания рендереров компонента, управляемого данными; ClassFactory занимается созданием и уничтожением экземпляров этого класса во время выполнения. ClassFactory передается ссылка на класс следующего вида:

```
var factory:ClassFactory = new ClassFactory(oreilly.cookbook.SevenTwoFac-
tory);
```

При условии, что класс SevenTwoFactory реализует интерфейс IFactory, все отлично сработает, и его можно будет передать PurgeList так, как показано в этом фрагменте:

```
<cookbook:PurgeList id="list" itemRenderer="{factory}" width="300"/>
```

В следующем примере представлен пользовательский список, который предоставляет доступ к одному методу – защищенному методу purgeItemRenderers класса ListBase:

```
public function clearList():void{
    this.purgeItemRenderers();
    this.invalidateDisplayList();
}
```

Теперь взгляните, как `ClassFactory` **получает ссылку на класс, а затем назначается** `itemRenderer` **компонента** `List`**:**

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
  initialize="setType()" mlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var factory:ClassFactory;

            private function setType():void
            {
                factory = new ClassFactory(SevenTwoFactory);
                factory.properties =
                    {type:SevenTwoFactory.HORIZONTAL};
            }
        ]]>
    </mx:Script>
    <cookbook:PurgeList id="list" itemRenderer="{factory}"
        width="300"/>
    <!-- Переключение между горизонтальной и вертикальной
         прорисовкой элементов -- >
    <mx:Button id="toggleStyle" toggle="true" click="toggleStyle.selected ?
        factory.properties = {type:SevenTwoFactory.VERTICAL} :
        factory.properties = {type:SevenTwoFactory.HORIZONTAL}"
        label="style"/>
        <!-- Перерисовка всего списка
             с использованием новых рендереров -- >
        <mx:Button id="toggleDP" toggle="true" click="list.clearList(),
             list.dataProvider = DataHolder.genericCollectionOne,
             this.invalidateDisplayList()" label="generate"/>
</mx:HBox>
```

Объект `properties` **класса** `ClassFactory` **передается новому объекту класса, сгенерированному фабрикой. Центральная роль в классах-фабриках отводится интерфейсу** `IFactory`**, требующему присутствия метода** `newInstance` **со следующей сигнатурой:**

```
public function newInstance():* {
    return new MyClass();
}
```

Объект `ClasFactory` **вызывает этот метод каждый раз, когда потребуется создать новый экземпляр класса.** `ClassFactory` **пытается передать полученные свойства создаваемому экземпляру класса. В предыдущем примере** `ClassFactory` **передает новому экземпляру класса объект**

```
factory.properties = {type:SevenTwoFactory.HORIZONTAL};
```

Следующий рендерер сохраняет значение, заданное в `properties`**, функцией** `set`**:**

```
public function set type(value:String):void {
    _type = value;
    invalidateDisplayList();
}
```

ClassFactory **пытается передать каждое свойства в объекте** properties **новому экземпляру класса. Именно это обстоятельство позволяет передавать новые данные рендереру во время выполнения и перерисовывать все рендереры с использованием новой информации. Для рендерера, используемого в этом примере, вместо контейнерных классов из пакета** mx.containers **используется** UIComponent**; этим обеспечивается максимальная скорость перерисовки. По той же причине вместо компонентов** mx.controls.Text **или** Label **используется компонент** flash.text.TextField**. При использовании рендереров (и особенно при их перерисовке) особенно важна скорость, а низкоуровневые компоненты ActionScript экономят время и повышают быстродействие.**

```
package oreilly.cookbook
{
    import flash.text.TextField;
    import mx.controls.Label;
    import mx.controls.listClasses.IListItemRenderer;
    import mx.core.IFactory;
    import mx.core.UIComponent;
    import mx.events.FlexEvent;
    // Класс реализует интерфейсы IListItemRenderer
    // (обеспечивает правильное взаимодействие с List)
    // и IFactory (обеспечивает возможность
    // использования с фабрикой).
    public class SevenTwoFactory extends UIComponent
        implements IFactory, IListItemRenderer
    {
        // Два типа раскладки, которые будут
        // использоваться при назначении 'type'
        public static const HORIZONTAL:String = "horizontal";
        public static const VERTICAL:String = "vertical";
        // По умолчанию используется
        // горизонтальная раскладка
        private var _type:String = HORIZONTAL;

        private var _data:Object;
        // Три компонента TextField
        private var nameLabel:TextField;
        private var ageLabel:TextField;
        private var appearanceLabel:TextField;
        // Свойство, заданное в объекте
        // ClassFactory.properties
        public function set type(value:String):void {
            _type = value;
            invalidateDisplayList();
        }
```

```
public function get data():Object {
    return _data;
}
// Необходимо для определения правильного размера
// рендерера.
override protected function measure():void {
    super.measure();
    if(this._type == HORIZONTAL) {
        measuredHeight = nameLabel.height;
        measuredWidth = nameLabel.width +
            ageLabel.width + appearanceLabel.width + 10;
    } else {
        measuredWidth = appearanceLabel.width;
        measuredHeight = nameLabel.height +
            ageLabel.height + appearanceLabel.height + 10;
    }
    height = measuredHeight;
    width = measuredWidth;
    trace(" w "+this.measuredWidth+ " "+
        this.measuredHeight+" "+this.width+" "+this.height);
}

// Заполнение полей TextField правильными данными,
// полученными в результате разбора. Экземпляр
// TextField создается только в том случае, если он
// действительно необходим.
public function set data(value:Object):void {
    _data = value;
    if(_data.name != null && nameLabel == null) {
        nameLabel = instantiateNewLabel(_data.name);
    } else {
        nameLabel.text = _data.name;
    }
    if(_data.age != null && ageLabel == null) {
        ageLabel = instantiateNewLabel(_data.age);
    } else {
        ageLabel.text = _data.age;
    }
    if(_data.appearance != null && appearanceLabel == null) {
        appearanceLabel = instantiateNewLabel(_data.appearance);
    } else {
        appearanceLabel.text = _data.appearance;
    }
    setStyle("backgroundColor", 0xddddff);
    invalidateProperties();
    dispatchEvent(new FlexEvent(FlexEvent.DATA_CHANGE));
}
// UIComponent не обладает логикой раскладки,
// поэтому все придется делать самостоятельно.
override protected function updateDisplayList(
unscaledWidth:Number,
```

```
            unscaledHeight:Number):void {
                super.updateDisplayList(unscaledWidth, unscaledHeight);
                var sum:Number = 0;
                if(this._type == HORIZONTAL) {
                    nameLabel.x = 0;
                    sum += nameLabel.width;
                    ageLabel.x = sum;
                    sum += ageLabel.width;
                    appearanceLabel.x = sum;
                } else {
                    nameLabel.y = 0;
                    sum += nameLabel.height;
                    ageLabel.y = sum;
                    sum += ageLabel.height;
                    appearanceLabel.y = sum;
                }
            }

            private function instantiateNewLabel(
            value:*):TextField {
                var text:TextField = new TextField();
                addChild(text);
                text.text = String(value);
                text.width = text.textWidth + 5;
                text.height = text.textHeight + 5;
                return text;
            }

            // Метод-фабрика, возвращающий новый экземпляр
            // для каждого запрашиваемого рендерера
            public function newInstance():* {
                return new SevenTwoFactory();
            }
        }
    }
```

7.3. Обращение к владельцу рендерера

Задача

Требуется обеспечить доступ из рендерера к создавшему его компоненту.

Решение

Реализуйте интерфейс `IDropInListItemRenderer` **и обратитесь к свойству** `owner` **объекта рендерера.**

Обсуждение

Интерфейс `IDropInListItemRenderer` **предоставляет рендереру доступ не только к переданным ему данным, через свойство** `owner` **типа** `BaseListData`

рендерер также может обратиться к компоненту List или DataGridColumn, который является владельцем рендерера. Тип mx.controls.listClasses.BaseListData определяет следующие свойства:

columnIndex : int

Индекс столбца компонента на базе List относительно текущих видимых столбцов; первый столбец обозначается индексом 1.

owner : IUIComponent

Объект List или DataGridColumn, являющийся владельцем рендерера.

rowIndex : int

Индекс строки компонента DataGrid, List или Tree относительно текущих видимых строк компонента; первая строка обозначается индексом 1.

uid : String

Уникальный идентификатор элемента. Всем элементам в itemRenderer назначаются уникальные идентификаторы, чтобы даже при полном совпадении данных двух и более itemRenderer компонент ListBase мог различить их.

При создании itemRenderer, реализующего интерфейс IDropInListItemRenderer, будет задано свойство listData рендерера, и рендерер получит доступ к переданному ему объекту BaseListData. После заполнения данных рендерер проверяет тип компонента-владельца, задавшего данные, и в зависимости от полученной информация либо отслеживает его имя, либо вызывает пользовательский метод владельца.

В следующем примере использована простая реализация этого процесса, которая демонстрирует, что рендерер может задать BaseListData, определить тип создавшего его компонента List и вызвать его методы.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="50" implements=
"mx.controls.listClasses.IDropInListItemRenderer">
    <mx:Script>
        <![CDATA[

            import mx.controls.DataGrid;
            import mx.controls.List;
            import mx.controls.dataGridClasses.DataGridColumn;

            import mx.controls.listClasses.BaseListData;

            // Сохранение элемента данных списка
            // на случай, если позднее потребуется
            // с ним что-нибудь сделать.
            private var _listData:BaseListData;
            [Bindable("dataChange")]
```

```
            // Метод get
            public function get listData():BaseListData
            {
                return _listData;
            }
            // Метод set
```

После передачи компоненту значения `BaseListData` **рендерер получает доступ к своему создателю через свойство** `owner` **объекта** `BaseListData`**. Это позволяет рендереру читать дополнительные данные и вызывать методы родительского компонента.**

```
        public function set listData(
        value:BaseListData):void
        {
            _listData = value;
            if(value.owner is DataGridColumn) {
                trace(" DataGridColumn ");
            } else if (value.owner is List) {
                trace(" List ");
            } else if (value.owner is CustomDataGrid) {
                trace(" CustomDataGrid ");
                (value.owner as
                    CustomDataGrid).checkInMethod(this);
            }
        }

        override public function set data(
        value:Object):void {
            nameTxt.text = value.name;
            appearanceTxt.text = value.appearance;
        }
    ]]>
</mx:Script>
<mx:Canvas backgroundColor="#3344ff">
    <mx:Label id="nameTxt"/>
</mx:Canvas>
<mx:Label id="appearanceTxt"/>
</mx:VBox>
```

Реализация `IDropInListRenderer`**, используемая с простой пользовательской версией** `DataGrid`**, выглядит так:**

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cookbook="oreilly.cookbook.*">
    <mx:List
        itemRenderer="oreilly.cookbook.IDropInListRenderer"
        dataProvider="{DataHolder.genericCollectionOne}"/>
    <!-- При создании пользовательского компонента с новым
         пространством имен cookbook столбцы должны
         объявляться в пространстве cookbook -->
    <cookbook:CustomDataGrid
        dataProvider="{DataHolder.genericCollectionOne}">
```

```
            <cookbook:columns>
                <!-- Так как dataField не объявляется,
                     рендереру передается весь объект данных -->
                <mx:DataGridColumn    itemRenderer=
                    "oreilly.cookbook.IDropInListRenderer"/>
                <mx:DataGridColumn dataField="age"/>
            </cookbook:columns>
        </cookbook:CustomDataGrid>
    </mx:HBox>
```

Остается привести код пользовательского компонента `DataGrid` с единственным методом, который вызывается рендерером при задании свойства `data`:

```
<mx:DataGrid xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[

            public function checkInMethod(obj:*):void {
                trace(" hello from my renderer "+obj);
            }
        ]]>
    </mx:Script>
</mx:DataGrid>
```

7.4. Совмещение рендерера с редактором

Задача

Требуется создать единый компонент, который может использоваться и в качестве рендерера, и в качестве редактора. Дополнительно требуется передать редактору массив значений, которые могут редактироваться пользователем в компоненте `ComboBox`.

Решение

Создайте компонент, реализующий интерфейс `IDropInLisItemRenderer` и имеющий два состояния: в одном состоянии компонент выполняет функции редактора, а в другом – функции рендерера. При изменении данных передайте родительскому контейнеру событие `ITEM_EDIT_END`, чтобы контейнер мог задать новое значение в `dataProvider`.

Обсуждение

Рендереры элементов легко совмещаются с функциями редактора элементов. Класс `List`, расширяемый классом `DataGridColumn`, определяет свойство `rendererIsEditor`. Если задать этому свойству значение `true`, компонент `List` при двойном щелчке на `itemRenderer` не создает редактор по умолчанию, а просто разрешает `itemRenderer` взять редактирование на себя.

Чтобы класс List изменил dataProvider при завершении редактирования, рендереру достаточно передать событие ListEvent или DataGridEvent (в зависимости от родителя) с типом ITEM_EDIT_END. В следующем примере DataGridEvent используется для передачи индексов редактируемой строки и столбца для обновления dataProvider. Объект DataGridEvent создается и передается методом setNewData, который вызывается при изменении компонента ComboBox, используемого для редактирования:

```
private function setNewData():void
{
    _data.selected = selectCB.selectedItem;
    dispatchEvent(
        new DataGridEvent(DataGridEvent.ITEM_EDIT_END, true,
        true, listData.columnIndex, 'selected',
       _listData.rowIndex));
}
```

В следующем примере тип данных, используемых компонентом, содержит поля имени, возраста и внешнего вида, массив дополнительных параметров, которые могут редактироваться пользователем, а также свойство selected, которое определяет, какой из элементов массива находится в выделенном состоянии:

```
{name:"Todd Anderson", age:31, appearance:"Intimidating", extras:["bar",
"foo","baz"], selected:"bar"}
```

Подобные типы данных очень часто встречаются на практике. Для рендереров особенно важны свойство extras, определяющее значение dataProvider компонента ComboBox, и свойство selected, определяющее выделенную строку ComboBox. Обычно в DataGrid рендереру передается только одно значение; но если DataGridColumn не определяет свойства dataField, то рендереру передается весь объект:

```
<mx:DataGrid dataProvider="{DataHolder.genericCollectionTwo}"
width="450" itemEditEnd="checkEditedItem(event)" id="dg">
    <mx:columns>
        <mx:DataGridColumn dataField="age"/>
        <mx:DataGridColumn dataField="appearance"/>
        <mx:DataGridColumn
            itemRenderer="oreilly.cookbook.ComboBoxRenderer"
            editable="true" rendererIsEditor="true"/>
    </mx:columns>
</mx:DataGrid>
```

Обратите внимание на свойство rendererIsEditor, заданное для столбца. Рендерер показан в следующем фрагменте:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="100"
doubleClickEnabled="true" currentState="display"
implements=
"mx.controls.listClasses.IDropInListItemRenderer">
```

```
<mx:Script>
    <![CDATA[
        import mx.events.DataGridEvent;
        import mx.events.ListEvent;

        import mx.controls.listClasses.BaseListData;

        // Внутренняя переменная для значения свойства.
        private var _listData:BaseListData;
        private var _data:Object = {};
```

После назначения данных для рендерера компонент начинает прослушивать событие `MouseEvent.DOUBLE_CLICK`**, по которому произойдет переход в состояние редактирования:**

```
override public function set data(value:Object):void {
    _data = value;
    if(_data.selected != null) {
        addEventListener(
            MouseEvent.DOUBLE_CLICK, startEdit);
    }
}
override public function get data():Object {
    return _data;
}

private function startEdit(event:Event):void {
    if(currentState == "display") {
        currentState = "edit";
        addEventListener(FocusEvent.FOCUS_OUT, endEdit);
    }
}

private function endEdit(event:Event):void {
    currentState = "display";
}

// Включение привязки для свойства listData.
[Bindable("dataChange")]
public function get listData():BaseListData {
    return _listData;
}

// Определение метода set
public function set listData(value:BaseListData):void {
    _listData = value;
}

private function setNewData():void {
    _data.selected = selectCB.selectedItem;
    dispatchEvent(new DataGridEvent(
        DataGridEvent.ITEM_EDIT_END, true,
```

```
            true,_listData.columnIndex, 'selected',
            _listData.rowIndex));
    }
]]>
</mx:Script>
```

В этом примере тег `States` используется для перевода компонента из состояния рендерера, в котором данные просто отображаются, в состояние редактора элементов, в котором компонент `ComboBox` может использоваться для редактирования данных:

```
<mx:states>
    <mx:State name="edit">
        <mx:AddChild>
            <mx:ComboBox id="selectCB"
                addedToStage="selectCB.dataProvider = _data.extras;
                    selectCB.selectedItem = _data.selected"
                change="setNewData()"/>
        </mx:AddChild>
    </mx:State>
    <mx:State name="display">
        <mx:AddChild>
            <mx:Text id="text"
                addedToStage="text.text = _data.selected"/>
        </mx:AddChild>
    </mx:State>
</mx:states>
</mx:Canvas>
```

7.5. Создание редактора для работы с несколькими полями данных

Задача

Требуется создать редактор элементов с возможностью редактирования типов данных, состоящих из нескольких полей, например пользовательских объектов.

Решение

Создайте редактор, который будет возвращать отредактированные элементы в свойстве `data`. Создайте в компоненте `List` слушателя события `itemEditEnd`, который будет отменять событие и обращаться к свойству `data` экземпляра `itemEditorInstance` компонента `List`.

Обсуждение

Компонент `List` предоставляет свойство `editorDataField`, которое отлично подходит для отдельных полей или свойств. Но если редактор работает с несколькими полями, стандартное поведение `List` и `DataGridColumn`

необходимо отменить, а данные полей должны читаться из itemEditorInstance.

Метод processData в этом фрагменте используется для обработки события itemEditEnd:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="700"
height="300">
    <mx:Script>
        <![CDATA[
            import mx.events.ListEvent;
            import mx.collections.ArrayCollection;
            import oreilly.cookbook.MultipleDataTypeEditor;

            [Bindable]
            private var arr:ArrayCollection =
              new ArrayCollection([{age:12, name:"Joe"},
              {age:16, name:"Jorge"}, {age:19, name:"Jojo"},
              {age:2, name:"James"},
              {age:12, name:"Joaquin"}]);

            public function processData(
            event:ListEvent):void {
                // Запрет на обратное копирование
                // данных в компонент.
                event.preventDefault();
                // Получение новой метки от редактора.
                list.editedItemRenderer.data = MultipleDataTypeEditor(
                    list.itemEditorInstance).data;
                // Закрытие редактора ячейки
                list.destroyItemEditor();
                // Оповещение компонента списка
                // о перерисовке изображения.
                list.dataProvider.notifyItemUpdate(
                    list.editedItemRenderer);
            }
        ]]>
    </mx:Script>
    <mx:List id="list" itemEditor="oreilly.cookbook.MultipleDataTypeEditor"
    dataProvider="{arr}" itemEditEnd="processData(event)"
    itemRenderer="oreilly.cookbook.MultipleDataTypeRenderer"
    width="350" editable="true">
    </mx:List>
</mx:Canvas>
```

Метод preventDefault события отменяет стандартное поведение itemEditor. Он вызывается для того, чтобы компонент List не пытался прочитать свойство text из itemEditor, как это обычно делается. Вместо этого данные читаются из itemRendererInstance, и прочитанное значение записывается в данные editedItemRenderers. Наконец, вызов notifyItemUpdate для dataProvider компонента List гарантирует, что новые данные, переданные редактором, будут отражены в dataProvider.

Реализация `itemEditor`, использованная в предшествующем `List`, просто возвращает значения двух объектов `TextInput` при обращении к свойству `data`. Программная реализация редактора выглядит так:

```
package oreilly.cookbook
{
    import mx.containers.Canvas;
    import mx.controls.TextInput;

    public class MultipleDataTypeEditor extends Canvas
    {
        private var nameField:TextInput;
        private var ageField:TextInput;

        public function MultipleDataTypeEditor() {
            super();
            nameField = new TextInput();
            ageField = new TextInput();
            addChild(nameField);
            nameField.focusEnabled = false;
            addChild(ageField);
            ageField.focusEnabled = false;
            ageField.x = 100;
            this.horizontalScrollPolicy = "none";
        }

        override public function set data(
        value:Object):void {
            super.data = value;
            nameField.text = value.name;
            ageField.text = value.age;
        }

        override public function get data():Object {
            return {name:nameField.text, age:ageField.text};
        }
    }
}
```

7.6. Объекты SWF в меню

Материал предоставлен Рико Зунига (Rico Zuniga).

Задача

Требуется вывести SWF или объект графического изображения в меню.

Решение

Используйте объект `itemRenderer` для загрузки файлов SWF и настройки меню.

Обсуждение

Первым шагом по модификации меню должно стать создание файлов SWF, содержащих шрифты и графику, и сохранение их в папке. В приведенном примере используется папка `swf`. Для создания файлов можно использовать Flash, Flex или любой другой инструмент. Созданные файлы сохраняются в папке. В нашем примере папка swf содержит файл SWF со статическим текстом, оформленным нужным шрифтом.

Далее создайте компонент-рендерер на основе `Canvas`, содержащий компонент `SWFLoader`. Для работы в контексте пользовательских меню в `Canvas` должен быть реализован интерфейс `IMenuItemRenderer`. Чтобы решить эту задачу в MXML, задайте нужный интерфейс свойству `implements`. Также необходимо переопределить методы `get` и `set` интерфейса `IMenuItemRenderer` для его свойства `menu`, но в нашем примере никакие дополнения не требуются. В компоненте `SWFLoader` свойству `source` задается значение `data.swf`; оно представляет элемент `dataProvider` компонента `Menu`, содержащий путь к файлам SWF. Программный код выглядит так:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100" height="25" verticalScrollPolicy="off"
    horizontalScrollPolicy="off"
    xmlns:external="flash.external.*"
    implements="mx.controls.menuClasses.IMenuItemRenderer">
    <mx:Script>
    <![CDATA[
    import mx.controls.Menu;

    public function get menu():Menu {
    return null;
    }

    public function set menu(value:Menu):void {
    }

    ]]>
    </mx:Script>
    <mx:SWFLoader source="{data.swf}" width="100"
    height="25" horizontalCenter="0"
    verticalCenter="0"/>
</mx:Canvas>
```

Далее в главном приложении необходимо создать пользовательскую реализацию `Menu`. В свойстве `Menu.dataProvider` задается `menuData` — массив объектов, представляющих структуру `Menu`. Добавьте свойство `swf`, представляющее путь к файлу SWF, отображаемому для каждого объекта команды меню. Для меню также необходимо задать `itemRenderer`, создав новый объект `ClassFactory` и передав его конструктору компонент рендерера; необходимость этого шага объясняется тем, что свойство `itemRenderer` класса `Menu` должно содержать объект, реализующий интерфейс `IFactory`. Код главного приложения:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
creationComplete="init();" backgroundGradientColors="[#c0c0c0, #ffffff]">
    <mx:Script>
    <![CDATA[
    import mx.events.MenuEvent;
    import mx.controls.Menu;
    private var menu:Menu;
    private function init():void {
        var menuData:Array = [
        {swf:'swf/coolfonts.swf', children: [
        {label: "SubMenuItem A-1", swf:'swf/meridiana.swf'},
        {label: "SubMenuItem A-2", swf:'swf/virinda.swf'}
        ]},
        {swf:'swf/scriptfonts.swf', children: [
        {label: "SubMenuItem A-1", swf:'swf/monotypecorsiva.swf'},
        {label: "SubMenuItem A-2", swf:'swf/comicsansms.swf'}
        ]}
    ];
    menu = Menu.createMenu(this, menuData);
    menu.itemRenderer = new ClassFactory(FontItemRenderer);
    }
    ]]>
    </mx:Script>
    <mx:Button x="10" y="10" label="Show Menu" id="btnShowMenu" click=
"menu.show(btnShowMenu.x,btnShowMenu.y+btnShowMenu.height);"/>
</mx:Application>
```

7.7. Выбор DataGridColumn с CheckBoxHeaderRenderer

Материал предоставлен Беном Клинкинбердом (Ben Clinkinbeard).

Задача

Требуется создать компонент `DataGrid`, в одном из столбцов которого в качестве `itemRenderer` используется `CheckBox`, а `headerRenderer` этого столбца представляет собой компонент `CheckBox`, используемый для установки/снятия выделения со всех ячеек столбца.

Решение

Создайте класс, используемый в качестве `headerRenderer`. Определите в этом классе метод, который будет передавать событие родительскому компоненту `DataGrid` и задавать свойство для каждого `itemRenderer`.

Обсуждение

Рендереры заголовков Flex отличаются от рендереров элементов как по способу создания, так и по жизненному циклу. На протяжении су-

ществования DataGrid экземпляры headerRenderer многократно инициализируются заново, поскольку практически любое действие с любым заголовком DataGrid приводит к сбросу headerRenderer. Как следствие, все данные с поддерживаемым состоянием должны храниться за пределами DataGrid. Для этой цели следует использовать класс mx.core.ClassFactory, который обычно работает незаметно для разработчика. При назначении класса рендерера:

```
<mx:DataGridColumn itemRenderer="mx.controls.CheckBox" />
```

компилятор MXML автоматически преобразует этот код в конструкцию следующего вида:

```
var temp : mx.core.ClassFactory = new mx.core.ClassFactory();
temp.generator = mx.controls.CheckBox;
return temp;
```

Как видите, указанный вами класс назначается свойству generator. При создании строк DataGrid вызывается метод newInstance соответствующего экземпляра ClassFactory, который создает и возвращает свежий экземпляр класса, заданного в свойстве generator. Такая схема удобна тем, что вам достаточно указать, какой класс должен использоваться для отображения данных, но у нее есть свои недостатки. Главный недостаток заключается в том, что вы не передаете аргументы при вызове конструктора, а это усложняет создание классов сколько-нибудь универсальных рендереров, подходящих для разных сценариев. К счастью, ClassFactory предоставляет механизм, упрощающий создание рендеров для повторного использования. Свойство properties относится к типу Object и используется для инициализации экземпляров класса рендерера. ClassFactory перебирает свойства, публикуемые объектом properties, в цикле for...in и задает те же значения в рендерере.

На основании того, что было сказано ранее о классе ClassFactory и рендерерах заголовков, мы создаем несколько переменных:

```
// Переменная для хранения состояния рендерера заголовка
public var selectAllFlag:Boolean;
[Bindable]
public var hr:ClassFactory;
```

В переменной selectAllFlag хранится флаг, который указывает, должны ли быть выделены все ячейки столбца. Переменная hr задается свойству headerRenderer класса DataGridColumn и помечается [Bindable] для удобства.

На следующем шаге определяется фактический экземпляр ClassFactory; это можно сделать в обработчике события creationComplete того файла, в котором находится ваш код DataGrid:

```
hr = new ClassFactory(CenteredCheckBoxHeaderRenderer);
hr.properties = {stateHost: this, stateProperty: "selectAllFlag"};
```

Перед тем как перейти к рассмотрению кода рендерера, обратим внимание на DataGridColumn:

```
<mx:DataGridColumn width="30"
sortable="false"
dataField="addToCart"
headerRenderer="{hr}"
itemRenderer="CenteredCheckBoxItemRenderer" />
```

Обратите внимание: свойству sortable задано значение false. Это необходимо для того, чтобы щелчки регистрировались как операции с CheckBox, а не как операции сортировки столбцов.

Переходим к классу рендерера заголовка:

```
package {
    import flash.display.DisplayObject;
    import flash.events.MouseEvent;
    import flash.text.TextField;
    import mx.controls.CheckBox;
    import mx.controls.DataGrid;
    public class CenteredCheckBoxHeaderRenderer extends CenteredCheckBox  {
        // Переменные используются для работы с внешним
        // свойством, в котором хранится состояние выделения
        public var stateHost:Object;
        public var stateProperty:String;

        // Функция вызывается многократно в ходе (повторной)
        // инициализации. Состояние выделения задается
        // на основании внешнего свойства.
        override public function set data(value:Object):void  {
            selected = stateHost[stateProperty];
        }

        // Функция определяется в mx.controls.CheckBox.
        // Является обработчиком по умолчанию для события click.
        override protected function
        clickHandler(event:MouseEvent):void  {
            super.clickHandler(event);
            // Эта строка важна - она обновляет внешнюю переменную,
            // созданную для хранения состояния.
            stateHost[stateProperty] = selected;
        }
    }
}
```

Метод set data игнорирует переданный ему аргумент (ссылка на экземпляр DataGridColumn, содержащий рендерер) и устанавливает состояние выделения по внешней флаговой переменной, на которую мы указали рендереру в свойстве properties. И наоборот, в функции clickHandler внешняя флаговая переменная задается в соответствии с новым состоянием выделения.

Пример получился довольно простым, однако уметь пользоваться ClassFactory полезно, потому что этот класс, пожалуй, является лучшим способом создания хоть сколько-нибудь нетривиальных рендереров, рассчитанных на повторное использование. Он также может применяться и для рендереров элементов; используйте ту же методологию задания свойств и значений для экземпляров класса рендерера.

Остается сделать последний шаг: связать операцию щелчка на флажке в заголовке с установкой/снятием всех компонентов CheckBox в рендерере элементов. Для этого следует просто прослушивать событие MouseEvent.CLICK в родительском файле DataGrid, так как эти события будут «всплывать» при каскадной передаче, и выполнить необходимые действия здесь. Убедившись в том, что событие поступило от рендерера заголовка, переберите данные и задайте соответствующему полю данных правильное значение для каждого элемента. Также для каждого элемента необходимо вызвать itemUpdated; это приведет к перерисовке рендереров, чтобы они точно соответствовали своим данным. Пример:

```
private function onCheckBoxHeaderClick(event:MouseEvent):void
{
    // Убедиться в том, что событие click
    // получено от заголовка
    if(event.target is CenteredCheckBoxHeaderRenderer)
    {
        // Перебор данных в цикле
        for each(var obj:Object in dg.dataProvider)
        {
            // Обновление значения на основании
            // состояния CheckBox
            obj.addToCart =
                CenteredCheckBoxHeaderRenderer(
                    event.target).selected;
            // Оповещение об изменении элемента коллекции
            ListCollectionView(dg.dataProvider).itemUpdated(
            obj, "addToCart");
        }
    }
}
```

7.8. Создание автономного рендерера CheckBox для использования в DataGrid

Материал предоставлен Беном Клинкинбердом (Ben Clinkinbeard).

Задача

Требуется создать itemRenderer для компонента CheckBox, используемого в DataGrid. Компонент CheckBox должен выравниваться по центру независимо от размера DataGridColumn.

Решение

Расширьте класс CheckBox и переопределите метод updateDisplayList так, чтобы компонент правильно выравнивался по центру. В обработчике clickHandler компонента CheckBox задайте данным родительского компонента DataGrid значение true или false в зависимости от состояния CheckBox.

Обсуждение

Следующий класс может использоваться как полностью автономный рендерер, автоматически выравнивающий свой компонент по центру родительского экземпляра DataGridColumn. Хотя класс предоставляет всю функциональность прорисовки элементов, событие MouseEvent все равно передается наверх – на тот случай, если вы захотите перехватить его для других целей. Например, в файле с DataGrid можно определить слушателя, который перехватывает MouseEvent и обновляет метку сообщением о количестве установленных флажков:

```
package
{
    import flash.display.DisplayObject;
    import flash.events.MouseEvent;
    import flash.text.TextField;
    import mx.controls.CheckBox;
    import mx.controls.dataGridClasses.DataGridListData;

    public class CenteredCheckBoxItemRenderer extends CheckBox
    {
        // Функция определяется mx.controls.CheckBox;
        // обработчик по умолчанию для события click.
        override protected function clickHandler(event:MouseEvent):void
        {
            super.clickHandler(event);
            // Важная строка - обновляет поле данных,
            // соответствующее данному CheckBox.
            data[DataGridListData(listData).dataField] = selected;
        }

        // Выравнивание флажка по центру
        override protected function
        updateDisplayList(w:Number, h:Number):void
        {
            super.updateDisplayList(w, h);
            var n:int = numChildren;
            for (var i:int = 0; i < n; i++)
            {
                var c:DisplayObject = getChildAt(i);
                // Компонент CheckBox состоит из значка
                // и текстовой метки. Игнорируем метку
                // и выравниваем по центру значок.
```

```
                    if (!(c is TextField))
                    {
                        c.x = Math.round((w - c.width) / 2);
                        c.y = Math.round((h - c.height) / 2);
                    }
                }
            }
        }
    }
```

В этом решении стоит обратить внимание на два момента. Во-первых, оно не реализует `ClassFactory`; это означает, что в атрибуте `itemRenderer` компонента `DataGrid` можно просто ввести полное имя класса, и оно сработает (как в случае с `mx.controls.CheckBox`), а в вашем файле потребуется на одну привязываемую переменную меньше. Это возможно благодаря тому, что `CheckBox` (а соответственно и `CenteredCheckBoxItemRenderer`) реализуют интерфейс `IDropInListItemRenderer`. В данном случае компонент знает, как задать свое состояние выделения на основании переданных ему данных. Второе серьезное различие неразрывно связано с этим фактом: вам уже не придется переопределять метод `set data`, так как вы получаете соответствующую функциональность автоматически.

Самая важная строка в классе `itemRenderer` обусловлена этим же обстоятельством. Хотя выборка и отображение данных обеспечиваются автоматически, функциональность сохранения данных необходимо реализовать самостоятельно. Правильное назначение данных – единственное, что отличает нашу версию `CheckBox` от стандартной (если не считать кода выравнивания по центру), и эта задача решается одной строкой кода:

```
data[DataGridListData(listData).dataField] = selected;
```

Свойство `listData` (определяемое интерфейсом `IDropInListItemRenderer`) в этой строке используется для выборки и обновления конкретного фрагмента данных, отображаемого компонентом в `DataGrid`.

7.9. Эффективное отображение графики в рендерере

Задача

Требуется отобразить графическое изображение в `itemRenderer` наиболее эффективным способом.

Решение

Создайте новый класс рендерера и используйте метод `commitProperties` для обращения к владельцу `itemRenderer` с использованием свойства `listData` интерфейса `IDropInItemRenderer`. Метод родителя может вернуть встроенный объект графического изображения, добавляемый в список отображения рендерера.

Обсуждение

Алекс Харуи (Alex Harui), один из архитекторов классов `List` и `ListBase` для Flex 3, написал в своем блоге: «Можно засунуть в `Canvas` тег `Image`, но это не лучшее решение». Почему? Потому что Flex Framework упрощает нашу жизнь сложной серией вызовов для вычисления размеров компонента, перерисовки всех дочерних компонентов и самого компонента. Чем больше вы работаете в Flex Framework, тем более вы склонны выполнять всю работу обычными средствами Framework, не возлагая лишнюю вычислительную нагрузку на Flash Player.

Рендерер в этом примере реализует интерфейсы `IListItemRenderer` и `IDropInListItemRenderer`; это обеспечивает гибкое использование рендерера с минимальными побочными затратами ресурсов. Так как компонент расширяет тег `UIComponent`, необходимо переопределить метод `measure`, чтобы компонент точно сообщал свой размер родительскому компоненту `DataGrid`. Реализация метода `measure` основана на вычислении размеров изображения и задании свойств `measuredWidth` и `measuredHeight` компонента с использованием полученных данных.

Этот рецепт демонстрирует еще один интересный прием: обращение к классу `Image`, содержащемуся в родительском компоненте. Изображение не передается самому рендереру. В нашем примере рендерер обращается к родителю с запросом и получает ссылку на класс встроенного изображения, которая затем используется для вывода.

```
if (listData) {
    // Удаление старого изображения (если оно есть)
    if (img) {
        removeChild(img);
    }
    if(_imgClass == null) {
        var _imgClass:Class =
        UIComponent(owner).document[listData.label];
    }
    img = new _imgClass();
    addChild(img);
}
```

Обращение к родителю осуществляется по ссылке на документ-владельца `listData`. Объект `document` в `listData` в данном случае представляет собой объект `DataGridColumn`, использующий рендерер. Если рендереру передается значение, оно используется для создания изображения, а если нет – происходит обращение к `DataGridColumn`, и изображение берется оттуда. Код рендерера выглядит так:

```
package oreilly.cookbook
{
    import flash.display.DisplayObject;
    import mx.events.FlexEvent;
    import mx.controls.listClasses.BaseListData;
    import mx.controls.listClasses.IDropInListItemRenderer;
```

```
import mx.controls.listClasses.IListItemRenderer;
import mx.core.UIComponent;

public class SevenSixRenderer extends UIComponent
implements IDropInListItemRenderer, IListItemRenderer
{
    private var _data:Object;
    private var img:DisplayObject;
    private var _listData:BaseListData;
    private var _imgClass:Class;

    [Bindable("dataChange")]
    public function get data():Object {
        return _data;
    }

    public function set data(value:Object):void {
        _data = value;
        if(_data.imgClass != null) {
            _imgClass =_data.imgClass;
        }
        // Свойства объявляются недействительными,
        // чтобы гарантировать их обновление.
        invalidateProperties();
        dispatchEvent(new FlexEvent(FlexEvent.DATA_CHANGE));
    }

    [Bindable("dataChange")]
    public function get listData():BaseListData {
        return _listData;
    }

    public function set listData(value:BaseListData):void {
        _listData = value;
    }
    override protected function commitProperties():void {
        super.commitProperties();
        // Иногда свойство listdata рендерера может
        // быть равно null; это не должно приводить
        // к ошибкам времени выполнения.
        if (listData) {
            // Удаление старого изображения
            // (если оно есть)
            if (img) {
                removeChild(img);
            }
            if(_imgClass == null) {
                var _imgClass:Class =
                UIComponent(owner).
                document[ listData.label];
            }
```

```
                img = new _imgClass();
                addChild(img);
            }
        }

        /* Теперь, когда мы располагаем информацией
           об изображении, можно создать его экземпляр */
        override protected function measure():void {
            super.measure();
            if (img) {
                measuredHeight = img.height;
                measuredWidth = img.width;
            }
        }

        /* Обеспечить правильное размещение изображения */
        override protected function
        updateDisplayList(w:Number, h:Number):void {
            super.updateDisplayList(w, h);
            if (img) {
                img.x = (w - img.width) / 2;
            }
        }
    }
}
```

В приведенном ранее переопределении метода `commitProperties`**, если** `Image` **не передается в данных, рендерер обращается к владельцу** `BaseListData` **для вызова его функции** `labelFunction`**. Данная возможность может использоваться для определения стандартных параметров изображения в** `DataGrid` **или** `List`**. Если значение передается рендереру в данных, рендерер использует его; в противном случае рендерер вызывает метод** `getImage`**, заданный в** `labelFunction`**, и выводит полученное изображение. Код реализации рендерера выглядит так:**

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
    width="700" height="300">
    <mx:Script>
        <![CDATA[

        import mx.collections.ArrayCollection;

        [Embed(source="../assets/foo.jpeg")]
        private var img:Class;

        [Embed(source="../assets/bar.jpeg")]
        public var img2:Class;

        // Обычная коллекция для хранения тривиальных данных
        [Bindable]
        private var genericCollectionOne:ArrayCollection =
```

```
            new ArrayCollection([{name: "josh noble", age:30,
            appearance:"Somewhat wild"}, {name:"Abey George", age:32,
            appearance:"Pretty tight", imgClass:img},
            {name:"Todd Anderson", age:31, appearance:"Intimidating"},
            {name:"Ryan Taylor", age:25, appearance:"Boyishly Handsome",
            imgClass:img}, {name:"Steve Weiss", age:36,
            appearance:"George Clooney-ish"}]);
        // Вызов этого метода используется в том случае,
        // если свойство imgClass равно null.
        private function getImage(o:Object, c:DataGridColumn):String
        {
            return "img2";
        }

        ]]>
    </mx:Script>
    <mx:DataGrid dataProvider="{genericCollectionOne}">
        <mx:columns>
            <mx:DataGridColumn dataField="age"/>
            <mx:DataGridColumn dataField="name"/>
            <mx:DataGridColumn dataField="appearance"/>
            <mx:DataGridColumn
            itemRenderer="oreilly.cookbook.SevenSixRenderer"
            labelFunction="getImage " dataField="imgClass"/>
        </mx:columns>
    </mx:DataGrid>
</mx:HBox>
```

7.10. Стилевое оформление itemRenderer и itemEditor во время выполнения

Задача

Требуется задать стилевую информацию для `itemRenderer` или `itemEditor` в `DataGrid`.

Решение

Переопределите метод `makeRowsAndColumns` класса `ListBase`, расширяемого `List` и `DataGridColumn`, для задания стилей рендереров.

Обсуждение

Задание стилей `itemRenderer` компонента `List` или `DataGridColumn` сводится к простому перебору `itemRenderer` в цикле и заданию стиля. Это должно происходить при задании стилей и при вызове метода `makeRowsAndColumns` класса `ListBase`, расширяемого компонентами `DataGridColumn` и `List`.

Для обращения к рендерерам элементов в `List` можно использовать метод `indexToItemRenderer` класса `ListBase`. Перебор всех рендереров элементов

с использованием свойства `rowCount` **класса** `ListBase` **гарантирует, что вы будете пытаться обращаться только к уже созданным рендерерам. Впрочем, простого перебора рендереров при задании стиля недостаточно; для оптимизации быстродействия рендереры элементов создаются только для видимых строк. При прокрутке** `List` **или** `DataGrid` **данные следующей видимой строки передаются соответствующим рендерам для создания эффекта прокрутки. Таким образом, при каждой перерисовке рендерера необходимо проверить, правильно ли задано значение** `styleName`**, и если нет – задать стиль и установить флаг** `invalidateDisplayList` **для рендереров:**

```
<mx:List xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[

            import mx.core.UIComponent;
            private var _rendererStyle:String;
            // Самый эффективный способ, гарантирующий,
            // что перерисовка рендереров осуществляется
            // только в случае необходимости.
            override protected function makeRowsAndColumns(
                left:Number, top:Number, right:Number, bottom:Number,
                firstCol:int, firstRow:int, byCount:Boolean=false,
                rowsNeeded:uint=0.0):Point
            {
                var pt:Point;
                pt = super.makeRowsAndColumns(left, top, right, bottom,
                    firstCol, firstRow, byCount, rowsNeeded);
                if(_rendererStyle != null) {
                    rendererStyle = _rendererStyle;
                }
                return pt;
            }

            public function set rendererStyle(styleName:String):void {
                _rendererStyle = styleName;
                if( collection != null ) {
                    var i:int = 0;
                    do {
                        try{
                           var comp:UIComponent = (indexToItemRenderer(i) as
                                UIComponent);
                           if(comp.styleName ==_rendererStyle){
                                comp.styleName = _rendererStyle;
                                comp.invalidateDisplayList();
                           } else { continue; }
                        } catch (err:Error){}
                        i++;
                    } while ( i < rowCount )
                }
            }
```

```
            public function get rendererStyle():String {
                return _rendererStyle;
            }
        ]]>
    </mx:Script>
</mx:List>
```

Чтобы задать стиль рендерера, просто обновите свойство `rendererStyle` **списка по щелчку на кнопке; это приведет к обновлению рендереров в компоненте** `List`**:**

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="700"
xmlns:cookbook="oreilly.cookbook.*">
    <mx:Style>
        .firstStyle{
            color:#999999;
        }
        .secondStyle{
            color:#3344ff;
        }
    </mx:Style>
    <mx:Button toggle="true"
    click="list.rendererStyle == 'firstStyle' ?
    list.rendererStyle = 'secondStyle' :
    list.rendererStyle = 'firstStyle'" label="TOGGLE" />
    <cookbook:StylingRendererList id="list"
        dataProvider="{oreilly.cookbook.DataHolder.simpleArray}"
        rendererStyle="firstStyle" width="200"/>
</mx:VBox>
```

7.11. Состояния и переходы в itemEditor

Задача

Требуется создать эффекты, воспроизводимые в начале и завершении редактирования.

Решение

Создайте в `itemEditor` **состояние, которое будет воспроизводить эффект с использованием** `Transition` **в начале редактирования.**

Обсуждение

Определить эффект, воспроизводимый сразу же после завершения редактирования данных, несложно – присоедините к `List` **объект** `DefaultListEffect`**. Свойство** `defaultChangeEffect` **компонента** `List` **обеспечивает создание эффекта после закрепления изменений, внесенных в** `itemEditor`**:**

```
<mx:List xmlns:mx="http://www.adobe.com/2006/mxml"
width="400" height="300" dataChangeEffect="{baseEffect}"
editable="true">
```

```
    <mx:DefaultListEffect id="baseEffect" color="#ffff00"
    fadeInDuration="300" fadeOutDuration="200"/>
</mx:List>
```

Этот код воспроизведет эффект *после* того, как пользователь завершит редактирование в `itemEditor`. Чтобы воспроизвести эффект *в начале* редактирования данных в `itemEditor`, необходимо создать пользовательский рендерер, способный выполнять функции редактора. Рендереру необходимо лишь задать данные для компонента `TextInput` и определить объект перехода `Transition`, который должен воспроизводиться при инициализации редактора.

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
currentState="base" width="100" height="20"
creationComplete="currentState = 'init'" focusEnabled="true"
backgroundColor="#ffff00">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var _data:Object;

            override public function set data(
            value:Object):void {
                _data = value;
            }

            override public function get data():Object {
                return _data;
            }

            // Для задания текста поля ввода
            // в самом начале
            public function set text(value:String):void {
                input.text = value;
            }

            // Необходимо для того, чтобы редактор элемента
            // вернул правильное значение при его
            // чтении списком
            public function get text():String {
                return input.text;
            }
        ]]>
    </mx:Script>
    <mx:transitions>
        <!-- Переход воспроизводится, когда компонент
             готов к отображению -->
        <mx:Transition fromState="*" toState="init">
            <mx:Fade alphaFrom="0" alphaTo="1"
            duration="500" target="{this}"/>
        </mx:Transition>
```

```
        <mx:Transition fromState="init" toState="*">
            <mx:Fade alphaFrom="1" alphaTo="0"
            duration="500" effectEnd=
            "this.dispatchEvent(
             new Event('finishTransition', true))"
            target="{this}"/>
        </mx:Transition>
    </mx:transitions>
    <mx:states>
        <mx:State name="base"/>
        <mx:State name="init"/>
    </mx:states>
    <mx:TextInput id="input"
    creationComplete="input.text = String(_data),
    input.setFocus()"/>
</mx:Canvas>
```

7.12. Создание CheckBox для компонента Tree

Задача

Требуется создать компонент `CheckBox` с тремя состояниями, который будет использоваться в качестве рендерера компонента `Tree`.

Решение

Решение имеет три основных аспекта:

1. создание `TreeItemRenderer` для размещения компонента `CheckBox` в каждом узле дерева.
2. прорисовка маленького черного квадратика над `CheckBox` в третьем состоянии.
3. включение в модель данных дерева атрибута, представляющего состояние `CheckBox`.

Обсуждение

Иерархические деревья часто используются для представления файловых систем. Пользователю часто требуется выделить несколько элементов в нескольких папках для выполнения общего действия; следовательно, нужен визуальный механизм для обозначения выделенных узлов. Как правило, выделение обозначается флажком, но для нашей задачи нужен флажок с тремя состояниями.

Операции установки или снятие выделения с родительского узла должны приводить к установке или снятию выделения с дочерних узлов. Чтобы представить родительский узел с дочерними узлами, одна часть из которых выделена, а другая нет, флажка с двумя состояниями недостаточно. Понадобится третье состояние, не представимое логическим свойством `selected`.

Проблема решается при помощи класса `TreeItemRenderer` **с пользовательским классом ActionScript** `CheckTreeRenderer`**. Класс** `CheckTreeRenderer` **расширяет** `TreeItemRenderer` **и предоставляет нестандартную функциональность поддержки третьего состояния:**

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init();" >
<mx:Script>
<![CDATA[

    import mx.collections.*;

    [Bindable]
    public var folderList:XMLList =
        <>
        <folder state="unchecked"
            label="Marketing Collateral" isBranch="true" >
            <folder state="unchecked" isBranch="true"
                label="Media, PR, and Communications" >
                <folder state="unchecked" isBranch="false"
                    label="Article Reprint Disclaimers" />
                <folder state="unchecked" isBranch="false"
                    label="Articles Reprints" />
                <folder state="unchecked" isBranch="false"
                    label="Interviews and Transcripts" />
                <folder state="unchecked" isBranch="false"
                    label="Press Kits" />
                <folder state="unchecked" isBranch="false"
                    label="Press Releases" />
                <folder state="unchecked" isBranch="false"
                    label="Quick Hits" />
                <folder state="unchecked" isBranch="false"
                    label="Rep Talking Points" />
                <folder state="unchecked" isBranch="false"
                    label="Special Updates" />
                <folder state="unchecked" isBranch="false"
                    label="White Papers" />
            </folder>

            <folder state="unchecked" isBranch="true"
                label="Forms and Applications" >
                <folder state="unchecked" isBranch="false"
                    label="Applications" />
                <folder state="unchecked" isBranch="false"
                    label="Forms" />
            </folder>
        </folder>
        </>;

    [Bindable]
    public var folderCollection:XMLListCollection;
```

```
        private function init() : void
        {
           folderCollection = new XMLListCollection(folderList);
           checkTree.dataProvider = folderCollection;
        }

    ]]>
    </mx:Script>
        <mx:Tree
            id="checkTree"
            itemRenderer="oreilly.cookbook.CheckTreeRenderer"
            labelField="@label"
            width="100%" height="100%" >
        </mx:Tree>
    </mx:Application>
```

Реализация класса `CheckTreeRenderer`**, используемого для отображения** `CheckBox` **в** `Tree`**:**

```
    package oreilly.cookbook
    {
        import mx.controls.Image;
        import mx.controls.Tree;
        import mx.controls.treeClasses.*;
        import mx.collections.*;
        import mx.controls.CheckBox;
        import mx.controls.listClasses.*;
        import flash.events.Event;
        import flash.events.MouseEvent;
        import mx.events.FlexEvent;
        import flash.display.DisplayObject;
        import flash.events.MouseEvent;
        import flash.xml.*;
        import mx.core.IDataRenderer;

    public class CheckTreeRenderer extends TreeItemRenderer
    {
```

Создание `CheckBox` **и** `Image`**:**

```
        protected var myImage:Image;
        protected var myCheckBox:CheckBox;
        // Задание свойств изображения
        private var imageWidth:Number = 6;
        private var imageHeight:Number = 6;
        private var inner:String = "assets/inner.png";
        static private var STATE_SCHRODINGER:String =
            "schrodinger";
        static private var STATE_CHECKED:String = "checked";
        static private var STATE_UNCHECKED:String = "unchecked";

        public function CheckTreeRenderer ()
        {
```

```
        super();
        mouseEnabled = false;
    }
    private function toggleParents (item:Object, tree:Tree,
    state:String):void
    {
        if (item == null)
        {
            return;
        }
        else
        {
            item.@state = state;
            toggleParents(tree.getParentItem(item), tree,
                getState (tree, tree.getParentItem(item)));
        }
    }

    private function toggleChildren (item:Object, tree:Tree,
    state:String):void
    {
        if (item == null) {return;}
        else {
            item.@state = state;
            var treeData:ITreeDataDescriptor = tree.dataDescriptor;
            if (treeData.hasChildren(item)) {
                var children:ICollectionView = treeData.getChildren (item);
                var cursor:IViewCursor = children.createCursor();
                while (!cursor.afterLast) {
                    toggleChildren(cursor.current, tree, state);
                    cursor.moveNext();
                }
            }
        }
    }
```

Обработчик компонента `Image` делегирует большую часть логики выделения обработчику `CheckBox`. В конце концов, компонент `Image` предназначен только для третьего состояния. Для каждого узла, на котором делается щелчок, обработчик `CheckBox` переключает состояние дочерних узлов, а затем изменяет состояние родителя(ей). Дочерние узлы могут переводиться только в состояние `CHECKED` или `UNCHECKED`, а родители также могут перейти в третье состояние. Это состояние, названное `SCHRODINGER`, возникает тогда, когда одна часть дочерних узлов находится в состоянии `CHECKED`, а другая в состоянии `UNCHECKED` и/или `SCHRODINGER`.

Состояние родительского узла определяется просмотром состояния дочерних узлов. За решение этой задачи отвечает метод `getState`:

```
private function getState(tree:Tree, parent:Object):String {
    var noChecks:int = 0;
    var noCats:int = 0;
```

```
        var noUnChecks:int = 0;
        if (parent != null) {
            var treeData:ITreeDataDescriptor = tree.dataDescriptor;
            var cursor:IViewCursor =
                treeData.getChildren(parent).createCursor();
            while (!cursor.afterLast) {
                if (cursor.current.@state == STATE_CHECKED){
                    noChecks++;
                }
                else if (cursor.current.@state == STATE_UNCHECKED) {
                    noUnChecks++
                }
                else {
                    noCats++;
                }
                cursor.moveNext();
            }
        }
        if ((noChecks > 0 && noUnChecks > 0) || (noCats > 0 && noChecks>0)) {
            return STATE_SCHRODINGER;
        } else if (noChecks > 0) {
            return STATE_CHECKED;
        } else {
            return STATE_UNCHECKED;
        }
    }
    private function imageToggleHandlder(event:MouseEvent):void {
        myCheckBox.selected = !myCheckBox.selected;
        checkBoxToggleHandler(event);
    }
```

Щелчки мышью должны обрабатываться обоими дочерними компонентами, CheckBox **и** Image**, поэтому для каждого из них необходимо создать** EventListener**:**

```
    private functioncheckBoxToggleHandler(event:MouseEvent):void
    {
        if (data)
        {
            var myListData:TreeListData = TreeListData(this.listData);
            var selectedNode:Object = myListData.item;
            var tree:Tree = Tree(myListData.owner);
            var toggle:Boolean = myCheckBox.selected;
            if (toggle) {
                toggleChildren(data, tree, STATE_CHECKED);
            } else {
                toggleChildren(data, tree, STATE_UNCHECKED);
            }
            var parent:Object = tree.getParentItem (data);
            toggleParents (parent, tree, getState (tree, parent));
        }
    }
```

Переопределение метода `createChildren`**, ответственного за создание каждого узла дерева:**

```
override protected function createChildren():void {
    super.createChildren();
    myCheckBox = new CheckBox();
    myCheckBox.setStyle( "verticalAlign", "middle" );
    myCheckBox.addEventListener( MouseEvent.CLICK, checkBoxToggleHandler );
    addChild(myCheckBox);
    myImage = new Image();
    myImage.source = inner;
    myImage.addEventListener( MouseEvent.CLICK, imageToggleHandler );
    myImage.setStyle( "verticalAlign", "middle" );
    addChild(myImage);
}

private function setCheckState (checkBox:CheckBox,
value:Object, state:String):void {
    if (state == STATE_CHECKED) {
        checkBox.selected = true;
    }
    else if (state == STATE_UNCHECKED) {
        checkBox.selected = false;
    }
    else if (state == STATE_SCHRODINGER) {
        checkBox.selected = false;
    }
}

override public function set data(value:Object):void {
    if(value != null) {
        super.data = value;
        setCheckState (myCheckBox, value, value.@state);
        if(TreeListData(super.listData).item.@type == 'dimension') {
            setStyle("fontStyle", 'italic');
        } else {
            if (this.parent != null) {
                var _tree:Tree = Tree(this.parent.parent);
                _tree.setStyle("defaultLeafIcon", null);
            }
            setStyle("fontStyle", 'normal');
        }
    }
}
override protected function updateDisplayList(
    unscaledWidth:Number, unscaledHeight:Number):void
{
    super.updateDisplayList(unscaledWidth, unscaledHeight);
    if(super.data)
    {
        if (super.icon != null) {
            myCheckBox.x = super.icon.x;
```

```
                myCheckBox.y = 2;
                super.icon.x = myCheckBox.x + myCheckBox.width + 17;
                super.label.x = super.icon.x + super.icon.width + 3;
            } else {
                myCheckBox.x = super.label.x;
                myCheckBox.y = 2;
                super.label.x = myCheckBox.x + myCheckBox.width + 17;
            }
            if (data.@state == STATE_SCHRODINGER) {
                myImage.x = myCheckBox.x + 4;
                myImage.y = myCheckBox.y + 4;
                myImage.width = imageWidth;
                myImage.height = imageHeight;
            } else {
                myImage.x = 0;
                myImage.y = 0;
                myImage.width = 0;
                myImage.height = 0;
            }
        }
    }
```

Теперь рендерер не только отображает разные состояния в `CheckBox`, но и читает и обновляет данные компонента `Tree`, в котором он используется.

7.13. Изменение размеров в рендерерах List

Задача

Требуется создать рендерер, размеры которого изменяются при выделении.

Решение

Создайте рендерер элементов, реализующий интерфейс `IDropInListItemRenderer`. Используйте `listData` для добавления объектов `eventListener` событий `Scroll` и `Change` родительского списка. При каждом обнаружении события `Change` или `Scroll` проверьте, соответствует ли свойство `data` объекта `itemRenderer` данным `selectedItem` в компоненте `List`. Если данные соответствуют, задайте текущее состояние рендерера, а если нет – верните `currentState` к базовому состоянию.

Обсуждение

Важно помнить, что рендерер элементов многократно используется компонентом `List`. Когда вы пытаетесь задать состояние рендерера при выделении, данные рендерера необходимо сравнивать со свойством `selectedItem` компонента `List` или `DataGridColumn`, а не с `selectedIndex`, потому что `selectedIndex` не сообщает, какой именно рендерер является

выделенным. Более того, выделенный рендерер в результате прокрутки может оказаться очень далеко.

Так как `List` или `DataGridColumn` доступны через свойство `listData`, вы можете добавить обработчики событий, а также создать обработчик, который будет проверять данные и текущее состояние рендерера:

```
private function resizeEventHandler(event:Event):void {
    if((_listData.owner as List).selectedIndex ==
        ArrayCollection((_listData.owner as List).dataProvider).
        getItemIndex(this.data) &&
        currentState != "selected") {
        trace(" functions " +_listData.rowIndex + "  "
            +(_listData.owner as List).selectedIndex);
        currentState = "selected";
    } else if((_listData.owner as List).selectedIndex != ArrayCollection(
         (_listData.owner as List).dataProvider).getItemIndex(this.data) &&
         currentState == "selected") {
         currentState = "base";
    }
}
```

Это решение можно усовершенствовать несколькими способами, чтобы сделать его более универсальным, но мы ограничимся простым примером, предполагающим, что вы всегда используете `ArrayCollection`. Взгляните:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" height="30"
currentState="base"
implements="mx.controls.listClasses.IDropInListItemRenderer"
verticalScrollPolicy="off">
    <mx:Script>
        <![CDATA[
            import mx.controls.List;
            import mx.events.ListEvent;
            import mx.controls.listClasses.BaseListData;
            import mx.collections.ArrayCollection;

            private function resizeFocusInHandler(event:Event):void {
              if((_listData.owner as List).selectedIndex ==
              ArrayCollection((_listData.owner as List).
              dataProvider).getItemIndex(this.data) &&
              currentState != "selected") {
              trace(" functions " +_listData.rowIndex +
              " " +(_listData.owner asList).selectedIndex);
              currentState = "selected";
              } else if((_listData.owner as List).selectedIndex !=
              ArrayCollection((_listData.owner as
              List).dataProvider).getItemIndex(this.data) &&
              currentState == "selected") {
              currentState = "base";
              }
            }
```

```
            override public function set data(value:Object):void {
                txt.text = value as String;
            }

            override public function get data():Object {
                return txt.text;
            }

            // Внутренняя переменная для значения свойства
            private var _listData:BaseListData;

            // Включение привязки для свойства listData
            [Bindable("dataChange")]

            // Определение метода get
            public function get listData():BaseListData {
                return _listData;
            }

            // Назначение слушателей событий Change и Scroll,
            // которые будут передаваться компонентом List или Column.
            public function set listData(value:BaseListData):void {
                _listData = value;
                _listData.owner.addEventListener(
                    ListEvent.CHANGE, resizeFocusInHandler);
                _listData.owner.addEventListener(
                    Event.SCROLL, resizeFocusInHandler);
            }
        ]]>
    </mx:Script>
    <mx:transitions>
        <mx:Transition fromState="*" toState="selected">
            <mx:Resize heightTo="60" target="{this}"/>
        </mx:Transition>
        <mx:Transition fromState="selected" toState="*">
            <mx:Resize heightTo="30" target="{this}"/>
        </mx:Transition>
    </mx:transitions>
    <mx:states>
        <mx:State name="base"/>
        <mx:State name="selected">
            <mx:AddChild>
                <mx:HBox>
                <mx:Label text="some text"/>
                <mx:Label text="{'some text = '+txt.text}"/>
                </mx:HBox>
            </mx:AddChild>
        </mx:State>
    </mx:states>
    <mx:Text id="txt"/>
</mx:VBox>
```

8

Графика, видео и звук

Графика, *видео* и *звук* – весьма обширная область, которую невозможно сколько-нибудь адекватно описать в одной главе, поэтому мы сконцентрируемся на самых типичных вопросах. По мере того как Flex становится основным механизмом передачи видео по Интернету, а Flex Framework все чаще применяется для создания приложений, работающих с фотографиями и MP3, умение работать с этими мультимедийными элементами начинает играть более важную роль.

Flash Player предоставляет многоуровневый инструментарий для работы с графикой и звуком. К первому уровню относятся классы `Image` и `VideoDisplay`, а также классы MXML, которые упрощают работу с графикой и видео и позволяют быстро интегрировать эти ресурсы в приложение. На следующем уровне находится пакет `flash.media` с классами `Video`, `Sound`, `SoundTransform`, `Camera` и `Microphone`; дополняющие их классы `Loader`, `NetConnection` и `NetStream` объединены в пакет `flash.net`. Эти классы позволяют более точно управлять интеграцией звука, видео и графики в приложение, но и на их освоение потребуется чуть больше времени. Наконец, можно спуститься до уровня байтов, из которых складываются все данные в Flash Player: он представлен классами `BitmapData` и `ByteArray`. Они позволяют не только работать с данными растровых изображений, загружаемых в Flash Player, но и создавать новые растровые изображения и осуществлять потоковую передачу данных.

Во многих примерах этой главе используются манипуляции с графикой и видео на нижнем (растровом) уровне. Это отнюдь не так сложно, как может показаться на первый взгляд, потому что Flash Player предоставляет ряд вспомогательных методов для работы с классом `BitmapData`, а манипуляции с низкоуровневыми данными значительно повышают эффективность приложения. Также в примерах этой главы широко используется класс `NetStream` для работы с видеоданными, микрофонами и камерами пользователей. `NetStream` является эффективным средством

приема и передачи информации приложениям, работающим на стороне сервера.

8.1. Загрузка и отображение графики

Задача

Требуется вывести графическое изображение в компоненте Flex.

Решение

Воспользуйтесь директивой `Embed` для включения изображения в файл SWF или загрузите его во время выполнения.

Обсуждение

Flex поддерживает возможность импортирования файлов GIF, JPEG, PNG и SWF на стадии компиляции или выполнения, и файлов SVG на стадии компиляции посредством встраивания (embedding). Выбор механизма зависит от типов файлов и параметров приложения. Все встроенные изображения становятся частью файла SWF, и их загрузка не требует дополнительных затрат времени. С другой стороны, они увеличивают размер приложения, а это замедляет процесс инициализации. Кроме того, при интенсивном использовании встроенной графики приходится перекомпилировать приложение при каждом изменении изображений.

Ресурсы также можно загружать во время выполнения; для этого свойству `source` изображения задается URL-адрес, или результат операции загрузки представляется в виде объекта `BitmapAsset`. Ресурсы могут загружаться как из локальной файловой системы, в которой выполняется файл SWF, так и с удаленных систем (как правило, посредством запроса HTTP по сети). Графические изображения существуют независимо от приложения, а их изменение не требует перекомпиляции (при условии, что имя модифицированного изображения осталось прежним).

Любой файл SWF может работать с внешними ресурсами только одного типа – либо локальными, либо сетевыми; обращение к обоим типам невозможно. Тип доступа определяется флагом `use_network` при компиляции приложения. Если флаг `use-network` равен `false`, приложение может работать с ресурсами локальной файловой системы, но сетевые ресурсы для него недоступны. По умолчанию используется значение `true`, которое позволяет работать с ресурсами по сети, но не в локальной файловой системе.

Чтобы встроить файл графического изображения в приложение, воспользуйтесь свойством метаданных `Embed`:

```
[Embed(source="../assets/flag.png")]
private var flag:Class;
```

Теперь объект Class может быть назначен источником графического изображения:

```
var asset:BitmapAsset = new flag() as BitmapAsset;
img3rd.source = asset;
```

Также свойству Source можно назначить URL-адрес в локальной или внешней файловой системе:

```
<mx:Image source="http://server.com/beach.jpg"/>
```

Пример кода:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;
            [Embed(source="../assets/flag.png")]
            private var flag:Class;
            private function imgMod():void
            {
                var asset:BitmapAsset = new flag() as BitmapAsset;
                img3rd.source = asset;
            }
        ]]>
    </mx:Script>
    <mx:Image source="../assets/flag.png"/>
    <mx:Image source="{flag}"/>
    <mx:Image id="img3rd" creationComplete="imgMod()"/>
</mx:VBox>
```

8.2. Отображение видео

Задача

Требуется воспроизвести файл FLV в приложении.

Решение

Используйте класс VideoDisplay в своем приложении. Для запуска и приостановки воспроизведения используются объекты Button.

Обсуждение

Класс VideoDisplay инкапсулирует объект flash.media.Video, существенно упрощая включение видео в этот объект. Атрибуту source объекта VideoDisplay задается URL-адрес файла FLV, а параметр autoplay задается равным true, чтобы объект автоматически создал экземпляр NetStream и начал потоковую передачу видео:

```
<mx:VideoDisplay source="http://localhost:3001/Trailer.flv" id="vid"
autoplay="true"/>
```

В следующем примере создаются кнопки для воспроизведения, временной и полной остановки видеоролика с использованием методов, определяемых классом VideoDisplay:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:VideoDisplay
        source="http://localhost:3001/Trailer.flv"
        id="vid" autoPlay="false" autoRewind="true"/>
    <mx:HBox>
        <mx:Button label="Play" click="vid.play();"/>
        <mx:Button label="Pause" click="vid.pause();"/>
        <mx:Button label="Stop" click="vid.stop();"/>
    </mx:HBox>
</mx:VBox>
```

8.3. Воспроизведение и приостановка файлов MP3

Задача

Требуется реализовать воспроизведение серии файлов MP3.

Решение

Используйте классы Sound и SoundChannel. Реализуйте прогрессивную загрузку при выборе пользователем нового файла MP3.

Обсуждение

Метод play класса Sound возвращает объект SoundChannel, который предоставляет доступ к методам и свойствам для управления балансом между левым и правым динамиками, а также приостановки и возобновления воспроизведения.

Допустим, для загрузки и воспроизведения аудиофайла используется следующий фрагмент:

```
var snd:Sound = new Sound(new URLRequest("sound.mp3"));
var channel:SoundChannel = snd.play();
```

Буквально приостановить воспроизведение в коде ActionScript невозможно; возможна только полная остановка методом SoundChannel.stop. Впрочем, начать воспроизведение можно с произвольной позиции. Зафиксируйте текущую позицию в аудиофайле на момент остановки, а затем начните воспроизведение с указанной позиции.

Во время воспроизведения звука текущая позиция в звуковом файле обозначается свойством SoundChannel.position. Сохраните его значение перед остановкой звука:

```
var pausePosition:int = channel.position;
channel.stop();
```

Передайте при вызове `play` **ранее сохраненное значение, чтобы воспроизведение продолжилось с той точки, на которой оно было остановлено:**

```
channel = snd.play(pausePosition);
```

В следующем примере компонент `ComboBox` **используется для выбора файла MP3, а класс** `SoundChannel` **обеспечивает остановку воспроизведения (временную или полную):**

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            public var sound:Sound;
            public var chan:SoundChannel;
            public var pausePos:int = 0;

            private const server:String = "http://localhost:3001/"
            private var dp:ArrayCollection =
                 new ArrayCollection(["Plans.mp3", "Knife.mp3",
                 "Marla.mp3", "On a Neck, On a Spit.mp3", "Colorado.mp3"])

            private function loadSound():void {
                if(chan != null) {
                    // Воспроизведение необходимо остановить,
                    // чтобы не возникло наложение звука.
                    chan.stop();
                }
                // Повторное создание звукового объекта, очистка буфера
                // и назначение слушателя события.
                sound = new Sound();
                sound.addEventListener(Event.SOUND_COMPLETE,
                    soundComplete);
                var req:URLRequest = new URLRequest(server +
                    cb.selectedItem as String);
                sound.load(req);
                pausePos = 0;
                chan = sound.play();
            }
            //
            private function soundComplete(event:Event):void {
                cb.selectedIndex++;
                sound.load(new URLRequest(server
                    + cb.selectedItem as String));
                chan = sound.play();
            }
            private function playPauseHandler():void
            {
                if(pausePlayBtn.selected){
                    pausePos = chan.position;
                    chan.stop();
                } else {
```

```
                    chan = sound.play(pausePos);
                }
            }
        ]]>
    </mx:Script>
    <mx:ComboBox creationComplete="cb.dataProvider=dp"
        id="cb" change="loadSound()"/>
    <mx:Button label="start" id="pausePlayBtn"
        toggle="true" click="playPauseHandler()"/>
    <mx:Button label="stop" click="chan.stop()"/>
</mx:HBox>
```

8.4. Позиционирование и управление громкостью для звукового файла

Задача

Требуется создать шкалу для перехода к различным позициям файла MP3 и компонент для управления громкостью воспроизведения.

Решение

Чтобы начать воспроизведение с конкретной позиции, передайте параметр `time` методу `Sound.play`. Метод создает новый объект `SoundTransform`, который назначается свойству `soundTransform` объекта `SoundChannel`.

Обсуждение

Методу `play` объекта `Sound` при вызове передается параметр, определяющий начальную позицию воспроизведения:

```
public function play(startTime:Number = 0, loops:int = 0,
sndTransform: SoundTransform = null):SoundChannel
```

Метод создает новый объект `SoundChannel` для воспроизведения звука и возвращает его вызывающей стороне; это дает возможность остановить воспроизведение и продолжить его с произвольной позиции.

Чтобы управлять громкостью воспроизведения, передайте объект `SoundTransform` объекту `SoundChannel`. В приведенном примере мы создаем новый объект `SoundTransform` с нужными параметрами и передаем его текущему объекту `SoundChannel`:

```
var trans:SoundTransform = new SoundTransform(volumeSlider.value);
chan.soundTransform = trans;
```

Класс `SoundTransform` получает следующие параметры:

```
SoundTransform(vol:Number = 1, panning:Number = 0)
```

Значения второго параметра лежат в интервале от -1.0 (полное смещение влево; правый динамик не воспроизводит звук) до 1.0 (полное смещение вправо). Полный код примера:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="loadSound()">
    <mx:Script>
        <![CDATA[

            private var sound:Sound;
            private var chan:SoundChannel;

            private function loadSound():void {
                sound = new Sound(new URLRequest(
                    "http://localhost:3001/Plans.mp3"));
                chan = sound.play();
            }

            private function scanPosition():void {
                chan.stop();
                // Делим на 10, т.к. шкала Slider размечена в интервале
                // 0 - 10, а нам нужно значение в интервале 0 - 1.0.
                chan = sound.play(positionSlider.value/10 *
                    sound.length);
            }

            private function scanVolume():void
            {
                var trans:SoundTransform = new
                    SoundTransform(volumeSlider.value,
                    (panSlider.value - 5)/10);
                chan.soundTransform = trans;
            }
        ]]>
    </mx:Script>
    <mx:Label text="Position"/>
    <mx:HSlider change="scanPosition()" id="positionSlider"/>
    <mx:Label text="Volume"/>
    <mx:HSlider change="scanVolume()" id="volumeSlider"/>
    <mx:Label text="Pan"/>
    <mx:HSlider change="scanVolume()" id="panSlider"/>
</mx:VBox>
```

8.5. Объединение изображений

Задача

Требуется объединить несколько графических изображений во время выполнения, с использованием фильтров для их модификации.

Решение

Преобразуйте изображения в объекты `BitmapData`. Используйте метод `combine` класса `BitmapData` для объединения графических данных в новое изображение.

Обсуждение

Классы BitmapData и Bitmap представляют собой мощные инструменты для обработки графики во время выполнения и создания новых эффектов. Эти два класса часто используются совместно, но между ними существует ряд важных различий. В классе BitmapData инкапсулируются фактические данные изображения, а в классе Bitmap – экранный объект, который можно включить в список отображения. Объект BitmapData создается и выводится так, как показано в следующем фрагменте:

```
var bitmapAsset:BitmapAsset = new BitmapAsset(img1.width,img1.height);
bitmapAsset.draw(img1);
```

Сначала задайте высоту и ширину BitmapAsset, проследите за тем, чтобы объект имел правильные размеры, а затем выведите все данные из графического изображения. В результате графические данные сохраняются в виде объекта растрового изображения, с которым можно выполнять различные операции. В следующем примере метод colorTransform манипулирует с цветовыми данными объекта BitmapData; для слияния двух объектов растровых изображений используется метод merge. Метод colorTransform изменяет цвет отображаемого объекта или BitmapData в соответствии со значениями, переданными в конструкторе:

```
ColorTransform(redMultiplier:Number = 1.0, greenMultiplier:Number = 1.0,
blueMultiplier:Number = 1.0, alphaMultiplier:Number = 1.0,
redOffset:Number = 0, greenOffset:Number = 0, blueOffset:Number = 0,
alphaOffset:Number = 0)
```

При применении объекта ColorTransform к отображаемому объекту новые значения цветовых каналов вычисляются по следующим формулам (н – новое значение, с – старое значение):

- $\text{Красный}_{\text{н}} = (\text{красный}_{\text{с}} * \text{redMultiplier}) + \text{redOffset}$
- $\text{Зеленый}_{\text{н}} = (\text{зеленый}_{\text{с}} * \text{greenMultiplier}) + \text{greenOffset}$
- $\text{Синий}_{\text{н}} = (\text{синий}_{\text{с}} * \text{blueMultiplier}) + \text{blueOffset}$
- $\text{Альфа}_{\text{н}} = (\text{альфа}_{\text{с}} * \text{alphaMultiplier}) + \text{alphaOffset}$

Метод merge класса BitmapData имеет следующую сигнатуру:

```
merge(sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point,
redMultiplier :uint, greenMultiplier:uint, blueMultiplier:uint,
alphaMultiplier:uint):void
```

Параметры:

sourceBitmapData:BitmapData

Входное растровое изображение. Возможно использование как другого, так и текущего объекта BitmapData.

sourceRect:Rectangle

Прямоугольник, определяющий область изображения с исходными данными.

`destPoint:Point`

Точка приемного изображения (текущий экземпляр BitmapData), соответствующая левому верхнему углу исходного прямоугольника.

`redMultiplier:uint`

Шестнадцатеричный множитель красного канала (тип `uint`).

`greenMultiplier:uint`

Шестнадцатеричный множитель зеленого канала (тип `uint`).

`blueMultiplier:uint`

Шестнадцатеричный множитель синего канала (тип `uint`).

`alphaMultiplier:uint`

Шестнадцатеричный множитель альфа-канала (тип `uint`).

Далее приводится полный листинг примера с элементами для изменения параметров `ColorTransform`:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="500" height="550"
creationComplete="imgMod()">
    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;
            import mx.controls.Image;

            [Embed(source="../assets/bigshakey.png")]
            private var shakey:Class;
            [Embed(source="../assets/mao.jpg")]
            private var mao:Class;

            // Наложение изображений с использованием данных vslider
            private function imgMod():void
            {
                var maoData:BitmapData = new BitmapData(firstImg.width,
                    firstImg.height);
                var shakeyData:BitmapData = new BitmapData(secondImg.width,
                    secondImg.height);
                maoData.draw(firstImg);
                shakeyData.draw(secondImg);
                maoData.colorTransform(new Rectangle(0, 0, maoData.width,
                    maoData.height), new ColorTransform(redSlider.value/10,
                    greenSlider.value/10, blueSlider.value/10,
                    alphaSlider.value/10));
                var red:uint = (uint(
                    redSlider.value.toString(16)) / 10) * 160;
                var green:uint = (uint(
                    greenSlider.value.toString(16)) / 10) * 160;
                var blue:uint = (uint(
                   blueSlider.value.toString(16)) / 10) * 160;
                var alpha:uint = (uint(
                   alphaSlider.value.toString(16)) / 10) * 160;
```

```
                shakeyData.merge(maoData,new Rectangle(0, 0,
                    shakeyData.width, shakeyData.height), new Point(0, 0),
                    red, green, blue, alpha);
                mainImg.source = new BitmapAsset(shakeyData);
            }
        ]]>
    </mx:Script>
    <mx:HBox>
        <mx:Image id="firstImg" source="{mao}" height="200" width="200"/>
        <mx:Image id="secondImg" source="{shakey}" height="200" width="200"/>
    </mx:HBox>
    <mx:HBox>
        <mx:Text text="Red"/>
        <mx:VSlider height="100" id="redSlider"
            value="5.0" change="imgMod()"/>
        <mx:Text text="Blue"/>
        <mx:VSlider height="100" id="blueSlider"
            value="5.0" change="imgMod()"/>
        <mx:Text text="Green"/>
        <mx:VSlider height="100" id="greenSlider"
            value="5.0" change="imgMod()"/>
        <mx:Text text="Alpha"/>
        <mx:VSlider height="100" id="alphaSlider"
            value="5.0" change="imgMod()"/>
    </mx:HBox>
    <mx:Image id="mainImg"/>
</mx:VBox>
```

8.6. Применение сверточного фильтра

Задача

Требуется изменить цвета, контраст или четкость изображения.

Решение

Создайте экземпляр `ConvolutionFilter` и привяжите свойства его внутренней матрицы к текстовым элементам, которые могут изменяться пользователем. Занесите фильтр в массив фильтров изображения, чтобы применить его.

Обсуждение

`ConvolutionFilter` – один из самых универсальных и сложных фильтров в пакете `flash.filters`. Он может использоваться для рельефного выделения, определения краев, размывки, повышения резкости и множества других эффектов. Всеми параметрами управляет объект `Matrix`, который представляет матрицу 3×3 и передается фильтру в конструкторе. `ConvolutionFilter` последовательно перебирает все пикселы исходного изображения и определяет его итоговый цвет как результат преобразо-

вания, определяемого значением текущего пиксела и его окружающих пикселов. Матрица, заданная в виде числового массива, указывает, в какой степени каждый соседний пиксел влияет на итоговое значение. Конструктор `ConvolutionFilter` имеет следующую сигнатуру:

```
ConvolutionFilter(matrixX:Number = 0, matrixY:Number = 0, matrix:Array =
    null, divisor:Number = 1.0, bias:Number = 0.0, preserveAlpha:Boolean =
    true, clamp:Boolean = true, color:uint = 0, alpha:Number = 0.0)
```

Параметры конструктора:

`matrixX:Number` (*по умолчанию = 0*)

Количество столбцов в матрице (по умолчанию 0).

`matrixY:Number` (*по умолчанию = 0*)

Количество строк в матрице (по умолчанию 0).

`matrix:Array` (*по умолчанию = null*)

Массив (матрица) параметров преобразования. Количество элементов в массиве должно быть равно `matrixX * matrixY`.

`divisor:Number` (*по умолчанию = 1.0*)

Делитель, используемый в ходе преобразования (по умолчанию 1). Делитель, равный сумме всех значений в матрице, нормализует общую интенсивность цвета. Значение 0 игнорируется, и вместо него используется значение по умолчанию.

`bias:Number` (*по умолчанию = 0.0*)

Смещение, прибавляемое к результату матричного преобразования (по умолчанию 0).

`preserveAlpha:Boolean` (*по умолчанию = true*)

Значение `false` означает, что величина альфа-канала не сохраняется и свертка применяется ко всем каналам. Значение `true` означает, что свертка применяется только к цветовым каналам. По умолчанию используется значение `true`.

`clamp:Boolean` (*по умолчанию = true*)

Для пикселов, выходящих за границы исходного изображения, при значении `true` изображение продлевается за границы посредством дублирования цветов граничных пикселов у соответствующего края. Значение `false` означает, что вместо них должен использоваться другой цвет, определяемый свойствами `color` и `alpha`. По умолчанию используется значение `true`.

`color:uint` (*по умолчанию = 0*)

Шестнадцатеричный код цвета пикселов, выходящих за границу исходного изображения.

`alpha:Number` (*по умолчанию = 0.0*)

Альфа-канал цвета-заменителя.

Стандартные примеры использования `ConvolutionFilter`**:**

```
new ConvolutionFilter(3,3,new Array(-5,0,1,1,-2,3,-1,2,1),1)
```

Эффект выявления контуров, т. е. в изображении остаются только области с наибольшим контрастом.

```
new ConvolutionFilter(3,3,new Array(0,20,0,20,-80,20,0,20,0),10)
```

Черно-белый контур.

```
new ConvolutionFilter(5,5,new Array(0,1,2,1,0,1,2,4,2,1,2,4,8,4,2,1,2,4,2,
1,0,1,2,1,0),50);
```

Эффект размывки.

```
new ConvolutionFilter(3,3,new Array(-2,-1,0,-1,1,1,0,1,2),0);
```

Эффект рельефного выделения.

Полный код примера:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    width="450" height="550">
    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;

            [Embed(source="../assets/mao.jpg")]
            private var mao:Class;

            private function convolve():void
            {
                var asset:BitmapAsset = new mao() as BitmapAsset;
                var convolution:ConvolutionFilter =
                    new ConvolutionFilter(matrixXSlider.value,
                    matrixYSlider.value,
                [input1.text, input2.text, input3.text,
                    input4.text, input5.text,input6.text],
                divisorSlider.value, biasSlider.value, true);
                var _filters:Array = [convolution];
                asset.filters = _filters;
                img.source = asset;
            }

        ]]>
    </mx:Script>
    <mx:Button click="convolve()" label="convolve away"/>
    <mx:HBox>
        <mx:Text text="Matrix X"/>
        <mx:VSlider height="100" id="matrixXSlider" value="5.0"
            change="convolve()"/>
        <mx:Text text="Matrix Y"/>
        <mx:VSlider height="100" id="matrixYSlider" value="5.0"
            change="convolve()"/>
        <mx:Text text="Divisor"/>
```

```
            <mx:VSlider height="100" id="divisorSlider" value="5.0"
                change="convolve()"/>
            <mx:Text text="Bias"/>
            <mx:VSlider height="100" id="biasSlider" value="5.0"
                change="convolve()"/>
            <mx:VBox>
                <mx:TextInput id="input1" change="convolve()" width="40"/>
                <mx:TextInput id="input2" change="convolve()" width="40"/>
                <mx:TextInput id="input3" change="convolve()" width="40"/>
                <mx:TextInput id="input4" change="convolve()" width="40"/>
                <mx:TextInput id="input5" change="convolve()" width="40"/>
                <mx:TextInput id="input6" change="convolve()" width="40"/>
            </mx:VBox>
        </mx:HBox>
        <mx:Image id="img"/>
    </mx:VBox>
```

8.7. Передача видео с камеры экземпляру FMS

Задача

Требуется передать поток данных с камеры пользователя экземпляру Flash Media Server (FMS) для использования в чате или другом приложении, требующем «живого» видеовывода.

Решение

Захватите поток данных с камеры методом `flash.media.Camera.getCamera` и присоедините его к экземпляру `NetStream`, который будет передаваться экземпляру Flash Media Server. Воспользуйтесь методом `publish` класса `NetStream` для отправки потока приложению для дальнейшей обработки.

Обсуждение

Метод `publish` сообщает экземпляру Flash Media Server, подключенному при помощи класса `NetConnection`, что `NetStream` будет отправлять информацию серверу. Что сервер сделает с этой информацией, зависит от приложения, но в методе `publish` определяются флаги, которые указывают серверу и Flash, как следует поступить с потоковой информацией. Метод `publish` имеет следующую сигнатуру:

```
publish(name:String = null, type:String = null):void
```

Параметры:

`name:String` (*default = null*)

Строковый идентификатор потока. При передаче значения `false` операция публикации останавливается. Клиенты, подписывающиеся на публикуемый поток, должны указать это же имя при вызове `NetStream.play`.

`type:String` **(*default* = *null*)**

Строка, определяющая режим публикации потока. Допустимые значения – `record`, `append` и `live` (по умолчанию). В режиме `record` Flash Player публикует и записывает «живые» данные, сохраняя их в новом файле FLV; имя файла совпадает со значением, переданным в параметре `name`. Файл сохраняется на сервере в подкаталоге того каталога, в котором находится серверное приложение. Если файл уже существует, он перезаписывается. В режиме `append` Flash Player публикует и записывает «живые» данные», присоединяя их к файлу FLV, имя которого совпадает со значением параметра `name`. Файл находится на сервере в подкаталоге того каталога, в котором находится серверное приложение. Если файл с именем, определяемым параметром `name`, не найден, он будет автоматически создан. Если параметр `type` не указан, а также в режиме `live`, Flash Player публикует «живые» данные без записи. Если в системе существует файл с именем, совпадающим со значением `name`, этот файл удаляется.

При записи потока с использованием Flash Media Server сервер создает файл FLV и сохраняет его в подкаталоге каталога приложения на сервере. Имя каталога потока соответствует имени экземпляра приложения, переданного при вызове `NetConnection.connect`. Сервер создает эти каталоги автоматически; вам не придется создавать их вручную для разных экземпляров приложения. Например, следующий фрагмент демонстрирует подключение к конкретному экземпляру приложения в каталоге `lectureSeries`. Файл с именем lecture.flv находится в подкаталоге /yourAppsFolder/lectureSeries/streams/Monday:

```
var myNC:NetConnection = new NetConnection();
myNC.connect("rtmp://server.domain.com/lectureSeries/Monday");
var myNS:NetStream = new NetStream(myNC);
myNS.publish("lecture", "record");
```

Если при вызове не передается значение, совпадающее с именем экземпляра, то данные, идентифицируемые свойством `name`, сохраняются в подкаталоге с именем /yourAppsFolder/appName/streams/_definst_ (сокращение от «default instance», т. е. «экземпляр по умолчанию»).

Метод может передать событие `netStatus` с несколькими информационными объектами. Например, если кто-то уже публикует поток с указанным именем, передается событие `netStatus` с кодом `NetStream.Publish.BadName`. За дополнительной информацией обращайтесь к описанию события `netStatus`.

В следующем примере создается подключение к серверу, а поток данных с камеры передается серверу:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"width="400" height="500"
creationComplete="setUpCam()">
    <mx:Script>
        <![CDATA[
            private var cam:Camera;
```

```
            private var nc:NetConnection;
            private var ns:NetStream;

            private function setUpCam():void
            {
                trace(Camera.names.join(","));
                // Без этого вызова Flash Player
                // не связывается с камерой на моем Macbook
                cam = flash.media.Camera.getCamera("2");
                vid.attachCamera(cam);
                nc = new NetConnection();
                nc.addEventListener(NetStatusEvent.NET_STATUS,netStatus);
                nc.connect("http://localhost:3002");
            }

            private function netStatus(event:NetStatusEvent):void
            {
                switch(event.info)
                {
                    case "NetConnection.Connect.Success":
                        ns = new NetStream(nc);
                        ns.attachCamera(cam, 20);
                        ns.attachAudio(Microphone.getMicrophone());
                        ns.publish("appname", "live");
                    break;
                }
            }
        ]]>
    </mx:Script>
    <mx:VideoDisplay id="vid" width="360" height="320"/>
</mx:Canvas>
```

8.8. Работа с микрофоном и индикатор уровня громкости

Задача

Требуется получить данные с микрофона и использовать полученную информацию для вывода индикатора уровня громкости.

Решение

Для работы с микрофоном используйте метод `Microphone.getMicrophone`. Информация об уровне громкости берется из свойства `mic.activityLevel` с равномерными интервалами.

Обсуждение

Класс `Microphone` предоставляет доступ к микрофону, установленному на компьютере. Чтобы вы могли использовать класс, пользователь должен

разрешить приложению Flash Player доступ к микрофону. Класс Microphone содержит данные о громкости звука, полученные от микрофона, и передает события в начале воспроизведения, а также при отсутствии звука в течение заданного периода времени.

Три свойства класса Microphone предназначены для наблюдения и управления работой микрофона. Свойство activityLevel (доступно только для чтения) обозначает уровень громкости, обнаруженный микрофоном, по шкале от 0 до 100. Свойство silenceLevel задает уровень громкости, необходимый для активизации микрофона и передачи события ActivityEvent.ACTIVITY. Свойство silenceLevel тоже использует шкалу от 0 до 100; значение по умолчанию равно 10. Свойство silenceTimeout определяет продолжительность интервала (в миллисекундах), в течение которого уровень громкости должен оставаться ниже silenceLevel, чтобы было выдано событие ActivityEvent.ACTIVITY, сообщающее об отсутствии звука в микрофоне. По умолчанию значение silenceTimeout равно 2000. Хотя свойства Microphone.silenceLevel и Microphone.silenceTimeout доступны только для чтения, их значения можно изменить методом Microphone.setSilenceLevel.

В следующем примере создается объект Microphone; при этом пользователю предлагается разрешить или запретить Flash Player доступ к микрофону. Затем при обнаружении активности микрофона по событию Activity добавляется слушатель события, выводящий текущее значение soundLevel на компоненте Canvas.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="createMic()">

<mx:Script>
    <![CDATA[
    import flash.media.Microphone;
    import flash.events.ActivityEvent;
    import flash.events.Event;
    import flash.events.StatusEvent;

    public var mic:Microphone;

    public function createMic():void
    {
        mic = Microphone.getMicrophone();
        mic.setLoopBack(true);
        mic.addEventListener(ActivityEvent.ACTIVITY, activity);
        mic.addEventListener(StatusEvent.STATUS, status);
        mic.addEventListener(Event.ACTIVATE, active);
    }

    private function active(event:Event):void
    {
        trace(' active ');
    }
```

```
        private function status(event:StatusEvent):void
        {
            trace("status");
        }

        private function activity(event:ActivityEvent):void
        {
            trace("active ");
            addEventListener(Event.ENTER_FRAME, showMicLevel);
        }

        private function showMicLevel(event:Event):void
        {
            trace(mic.gain + " " + mic.activityLevel +
                "  "+mic.silenceLevel+" "+mic.rate);
            level.graphics.clear();
            level.graphics.beginFill(0xccccff, 1);
            level.graphics.drawRect(0, 0, (mic.activityLevel * 30), 100);
            level.graphics.endFill();
        }

        ]]>
    </mx:Script>
    <mx:Canvas width="300" height="50" id="level"/>
    </mx:VBox>
```

8.9. Сглаживание видео в приложении Flex

Задача

Требуется обеспечить сглаживание видеороликов, воспроизводимых в приложении.

Решение

Создайте пользовательский компонент, содержащий компонент `flash.media.Video`. Задайте свойству `smoothing` компонента `Video` значение `true`.

Обсуждение

Чтобы включить сглаживание видео (т. е. чтобы на изображении не были четко видны отдельные пикселы), необходимо использовать объект `flash.media.Video`. Сглаживание видео, как и сглаживание статической графики, требует бóльших затрат вычислительных ресурсов, чем при выводе несглаженной графики, и может замедлить воспроизведение для крупных или высококачественных роликов.

Компонент Flex `VideoDisplay` не позволяет задать свойство `smoothing` содержащегося в нем объекта `flash.media.Video`, поэтому вам придется создать отдельный компонент, включающий низкоуровневый компонент Flash `Video`, и задать свойство `smoothing`:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="setup()">
    <mx:Script>
        <![CDATA[

            private var vid:Video;

            private var nc:NetConnection;
            private var ns:NetStream;
            private var metaDataObj:Object = {};

            private function setup():void {
                vid = new Video(this.width, this.height);
                vid.smoothing = true;
                this.rawChildren.addChild(vid);
                vid.y = 50;
                this.invalidateDisplayList();
            }

            private function startVid():void {
                nc = new NetConnection();
                nc.addEventListener(NetStatusEvent.NET_STATUS,
                    netStatusHandler);
                nc.connect(null);
            }

            private function netStatusHandler(event:NetStatusEvent):void {
               ns = new NetStream(nc);
               metaDataObj.onMetaData = this.onMetaData;
               ns.client = metaDataObj;
               vid.attachNetStream(ns);
               ns.play("http://localhost:3001/Trailer.flv");
            }

            private function onMetaData(obj:Object):void {
                var i:int = 0;
                for each(var prop:Object in obj)
                {
                    trace(obj[i] + " : " + prop);
                    i++;
                }
                trace(obj.duration + " " + obj.framerate +
                    " "+obj.bitrate);
            }

        ]]>
    </mx:Script>
    <mx:Button click="startVid()" label="load" x="50"/>
    <mx:Button click="ns.resume()" label="resume" x="120"/>
    <mx:Button click="ns.pause()" label="pause" x="190"/>
</mx:Canvas>
```

8.10. Проверка коллизий на уровне пикселов

Задача

Требуется проверить, не возникают ли коллизии у изображений с альфа-прозрачностью с другими изображениями.

Решение

Выведите данные обоих изображений на объект `BitmapData` и воспользуйтесь методом `BitmapData.hitTest`.

Обсуждение

Объект `BitmapData` обладает методом `hitTest`, который напоминает метод `hitTest` класса `DisplayObject`, с одним существенным исключением: в отличие от метода `hitTest` класса `DisplayObject` возвращающего `true` в случае нахождения точки в границах объекта, метод `hitTest` класса `BitmapData` возвращает `true`, если пиксел в заданной точке находится выше заданного порога альфа-прозрачности.

Метод имеет следующую сигнатуру:

```
public function hitTest(firstPoint:Point, firstAlphaThreshold:uint,
    secondObject:Object, secondBitmapDataPoint:Point = null,
    secondAlphaThreshold:uint = 1):Boolean
```

Для непрозрачных изображений для этого метода конструируется полностью непрозрачный прямоугольник. Для выполнения проверки принадлежности (hit testing) на уровне пикселов с учетом прозрачности оба изображения должны быть прозрачными. При проверке двух прозрачных изображений параметры альфа-порога управляют тем, какие значения альфа-канала (от 0 до 255) считаются непрозрачными. Метод имеет следующие параметры:

`firstPoint:Point`

Позиция левого верхнего угла изображения `BitmapData` в произвольной системе координат. Та же система координат используется для определения параметра `secondBitmapPoint`.

`firstAlphaThreshold:uint`

Максимальное значение альфа-канала, при котором пиксел считается непрозрачным в контексте проверки принадлежности.

`secondObject:Object`

Объект `Rectangle`, `Point`, `Bitmap` или `BitmapData`.

`secondBitmapDataPoint:Point` (*по умолчанию = null*)

Координаты пиксела во втором объекте `BitmapData`. Параметр используется только в том случае, если значение `secondObject` представляет собой объект `BitmapData`.

`secondAlphaThreshold:uint` ***(по умолчанию = 1)***

Максимальное значение альфа-канала, при котором пиксел считается непрозрачным во втором объекте `BitmapData`. Параметр используется только в том случае, если значение `secondObject` представляет собой объект `BitmapData`, а оба объекта `BitmapData` прозрачны.

В следующем примере проверяются возможные коллизии каждого угла прямоугольного изображения с файлом PNG:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
width="1500" height="900">
    <mx:Script>
        <![CDATA[

            import flash.display.BlendMode;

            private var mainBmp:BitmapData;
            private var dragBmp:BitmapData;
            private var hasDrawn:Boolean = false;

            private function loaded():void{
                if(!hasDrawn){
                    mainBmp = new BitmapData(mainImg.width,
                        mainImg.height, true, 0x00000000);
                    dragBmp = new BitmapData(dragImg.width,
                        dragImg.height, true, 0x00000000);
                    hasDrawn = true;
                    this.addEventListener(Event.ENTER_FRAME, showHits);
                }
            }

            private function showHits(event:Event):void
            {
                mainBmp.draw(mainImg);
                dragBmp.draw(dragImg);
                if(mainBmp.hitTest(new Point(0,0), 0xff,
                    dragImg.getBounds(this).topLeft)){
                    trace(" true ");
                    return;
                }
                if(mainBmp.hitTest(new Point(0,0), 0xff,
                    dragImg.getBounds(this).bottomRight)){
                    trace(" true ");
                    return;
                }
                if(mainBmp.hitTest(new Point(0,0), 0xff,
                    new Point(dragImg.getBounds(this).left,
                    dragImg.getBounds(this).bottom))){
                    trace(" true ");
                    return;
                }
```

```
                if(mainBmp.hitTest(new Point(0,0), 0xff,
                    new Point(dragImg.getBounds(this).right,
                    dragImg.getBounds(this).top))){
                    trace(" true ");
                    return;
                }
                trace(" false ");
            }

        ]]>
    </mx:Script>
    <mx:Image id="mainImg" source="../assets/alphapng.png"
        cacheAsBitmap="true"/>
    <mx:Image cacheAsBitmap="true" id="dragImg"
        mouseDown="dragImg.startDrag(false, this.getBounds(stage)),
        loaded()" rollOut="dragImg.stopDrag()"
        mouseUp="dragImg.stopDrag()" source="../assets/bigshakey.png"/>
</mx:Canvas>
```

Код возвращает `false`, если альфа-каналы пикселов первого изображения в заданных точках не превышают пороговых значений, заданных при вызове метода `hitTest`. На рис. 8.1 два светло-синих квадрата расположены внутри файла PNG с альфа-прозрачностью. Стаканчик с коктейлем представляет собой отдельное изображение, которое в данный момент не перекрывается ни с одной областью PNG, обладающей достаточно высокими значениями альфа-канала. На рис. 8.2 стаканчик соприкасается с квадратом, и метод возвращает `true`.

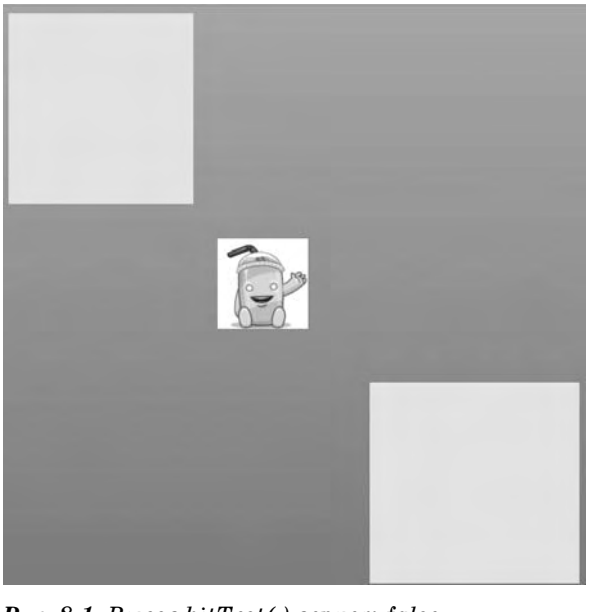

***Рис. 8.1.** Вызов hitTest( ) вернет false*

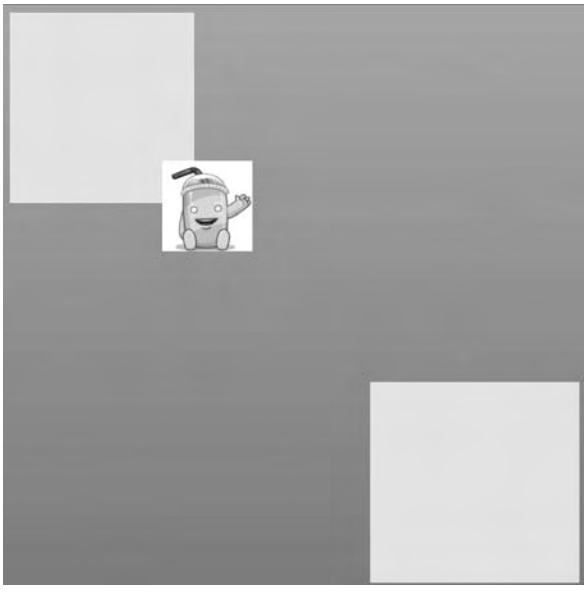

***Рис. 8.2.** Вызов hitTest( ) вернет true*

8.11. Чтение и сохранение изображения с веб-камеры

Задача

Требуется прочитать изображение с веб-камеры пользователя и сохранить его на сервере.

Решение

Создайте объект `Camera` и присоедините его к объекту `Video`. Затем создайте кнопку, которая читает растровое изображение из объекта `Video` и передает его данные серверному сценарию, который сохраняет изображение.

Обсуждение

Чтобы сохранить изображение с веб-камеры, создайте растровое изображение на базе объекта `Video`, отображающего информацию с веб-камеры. Однако Flash Player не предоставляет доступ к потоку данных, прочитанному с камеры; прежде чем использовать данные, необходимо предварительно воспроизвести их в растровом формате.

После того как данные будут сохранены в объекте `BitmapData`, их можно передать экземпляру класса `JPEGEncoder` для преобразования графики

в данные JPEG. Чтобы сохранить JPEG на сервере, включите данные в объект URLRequest и отправьте их методом navigateToURL. Пример:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="500" creationComplete="setUpCam()">
    <mx:Script>
        <![CDATA[
            import flash.net.navigateToURL;
            import flash.net.sendToURL;
            import mx.graphics.codec.JPEGEncoder;

            private var cam:Camera;
            private function setUpCam():void {
                cam = flash.media.Camera.getCamera("2");
                vid.attachCamera(cam);
            }

            private function saveImage():void {
                var bitmapData:BitmapData =
                    new BitmapData(vid.width, vid.height);
                bitmapData.draw(vid);
                var encode:JPEGEncoder = new JPEGEncoder(100);
                var ba:ByteArray = encode.encode(bitmapData);
                var urlRequest:URLRequest =
                    new URLRequest("/jpg_reader.php");
                urlRequest.method = "POST";
                var urlVars:URLVariables = new URLVariables();
                urlVars.pic = ba;
                urlRequest.data = urlVars;
                flash.net.navigateToURL(urlRequest, "_blank");
            }
        ]]>
    </mx:Script>
    <mx:VideoDisplay id="vid" width="360" height="320"/>
    <mx:Button label="Take Picture Now" click="saveImage()"/>
</mx:Canvas>
```

8.12. Объединение изображений

Задача

Требуется объединить несколько изображений в одно.

Решение

Задайте свойство blendMode для объединяемых изображений.

Обсуждение

Каждый объект DisplayObject определяет свойство blendMode, управляющее отображением объекта; в частности, оно управляет альфа-

прозрачностью и способом отображения других объектов DisplayObject, находящихся под текущим объектом в списке отображения DisplayList. Режимы наложения blendMode хорошо знакомы каждому, кто когда-либо работал с Adobe Photoshop или After Effects:

`BlendMode.ADD ("add")`

Анимированный эффект осветляющего растворения с переходом между двумя изображениями.

`BlendMode.ALPHA ("alpha")`

Прозрачность основного изображения применяется к фону.

`BlendMode.DARKEN ("darken")`

Затемнение.

`BlendMode.DIFFERENCE ("difference")`

Создание более ярких, контрастных цветов.

`BlendMode.ERASE ("erase")`

Стирание части фона с использованием альфа-канала основного изображения.

`BlendMode.HARDLIGHT ("hardlight")`

Создание эффекта тени.

`BlendMode.INVERT ("invert")`

Инверсия фона.

`BlendMode.LAYER ("layer")`

Создание временного буфера для предварительного формирования отображаемого объекта.

`BlendMode.LIGHTEN ("lighten")`

Осветление.

`BlendMode.MULTIPLY ("multiply")`

Создание эффектов тени и визуальной глубины.

`BlendMode.NORMAL ("normal")`

Пикселы накладываемого изображения заменяют пикселы базового изображения.

`BlendMode.OVERLAY ("overlay")`

Создание эффекта тени.

`BlendMode.SCREEN ("screen")`

Создание подсветки и бликов.

`BlendMode.SUBTRACT ("subtract")`

Анимированный эффект затемняющего растворения с переходом между двумя изображениями.

Пример использования различных режимов смешения с двумя объектами `Image`:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800"
    height="800">
    <mx:Script>
        <![CDATA[
            import flash.display.BlendMode;
        ]]>
    </mx:Script>
    <mx:Image id="img1" mouseDown=
        "img1.startDrag(false, this.getBounds(stage)),
        swapChildren(img1, img2)" rollOut="img1.stopDrag()"
        mouseUp="img2.stopDrag()" source="../assets/mao.jpg"/>
    <mx:Image id="img2" mouseDown=
        "img2.startDrag(false, this.getBounds(stage)),
        swapChildren(img2, img1)" rollOut="img2.stopDrag()"
        mouseUp="img2.stopDrag()" source="../assets/bigshakey.png"/>
    <mx:HBox>
        <mx:CheckBox id="chb" label="which one"/>
        <mx:ComboBox id="cb" dataProvider=
            "{[BlendMode.ADD, BlendMode.ALPHA, BlendMode.DARKEN,
            BlendMode.DIFFERENCE, BlendMode.ERASE, BlendMode.HARDLIGHT,
            BlendMode.INVERT, BlendMode.LAYER, BlendMode.LIGHTEN,
            BlendMode.MULTIPLY, BlendMode.NORMAL, BlendMode.OVERLAY,
            BlendMode.SCREEN, BlendMode.SUBTRACT]}"
        change="chb.selected ?
            img1.blendMode = cb.selectedItem as String :
            img2.blendMode = cb.selectedItem as String"/>
    </mx:HBox>
</mx:Canvas>
```

8.13. Использование опорных точек в данных FLV

Задача

Требуется использовать опорные точки (cue points), встроенные в файл FLV, во время воспроизведения.

Решение

Используйте событие `onCuePoint` класса `NetStream` для создания метода-обработчика, активизируемого при достижении опорной точке в видеопотоке.

Обсуждение

Опорной точкой называется значение, вставленное в определенную временную позицию видеоролика в файле FLV. Это значение представляет собой имя или объект данных в хеш-таблице. Обычно опорные точки вставляются в FLV при кодировании файла, и их значения определяются

на этой стадии. Объект Flex VideoDisplay использует класс mx.controls.videoclasses.CuePointManager для обнаружения опорных точек и чтения данных из них. Чтобы вы лучше поняли принципы работы с опорными точками, рассмотрим пример с использованием объекта flash.media.Video.

При подключении объекта NetConnection и создании экземпляра NetStream необходимо настроить объект для передачи всех метаданных и событий опорных точек методам-обработчикам:

```
var obj:Object = new Object();
obj.onCuePoint = onCuePoint;
obj.onMetaData = onMetaData;
ns.client = obj;
```

Настройка должна быть выполнена до вызова метода NetStream.play. Обратите внимание: в следующем листинге каждое из событий onMetaData и onCuePoint получает параметр Object:

```
import flash.events.NetStatusEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;
import mx.core.UIComponent;

public class CuePointExample extends UIComponent
{
    private var ns:NetStream;
    private var nc:NetConnection;
    private var obj:Object = {};
    private var vid:Video;

    public function CuePointExample () {
        super();
        vid = new Video();
        addChild(vid);
        nc = new NetConnection();
        nc.addEventListener(NetStatusEvent.NET_STATUS,
            netStatusEventHandler);
        nc.connect(null);
    }
    private function netStatusEventHandler(
        event:NetStatusEvent):void {
        ns = new NetStream(nc);
        obj.onCuePoint = onCuePoint;
        obj.onMetaData = onMetaData;
        ns.client = obj;
        ns.play("http://localhost:3001/test2.flv");
        vid.attachNetStream(ns);
    }
    private function onCuePoint(obj:Object):void {
        trace(obj.name+" "+obj.time+" "+obj.length+" ");
```

```
            for each(var o:String in obj.parameters) {
                trace(obj[o]+" "+o);
            }
        }
        private function onMetaData(obj:Object):void{
        }
    }
```

Использование `mx.controls.VideoDisplay` значительно упрощает работу с объектами опорных точек. В отличие от предыдущего примера, при использовании события `CuePointEvent`, переданного `CuePointManager`, полученное событие обладает только тремя свойствами: `cuePointTime`, `cuePointName` и `cuePointType`. Если вам потребуется дополнительная или другая информация об опорной точке, напишите собственный класс, возвращающий данные опорной точки, и задайте его свойству `cuePoint-Manager` объекта `VideoDisplay`. Полный код выглядит примерно так:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.events.CuePointEvent;

            private function onCuePoint(event:CuePointEvent):void {
                trace(event.cuePointName +" "+event.cuePointTime+
                " "+event.cuePointType+" ");
            }

        ]]>
    </mx:Script>
    <mx:VideoDisplay id="vid" cuePoint="onCuePoint(event)"/>
</mx:VBox>
```

8.14. Создание шкалы позиционирования

Задача

Требуется создать элемент управления, при помощи которого пользователь сможет управлять воспроизведением видеоролика.

Решение

Создайте объект `Sprite` с поддержкой перетаскивания; прослушивайте все события `DragEvent`, передаваемые им. В обработчике `DragEvent` задайте величину смещения (прямого или обратного) в потоке `NetStream`, поставляющем видеоданные для объекта `Video`.

Обсуждение

Для задания новой позиции, с которой может продолжаться воспроизведение видеоролика, можно воспользоваться любым перетаскиваемым экранным объектом. В следующем примере метод `seek` объекта `NetStream`

начинает воспроизведение с заданной (в секундах) позиции от начала видеопотока:

```
ns.seek((playhead.x/timeline.width) * length);
```

Чтобы определить, к какой секунде видеоролика пользователь хочет перейти, следует разделить позицию перетаскиваемого объекта `Sprite` на ширину шкалы и умножить на продолжительность ролика. `NetStream` найдет соответствующие кадры в видео и перезапустит потоковую передачу с этой точки.

```
import flash.display.Sprite;
import flash.events.MouseEvent;
import flash.events.NetStatusEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;
import mx.core.UIComponent;

public class Scrubber extends UIComponent
{
    private var playhead:Sprite;
    private var timeline:Sprite;
    private var ns:NetStream;
    private var nc:NetConnection;
    private var obj:Object = {};
    private var length:int;
    private var vid:Video;

    public function Scrubber () {
        super();
        playhead = new Sprite();
        addChild(playhead);
        playhead.graphics.beginFill(0x0000ff, 1);
        playhead.graphics.drawCircle(0, 0, 5);
        playhead.graphics.endFill();
        playhead.addEventListener(MouseEvent.MOUSE_DOWN, startSeek);
        timeline = new Sprite();
        timeline.graphics.beginFill(0xcccccc, 1);
        timeline.graphics.drawRect(0, 0, 200, 10);
        timeline.graphics.endFill();
        addChild(timeline);
        timeline.addChild(playhead);
        playhead.y = 4;
        vid = new Video();
        addChild(vid);
        vid.y = 100;

        nc = new NetConnection();
        nc.addEventListener(NetStatusEvent.NET_STATUS, netStatus);
        nc.connect(null);
    }
```

```
    private function netStatus(event:NetStatusEvent):void {
        obj.onMetaData = onMetaData;
        ns = new NetStream(nc);
        ns.client = obj;
        vid.attachNetStream(ns);
        ns.play("http://localhost:3001/test.flv");
    }
    private function onMetaData(obj:Object):void {
        length = obj.duration;
        trace(length);
    }
    private function startSeek(mouseEvent:MouseEvent):void {
        playhead.startDrag(false, timeline.getBounds(this));
        addEventListener(MouseEvent.MOUSE_MOVE, seek);
        playhead.addEventListener(MouseEvent.ROLL_OUT, endSeek);
        playhead.addEventListener(MouseEvent.MOUSE_UP, endSeek);
    }
    private function seek(mouseEvent:MouseEvent):void {
        ns.seek((playhead.x/timeline.width) * length);
    }
    private function endSeek(mouseEvent:MouseEvent):void {
        removeEventListener(MouseEvent.MOUSE_MOVE, seek);
        playhead.stopDrag();
    }
```

8.15. Чтение данных ID3 из файла MP3

Задача

Требуется прочитать данные ID3 из файла MP3.

Решение

Используйте событие `Event.ID3`, передаваемое после завершения разбора данных ID3.

Обсуждение

После завершения разбора данных ID3 в загруженном файле MP3 класс `Sound` передает событие. Данные сохраняются в объекте `ID3Info`, который определяет переменные для обращения ко всем свойствам в начальных байтах MP3:

```
private var sound:Sound;

public function _8_16()
{
    sound = new Sound();
    sound.addEventListener(Event.ID3, onID3InfoReceived);
    sound.load(new URLRequest("../assets/1.mp3"));
}
```

```
private function onID3InfoReceived(event:Event):void
{
    var id3:ID3Info = event.target.id3;
    for (var propName:String in id3)
    {
        trace(propName + " = " + id3[propName]);
    }
}
```

В песне, которую я слушал во время работы над этим рецептом, эта информация выглядит так:

```
TCON = Alternative & Punk
TIT2 = The Pink Batman
TRCK = 2/9
TPE1 = Dan Deacon
TALB = Spiderman Of The Rings
TCOM = Dan Deacon
```

Информация ID3 в файле MP3 представляет собой обычную группу байтов, записанных в определенном порядке. Байты читаются и преобразуются в строки или целые числа. MP3 – единственный формат аудиофайлов, поддерживаемый Flash Player в исходном виде. Впрочем, разработчик Бенджамин Доблер (Benjamin Dobler) из RichApps (*www.richapps.de*) проделал выдающуюся работу по поддержке формата WAV. Обеспечить воспроизведение файла WAV в Flash Player чуть сложнее. Если вас заинтересует эта тема, зайдите на сайт Бенджамина и посмотрите сами. Разбор данных из файла WAV выглядит примерно так:

```
public var bytes:ByteArray;
public var chunkId:String;
public var chunkSize:int;
public var chunkFormat:String;
public var subchunk1Id:String;
public var subchunk1Size;
public var audioFormat;
public var channels;
public var sampleRate;
public var bytesPersecond;
public var blockAlign;
public var bitsPerSample;
public var dataChunkSignature:String;
public var dataChunkLength;

public function read(bytes:ByteArray):void{
    this.bytes = bytes;
    // Чтение заголовка
    bytes.endian = "littleEndian";
    chunkId = bytes.readMultiByte(4,"utf"); //RIFF
    chunkSize = bytes.readUnsignedInt();
    chunkFormat = bytes.readMultiByte(4,"utf"); //WAVE
```

```
        subchunk1Id = bytes.readMultiByte(4,"iso-8859-1");
                                        // Сигнатура заголовка
        subchunk1Size = bytes.readInt(); // 16 4 <fmt length>
        audioFormat = bytes.readShort(); // 20 2 <format tag> sample
        channels = bytes.readShort(); // 22 2 <channels>
                                      // 1 = mono, 2 = stereo
        sampleRate = bytes.readUnsignedInt();// 24 4 <sample rate>
        bytesPersecond = bytes.readUnsignedInt(); //28 4
                    // <bytes/second> Sample-Rate * Block-Align
        blockAlign = bytes.readShort(); // 32 2 <block align>
                                        // channel * bits/sample / 8
        bitsPerSample = bytes.readUnsignedShort();
                                    // 34 <bits/sample> 8, 16 or 24
        dataChunkSignature = bytes.readMultiByte(4,"iso-8859-1"); //RIFF
        dataChunkLength = bytes.readInt();
    }
```

Если заголовок читается из файла AU, он выглядит так:

```
    public var bytes:ByteArray;
    public var magicId;
    public var header;
    public var datasize;
    public var channels;
    public var comment;
    public var sampleRate;
    public var encodingInfo;

    public function read(bytes:ByteArray):void{
        this.bytes = bytes;
        // Чтение заголовка
        bytes.endian = "bigEndian";
        magicId = bytes.readUnsignedInt();
        header = bytes.readInt();
        datasize = bytes.readUnsignedInt();
        encodingInfo = bytes.readUnsignedInt();
        sampleRate = bytes.readUnsignedInt();
        channels = bytes.readInt();
        comment = bytes.readMultiByte(uint(header)-24, "utf");
    }
```

MP3 – самый простой формат для чтения служебных данных, но отнюдь не единственный.

8.16. Отображение пользовательской анимации во время загрузки

Задача

Требуется вывести пользовательскую анимацию на время загрузки изображения.

Решение

Создайте пользовательскую графику и прослушайте событие ProgressEvent.PROGRESS во время загрузки. Выведите изображение при помощи свойств bytesLoaded и bytesTotal.

Обсуждение

Существует два основных механизма отображения графики при использовании компонента Image: назначение источника для класса Image в MXML и передача URL-адреса для загрузки с последующим вызовом метода img.load:

```
img.load("http://thefactoryfactory.com/beach.jpg");
```

Но прежде чем загружать изображение, необходимо назначить слушателя, обеспечивающего обработку событий ProgressEvent:

```
img.addEventListener(ProgressEvent.PROGRESS, progress);
```

В методе progress, обрабатывающем событие ProgressEvent.PROGRESS, для перерисовки UIComponent используется свойство bytesLoaded класса Image:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="loadImage()">

    <mx:Script>
        <![CDATA[

            private var m:Matrix;

            private function loadImage():void {
                var m:Matrix = new Matrix();
                m.createGradientBox(450, 40);
                img.addEventListener(ProgressEvent.PROGRESS, progress);
                img.load("http://thefactoryfactory.com/beach.jpg");
            }

            private function progress(event:Event):void{
                grid.graphics.clear();
                grid.graphics.beginGradientFill("linear",
                    [0x0000ff, 0xffffff], [1, 1], [0x00, 0xff], m);
                grid.graphics.drawRect(0, 0,
                    (img.bytesLoaded / img.bytesTotal) * 300, 40);
                grid.graphics.endFill();
            }
        ]]>
    </mx:Script>

    <mx:Canvas id="grid" height="40" width="300"/>
    <mx:Image id="img" y="40"/>
</mx:Canvas>
```

8.17. Отправка графики в приложениях Flex

Задача

Требуется средствами Flex реализовать отправку графических изображений на сервер.

Решение

Создайте объект FileReference с присоединением соответствующих фильтров, чтобы пользователи могли передавать файлы только графических типов. Создайте обработчик события complete от объекта FileReference и передайте изображение сценарию на стороне сервера.

Обсуждение

Функция отправки графики в Flex, как и в Flash, базируется на использовании класса FileReference. Объект FileReference создает окно передачи с использованием стандартного окна и графического оформления броузера, и когда пользователь выберет изображение для передачи – отправляет его через Flash Player. Чтобы узнать о выборе файла пользователем, добавьте слушателя события для объекта FileReference:

```
fileRef.addEventListener(Event.SELECT, selectHandler);
```

Добавьте метод для отправки файла, выбранного пользователем:

```
private function selectHandler(event:Event):void {
    var request:URLRequest = new URLRequest(
        "http://thefactoryfactory.com/upload2.php");
    fileRef.upload(request, "Filedata", true);
}
```

Отправленный графический файл передается сценарию PHP для сохранения:

```
package oreilly.cookbook
{
    import mx.core.UIComponent;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.events.Event;

    public class _8_17 extends UIComponent
    {
        private var fileRef:FileReference;

        public function _8_17() {
            super();
            startUpload();
        }
        private function startUpload():void {
```

```
            // Перечисление всех типов файлов,
            // отправка которых будет разрешена пользователю
            var imageTypes:FileFilter = new FileFilter(
                "Images (*.jpg, *.jpeg, *.gif, *.png)",
                "*.jpg; *.jpeg; *.gif; *.png");
            var allTypes:Array = new Array(imageTypes);
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT,
                selectHandler);
            fileRef.addEventListener(Event.COMPLETE,
                completeHandler);
            // Приказываем объекту FileRefence
            // принимать только графические файлы
            fileRef.browse(allTypes);
        }

        private function selectHandler(event:Event):void {
            var request:URLRequest = new URLRequest(
                "http://thefactoryfactory.com/upload2.php");
            fileRef.upload(request, "Filedata", true);
        }

        private function completeHandler(event:Event):void {
            trace("uploaded");
        }
    }
}
```

Отправленный файл обрабатывается на сервере, например перемещается в папку с именем images:

```
$file_temp = $_FILES['file']['tmp_name'];
$file_name = $_FILES['file']['name'];
$file_path = $_SERVER['DOCUMENT_ROOT']."/images";

// Проверка дубликатов
if(!file_exists($file_path."/".$file_name)) {
    //
    $filestatus = move_uploaded_file(
    $file_temp,$file_path."/".$file_name);
    if(!$filestatus) {
        // Ошибка при передаче
    }
}
```

8.18. Сравнение двух растровых изображений

Задача

Требуется сравнить два растровых изображения и вывести визуальную сводку различий между ними.

Решение

Прочитайте растровые данные из двух изображений, воспользуйтесь методом `compare` для их сравнения. Назначьте результат сравнения источником третьего изображения.

Обсуждение

Метод `compare` класса `BitmapData` возвращает объект `BitmapData` с информацией обо всех пикселах, различающихся в двух изображениях. Если два объекта `BitmapData` имеют одинаковые размеры (ширину и длину), метод возвращает новый объект `BitmapData`, в котором каждый пиксел определяется разностью пикселов двух исходных объектов. Если пикселы совпадают, итоговому пикселу присваивается значение 0x00000000. Если пикселы имеют различающиеся коды RGB (без учета альфа-канала), итоговому пикселу присваивается значение 0xFFRRGGBB, где RR/GG/BB – разности красной, зеленой и синей составляющих. Различия в альфа-канале при этом игнорируются. Если пикселы различаются только альфа-каналом, пикселу присваивается значение 0xZZFFFFFF, где ZZ – разность альфа-составляющих.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="800">
    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;

            private function compare():void {
                var bmpd1:BitmapData =
                    new BitmapData(img1.width, img1. height);
                var bmpd2:BitmapData =
                    new BitmapData(img2.width, img2. height);
                bmpd1.draw(img1)
                bmpd2.draw(img2);
                var diff:BitmapData = bmpd2.compare(bmpd1) as BitmapData;
                var bitmapAsset:BitmapAsset = new BitmapAsset(diff);
                img3.source = bitmapAsset;
            }
        ]]>
    </mx:Script>
    <mx:Image id="img1" source="../assets/mao.jpg"
        height="200" width="200"/>
    <mx:Image id="img2" source="../assets/bigshakey.png"
        height="200" width="200"/>
    <mx:Button click="compare()" label="compare"/>
    <mx:Image id="img3"/>
</mx:VBox>
```

9

Скины и стили

Благодаря мощным средствам управления раскладкой Flex Framework в сочетании со стандартной темой Halo Aeon приложения выглядят достаточно привлекательно даже без дополнительных усилий по их оформлению. Простота построения интерфейса из контейнеров и компонентов сопоставима с простотой настройки их оформления при помощи скинов и стилей.

Впрочем, название главы не совсем точно: стили и скины в Flex **не являются** разнородными концепциями. Они используются в сочетании друг с другом для обеспечения визуальной уникальности приложения. Скины назначаются свойствами стилей и могут использовать другие объявленные значения стилей. С другой стороны, стилевое оформление и применение скинов следуют разным принципам, и в этой главе вы узнаете, как извлечь из них максимум пользы.

Стили представляют собой значения свойств, инструкции по назначению цветов, размеров и шрифтов, которые изменяют внешний вид компонентов и могут задаваться на программном уровне – как на стадии компиляции, так и на стадии выполнения. Стилевые свойства определяются разными способами: встроенными при объявлении компонента, применением стиля вызовом `setStyle` или в стилевых таблицах CSS (Cascading Style Sheets). Таблицы CSS могут использоваться как для локального определения стилей в файлах MXML, так и во внешних файлах.

Скины назначаются как стилевые свойства и используются для изменения элементов визуального оформления компонента. Они могут быть представлены как графическими элементами (как, например, файл изображения или SWF), так и программными классами, использующими API графического вывода. Скиновое оформление компонента может привести к изменению и даже замене некоторых его визуальных элементов.

Соответственно некоторые стилевые назначения могут игнорироваться при назначении скина.

Чтобы вы лучше поняли, как совместная работа этих двух процессов изменяет внешний вид компонентов, рассмотрим визуальное оформление обычной кнопки. Существует несколько стилевых свойств, которые можно назначить кнопке, в частности, относящихся к шрифтовому оформлению и выравниванию. В их число также входят свойства для определения скинов разных состояний экземпляра `Button`.

Для примера возьмем небольшое подмножество стилей, применяемых к `Button`:

```
cornerRadius="4"
fillAlphas="[0.6, 0.4]"
fillColors="[0xE6EEEE, 0xFFFFFF]"
upSkin="mx.skins.halo.ButtonSkin"
```

Значения свойств в этом укороченном списке стилей кнопок представляют собой значения по умолчанию из файла defaults.css, входящего в Framework. Значение по умолчанию свойства `upSkin` экземпляра кнопки представляет собой класс `mx.skins.halo.ButtonSkin` из пакета скинов Halo.

Используя графический API, этот программный скин прорисовывает экземпляр `Button` в «отжатом» состоянии; при этом учитываются значения других свойств: `cornerRadius`, `fillAlphas` и `fillColors`.

В этой главе описаны процессы стилевого и скинового оформления компонентов, а также возможности их эффективного использования для модификации внешнего вида и поведения приложения.

Сетевое приложение **Flex Style Explorer дает возможность изменять** стили различных компонентов во время выполнения, с просмотром полученных определений CSS. Приложение доступно по адресу *http://www.adobe.com/go/flex_styles_explorer_app*.

9.1. Использование таблиц CSS для стилевого оформления компонентов

Задача

Требуется изменить внешний вид компонентов при помощи каскадных таблиц CSS.

Решение

Объявите стилевые свойства с использованием селектора класса или селектора типа.

Обсуждение

Таблицы CSS могут использоваться для стилевого оформления компонентов пользовательского интерфейса в ваших приложениях. Если вы уже знакомы со стилевым оформлением элементов в документах HTML, синтаксис CSS для Flex покажется вам в целом знакомым. Селекторы классов определяются для назначения стилей нескольким экземплярам компонентов, а селекторы – для всех экземпляров компонента в списке отображения.

Селекторы классов могут многократно объявляться и использоваться для разных компонентов приложения. Синтаксис селектора класса состоит из точки, за которой следует имя стиля по схеме Camel Caps, начиная со строчного символа, например `.myCustomStyle`. В следующем примере стиль, созданный локально в теге `<mx:Style>`, назначается компоненту `Label` с использованием атрибута `styleName`:

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

    <mx:Style>
        .labelStyle {
            font-family: 'Arial';
            font-size: 20px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>

    <mx:Label text="Hello Flex!" styleName="labelStyle" />
</mx:Application>
```

Селектор `.labelStyle` в этом примере определяет стилевые свойства, относящиеся к шрифтовому оформлению надписи. Стиль применяется как встроенный, с указанием его имени в качестве значения атрибута `styleName` экземпляра `Label`. Обратите внимание: при назначении стиля начальная точка отсутствует.

Селекторы типов могут использоваться для назначения своего типа для всех экземпляров компонента. Чтобы объявить селектор типа, используйте имя класса оформляемого компонента, как показано в следующем примере:

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

    <mx:Style>
        Label {
            font-family: 'Arial';
            font-size: 20px;
```

```
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>

    <mx:Label text="Hello Flex!" />
</mx:Application>
```

Итак, вы можете объявлять селекторы типов и классов для стилевого оформления компонентов, однако определяемые свойства могут заменяться на стадии компиляции встроенными определениями в объявлениях компонентов.

В следующем примере стиль применяется к двум экземплярам компонента `Label`, причем второй компонент заменяет стилевое свойство `fontSize`:

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

    <mx:Style>
        Label {
            font-family: 'Arial';
            font-size: 20px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>

    <mx:Label text="Hello Flex!" />
    <mx:Label text="Hello Flex!" fontSize="12" />
</mx:Application>
```

Применение встроенных значений стилевых свойств на стадии компиляции не ограничивается стилями CSS, объявляемыми локально в теге `<mx:Style>`, как в данном примере. Значение стилевых свойств, объявленных как встроенные, могут заменять стили из внешних стилевых файлов, загружаемых как на стадии выполнения, так и на стадии компиляции.

Вероятно, вы заметили, что имя встроенного свойства `fontSize` в этом примере записано по схеме «Camel Caps», тогда как имена стилевых свойств в объявлении CSS разделяются дефисами. Объявления, разделяемые дефисами, следуют традиции CSS и могут использоваться для объявления стилевых свойств во внешних таблицах стилей и в тегах `<mx:Style>`. Применение схемы «Camel Caps» для стилевых свойств, объявляемых встроенными в MXML, соответствует стандартам ActionScript, не поддерживающим разделение дефисами. Во внутренней реализации объявления с разделением дефисами в CSS преобразуются в «Camel Caps» и используются для получения объявлений свойств методом `getStyle`.

9.2. Переопределение стиля по умолчанию для Application

Материал предоставлен Марко Касарио (Marco Casario).

Задача

Требуется изменить стиль, назначаемый по умолчанию главному контейнеру `Application`.

Решение

Задайте атрибуту `styleName` главного приложения значение `plain`.

Обсуждение

Корневой контейнер приложения Flex – `Application` – представляет область, прорисовываемую Flash Player.

Контейнер `Application` обладает свойствами по умолчанию, определяющими его визуальный стиль и оформление. Например, с тегом `Application` связываются атрибуты `horizontalGap` и `verticalGap` (горизонтальное и вертикальное расстояние между дочерними компонентами), равные соответственно 8 и 6 пикселам.

В некоторых ситуациях бывает нужно сбросить все свойства по умолчанию для тега `Application`. Для решения этой задачи атрибуту `styleName` задается значение `plain`:

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

При задании этому атрибуту значения `plain` происходит следующее:

- Все отступы задаются равными 0
- Цвету фона задается значение #FFFFFF
- Фоновое изображение удаляется
- Дочерние компоненты выравниваются по левому краю

Стиль `plain` определяется в файле Framework с именем `defaults.css`. Чтобы узнать, как он инициализирует свойства, взгляните на селектор стиля `.plain` в таблице стилей Flex Framework по умолчанию:

```
.plain
{
    backgroundColor: #FFFFFF;
    backgroundImage: "";
    horizontalAlign: "left";
    paddingBottom: 0;
    paddingLeft: 0;
```

```
        paddingRight: 0;
        paddingTop: 0;
    }
```

В этом и проявляется мощь селекторов классов: они позволяют назначить один стиль разным типам компонентов или же назначить разные стили одному типу компонентов.

В Flex также поддерживаются селекторы типов. В следующем примере в теге `<mx:Style>` объявляется селектор типа для приложения с таким же стилевым оформлением, как при задании атрибуту `styleName` значения `plain`:

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

        Application {
            backgroundColor: #ffffff;
            backgroundImage: '';
            paddingLeft: 0;
            paddingRight: 0;
            paddingTop: 0;
            paddingBottom: 0;
            horizontalAling: 'left';
        }

    </mx:Style>
</mx:Application>
```

9.3. Встроенные стили с использованием CSS

Задача

Требуется определить встроенные стили для компонентов с использованием CSS в приложении.

Решение

Определите стили локально или воспользуйтесь атрибутом `source` тега `<mx:Style>` для встраивания правил CSS из внешнего файла.

Обсуждение

Существует несколько способов определения встроенных стилей на стадии компиляции в приложениях Flex. В этом рецепте рассматривается определение стилей в синтаксисе CSS, встроенных в приложение. Использование CSS в Flex позволяет:

- объявлять стили локально в теге `<mx:Style>` файла MXML
- задать в атрибуте `source` тега `<mx:Style>` внешнюю таблицу стилей

В главном файле приложения из следующего примера локально объявляются два стиля классов, применяемые к экземплярам Label в приложении:

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

    <mx:Style>
        .header {
            font-family: 'Arial';
            font-size: 15px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
        .message {
            font-family: 'Arial';
            font-size: 12px;
            color: #336699;
        }
    </mx:Style>

    <mx:Label text="I have a header style!" styleName="header" />
    <mx:Text text="I have a message style!" styleName="message" />

</mx:Application>
```

В этом примере селекторы классов объявляются локально, после чего назначаются во встроенном формате компоненту Label и компоненту Text. Значения свойств, определяемые в каждом селекторе, применяются к соответствующим элементам экземпляров компонентов, а говоря более конкретно – изменяют шрифтовое оформление надписей. Хотя в приведенном примере стили объявляются локально для контейнера Application, они также могут объявляться в пользовательских компонентах MXML с использованием тега <mx:Style>.

Возможен и другой вариант: объявления стилей хранятся во внешнем файле CSS, имя которого задается в атрибуте source тега <mx:Style>. Например, следующий фрагмент кода можно сохранить в файле с расширением .css для последующего встраивания:

```
.header {
    font-family: 'Arial';
    font-size: 15px;
    font-weight: 'bold';
    color: #FFFFFF;
}
.message {
    font-family: 'Arial';
    font-size: 12px;
    color: #336699;
}
```

Чтобы встроить внешний файл CSS в приложение, укажите путь к файлу в атрибуте `source` тега `<mx:Style>`. Следующий фрагмент с встраиванием файла CSS приводит к такому же визуальному результату, что и предыдущий пример:

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

    <mx:Style source="assets/styles/label_styles.css" />

    <mx:Label text="Hello!" styleName="header" />
    <mx:Text text="Welcome to the example." styleName="message" />
</mx:Application>
```

Хотя стили по умолчанию для приложений Flex объявляются в файле `defaults.css`, входящем в Framework, существует немало других тем оформления, которые можно использовать вместо него. Поищите в каталоге *<установочный_каталог_flex sdk>*/frameworks/themes.

9.4. Переопределение базовых стилевых свойств

Задача

Требуется изменить базовое стилевое свойство в экземпляре компонента.

Решение

Задайте значение стилевого свойства во встроенном формате, с использованием атрибута или дочернего тега компонента.

Обсуждение

Стилевые свойства можно задавать как во встроенном формате, так и в объявлениях дочерних тегов компонентов. В любом случае определение стиля переопределяет значения, уже определенные в приложении (локально или во внешнем файле). Пример:

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

    <mx:Style>
        Label {
            font-family: 'Arial';
            font-size: 20px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>
```

```
        <mx:Label text="Hello Flex!" color="#336699" />
        <mx:Text text="Hello Flex!" fontStyle="italic">
            <mx:fontFamily>Verdana</mx:fontFamily>
        </mx:Text>
    </mx:Application>
```

Здесь селектор типа объявляется локально в главном приложении, после чего стилевые свойства назначаются экземплярам `Label`. По сути, селектор типа, объявленный в теге `<mx:Script>`, переопределяет стиль по умолчанию из файла defaults.css Flex Framework, а элементы компонентов заменяют конкретные стилевые свойства этого объявления.

Компонент `Label` устанавливает цвет шрифта, отличный от того, который был задан в селекторе типа для экземпляров `Label`. Компонент `Text` наследует все стили, объявленные в селекторе типа, потому что `Text` является субклассом `Label`. Соответственно шрифтовые атрибуты, определенные для `Label`, задаются дочерним элементам компонента `Text`. Экземпляр `Text` задает значение свойства `fontStyle`, не определенного в селекторе, и заменяет `fontFamily` в дочернем теге; в данном случае Arial заменяется шрифтом Verdana.

Замена стилей во встроенных объявлениях и дочерних тегах – удобный способ замены оформления отдельных компонентов, обладающих общим стилем согласно объявлениям в селекторе типа или класса, на стадии компиляции.

См. также

Рецепт 9.3.

9.5. Настройка стилей во время выполнения

Задача

Требуется настроить значения стилевых свойств, заданные компонентам, во время выполнения.

Решение

Используйте метод `setStyle` для повторного назначения свойств.

Обсуждение

Метод `setStyle` наследуется всеми субклассами `mx.core.UIComponent`. Он позволяет задавать стилевым свойствам нужные значения во время выполнения. В аргументах `setStyle` передается имя стиля и присваиваемое значение:

```
myContainer.setStyle( "backgroundColor", 0xFFFFFF );
```

Как говорилось ранее, стили компонентов могут определяться во время компиляции с использованием локальных или внешних таблиц стилей.

Метод `setStyle` **позволяет изменить значения этих свойств после компиляции.**

В следующем примере метод `setStyle` **используется для задания цветовых свойств компонентов в ответ на событие** `click` **экземпляра** `Button`**:**

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

        VBox {
            backgroundColor: #CCCCCC;
            verticalAlign: 'middle';
            horizontalAlign: 'center';
        }
        .header {
            font-family: 'Arial';
            font-size: 15px;
            font-weight: 'bold';
            color: #FFFFFF;
        }
    </mx:Style>
    <mx:Script>
        <![CDATA[
           // Цветовым свойствам задаются случайные значения
            private function jumbleColors():void
            {
                holder.setStyle( "backgroundColor",
                getRandomColor() );
                labelField.setStyle( "color",
                getRandomColor() );
            }

            private function getRandomColor():uint
            {
                return randomValue() << 16 ^ randomValue()
                << 8 ^ randomValue();
            }

            private function randomValue():Number
            {
                return (Math.random() * 512) - 255;
            }
        ]]>
    </mx:Script>
    <mx:VBox id="holder" width="200" height="200">
        <mx:Label id="labelField" text="Hello Flex!"
        styleName="header" />
        <mx:Button label="jumble colors" click="jumbleColors();" />
    </mx:VBox>
</mx:Application>
```

После создания экземпляров этих компонентов им назначаются соответствующие стили: VBox оформляется с использованием селектора типа, а Label назначает стиль .header через атрибут styleName. Обработчик события click кнопки задает новые значения цветовым свойствам экземпляров VBox и Label. Хотя этот пример демонстрирует переопределение стилевых свойств, объявленных локально по отношению к приложению, стили во внешних файлах CSS также могут переопределяться во время выполнения с использованием метода setStyle.

См. также

Рецепты 9.3 и 9.4.

9.6. Загрузка CSS во время выполнения

Задача

Требуется свести к минимуму размер файла SWF; для этого внешние файлы CSS загружаются во время выполнения (вместо встраивания стилей во время компиляции).

Решение

Воспользуйтесь программой командной строки mxmlc, входящей в Flex 3 SDK, для упаковки файла CSS. Загрузите внешний файл CSS во время выполнения при помощи класса mx.styles.StyleManager.

Обсуждение

Загрузка стилей во время выполнения позволяет вносить изменения в определения стилей без перекомпиляции приложения. Загрузка стилевого файла SWF во время выполнения осуществляется методом loadStyleDeclarations класса StyleManager.

Вместо файла CSS класс StyleManager загружает файл SWF со стилевой информацией. Чтобы создать стилевой файл SWF, откомпилируйте файл CSS в формат SWF программой mxmlc из Flex SDK. Для начала создайте файл с расширением .css и включите в него объявления следующего вида:

```
VBox {
    backgroundColor: #CCCCCC;
    verticalAlign: 'middle';
    horizontalAlign: 'center';}
.header {
    font-family: 'Arial';
    font-size: 15px;
    font-weight: 'bold';
    color: #FFFFFF;
}
```

Сохраните файл под именем MyStyles.css, откройте окно командной строки. Проследите за тем, чтобы путь к каталогу /bin установки Flex был включен в системную переменную PATH. Введите следующую команду в приглашении командной строки и нажмите Enter:

```
> mxmlc MyStyles.css
```

Команда создает файл SWF с таким же именем, как у переданного файла CSS. В приведенном примере файл MyStyles.swf создается в текущем каталоге на момент запуска компилятора.

Чтобы загрузить стилевой файл SWF и применить его к компонентам с созданными экземплярами, вызовите метод `loadStyleDeclarations` класса `StyleManager` с аргументом `update=true`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="appComplete();">
    <mx:Script>
        <![CDATA[

            // Загрузка внешнего стилевого файла SWF
            // и принудительная перерисовка.
            private function appComplete():void
            {
                StyleManager.loadStyleDeclarations(
                "assets/styles/MyStyles.swf", true );
            }
        ]]>
    </mx:Script>
    <mx:VBox id="holder" width="200" height="200">
        <mx:Label id="labelField" text="Hello Flex!" styleName="header" />
    </mx:VBox>
</mx:Application>
```

Объявления из файла CSS применяются к компонентам при создании приложения и успешной загрузке откомпилированного стилевого файла SWF. При желании можно организовать отслеживание прогресса, успешного или ошибочного результата загрузки стилевого файла SWF; для этого определяются слушатели событий для экземпляра `IEventDispatcher`, возвращаемого методом `loadStyleDeclarations`:

```
private function appComplete():void
{
    // Прослушивание события complete для загрузки
    // внешнего стилевого файла SWF
    var dispatcher:IEventDispatcher =
    StyleManager.loadStyleDeclarations(
    "assets/styles/MyStyles.swf", true );
        dispatcher.addEventListener( StyleEvent.COMPLETE,
        styleCompleteHandler );
}
```

```
private function styleCompleteHandler( evt:StyleEvent ):void
{
    trace( "Styles Loaded!" );
}
```

Если экземпляр IEventDispatcher, возвращаемый методом loadStyleDeclarations, будет использоваться для отслеживания хода загрузки стилевого файла SWF, вы можете задать аргумент update=false и на программном уровне назначать стилевые свойства компонентам в обработчиках событий так, как сочтете нужным.

Если внешний файл CSS назначается на стадии компиляции с использованием атрибута source тега <mx:Style>, компилятор проверяет, существует ли указанный файл. При загрузке стилевого файла SWF на стадии выполнения с использованием StyleManager такая проверка не выполняется. Это дает возможность создать файл в любой момент работы приложения – как до, так и после его развертывания.

9.7. Объявление стилей во время выполнения

Задача

Требуется объявить и настроить стили компонентов Flex во время выполнения на уровне кода ActionScript.

Решение

Создайте объекты mx.styles.CSSStyleDeclaration; свяжите их с именем селектора, которое должно быть сохранено в классе mx.styles.StyleManager.

Обсуждение

Объект CSSStyleDeclaration содержит стилевые свойства и значения, которые могут задаваться и настраиваться во время выполнения программы. Когда вы определяете правило CSS во внешнем файле или локально с использованием тега <mx:Style>, Flex автоматически создает объект CSSStyleDeclaration для каждого селектора, объявленного во время компиляции. Чтобы создать объявление стиля во время выполнения, используйте метод StyleManager.setStyleDeclaration, в аргументах которого передаются строка с идентификатором селектора и объект CSSStyleDeclaration. Для обращения к объектам CSSStyleDeclaration, созданным во время выполнения или во время компиляции, используется метод StyleManager.getStyleDeclaration.

Связывание имен селекторов с объектом CSSStyleDeclaration, созданным во время выполнения, происходит по тем же правилам, что и для встроенных объявлений стилей. Стилевые правила для каждого экземпляра класса применяются с использованием селектора типа, который является строковым представлением имени класса. Пример:

```
var vboxStyle:CSSStyleDeclaration = new CSSStyleDeclaration();
vboxStyle.setStyle( "backgroundColor", 0xFF0000 );
StyleManager.setStyleDeclaration( "VBox", vboxStyle, false );
```

Объявления стилей также могут связываться с экземплярами при помощи селекторов классов. Селектор класса начинается с символа «точка» и применяется ко всем компонентам, у которых свойству `styleName` задано соответствующее значение:

```
var redBoxStyle:CSSStyleDeclaration = new CSSStyleDeclaration();
redBoxStyle.setStyle( "backgroundColor", 0xFF0000 );
StyleManager.setStyleDeclaration( ".redBox", redBoxStyle, false );
```

Хотя назначение объявлений `StyleManager` во время выполнения происходит по тем же правилам, как при преобразовании CSS во время компиляции, стилевые свойства не могут создаваться методом `setStyle` экземпляра `CSSStyleDeclaration` с использованием дефисов (например, `font-family`), разрешенных во внешних файлах или в тегах `<mx:Style>`.

Назначение стилей во время выполнения с использованием `StyleManager` приводит к замещению всех объявлений, созданных ранее и сохраненных под тем же именем селектора.

В следующем примере объекты `CSSStyleDeclaration` создаются при инициализации приложения, после чего к дочерним компонентам применяются обновления стилей:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    initialize="init();">

    <mx:Script>
        <![CDATA[
            // Создание и сохранение
            // новых объявлений стилей.
            private function init():void
            {
                StyleManager.registerColorName(
                "cookbookBlue", 0x339966 );
                var headerStyleDeclaration:
                CSSStyleDeclaration =
                    new CSSStyleDeclaration();
                headerStyleDeclaration.setStyle(
                    "fontWeight", "bold" );
                headerStyleDeclaration.setStyle( "fontSize", 15 );
                headerStyleDeclaration.setStyle( "color",
                    StyleManager.getColorName(
                    "cookbookBlue" ) );
                var msgStyleDeclaration:CSSStyleDeclaration =
                    new CSSStyleDeclaration();
```

```
                msgStyleDeclaration.setStyle( "fontSize", 12 );
                msgStyleDeclaration.setStyle( "color",
                    StyleManager.getColorName( "haloSilver" ) );
                StyleManager.setStyleDeclaration( ".header",
                    headerStyleDeclaration, false );
                StyleManager.setStyleDeclaration( ".message",
                    msgStyleDeclaration, true );
            }
            // Уничтожение ранее созданных стилей
            private function clickHandler():void
            {
                StyleManager.clearStyleDeclaration(
                ".header", false );
                StyleManager.clearStyleDeclaration(
                ".message", true );
            }
        ]]>
    </mx:Script>
    <mx:Label text="I'm a header styled through the
    StyleManager"
        styleName="header"
        />
    <mx:Text text="I'm a message styled through the
    StyleManager"
        styleName="message"
        />
    <mx:Button label="clear styles" click="clickHandler();" />
</mx:Application>
```

В этом примере в StyleManager объявляются и сохраняются два селектора класса. Стиль применяется к соответствующим компонентам через встроенное свойство styleName. Последний параметр метода setStyleDeclaration класса StyleManager содержит флаг обновления стилей для ранее созданных компонентов. Назначение новых объявлений стилей требуют весьма существенных затрат вычислительных ресурсов. Как следствие, вызовы обновления экрана при добавлении стилей в StyleManager обычно откладываются до объявления последнего стиля.

При назначении стиля компоненту (например, через свойство styleName) происходит замена стиля по умолчанию для этого компонента. В приведенном примере при уничтожении стилей компонентов в методе clickHandler компоненты возвращаются к стилю по умолчанию.

Удаление стилей производится методом StyleManager.clearStyleDeclaration. Как и StyleManager.setStyleDeclaration, этот метод получает параметр с флагом немедленного обновления стилей приложения. В рассмотренном примере обновление изображения после очистки стилей классов выполняется в завершающем вызове метода StyleManager.clearStyleDeclaration.

Класс StyleManager также позволяет связать цветовое значение со строковым ключом; для решения этой задачи используется метод registerColor-

Name. Метод StyleManager.getColorName предоставляет доступ к цветовым значениям, созданным во время выполнения – а также определенным в стилях по умолчанию Flex Framework. В приведенном примере данная возможность продемонстрирована назначением свойству color класса .message цвета с обозначением haloSilver.

9.8. Создание пользовательских стилевых свойств у компонентов

Задача

Требуется создать и опубликовать пользовательские стилевые свойства, изначально отсутствующие в компоненте.

Решение

Добавьте в компонент стилевые метаданные. Для получения значений свойств следует использовать метод getStyle.

Обсуждение

Все компоненты Flex Framework обладают стандартными стилевыми свойствами, но вы также можете объявлять дополнительные стилевые свойства для пользовательских компонентов при помощи тега метаданных [Style]. Включение определений стилей в тег <mx:Metadata> позволяет задавать значения свойств в MXML при объявлении компонента. Значения стилевых свойств пользовательских компонентов также могут задаваться при помощи CSS и метода setStyle. Все эти способы приводят к принудительному обновлению изображения, как и при изменении стандартных стилевых свойств.

Включение дополнительных стилевых свойств в пользовательские компоненты позволяет вам указать, какие свойства изменяют визуальное оформление компонента. В следующем примере представлен пользовательский компонент, который является субклассом mx.containers.Box. В теге <mx:Metadata> этого компонента перечислены пользовательские стили, используемые для настройки его внешнего вида:

```
<mx:Box
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%"
    horizontalAlign="center" verticalAlign="middle"
    creationComplete="creationHandler();">
    <mx:Metadata>
        [Style(name="teeterAngle", type="Number")]
        [Style(name="activeColor", type="uint", format="Color")]
        [Style(name="idleColor", type="uint", format="Color")]
    </mx:Metadata>
    <mx:Script>
        <![CDATA[
```

```
            import mx.effects.easing.Bounce;
            [Embed("assets/fonts/verdana.TTF",
                fontName="MyFont")]
            public var verdana_font:Class;
            private function creationHandler():void
            {
                playAnim();
            }
            private function effectStartHandler():void
            {
                holder.setStyle( "backgroundColor",
                    getStyle( "activeColor" ) );
                holder.enabled = false;
                field.text = "Wheeeee!";
            }
            private function effectEndHandler():void
            {
                holder.setStyle( "backgroundColor",
                    getStyle( "idleColor" ) );
                holder.enabled = true;
                field.text = "Again! Click Me";
            }
            private function playAnim():void
            {
                rotater.play( [this], true );
            }
        ]]>
    </mx:Script>
    <mx:Rotate id="rotater"
        effectStart="effectStartHandler();"
        effectEnd="effectEndHandler();"
        originX="{width}" originY="{height}"
        angleFrom="0" angleTo="{getStyle('teeterAngle')}"
        easingFunction="{Bounce.easeIn}" duration="500"
        />
    <mx:VBox id="holder"
        width="60%" height="60%"
        horizontalAlign="center"
        verticalAlign="middle"
        click="playAnim();">
        <mx:Text id="field" width="100%"
            selectable="false"
            fontFamily="MyFont"
            />
    </mx:VBox>
</mx:Box>
```

Начальный угол анимации `Rotate`**, объявленной в этом примере, задается относительно стилевого свойства** `teeterAngle`**, а цвет фона потомка** `VBox` **обновляется независимо от состоянии анимации. Допустимые пользовательские стилевые свойства этого компонента перечислены в теге** `<mx:Metadata>`**, а для обращения к соответствующим значениям исполь-**

зуется метод getStyle. Объявление пользовательских стилевых свойств в теге <mx:Metadata> позволяет определять их значения во встроенном формате при объявлении пользовательского компонента.

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cookbook="oreilly.cookbook.*"
    layout="vertical">

    <cookbook:ChompBox
        width="200" height="200"
        backgroundColor="#DDDDDD"
        textAlign="center"
        teeterAngle="45"
        activeColor="#FFDD33" idleColor="#FFFFFF"
        />

</mx:Application>
```

Несмотря на то что в компонент можно добавлять пользовательские стилевые свойства, стандартные стилевые свойства остаются доступными для настройки, как видно из задания стиля textAlign в объявлении пользовательского компонента. Экземпляр ChompBox не обрабатывает значение свойства textAlign, позволяя базовому классу mx.containers.Box управлять его применением в дочерних компонентах через наследование.

9.9. Использование нескольких тем оформления в одном приложении

Материал предоставлен Андреем Кристианом (Andrei Cristian).

Задача

Требуется применить разные темы оформления к разным компонентам одного приложения.

Решение

Используйте свойство themeColor контейнера для назначения цветовых значений.

Обсуждение

Благодаря свойству themeColor компонента <mx:Canvas> в оформлении компонентов приложения можно использовать несколько разных тем оформления. Применение цветов из тем влияет на изменение внешнего вида компонента при наведении мыши, выделении и других визуальных операциях.

В примере этого рецепта в контейнеры <mx:Canvas> включатся дочерние компоненты для демонстрации трех тем оформления. По умолчанию в глобальном определении стиля из файла defaults.css Flex Framework приложениям Flex назначается цвет темы haloBlue, однако существуют и другие темы оформления, причем в одном приложении можно использовать любое количество тем.

Начните с создания провайдера данных dp, включите для него привязку. Затем объявите три экземпляра Canvas с компонентами, у которых будет устанавливаться атрибут themeColor. Для дочерних компонентов mx.controls.ComboBox и mx.controls.DataGrid каждого экземпляра Canvas используется общий провайдер данных dp. Программный код выглядит так:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="636"
    height="241">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var dp:Array =
            [
                        {label:'Carole King', Album:'Tapestry', data:1},
                        {label:'Paul Simon', data:2},
                        {label:'Original Cast', data:3},
                        {label:'The Beatles', data:4}
            ];

        ]]>
    </mx:Script>

    <mx:Label x="10" y="7" text="Standard Theme"
        fontWeight="bold" color="#ffffff"/>
    <mx:Label x="218" y="7" text="Green Theme"
        fontWeight="bold" color="#ffffff"/>
    <mx:Label x="426" y="7" text="Silver Theme"
        fontWeight="bold" color="#ffffff"/>
    <mx:Canvas x="10" width="200" height="200"
        verticalCenter="10.5">
        <mx:ComboBox x="10" y="10" dataProvider="{dp}"
            width="180"></mx:ComboBox>
        <mx:DataGrid x="10" y="40" width="180" height="120"
            dataProvider="{dp}">
            <mx:columns>
                <mx:DataGridColumn headerText="Id"
                    dataField="data" width="30" />
                <mx:DataGridColumn headerText="Artist"
                    dataField="label"  />
```

```
            </mx:columns>
        </mx:DataGrid>
        <mx:Button x="10" y="168" label="Button"/>
    </mx:Canvas>
    <mx:Canvas x="426" width="200" height="200"
        themeColor="haloSilver"
        verticalCenter="10.5" backgroundColor="#ffffff"
        cornerRadius="8" borderStyle="solid">
        <mx:ComboBox x="10" y="10" dataProvider="{dp}"
            width="180"></mx:ComboBox>
        <mx:DataGrid x="10" y="40" width="180" height="120"
            dataProvider="{dp}">
            <mx:columns>
                <mx:DataGridColumn headerText="Id"
                    dataField="data" width="30" />
                <mx:DataGridColumn headerText="Artist"
                    dataField="label"  />
            </mx:columns>

        </mx:DataGrid>
        <mx:Button x="10" y="166" label="Button"/>

    </mx:Canvas>
    <mx:Canvas x="218" width="200" height="200"
        themeColor="haloGreen" verticalCenter="10.5"
        backgroundColor="#ffffff" cornerRadius="8"
        borderStyle="solid" alpha="0.5">
        <mx:ComboBox x="10" y="10" dataProvider="{dp}"
            width="180"></mx:ComboBox>
        <mx:DataGrid x="10" y="40" width="180" height="120"
            dataProvider="{dp}">
            <mx:columns>
                <mx:DataGridColumn headerText="Id"
                    dataField="data" width="30" />
                <mx:DataGridColumn headerText="Artist"
                    dataField="label"  />
            </mx:columns>
        </mx:DataGrid>
        <mx:Button x="10" y="166" label="Button"/>
    </mx:Canvas>

</mx:Application>
```

Когда вы задаете значение свойства `themeColor` **для экземпляра контейнера, все дочерние компоненты в списке отображения этого контейнера оформляются указанным цветом темы посредством наследования. При этом можно использовать строковые ключи цветовых значений, доступные в Flex Framework (например,** `haloOrange`**), но также допускается использование любых корректных цветовых значений в шестнадцатеричном формате или пользовательских цветовых ключей, созданных методом** `StyleManager.registerColorName` **во время выполнения.**

9.10. Компиляция темы в файл SWC

Задача

Требуется упаковать стилевые файлы в файл SWC для последующей компиляции в приложении.

Решение

Создайте файл SWC средствами командной строки. Откомпилируйте приложение, используя параметр `theme` компилятора mxmlc.

Обсуждение

Файл SWC (Shockwave Component) представляет собой архив в формате PKZIP. Файлы SWC позволяют разработчикам передавать друг другу один архив вместо множества отдельных файлов. Однако формат SWC используется не только для создания и упаковки файлов MXML и ActionScript; в нем также можно упаковать файлы стилей. Для создания файлов SWC используется утилита compc из подкаталога /bin установочного каталога Flex SDK.

Предполагается, что путь к каталогу /bin установки Flex включен в системную переменную PATH. В следующем примере файл SWC с именем MyTheme.swc строится программой compc:

```
> compc -include-file MyStyles.css C:/mystyles/MyStyles.css -o MyTheme.swc
```

При использовании параметра `include-file` задаются два аргумента: имея файла, которое должно использоваться для обращения к стилевому ресурсу, и местонахождение ресурса в файловой системе. Допускается передача нескольких параметров `include-file` для упаковки всех стилевых ресурсов, необходимых для построения темы приложения.

Тема может назначаться во время компиляции приложения при помощи параметра `theme` компилятора mxmxc:

```
> mxmlc MyApplication.mxml -theme MyTheme.swc
```

Чтобы вы лучше поняли, как сгенерировать файл SWC с темой и откомпилировать файл в приложение, мы попробуем упаковать одну из дополнительных тем, входящих в поставку Flex SDK.

Откройте в режиме командной строки подкаталог темы Smoke в установочном каталоге SDK: <*каталог flex sdk*>/frameworks/themes/Smoke. Введите следующую команду (каталог /bin установки Flex SDK должен быть включен в системную переменную PATH):

```
> compc -include-file Smoke.css Smoke.css -include-file smoke_bg.jpg
smoke_bg.jpg-o SmokeTheme.swc
```

Команда создает файл `SmokeTheme.swc` в этом каталоге. Переместите файл SWC в каталог нового проекта и введите следующую разметку:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:cookbook="oreilly.cookbook.*"
    layout="vertical">
    <mx:Script>
        <![CDATA[

            [Bindable] private var dp:Array =
                [{label:"Josh Noble", data:0},
                {label:"Abey George", data:1},
                {label:"Todd Anderson", data:2}];

            private function clickHandler():void
            {
                messageField.text = "You chose: " +
                nameCB.selectedLabel;
            }
        ]]>
    </mx:Script>
    <mx:Panel title="Pick One:" width="50%" height="50%"
            paddingLeft="10" paddingTop="10">
        <mx:VBox width="100%">
            <mx:HBox width="100%">
                <mx:Text text="Choose:" styleName="windowStyles" />
                <mx:ComboBox id="nameCB" dataProvider="{dp}" />
                <mx:Button label="select" click="clickHandler();" />
            </mx:HBox>
            <mx:Text id="messageField" styleName="windowStyles" />
        </mx:VBox>
    </mx:Panel>
</mx:Application>
```

Сохраните файл как приложение MXML. Обратите внимание: в файле MXML нет ни локальных объявлений стилей, ни импорта внешних таблиц стилей. Однако свойству `styleName` каждого из двух компонентов `Text` задается значение `windowStyles`. Селектор класса `.windowStyles` объявляется в таблице стилей Smoke.css, ранее упакованной в файл SmokeTheme.swc.

При компиляции приложения с параметром `theme`, аргумент которого ссылается на **SmokeTheme.swc, стили применяются не только к экземплярам** `Text`, но и ко всем компонентам приложения.

Откройте командную строку, перейдите в каталог проекта с файлом MXML приложения и созданным ранее файлом SWC и введите следующую команду:

```
> mxmlc MyApplication.mxml -theme SmokeTheme.swc
```

Команда генерирует файл SWF приложения с встроенной темой из файла SWC. Откройте сгенерированный файл SWF из каталога проекта; вы увидите, что в оформлении приложения используется стиль Smoke из Flex SDK.

В файлах SWC могут упаковываться не только файлы CSS и графические файлы, но и файлы со шрифтами и классами скинов. Последние возможности более подробно рассматриваются в последующих рецептах.

См. также

Рецепты 9.2 и 9.6.

9.11. Встроенные шрифты

Задача

Требуется встроить шрифты в приложение, чтобы оформление текста оставалось единым на всех компьютерах независимо от того, какие шрифты установлены в системе пользователя.

Решение

Используйте тег `[Embed]` в коде ActionScript или директиву `@font-face` в CSS.

Обсуждение

Наличие в приложении встроенных шрифтов гарантирует, что стилевое оформление текста останется неизменным независимо от состава системных шрифтов на компьютере пользователя. Встраивание шрифтов осуществляется в коде ActionScript или CSS.

Следующий пример показывает, как встроить шрифты в приложение из кода ActionScript:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Script>
        <![CDATA[

            [Embed(source="assets/fonts/verdana.TTF",
            fontName="MyVerdana")]
            private var _verdana:Class;

        ]]>
    </mx:Script>
    <mx:Label text="i have no custom style." />
    <mx:Label text="i have verdana style!" fontFamily="MyVerdana" />
</mx:Application>
```

При использовании тега `[Embed]` для шрифтов можно указать атрибут `fontName`, используемый для назначения стилевого свойства `fontFamily` при создании экземпляра компонента. В нашем примере шрифт встраивается в приложение и для него объявляется соответствующая переменная типа `Class`.

Представленный фрагмент встраивает шрифт с простым (нормальным) начертанием. Этот шрифт может быть применен к любому компоненту и использован для вывода на экран. Но вы можете заметить, что некоторые компоненты, которые во внутренней реализации отображают дочерние компоненты с заданным стилем (как, например, `mx.controls.Button` со своей полужирной внутренней надписью), не выводят текст, если в приложение не встроено соответствующее начертание шрифта. Нельзя применить полужирное начертание к компоненту, если вы не встроите соответствующую гарнитуру и не установите атрибут `fontWeight`. Пример:

```
[Embed(source="assets/fonts/verdanab.TTF", fontName="MyVerdana"
fontWeight="bold")]

private var _verdanaBold:Class;
...
<mx:Button label="I have style" fontFamily="MyVerdana" />
```

Многие шрифты имеют четыре основных начертания: простое, полужирное, курсивное и полужирное курсивное. Все варианты начертаний могут встраиваться в приложения Flex, но должны объявляться отдельно с правильными атрибутами.

Встраивание шрифтов в ActionScript работает, однако управление встраиванием нескольких шрифтов с соответствующими переменными быстро становится громоздким и неудобным.

Шрифты также можно встраивать из CSS, как локально в файлах MXML, так и во внешнем варианте, с использованием директивы `@font-face`:

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

    <mx:Style>
        @font-face {
            src: url('assets/fonts/verdana.TTF');
            font-family: MyVerdana;
        }
        @font-face {
            src: url('assets/fonts/verdanai.TTF');
            font-family: MyVerdana;
            font-style: italic;
        }
        @font-face {
            src: local('Verdana');
            font-family: MyVerdana;
            font-weight: bold;
        }

        Application {
            font-family: MyVerdana;
```

```
        }
        .italicStyle {
            font-style: italic;
        }
    </mx:Style>

    <mx:Text text="i have MyVerdana fontFamily styling." />
    <mx:Label text="i am in italics with MyVerdana styling."
        styleName="italicStyle" />
    <mx:Button label="i am a button. i am bold" />
</mx:Application>
```

В приложение встраиваются три варианта начертания шрифта Verdana и для него назначается шрифт по умолчанию. Благодаря селектору типа для `Application` весь текст в приложении использует встроенный шрифт. Так как у экземпляра `Button` по умолчанию стилевое свойство `fontWeight` задается равным `bold`, нам в этом примере не приходится объявлять стиль для кнопки – полужирный шрифт встроен в приложение, а значение `fontFamily` по умолчанию задано в селекторе `Application`.

Для встраивания шрифта также можно задать его имя вместо местонахождения; для этого в правиле `@font-face` используется функция `local` вместо функции `url`:

```
@font-face {
    src: local('Verdana');
    font-family: MyVerdana;
    font-weight: bold;
}
```

При вызове функции `local` указывается не имя гарнитуры (например, `verdanab` в нашем примере), а имя семейства шрифтов. Полужирные, курсивные и полужирные курсивные начертания встраиваются посредством включения соответствующего свойства в объявление.

Внедрение шрифтов в приложение увеличивает размер файла SWF. В приведенных примерах встраивались все доступные наборы символов шрифтов. Для уменьшения размера файла можно объявить в правиле `@font-face` встраиваемые диапазоны символов:

```
@font-face {
     src: local('Verdana');
     font-family: MyVerdana;
     unicodeRange: U+0041-U+005A, U+0061-U+007A, U+002E;
}
```

В свойстве `unicodeRange` могут задаваться как диапазоны, так и отдельные символы. Подмножества в объявлении разделяются запятыми. Диапазоны задаются граничными символами, разделенными дефисом (-). В приведенном примере встраиваются символы верхнего и нижнего регистров стандартного латинского алфавита (от A до Z и от a до z) и символ «точка» (.).

9.12. Встраивание шрифтов из файла SWF

Задача

Требуется встроить шрифты из файла SWF для использования в приложении.

Решение

Создайте файл SWF со встроенными шрифтами. Объявите встраиваемые гарнитуры директивой `@font-face`.

Обсуждение

В приложение можно встроить несколько гарнитур из одного файла SWF со встроенными шрифтами. Главным преимуществом использования встроенных шрифтов из SWF, в отличие от шрифтов из шрифтового файла, является портируемость. Вы можете встроить шрифты, доступные в вашей системе, в файл SWF и передать его другому разработчику. Ему не придется беспокоиться о том, чтобы эти шрифты были установлены на его компьютере и были активны во время компиляции.

Чтобы встроить шрифты в файл SWF, создайте новый файл FLA в Flash IDE и разместите на сцене динамические текстовые поля для выбора шрифта и диапазонов символов (рис. 9.1).

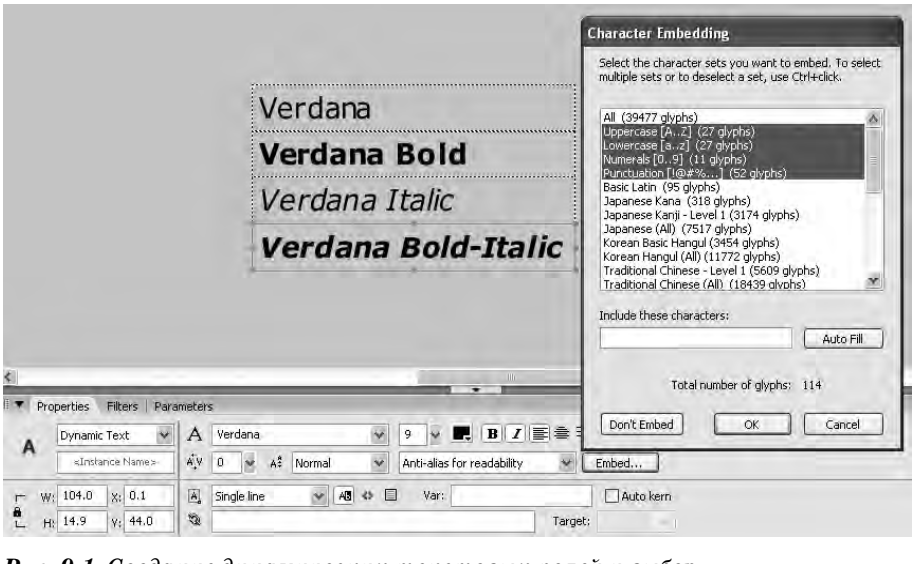

Рис. 9.1. *Создание динамических текстовых полей и выбор диапазонов символов*

В этом примере динамические текстовые поля добавляются с выбранным шрифтом Verdana и свойствами начертания. Как видно из рисунка,

полужирно-курсивное начертание применяется на панели свойств. При щелчке на кнопке Embed пользователю предлагается задать диапазоны символов, чтобы уменьшить размер файла. Сохраните файл FLA под тем именем, которое вам покажется подходящим для этого проекта (у меня файл примера был назван Verdana.fla), и опубликуйте его.

Объявление встроенных шрифтов из файла SWF напоминает работу со шрифтовыми файлами. Вместо того чтобы задавать местонахождение шрифтового файла для функции `url` или имя шрифта для функции `local`, мы используем директиву `url` для встраивания шрифтов из SWF:

```
@font-face {
    src: url("styles/fonts/Verdana.swf");
    font-family: Verdana;
}
```

Этот фрагмент встраивает простое (обычное) начертание шрифта Verdana из верхнего динамического текстового поля на рис. 9.1. Для каждого начертания, встраиваемого в файл SWF, необходимо объявить соответствующие значения свойств. Так, при встраивании шрифта, выбранного на рис. 9.1, свойству `fontWeight` задается значение `bold`, а свойству `fontStyle` – значение `italic`:

```
@font-face {
    src: url("styles/fonts/Verdana.swf");
    font-family: Verdana;
    font-weight: bold;
    font-style: italic;
}
```

Встраиваемые шрифты представляются субклассом для класса `flash.text.Font` из ActionScript API. Для получения списка шрифтов устройств и встроенных шрифтов используется статический метод `enumerateFonts` класса `Font`. Аргумент `enumerateFonts` содержит флаг включения устройств шрифтов в возвращаемый массив. В следующем примере таблица данных заполняется встроенными шрифтами из шрифтового файла SWF, объявленного в теге `<mx:Style>`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" creationComplete="creationHandler();">

    <mx:Style>
        @font-face {
            src: url("assets/fonts/Verdana.swf");
            font-family: Verdana;
        }
        @font-face {
            src: url("assets/fonts/Verdana.swf");
            font-family: Verdana;
            font-weight: bold;
        }
```

```
    @font-face {
        src: url("assets/fonts/Verdana.swf");
        font-family: Verdana;
        font-style: italic;
    }
    @font-face {
        src: url("assets/fonts/Verdana.swf");
        font-family: Verdana;
        font-weight: bold;
        font-style: italic;
    }
    Label {
        font-family: Verdana;
        font-size: 15px;
    }
    .verdanaBoldStyle {
        font-weight: bold;
    }
    .verdanaItalicStyle {
        font-style: italic;
    }
    .verdanaBoldItalicStyle {
        font-weight: bold;
        font-style: italic;
    }
</mx:Style>

<mx:Script>
    <![CDATA[

        [Bindable] private var fontArray:Array;
        [Bindable] private var selectedFontStyle:String;
        [Bindable] private var selectedFontWeight:String;
        private function creationHandler():void
        {
            var embeddedFonts:Array =
            Font.enumerateFonts();
            embeddedFonts.sortOn( "fontStyle",
                Array.CASEINSENSITIVE );
            fontArray = embeddedFonts;
        }
        private function changeHandler():void
        {
            var italic:RegExp = /italic/i;
            var bold:RegExp = /bold/i;
            selectedFontStyle =
                ( String(
                fontGrid.selectedItem.fontStyle
                ).match(italic) )
                ? 'italic'
                : 'normal';
```

```
                selectedFontWeight = ( String(
                    fontGrid.selectedItem.fontStyle
                    ).match(bold) )
                    ? 'bold'
                    : 'normal';
            }

        ]]>
    </mx:Script>
    <mx:Label text="Select a font form the grid below:" />
    <mx:DataGrid id="fontGrid" dataProvider="{fontArray}"
        change="changeHandler();">
        <mx:columns>
            <mx:DataGridColumn headerText="Font Name"
                dataField="fontName"width="150" />
            <mx:DataGridColumn headerText="Font Style"
                dataField="fontStyle" />
            <mx:DataGridColumn headerText="Font Type"
                dataField="fontType" />
        </mx:columns>
    </mx:DataGrid>

    <mx:Label width="100%" textAlign="center"
        text="{fontGrid.selectedItem.fontName + ' : ' +
        fontGrid.selectedItem.fontStyle}"
        fontFamily="{fontGrid.selectedItem.fontName}"
        fontStyle="{selectedFontStyle}"
        fontWeight="{selectedFontWeight}"
        />
</mx:Application>
```

См. также

Рецепт 9.10.

9.13. Скины со встроенными изображениями

Материал предоставлен Кристофером Шульцем (Kristofer Schultz).

Задача

Требуется использовать пользовательские изображения в оформлении визуальных элементов компонента.

Решение

Примените пользовательскую графику в формате JPEG, GIF или PNG при помощи стилевых свойств компонента. Эти атрибуты могут задаваться как непосредственно для экземпляра компонента в коде MXML, так и в контексте определения стиля CSS.

Обсуждение

По умолчанию встроенные темы Flex назначают компонентам программные классы скинов. Вы можете создать пользовательские программные скины или назначить графические элементы (в том числе и скины) для модификации визуального оформления компонентов. Компоненты, оформляемые с применением скинов, обычно имеют набор *состояний* скинов (также называемых фазами), отображаемых во время взаимодействия с пользователем. Соответственно при создании пользовательских графических скинов желательно учитывать различные состояния интерактивности компонента.

В следующем примере пользовательские изображения применяются к различным состояниям компонента `Button`. А именно, пример задает свойства `upSkin`, `overSkin`, `downSkin` и `disabledSkin` во встроенном формате в теге `Button`:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="#FFFFFF">

    <mx:Button label=""
        upSkin="@Embed('assets/images/text_button_up.png')"
        overSkin="@Embed('assets/images/
            text_button_over.png')"
        downSkin="@Embed('assets/images/
            text_button_down.png')"
        disabledSkin="@Embed('assets/images/
            text_button_disabled.png')"

    />
</mx:Application>
```

Директива компилятора `@Embed` обеспечивает принудительную упаковку графических ресурсов в файл SWF приложения. Принципиально, чтобы графика была встроена именно таким способом, чтобы переход кнопки из одного состояния в другое не сопровождался задержкой.

Поскольку в примере свойству `label` задается пустая строка (`''`), текст не будет выводиться поверх графических элементов. В графических файлах этого рецепта, содержащихся в примерах кода, надписи уже включены в графические изображения.

Если вы хотите, чтобы на экземплярах `Button` выводился текст, а изображения масштабировались в соответствии с изменяющейся длиной текста, воспользуйтесь функцией scale-9 Flex Framework для включения данных сетки в скиновые атрибуты. Значения сетки масштабирования зависят от размера и дизайна графических элементов. В следующем примере атрибуты скинов задаются в определении стиля CSS; стиль назначается нескольким экземплярам `Button` через атрибут `styleName`. При использовании директивы `Embed` в CSS префикс `@` отсутствует, как и при встраивании графики «на месте».

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal" backgroundColor="#FFFFFF">

    <mx:Style>
        .customButton {
            color: #FFFFFF;
            text-roll-over-color: #FFFFBB;
            text-selected-color: #9999FF;
            disabled-color: #333333;
            up-skin: Embed(
                source="assets/images/button_up.png",
                scaleGridTop="15",
                scaleGridBottom="20",
                scaleGridLeft="15",
                scaleGridRight="28");
            over-skin: Embed(
                source="assets/images/button_over.png",
                scaleGridTop="15",
                scaleGridBottom="20",
                scaleGridLeft="15",
                scaleGridRight="28");
            down-skin: Embed(
                source="assets/images/button_down.png",
                scaleGridTop="15",
                scaleGridBottom="20",
                scaleGridLeft="15",
                scaleGridRight="28");
            disabled-skin: Embed(
                source="assets/images/button_disabled.png",
                scaleGridTop="15",
                scaleGridBottom="20",
                scaleGridLeft="15",
                scaleGridRight="28");
        }
    </mx:Style>
    <mx:Button label="This button is enabled"
        styleName="customButton"
    />

    <mx:Button label="This button is disabled"
        styleName="customButton"
        enabled="false"
    />
</mx:Application>
```

Функция scale-9 Flex Framework позволяет определить девять секций изображения, масштабируемых независимо друг от друга. Применение scale-9 для задания атрибутов сетки масштабирования особенно полезно при внедрении графики с краями, которые должны остаться четкими и неискаженными, например закругленных углов графики кнопок в приведенном примере.

9.14. Применение скинов из файла SWF

Задача

Требуется сохранить в файле SWF библиотеку графики для встраивания скинов компонентов.

Решение

Создайте файл SWF с несколькими экспортируемыми символическими именами для разных состояний компонента.

Обсуждение

Библиотеки графических скинов в файлах SWF удобно использовать для передачи графического дизайна другому разработчику. Кроме того, библиотеки дают возможность использовать векторную графику в скинах.

На панели **Properties для символического имени в Flash IDE можно выбрать** имя класса и компоновку суперкласса, используемые для ссылок на символическое имя графического объекта. Так как при работе в Flash IDE используются библиотеки классов, доступные только для проектов на базе ActionScript, вам не удастся выбрать базовые классы из Flex Framework. Это нормально – символические имена при встраивании в приложения преобразуются в свои Flex-аналоги. Например, экземпляры `MovieClip`, помеченные для экспорта, преобразуются в объекты `MovieClipAsset`, которые затем включаются в вывод приложения Flex.

Создание ресурсов, расширяющих `MovieClip`, пригодится для графики с несколькими состояниями, но при использовании статической графики достаточно выбрать в качестве базового класса символического имени `flash.display.Sprite`. Объект `Sprite` будет преобразован в объект `SpriteAsset` при встраивании в приложение Flex.

Чтобы приступить к созданию статических графических скинов в Flash IDE, откройте новый файл FLA и создайте новое символическое имя `MovieClip` из библиотеки. Импортируйте графический файл на сцену экземпляра `MovieClip` или графическими средствами Flash IDE создайте векторные скины для своего приложения. Чтобы разрешить встраивание символического имени в приложение Flex, щелкните правой кнопкой мыши на имени в библиотеке, откройте панель свойств из контекстного меню и установите флажок Export for ActionScript. В поле Base class по умолчанию помещается значение `flash.display.MovieClip`. При желании его можно оставить, а можно и заменить на `flash.display.Sprite`, так как графика считается статической или не имеет состояний, зависящих от временной шкалы.

На рис. 9.2 изображена секция Advanced панели свойств с настройкой скинов для различных состояний и элементов полосы прокрутки. Здесь можно задать свойства для встраивания (также доступные с панели

Linkage) и включить сетку scale-9 установкой флажка Enabled guides for 9-slice scaling. Если вручную настроить направляющие **scale-9, компилятор Flex** автоматически задаст значения свойств сетки масштабирования (`scaleGridLeft`, `scaleGridRight`, `scaleGridTop`, `scaleGridBottom`).

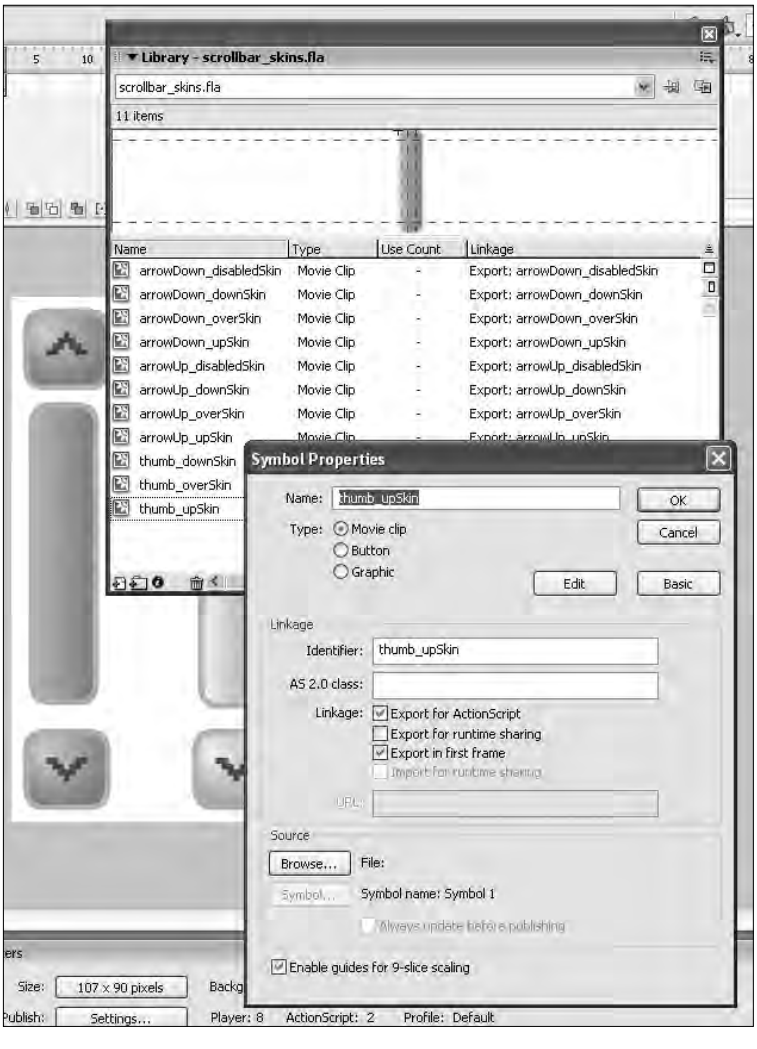

Рис. 9.2. *Создание и экспортирование векторной графики в файл SWF*

После того как скины будут созданы, каждому из них будет присвоен уникальный идентификатор компоновки, и они будут помечены для экспорта. Сохраните файл FLA под именем, подходящим для вашего проекта (в приведенном примере использовалось имя scrollbar_skins.fla), и опубликуйте ролик.

Встраивание графических символических имен из файлов SWF напоминает встраивание отдельных графических файлов. В директиве `Embed` устанавливается атрибут `source`, а свойству `symbol` задается идентификатор компоновки встраиваемого символического имени из библиотеки. Пример:

```
thumbUpSkin: Embed(source="styles/scrollbar_skins.swf", symbol="thumb_up-
Skin");
```

Также можно задать значение атрибута `symbol`, отделяя имя SWF от идентификатора компоновки знаком # в определении `source`:

```
thumbUpSkin: Embed(source="styles/scrollbar_skins.swf#thumb_upSkin");
```

В следующем примере сгенерированный файл SWF (scrollbar_skins.swf из примеров кода этой главы) используется для назначения скинов экземпляру класса `ScrollBar`. Класс `mx.controls.scrollClasses.ScrollBar` из архитектуры компонентов Flex состоит из нескольких элементов «с состояниями». Каждый элемент компонента полосы прокрутки имеет визуальное представление текущего состояния. Состояние может передавать информацию о текущем действии пользователя и доступности элемента. Например, когда пользователь наводит указатель мыши на дочерний компонент полосы прокрутки, компонент переходит в состояние «over». Назначение скинов для каждой фазы состояния каждого элемента реализуется при помощи соответствующих стилевых свойств класса `ScrollBar`.

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Style>
        ScrollBar
        {
            trackColors: #0099CC, #0099CC;
            borderColor: #99FFFF;
            thumbUpSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="thumb_upSkin");
            thumbOverSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="thumb_overSkin");
            thumbDownSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="thumb_downSkin");
            upArrowUpSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="arrowUp_upSkin");
            upArrowOverSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="arrowUp_overSkin");
            upArrowDownSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
```

```
                symbol="arrowUp_downSkin");
            upArrowDisabledSkin: Embed(source="assets/
                styles/scrollbar_skins.swf",
                symbol="arrowUp_disabledSkin");
            downArrowUpSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="arrowDown_upSkin");
            downArrowOverSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="arrowDown_overSkin");
            downArrowDownSkin: Embed(source="assets/styles/
                scrollbar_skins.swf",
                symbol="arrowDown_downSkin");
            downArrowDisabledSkin: Embed(source="assets/
                styles/scrollbar_skins.swf",
                symbol="arrowDown_disabledSkin");
        }

    </mx:Style>
    <mx:Label text="Keep typing to see the scroll!"
        color="0xFFFFFF" />
    <mx:TextArea width="200" height="150"
        text="Lorem ipsum dolor sit amet..."
        borderColor="0x0099CC" />
    <mx:HScrollBar width="200" enabled="false" />
</mx:Application>
```

Селектор типа для экземпляров `ScrollBar` объявляется локально в теге `<mx:Style>`. В объявлении определяются скины состояний для трех дочерних компонентов: бегунка и двух кнопок со стрелками.

Компонент `mx.controls.TextArea` отображает полосы прокрутки в зависимости от длины текста. Чтобы увидеть вертикальную полосу прокрутки, оформленную с использованием стилевого файла SWF, введите дополнительный текст в поле. В изображение включается экземпляр `mx.controls.HScrollBar`, который затем блокируется, чтобы показать, что скины применяются к обоим экземплярам полосы прокрутки (`VScrollBar` и `HScrollBar`), а стилевые свойства не зависят от направления.

См. также

Рецепт 9.12.

9.15. Скиновое оформление компонента на программном уровне

Задача

Требуется управлять прорисовкой визуальных элементов в компоненте без назначения графических скинов.

Решение

Создайте пользовательский класс скина, расширяющий класс `mx.skins.ProgrammaticSkin`, и переопределите защищенный метод `updateDisplayList`.

Обсуждение

Программное применение скинов, в отличие от графического, требует более глубокого понимания ActionScript. С другой стороны, разработчик получает больше возможностей для контроля над визуальным представлением компонента. Программные классы скинов представляют собой отображаемые объекты, которые используют API графического вывода для прорисовки элементов скинов и позволяют работать с другими стилевыми свойствами, недоступные в случае применения графических скинов.

Компоненты Flex делятся на два типа: контейнеры (containers) и элементы управления (controls). Контейнеры обладают граничными скинами, представляющими фоновое изображение, а у элементов управления обычно имеется набор скинов для представления состояний. Впрочем, существуют исключения: например, элемент управления `TextInput` использует граничный скин. В общем случае важно узнать визуальное строение компонента, прежде чем создавать для него пользовательский программный скин.

При создании программного скина для элемента управления следует расширять класс `mx.skins.ProgrammaticSkin`. При создании скина для контейнера тоже можно воспользоваться субклассом `ProgrammaticSkin`, но обычно субкласс создается на основе класса `Border` (обладающего свойством `borderMetrics`) или `RectangleBorder` (субкласс `Border` с поддержкой фоновых стилей). В табл. 9.1 перечислены базовые классы Flex API, используемые для создания пользовательских программных скинов.

Таблица 9.1. Базовые классы Flex API для создания пользовательских программных скинов

Класс	Использование
`mx.skins.ProgrammaticSkin`	Базовый класс для всех элементов скинов, прорисовывающих себя на программном уровне
`mx.skins.Border`	Базовый класс для рисования прямоугольных и непрямоугольных границ. Содержит свойство `borderMetrics`, является субклассом `ProgrammaticSkin`
`mx.skins.RectangularBorder`	Базовый класс для рисования прямоугольных границ. Субкласс `Border`. Поддерживает стили `backgroundImage`, `backgroundSize` и `backgroundAttachment`

При создании субклассов пользовательских скинов на основе этих базовых классов метод `updateDisplayList` класса `ProgrammaticSkin` переопределяется, и в него включаются вызовы методов графического вывода с учетом значений свойств. Пример:

```
package
{
    import mx.skins.ProgrammaticSkin;
    public class MyCustomSkin extends ProgrammaticSkin
    {
        public function MyCustomSkin() {}
        override protected function updateDisplayList(
            unscaledWidth:Number,
            unscaledHeight:Number ):void
        {
            // Получение значения свойства backgroundColor
            var backgroundColor:Number = getStyle(
                "backgroundColor" );
            // Реализация графического вывода
        }
    }
}
```

Метод `updateDisplayList` вызывается во внутренней реализации экземпляра класса каждый раз, когда возникнет необходимость в прорисовке или перерисовке элементов скина на основании изменений в свойствах. Таким образом, при переопределении этого метода можно получать значения стилевых свойств методом `getStyle` и использовать графический API для формирования изображения.

Чтобы задать размеры по умолчанию для элемента скина, переопределите доступные только для чтения свойства `measuredWidth` и `measuredHeight`:

```
package oreilly.cookbook
{
    import mx.skins.ProgrammaticSkin;
    public class MyCustomSkin extends ProgrammaticSkin
    {
        private var _measuredWidth:Number;
        private var _measuredHeight:Number;
        public function MyCustomSkin()
        {
            _measuredWidth = 120;
            _measuredHeight = 120;
        }
        // Возвращение константы ширины по умолчанию
        override public function get
            measuredWidth():Number
        {
            return _measuredWidth;
        }
```

```
        // Возвращение константы высоты по умолчанию
        override public function get measuredHeight():Number
        {
            return _measuredHeight;
        }

        override protected function updateDisplayList(
            unscaledWidth:Number, unscaledHeight:Number ):void
        {
            // Получение значения свойства backgroundColor
            var backgroundColor:Number = getStyle( "backgroundColor" );
            // Реализация графического вывода
        }
    }
}
```

Если не переопределить свойства `measuredWidth` и `measuredHeight`, доступные только для чтения, по умолчанию они задаются равными 0. В предыдущем примере компоненту с назначенным программным классом скина присваиваются размеры по умолчанию 120×120. Эти размеры могут быть переопределены при объявлении компонента в свойствах `width` и `height`.

Следующий фрагмент создает нестандартный скин обрамления, применяемый к стилевому свойству `borderSkin` контейнера:

```
package oreilly.cookbook
{
    import mx.graphics.RectangularDropShadow;
    import mx.skins.Border;

    public class CustomBorder extends Border
    {
        private var _dropShadow:RectangularDropShadow;
        private static const CNR_RAD:Number = 5;
        private static const POINT_POS:String = 'bl';
        private static const BG_COL:uint = 0x336699;
        private static const BG_ALPHA:Number = 1.0;
        public function CustomBorder()
        {
            super();
            _dropShadow = new RectangularDropShadow();
        }
        private function getCornerRadiusObj(
        rad:Number, pointPos:String ):Object
        {
            var pt:String = pointPos ? pointPos : POINT_POS;
            return {tl:rad, bl:0, tr:rad, br:rad};
        }
        override protected function updateDisplayList(
            unscaledWidth:Number,
            unscaledHeight:Number ):void
```

```
        {
            super.updateDisplayList( unscaledWidth,
                unscaledHeight );
            var cornerRadius:Number = getStyle(
                "cornerRadius" ) ?
                getStyle( "cornerRadius" ) :
                CustomBorder.CNR_RAD;
            var backgroundColor:Number = getStyle(
                "backgroundColor" ) ?
                getStyle( "backgroundColor" ) :
                CustomBorder.BG_COL;
            var backgroundAlpha:Number = getStyle(
                "backgroundAlpha" ) ?
                getStyle( "backgroundAlpha" ) :
                CustomBorder.BG_ALPHA;
            var cnrRadius:Object = getCornerRadiusObj(
                cornerRadius,
                getStyle( "pointPosition" ) );
            graphics.clear();
            drawRoundRect( 0, 0, unscaledWidth,
                unscaledHeight, cnrRadius , backgroundColor,
                backgroundAlpha );

            _dropShadow.tlRadius = cnrRadius.tl;
            _dropShadow.blRadius = cnrRadius.bl;
            _dropShadow.trRadius = cnrRadius.tr;
            _dropShadow.brRadius = cnrRadius.br;
            _dropShadow.drawShadow( graphics, 0, 0,
                unscaledWidth, unscaledHeight );
        }
    }
}
```

При каждом вызове метода `updateDisplayList` графический слой очищается и перерисовывается унаследованным методом `drawRoundRect`. При этом также применяется фильтр тени; в этом отношении программные скины превосходят графические. Эта методика открывает доступ к низкоуровневым функциям (в том числе и к фильтрам).

Программный скин может быть назначен любым способом, используемым для назначения стилевых свойств: во встроенном формате, методом `setStyle` или через CSS. В следующем примере продемонстрированы оба способа:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    initialize="initHandler();">
    <mx:Script>
        <![CDATA[
            import oreilly.cookbook.CustomBorder;
            private function initHandler():void
```

```
            {
                myVBox.setStyle( "borderSkin", CustomBorder );
            }
        ]]>
    </mx:Script>
    <mx:VBox width="100" height="50"
        borderSkin="oreilly.cookbook.CustomBorder" />
    <mx:VBox width="50" height="20"
        borderSkin="{CustomBorder}" />
    <mx:VBox id="myVBox" width="80" height="20" />
</mx:Application>
```

При назначении скина во встроенном формате или методом `setStyle` **можно импортировать класс и использовать сокращенное название или же задать скин с указанием полного имени класса. Если вы используете сокращенное имя для назначения класса скина во встроенном формате, необходимо добавить фигурные скобки (**`{}`**) для обработки импортированного класса.**

При использовании CSS необходимо задать полное имя класса, заключенное в директиву `ClassReference`**. В следующем примере скин** `CustomBorder` **назначается свойству** `borderSkin` **экземпляра** `mx.containers.VBox` **с использованием селектора типа:**

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Style>
        VBox {
            borderSkin: ClassReference(
                "oreilly.cookbook.CustomBorder");
            cornerRadius: 30;
            pointPosition: 'bl';
            backgroundColor: #999933;
            backgroundAlpha: 1;
            paddingLeft: 5;
            paddingRight: 5;
        }
    </mx:Style>
    <mx:VBox id="myBox"
        width="120" height="100"
        verticalAlign="middle" horizontalAlign="center">
        <mx:Text text="i'm a styled VBox!"
            textAlign="center" />
    </mx:VBox>
</mx:Application>
```

В переопределении метода `updateDisplayList` **скина** `CustomBorder` **другие стилевые свойства (**`cornerRadius`**,** `pointPosition` **и т. д.), объявленные в селекторе** `VBox`**, используются для нестандартной прорисовки фонового изображения экземпляра** `VBox`**.**

См. также

Рецепты 9.3 и 9.5.

9.16. Программное скиновое оформление элементов управления с состояниями

Задача

Требуется создать программный скин, обеспечивающий прорисовку различных состояний элемента управления.

Решение

Создайте субкласс на основе `mx.skins.ProgrammaticSkin` и используйте метод `updateDisplayList` для обновления изображения на основании значения свойства `name`. Назначьте пользовательский программный скин «на месте» (свойство `skin`) методом `setStyle` или в CSS.

Обсуждение

Как правило, элементы управления имеют несколько состояний или содержат дочерние компоненты, обладающие несколькими состояниями. Назначение программного скина в качестве стиля, обеспечивающего перерисовку состояний, равносильно назначению программного скина для прорисовки фона контейнера: значения стилевого свойства определяются по имени класса программного скина. Однако визуальные элементы скинов элемента управления и контейнера различаются, так как большинство элементов управления при прорисовке учитывает текущее состояние.

При создании класса программного скина для компонента с состояниями следует создать субкласс `ProgrammaticSkin` и переопределить защищенный метод `updateDisplayList`. В переопределении `updateDisplayList` можно использовать конструкцию `switch/case` для разных состояний. Значение свойства `name` для каждой фазы скина назначается внутренней реализацией оформляемого элемента управления и представляет собой строковое значение, совпадающее со значением стилевого свойства элемента.

В следующем примере создается программный скин, обновляющий изображение компонента в состояниях `upSkin`, `overSkin`, `downSkin` и `disabledSkin`:

```
package oreilly.cookbook
{
    import flash.filters.BevelFilter;
    import mx.skins.ProgrammaticSkin;
    public class CustomButtonSkin extends ProgrammaticSkin
    {
```

```
        private var _measuredWidth:Number = 100;
        private var _measuredHeight:Number = 100;
        // Переопределения measuredWidth и measuredHeight
        // задают размеры по умолчанию
        override public function get measuredWidth():Number
        {
            return _measuredWidth;
        }
        override public function get measuredHeight():Number
        {
            return _measuredHeight;
        }
        // Обновление изображения для разных фаз состояния
        override protected function updateDisplayList(
            unscaledWidth:Number, unscaledHeight:Number ):void
        {
            var backgroundAlpha:Number = 1.0;
            var bevelAngle:Number = 45;
            var backgroundColor:uint;
            // Значения свойств задаются в зависимости
            // от имени состояния
            switch( name )
            {
                case "upSkin":
                    backgroundColor = 0xEEEEEE;
                    break;
                case "overSkin":
                    backgroundColor = 0xDDDDDD;
                    break;
                case "downSkin":
                    backgroundColor = 0xDDDDDD;
                    bevelAngle = 245;
                    break;
                case "disabledSkin":
                    backgroundColor = 0xFFFFFF;
                    backgroundAlpha = 0.5;
                    break;
            }
            // Очистка изображения и перерисовка
            // со значениями, заданными в switch
            graphics.clear();
            drawRoundRect( 0, 0, unscaledWidth,
                unscaledHeight, null,
                backgroundColor, backgroundAlpha );
            // Применение фильтра bevel
            var bevel:BevelFilter = new BevelFilter( 2, bevelAngle );
            this.filters = [bevel];
        }
    }
}
```

Чтобы назначить программный скин, реагирующий на фазы состояния, следует задать имя его класса стилевому свойству `skin`. В следующем фрагменте CSS используется для задания полного имени класса `CustomButtonSkin`, упакованного в директиве `ClassReference`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Style>
        .customButton {
            skin: ClassReference(
                "oreilly.cookbook.CustomButtonSkin" );
        }
    </mx:Style>

    <mx:Button id="skinnedBtn"
        styleName="customButton"
        label="click me"
        />
    <mx:Button label="toggle enabled"
        click="{skinnedBtn.enabled = !skinnedBtn.enabled;}"
        />
</mx:Application>
```

Стилевое свойство `skin` принадлежит к числу новшеств Flex X SDK. В Flex 2 каждое состояние скина приходилось определять даже в том случае, если все значения задавались одинаковыми директивами `ClassReference`. В Flex 3 можно назначить одно свойство `skin`, которое будет использоваться для всех состояний. Впрочем, даже после определения свойства `skin` можно переопределять свойства состояний (`overSkin` и т. д.), что расширяет возможности настройки компонентов с состояниями.

См. также

Рецепт 9.14.

9.17. Создание анимированных скинов на основе файла SWF

Задача

Требуется создать кнопку с разными анимациями для каждого состояния.

Решение

Создайте файл FLA с символическим именем `MovieClip`, помеченным для экспорта, содержащим несколько меток кадров и анимаций. Пометьте класс `MovieClip` как расширяющий вашу анимацию при помощи тега метаданных `[Embed]`. Назначьте субкласс `mx.core.UIComponent` скином для

экземпляра `mx.controls.Button` и используйте унаследованное свойство `currentState` для управления текущим состоянием.

Обсуждение

Назначение анимированного скина из файла SWF напоминает назначение программного скина компоненту при его объявлении. Создание скина отличается тем, что между кадрами символического идентификатора применяются tween-анимации, в отличие от программной анимации элементов в компонентах.

Чтобы создать анимированный скин для встраивания, используйте символический идентификатор `MovieClip` из библиотеки документа FLA с определенными метками кадров, соответствующими состояниям. Этот символический идентификатор встраивается в определение класса при назначении свойств компоновки символическому имени в библиотеке FLA. После этого анимированный скин можно включить в список отображения по ссылке на класс, а для управления анимацией используются переходы к кадрам на основании фаз состояния компонента.

Чтобы приступить к созданию анимированных скинов, добавьте символический идентификатор `MovieClip` в Flash IDE; включите несколько меток кадров и анимаций между этими кадрами на временной шкале (рис. 9.3).

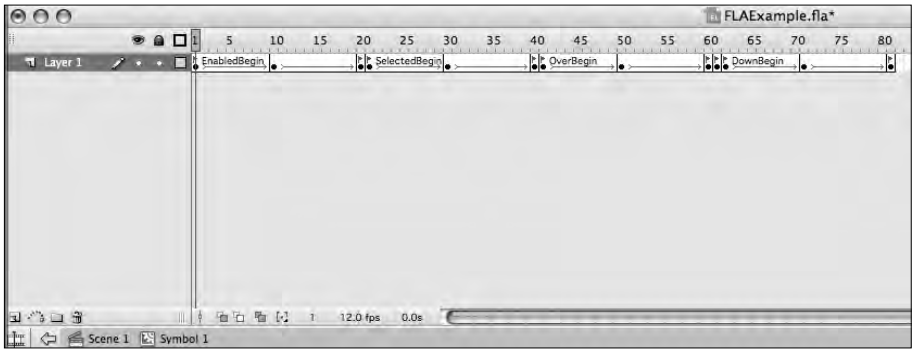

Рис. 9.3. *Создание нескольких меток кадров с промежуточными анимациями*

Ваши анимации могут быть существенно сложнее, чем анимации, представленные в этом примере. Очень важно, чтобы метки кадров были содержательными и соответствовали пяти главным состояниям кнопки: `up`, `down`, `selected`, `over` и `disabled`. В файле FLA на рис. 9.3 для каждого состояния добавляются две метки кадров: одна помечает начало анимации состояния, а другая – ее завершение (например, `OverBegin` и `OverEnd`). Пользовательский класс, реагирующий на фазы состояния, использует эти метки для продвижения к заданным кадрам. Следующим шагом должен стать экспорт символического идентификатора из Flash, чтобы он мог использоваться компилятором Flex.

Для предоставления доступа к кадрам символического идентификатора в файле SWF свойство *Base class* определяется как экземпляр класса `flash.display.MovieClip` и ему назначается имя класса. На рис. 9.4 созданному ранее многокадровому символическому имени `MovieClip` назначается имя класса `ButtonClip`, и оно помечается для экспорта.

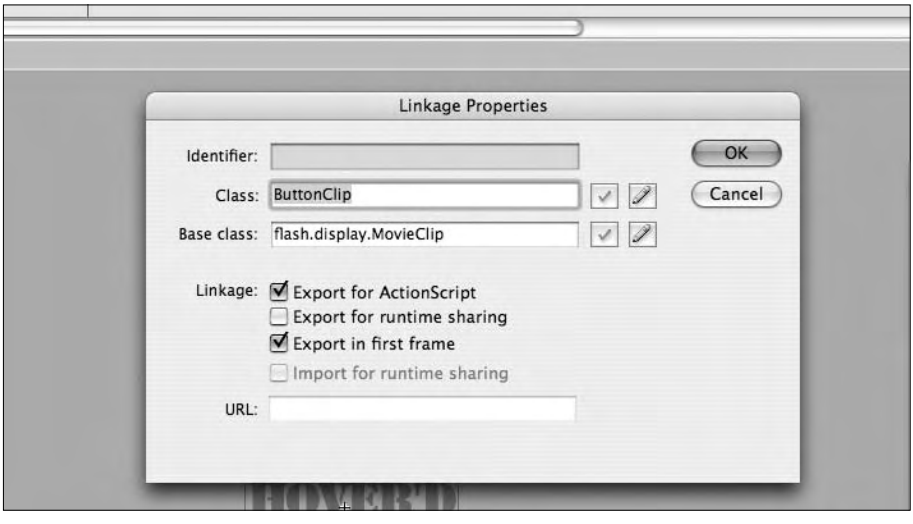

***Рис. 9.4.** Экспорт клипа с использованием уникального имени класса*

Для импортирования символического идентификатора из сгенерированного файла SWF в определении класса используется тег метаданных `[Embed]`. Атрибут `source` определяет местонахождение файла SWF, а атрибут `symbol` определяется именем класса, заданным в экспортированном символическом идентификаторе:

```
package oreilly.cookbook
{
    import flash.display.MovieClip;

    [Embed(source="assets/styles/FLAExample.swf",
        symbol="ButtonClip")]
    public class ButtonClass extends MovieClip{}
}
```

Класс выполняет функции пользовательского компонента, добавляемого в список отображения; он обеспечивает навигаци между tween-анимациями с использованием меток кадров. Экземпляр `ButtonClass` нельзя напрямую применить к списку отображения в приложении Flex, потому что он является субклассом компонента Flash, не реализующего интерфейс `mx.core.IFlexDisplayObject`. Чтобы включить экземпляр этого класса в список отображения, необходимо сделать его производным от `mx.core.UIComponent`.

Пользовательская версия `UIComponent`**, включающая экземпляр** `ButtonClass` **в список отображения, также переходит между кадрами анимации в соответствии с изменениями свойства** `currentState`**. Пример пользовательского скина, назначаемого экземпляру** `Button`**:**

```
package oreilly.cookbook
{
    import flash.display.MovieClip;
    import flash.events.Event;
    import mx.core.UIComponent;

    public class ButtonAnimatedSkin extends UIComponent
    {
        private var _state:String;
        private var _movieClipAsset:MovieClip;
        public function ButtonAnimatedSkin() {
            super();
        }
        // При создании дочерних компонентов
        // создается экземпляр нашего класса ButtonClass
        override protected function createChildren():void {
            super.createChildren();
            movieClipAsset = new ButtonClass();
            movieClipAsset.addEventListener(
            Event.ENTER_FRAME, enterFrameHandler);
            addChild(_movieClipAsset);
        }
        // Необходимо обеспечить корректную проверку
        // принадлежности. Хотя объект не видим на экране,
        // он все равно присутствует.
        override protected function updateDisplayList(
            unscaledWidth:Number,
            unscaledHeight:Number ):void {
            graphics.clear();
            graphics.beginFill( 0x000000, 0 );
            graphics.drawRect( 0, 0, _movieClipAsset.width,
            movieClipAsset.height );
            graphics.endFill();
            // Клип всегда находится в точке 0, 0
            _movieClipAsset.x = _movieClipAsset.width/2;
            _movieClipAsset.y = _movieClipAsset.height/2;
        }
        // Метод вызывается при изменении состояния кнопки.
        override public function set currentState(
            value:String ):void {
            state = value;
            // Некоторые из неиспользуемых значений;
            // разумеется, при желании для них можно
            // создать разные анимации.
            if(_state == "selectedUp")      _state = "selected";
            if(_state == "selectedDown")    _state = "down";
```

```
    if(_state == "selectedOver")    _state = "over";
    switch( _state )
    {
        case "over" :
            // Для каждого состояния просто
            // вызывается gotoAndPlay для кадра
            // соответствующего состояния
            movieClipAsset.gotoAndPlay("OverBegin");
            break;
        case "down":
            movieClipAsset.gotoAndPlay("DownBegin");
            break;
        case "selected":
            movieClipAsset.gotoAndPlay(
                "SelectedBegin");
            break;
        case "up" :
            movieClipAsset.gotoAndPlay(
                "EnabledBegin");
            break;
        case "disabled" :
            movieClipAsset.gotoAndPlay(
                "DisabledBegin");
            break;
    }
}
// При каждом переходе к очередному кадру
// необходимо проверить, не находится ли он
// в конце анимации какого-либо состояния.
// Если кадр действительно завершает анимацию,
// клип возвращается к началу цикла.
private function enterFrameHandler(event:Event):
    void {
    switch(_state){
        case "over" :
            if(_movieClipAsset.currentLabel ==
                "OverEnd") {
                _movieClipAsset.gotoAndPlay(
                "OverBegin");
            }
            break;
        case "down":
            if(_movieClipAsset.currentLabel ==
               "DownEnd") {
                movieClipAsset.gotoAndPlay(
               "DownBegin");
            }
            break;
        case "up" :
            if(_movieClipAsset.currentLabel ==
            "EnabledEnd") {
```

```
                        movieClipAsset.gotoAndPlay(
                        "EnabledBegin");
                    }
                    break;
                case "selected":
                    if(_movieClipAsset.currentLabel ==
                    "SelectedEnd") {
                        movieClipAsset.gotoAndPlay(
                        "SelectedBegin");
                    }
                    break;
                case "disabled" :
                    // Ничего делать не нужно, так как
                    // состояние disabled представлено
                    // всего одним кадром
                    break;
            }
        }
    }
}
```

В переопределении метода `createChildren` новый экземпляр `ButtonClass` создается и включается в список отображения. При каждом обновлении текущего состояния в классе `Button` класс `ButtonAnimatedSkin` использует новое состояние, чтобы приказать `ButtonClass` перейти к заданному кадру. Внутри метода `enterFrameHandler` находится цикл, проверяющий, достигнут ли конец анимации. Если анимация достигла завершающего кадра, клип возвращается к началу этого состояния анимации.

Скин `ButtonAnimatedSkin` назначается точно так же, как назначаются программные скины – определением класса со стилевым свойством `skin`:

```
<mx:Button toggle="true" skin="oreilly.cookbook.ButtonAnimatedSkin"
    y="300" x="300"/>
```

См. также

Рецепты 9.13–9.15.

9.18. Настройка предварительной загрузки

Задача

Требуется настроить информацию, выводимую во время загрузки и инициализации приложения Flex.

Решение

Создайте пользовательский экран предварительной загрузки. Для этого создайте субкласс, производный либо от `mx.preloaders.DownloadProgressBar` (используется приложениями по умолчанию), либо от `flash.`

display.Sprite, реализующий интерфейс mx.preloaders.IPreloaderDisplay interface.

Обсуждение

По умолчанию приложение Flex состоит из двух кадров. В первом кадре создается предварительный загрузчик, который передает серию событий, относящихся к загрузке и инициализации приложения. Стандартный индикатор загрузки обновляет свое изображение в соответствии с ходом обработки этих событий. По завершению загрузки системный менеджер переход к следующему кадру, в котором происходит собственно загрузка и инициализация приложения. После завершения инициализации приложения системный менеджер получает оповещение и удаляет изображение предварительного загрузчика.

Во внутренней реализации этот процесс обеспечивается системным менеджером, инициализирующим экземпляр класса mx.preloaders.Preloader для отслеживания загрузки и инициализации приложения. Экземпляр Preloader создает экземпляр индикатора загрузки, включает его в свой список отображения и задает свойству preloader реализации IPreloaderDisplay ссылку на самого себя. Определенный экземпляр Preloader может прослушивать разнообразные события, передаваемые предварительным загрузчиком (табл. 9.2).

Таблица 9.2. События, передаваемые классом Preloader

Событие	Описание
ProgressEvent.PROGRESS	Передается во время загрузки файла SWF
Event.COMPLETE	Передается при завершении загрузки файла SWF
FlexEvent.INIT_PROCESS	Передается во время инициализации приложения
FlexEvent.INIT_COMPLETE	Передается при завершении инициализации приложения. Реализация IPreloaderDisplay должна прослушивать это событие для своевременной выдачи события COMPLETE. Событие COMPLETE прослушивается передаваемым экземпляром Preloader для передачи системному менеджеру информации о том, что приложение готово к дальнейшему отображению
RslEvent.RSL_ERROR	Передается при неудачной попытке загрузки RSL (Runtime Shared Library)
RslEvent.RSL_PROGRESS	Передается во время загрузки RSL
RslEvent.RSL_COMPLETE	Передается после завершения загрузки RSL

Чтобы создать пользовательский индикатор прогресса для этих событий, следует определить субкласс, производный либо от класса mx.preloaders.DownloadProgressBar, либо от класса mx.display.Sprite, реализующего

mx.preloaders.IPreloaderDisplay **(слегка видоизмененная версия** DownloadProgressBar**).**

Класс DownloadProgressBar **определяет защищенные обработчики для событий, перечисленных в табл. 9.2. Создание субкласса, производного от** DownloadProgressBar**, позволяет переопределить эти обработчики для соответствующей модификации и обновления нестандартного загрузчика:**

```
package oreilly.cookbook
{
    import flash.events.Event;
    import flash.events.ProgressEvent;
    import flash.geom.Rectangle;
    import mx.preloaders.DownloadProgressBar;

    public class DPBSubclass extends DownloadProgressBar
    {
        public function DPBSubclass()
        {
            super();
            // Минимальное время отображения после загрузки
            MINIMUM_DISPLAY_TIME = 3000;
            // Текст, выводимый в ходе инициализации
            initializingLabel = "Download
                complete!\nInitializing...";
        }
        // Переопределение области вывода надписи
        // для нестандартного текста
        override protected function get labelRect():
            Rectangle
        {
            return new Rectangle(14, 5, 150, 30);
        }
        // Переопределение обработчика события загрузки
        // для вывода нестандартного текста
        override protected function progressHandler(
            event:ProgressEvent ):void
        {
            super.progressHandler(event);
            label = Math.round(
                event.bytesLoaded / 1000 ).toString()
                + "k of " + Math.round( event.bytesTotal /
                1000 ).toString()
                + "k";
        }
        // Переопределение, обеспечивающее вывод индикатора
        // прогресса во время инициализации и загрузки
        override protected function showDisplayForInit(
            elapsedTime:int,
            count:int):Boolean
        { return true; }
```

```
        override protected function
            showDisplayForDownloading( elapsedTime:int,
            event:ProgressEvent):Boolean
        { return true; }
    }
}
```

В этом примере обработчик `progressHandler`, определяемый классом `DownloadProgressBar`, используется для вывода нестандартного текста при получении события `PROGRESS`, передаваемого предварительным загрузчиком. Область отображения надписи также изменяется посредством переопределения `labelRect`. Во внутренней реализации область для вывода текстовых оповещений создается суперклассом. Чтобы назначить класс `DPBSubclass` пользовательским предварительным загрузчиком, укажите полное имя класса в свойстве `preloader` тега `<mx:Application>`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    preloader="oreilly.cookbook.DPBSubclass">
    <mx:Script>
        <![CDATA[
            // Встраивание большого звукового файла
            // для демонстрации загрузчика
            [Embed(source="assets/audio/audio.mp3")]
            private var _audio:Class;
        ]]>
    </mx:Script>
</mx:Application>
```

Переопределение открытого set-метода `preloader` в субклассе `DownloadProgressBar` позволяет назначить собственный обработчик на случай, если вы предпочитаете более точную обработку событий. Но если вам потребуются расширенные возможности контроля над визуальным оформлением нестандартного предварительного загрузчика, создайте субкласс `Sprite`, реализующий интерфейс `IPreloaderDisplay`.

При создании экземпляра `IPreloaderDisplay` необходимо определить реализации ряда свойств и методов. Значения свойств, относящихся к визуальным аспектам (например, `backgroundColor` и размеры сцены), назначаются при создании экземпляра пользовательского загрузчика, который затем вызывает методы `initialize`. Свойство `preloader` экземпляра `IPreloaderDisplay` содержит экземпляр `Preloader`, для которого реализация должна назначить обработчики событий.

Следующий фрагмент реализует `IPreloaderDisplay` для создания пользовательского индикатора загрузки:

```
package oreilly.cookbook
{
    import flash.display.Shape;
    import flash.display.Sprite;
```

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.text.TextField;
import flash.text.TextFormat;
import flash.utils.Timer;

import mx.events.FlexEvent;
import mx.preloaders.IPreloaderDisplay;
import mx.preloaders.Preloader;

public class CustomProgress extends Sprite implements
IPreloaderDisplay
{
    private var _bgAlpha:Number;
    private var _bgColor:uint;
    private var _bgImage:Object;
    private var _bgSize:String;
    private var _stageHeight:Number;
    private var _stageWidth:Number;
    private var _preloader:Preloader;
    private var _downloadBar:Shape;
    private var _initBar:Shape;
    private var _initField:TextField;

    public function CustomProgress()
    {
        _initField = new TextField();
        _initField.defaultTextFormat =
            new TextFormat( 'Arial', 12, 0xFFFFFF, true );
        _downloadBar = new Shape();
        addChild( _downloadBar );
        initBar = new Shape();
        addChild( _initBar );
    }
    // Инициализация свойств

    public function initialize():void
    {
        downloadBar.x = ( _stageWidth / 2 ) - 20;
        initBar.x = _downloadBar.x - 2;
        downloadBar.y = ( _stageHeight / 2 ) - 50;
        initBar.y = _downloadBar.y;
        initField.x = _initBar.x + 2;
        initField.y = _initBar.y + 100 - 15;
    }
    // Определение обработчиков событий
    // для экземпляра Preloader
    public function set preloader( obj:Sprite ):void
    {
        preloader = obj as Preloader;
```

```
        preloader.addEventListener(ProgressEvent.PROGRESS,
            downloadProgressHandler );
        preloader.addEventListener(FlexEvent.INIT_PROGRESS,
            initProgressHandler );
        preloader.addEventListener(FlexEvent.INIT_COMPLETE,
            initCompleteHandler );
    }

    public function get backgroundAlpha():Number
    {
        return _bgAlpha;
    }

    public function set backgroundAlpha(value:Number):void
    {
        _bgAlpha = value;
    }

    public function get backgroundColor():uint
    {
        return _bgColor;
    }

    public function set backgroundColor(value:uint):void
    {
        _bgColor = value;
    }

    public function get backgroundImage():Object
    {
        return _bgImage;
    }

    public function set backgroundImage(value:Object):void
    {
        _bgImage = value;
    }

    public function get backgroundSize():String
    {
        return _bgSize;
    }
    public function set backgroundSize(value:String):void
    {
        _bgSize = value;
    }

    public function get stageHeight():Number
    {
        return _stageHeight;
    }
```

```
public function set stageHeight(value:Number):void
{
    _stageHeight = value;
}

public function get stageWidth():Number
{
    return _stageWidth;
}

public function set stageWidth(value:Number):void
{
    _stageWidth = value;
}

// Обработка хода загрузки файла SWF
private function downloadProgressHandler(
    evt:ProgressEvent ):void
{
    var perc:Number = ( (
        evt.bytesLoaded / evt.bytesTotal ) * 100 );
    var top:Number = 100 - perc;
    downloadBar.graphics.clear();
    downloadBar.graphics.beginFill( 0xFF0000, 1 );
    downloadBar.graphics.moveTo( 0, 0 );
    downloadBar.graphics.lineTo( 10, 0 );
    downloadBar.graphics.lineTo( 10, perc * 0.9 );
    downloadBar.graphics.lineTo( 0, perc * 0.9 );
    downloadBar.graphics.lineTo( 0, 0 );
    downloadBar.graphics.endFill();

    initBar.graphics.clear();
    initBar.graphics.beginFill( 0xFFFFFF, 1 );
    initBar.graphics.moveTo( 0, 100 );
    initBar.graphics.lineTo( 2, 100 );
    initBar.graphics.lineTo( 2, top );
    initBar.graphics.lineTo( 0, top );
    initBar.graphics.lineTo( 0, 100 );
    initBar.graphics.endFill();
}
// Обработка хода инициализации приложения
private function initProgressHandler( evt:FlexEvent ):void
{
    _initField.text = "initializing...";
    addChild( _initField );
}
// Обработка завершения загрузки и инициализации.
private function initCompleteHandler( evt:FlexEvent ):void
{
    var timer:Timer = new Timer( 3000, 1 );
    timer.addEventListener(
```

```
            TimerEvent.TIMER_COMPLETE, notifyOfComplete );
            timer.start();
        }
        // Оповещение о завершении загрузки и инициализации.
        private function notifyOfComplete(
            evt:TimerEvent ):void
        {
            dispatchEvent( new Event( Event.COMPLETE ) );
        }
    }
}
```

Пользовательские загрузчики должны передавать событие `COMPLETE` после получения события `INIT_COMPLETE` от экземпляра `Preloader`; этим они оповещают системного менеджера о том, что все операции завершены, а загрузчик можно исключить из отображения. Соответственно, если вы в своем приложении переопределяете открытый set-метод `preloader` в субклассе `DownloadProgressBar` или создаете реализацию `IPreloaderDisplay`, не забудьте определить обработчик события `INIT_COMPLETE` и передать событие `COMPLETE`. В предшествующем примере событие `COMPLETE` отправлялось по таймеру, чтобы информация инициализации оставалась на экране чуть дольше обычного.

Класс `CustomProgress` задается в свойстве `preloader` при объявлении `<mx:Application>`, как было сделано в предыдущем рецепте. Пример:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    preloader="oreilly.cookbook.CustomProgress">
    <mx:Script>
        <![CDATA[
            // Встраивание большого звукового файла
            // для демонстрации загрузчика
            [Embed(source="assets/audio/audio.mp3")]
            private var _audio:Class;

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

Возможности модификации индикатора загрузки не ограничиваются API графического вывода. Также можно внедрять графические элементы (например, файлы изображений или SWF), влияющие на внешний вид и поведение предварительного загрузчика приложения.

10

Перетаскивание

Поддержка перетаскивания мышью в Flex Framework расширяет функциональность **RIA-приложений: у пользователя появляется воз**можность визуального перемещения данных из одного места в другое. Поддержка перетаскивания может быть включена в любой компонент, расширяющий класс `mx.core.UIComponent`. В операциях перетаскивания участвуют две стороны: *источник* и *приемник*. Любой экземпляр `UIComponent` может принимать операции перетаскивания, а некоторые списковые компоненты Flex (например, `List`, `Tree` и `DataGrid`) обладают встроенной поддержкой перетаскивания, упрощающей автоматизацию процесса перемещения данных между разными источниками и внутри одного компонента.

Операции перетаскивания инициируются *жестом* **(gesture) перета**скивания: выделите компонент или элемент нажатием кнопки мыши, а затем перетащите его в другое место, не отпуская кнопки. Во время жеста перетаскивания графическое изображение, называемое *посредником перетаскивания* (drag proxy), включается в список отображения и следует за мышью, обозначая перетаскиваемый объект. Наряду с посредником перетаскивания на экране отображаются встроенные значки, которые показывают, что указатель мыши находится над компонентом, который может стать ее приемником. Чтобы разрешить прием «сброшенных» объектов для компонента, следует назначить ему обработчики событий перетаскивания. *Приемник перетаскивания* (компонент, способный принять «сброшенный» объект) может проанализировать данные объекта-источника и определить, хранятся ли данные в формате, приемлемом для компонента. Исходный объект может быть либо скопирован, либо перемещен из одного компонента в другой или же в пределах одного компонента – в этом случае компонент одновременно является как источником, так и приемником перетаскивания.

Эта глава посвящена поддержке перетаскивания в Flex Framework и возможностям ее применения для улучшения взаимодействий с пользователем.

10.1. Использование класса DragManager

Задача

Требуется переместить данные из одной точки приложения в другую.

Решение

Используйте класс `mx.manager.DragManager` для управления операциями перетаскивания; прослушивайте события перетаскивания в приемнике.

Обсуждение

Класс `DragManager` управляет операциями перетаскивания в приложениях. В начале операции перетаскивания объект-источник включается в `DragManager` статическим методом `doDrag`. Компоненты, регистрирующие слушателей для событий, передаваемых `DragManager`, считаются *приемниками перетаскивания* и могут принимать объекты данных, доступные через `DragManager`.

Исходный объект данных, полученный `DragManager` от компонента-источника, может быть перемещен или скопирован. По умолчанию в операциях перетаскивания данные перемещаются, однако при необходимости вы можете самостоятельно реализовать копирование через `DragManager`.

Следующий пример реализует возможность перемещения компонента `Box` в контейнере `Canvas`:

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

    <mx:Script>
        <![CDATA[
            import mx.core.DragSource;
            import mx.core.IUIComponent;
            import mx.events.DragEvent;
            import mx.managers.DragManager;

            private static const FORMAT:String = "box";

            private function mouseDownHandler(evt:MouseEvent ):void
            {
                var initiator:IUIComponent =
                evt.currentTarget as IUIComponent;
```

```
                var dragSource:DragSource = new DragSource();
                dragSource.addData( initiator, FORMAT );

                DragManager.doDrag( initiator, dragSource, evt );
            }
            private function dragEnterHandler(evt:DragEvent ):void
            {
                if( evt.dragSource.hasFormat( FORMAT ) )
                {
                    DragManager.acceptDragDrop( Canvas(
                    evt.currentTarget ) );
                }
            }
            private function dropHandler(evt:DragEvent ):void
            {
                var box:Box = Box( evt.dragInitiator );
                box.x = evt.localX;
                box.y = evt.localY;
            }
        ]]>
    </mx:Script>

    <mx:Canvas id="canvas"
        backgroundColor="0xEEEEEE"
        width="300" height="300"
        dragEnter="dragEnterHandler(event);"
        dragDrop="dropHandler(event);">
        <mx:Box id="dragItem"
            width="20" height="20"
            backgroundColor="0x00FFCC"
            mouseDown="mouseDownHandler(event);"
            />
    </mx:Canvas>

</mx:Application>
```

При передаче события `mouseDown` экземпляром `<mx:Box>` вызывается метод `mouseDownHandler`, и в `DragManager` добавляется объект данных `DragSource`. Метод `DragManager.doDrag` инициирует операцию перетаскивания, а при вызове ему должны передаваться минимум три аргумента: ссылка на источник, объект `mx.core.DragSource` и объект `flash.events.MouseEvent`, активизировавший обработчик события и содержащий информацию о мыши, относящуюся к операции перетаскивания. Изображение, выводимое по умолчанию во время перетаскивания, представляет собой прямоугольник с альфа-прозрачностью. Это изображение (называемое *посредником перетаскивания*) и его свойства можно изменить методом `doDrag`, но в данном примере будут использоваться значения по умолчанию.

Назначение обработчиков для событий `dragEnter` и `dragDrop`, передаваемых `DragManager`, превращает компонент `Canvas` в приемник перетаски-

вания, инициированного компонентом Box. В методе dragEventHandler проверяется формат данных источника, изначально заданный методом doDrag, а вызов статического метода acceptDragDrop объекта DragManager включает поддержку приема «сбрасываемого» объекта. В параметре acceptDragDrop передается приемник, реагирующий на события перетаскивания, в частности на событие dragDrop. Метод dropHandler приложения реагирует на операцию «сброса» и перемещает перемещенный источник (в данном примере компонент Box) в соответствии с позицией указателя мыши на момент отпускания кнопки.

Хотя в предыдущем примере данные по умолчанию перемещаются из одного места в другое, вы можете без особого труда реализовать копирование. В следующем примере информация копируется из компонента Box с поддержкой перетаскивания в объект DragSource, который используется для создания нового экземпляра Box, включаемого в список отображения приемника:

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

    <mx:Script>
        <![CDATA[
            import mx.core.DragSource;
            import mx.events.DragEvent;
            import mx.managers.DragManager;

            private static const FORMAT:String = "box";

            private function mouseDownHandler( evt:MouseEvent ):void
            {
                var initiator:Box = evt.currentTarget as Box;
                var boxData:Object = new Object();
                boxData.width = initiator.width;
                boxData.height= initiator.height;
                boxData.backgroundColor = initiator.getStyle(
                    "backgroundColor" );
                var dragSource:DragSource = new DragSource();
                dragSource.addData( boxData, FORMAT );

                DragManager.doDrag( initiator, dragSource, evt );
            }
            private function dragEnterHandler( evt:DragEvent ):void
            {
                if( evt.dragSource.hasFormat( FORMAT ) )
                {
                    DragManager.acceptDragDrop( Canvas( evt.currentTarget )
);
                }
            }
            private function dropHandler( evt:DragEvent ):void
```

```
            {
                var boxData:Object = evt.dragSource.dataForFormat( FORMAT );
                var box:Box = new Box();
                box.width = boxData.width;
                box.height = boxData.height;
                box.setStyle( "backgroundColor", boxData.backgroundColor );
                box.x = evt.localX;
                box.y = evt.localY;
                canvas.addChild( box );
            }
        ]]>
    </mx:Script>

    <mx:Canvas id="canvas"
        backgroundColor="0xEEEEEE"
        width="300" height="300"
        dragEnter="dragEnterHandler(event);"
        dragDrop="dropHandler(event);">
        <mx:Box id="dragItem"
            width="20" height="20"
            backgroundColor="0x00FFCC"
            mouseDown="mouseDownHandler(event);"
            />
    </mx:Canvas>

</mx:Application>
```

В обработчике события `mouseDownHandler` создается обобщенный объект со свойствами, соответствующими свойствам компонента-источника `Box`. Объект включается в объект `DragSource`, а обращение к нему производится из метода `DragSource.dataForFormat` в обработчике события `dragDrop` приложения. Метод `dropHandler` создает новый экземпляр компонента `Box` со свойствами, переданными операцией перетаскивания, и включает его в список отображения контейнера `Canvas`.

10.2. Назначение посредника перетаскивания

Задача

Требуется настроить изображение, выводимое в ходе операции перетаскивания.

Решение

Передайте пользовательское изображение в необязательном параметре `dragImage` метода `DragManager.doDrag`.

Обсуждение

По умолчанию во время перетаскивания используется изображение, которое представляет собой прямоугольник с альфа-прозрачностью.

Экранный объект, отображаемый при инициализации перетаскивания, называется *посредником перетаскивания*. Чтобы изменить изображение, передайте экземпляр `IFlexDisplayObject` в параметре `dragImage`. Все компоненты Flex Framework могут назначаться в качестве посредников перетаскивания, так как они являются расширениями класса `mx.core.UIComponent`, реализующего интерфейс `IFlexDisplayObject`. Добавление компонента в качестве посредника перетаскивания дает точное представление о перетаскиваемом объекте, однако оно сопряжено с излишними затратами ресурсов. Класс `BitmapAsset` также реализует интерфейс `IFlexDisplayObject`; это удобный способ графического представления визуальных данных, перемещаемых внутри приложения.

В следующем примере объект `BitmapAsset` используется в качестве посредника в операции перетаскивания:

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

    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;
            import mx.core.DragSource;
            import mx.events.DragEvent;
            import mx.managers.DragManager;

            private var xoffset:Number;
            private var yoffset:Number;
            private static const FORMAT:String = "box";

            private function mouseDownHandler(evt:MouseEvent ):void
            {
                xoffset = evt.localX;
                yoffset = evt.localY;
                var initiator:Box = evt.currentTarget as Box;
                var proxyBox:BitmapAsset = new BitmapAsset();
                proxyBox.bitmapData = new BitmapData( initiator.width,
                    initiator.height );
                proxyBox.bitmapData.draw( initiator );
                var dragSource:DragSource = new DragSource();
                dragSource.addData( initiator, FORMAT );
                DragManager.doDrag( initiator, dragSource,
                    evt, proxyBox, 0, 0, 0.5 );
            }
            private function dragEnterHandler(evt:DragEvent ):void
            {
                if( evt.dragSource.hasFormat( FORMAT ) )
                {
                   DragManager.acceptDragDrop(Canvas( evt.currentTarget ) );
                }
            }
```

```
            private function dropHandler(evt:DragEvent ):void
            {
                var box:Box = Box( evt.dragInitiator );
                box.x = evt.localX - xoffset;
                box.y = evt.localY - yoffset;
            }
        ]]>
    </mx:Script>
    <mx:Canvas id="canvas"
        backgroundColor="0xEEEEEE"
        width="300" height="300"
        dragEnter="dragEnterHandler(event);"
        dragDrop="dropHandler(event);">
        <mx:Box id="dragItem"
            width="20" height="20"
            backgroundColor="0x00FFCC"
            mouseDown="mouseDownHandler(event);"
            />
    </mx:Canvas>
</mx:Application>
```

В начале операции перетаскивания, при вызове обработчика `mouseDownHandler` растровое представление источника выводится на экземпляр `BitmapData` нового экземпляра `BitmapAsset`. Экземпляр `BitmapAsset` указывается как посредник перетаскивания и воспроизводится как временное изображение для операции перетаскивания с альфа-прозрачностью 50%. Локальная по отношению классу ссылка на позицию нажатия кнопки мыши сохраняется в начале перетаскивания, а скопированный экземпляр `Box` добавляется в финальную позицию с эквивалентным смещением.

Ваши возможности по назначению посредника перетаскивания не ограничиваются растровым представлением источника; также при передаче данных между частями одного приложения можно использовать встроенную графику.

См. также

Рецепт 10.1.

10.3. Перетаскивание внутри списка

Задача

Требуется переместить и копировать данные в пределах одного спискового компонента.

Решение

Воспользуйтесь встроенной поддержкой перетаскивания в списковых компонентах.

Обсуждение

Поддержка перетаскивания в любом компоненте включается назначением обработчиков событий, поступающих от `DragManager`. Хотя поддержку перетаскивания для списковых компонентов Flex Framework (`List`, `Tree` и `DataGrid`) можно добавить и вручную, эти компоненты обладают встроенной поддержкой перетаскивания. Наряду с внутренней обработкой событий, поступающих от `DragManager`, списковые компоненты (компоненты, расширяющие `mx.controls.listClasses.ListBase`) содержат открытые методы и свойства для выполнения операций и управления данными.

В следующем примере представлен список, элементы которого могут перемещаться внутри коллекции посредством перетаскивания. Встроенная поддержка перетаскивания включается у компонента при помощи свойств `dragEnabled` и `dropEnabled`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            private function creationHandler():void
            {
                var collection:ArrayCollection =
                    new ArrayCollection(['Josh', 'Todd', 'Abey'] );
                contactList.dataProvider = collection;
            }
        ]]>
    </mx:Script>
    <mx:Panel title="Contact List:"
        width="200" height="200">
        <mx:List id="contactList"
            width="100%" height="100%"
            dragEnabled="true"
            dropEnabled="true"
            dragMoveEnabled="true"
            />
    </mx:Panel>

</mx:Application>
```

Свойствам `dragEnabled`, `dropEnabled` и `dragMoveEnabled` компонента `List` в этом примере задается значение `true`. Свойства `dragEnabled` и `dropEnabled` фактически разрешают компоненту реагировать на события, поступающие от класса `DragManager`. Свойство `dragMoveEnabled` представляет собой флаг, указывающий, какая операция должна выполняться с копируе-

мым элементом списка – перемещение или копирование. По умолчанию свойство задается равным `false`, т. е. разрешается копирование в список данных источника. Если задать свойству `dragMoveEnabled` значение `true`, то при завершении перетаскивания элемент удаляется из предыдущей позиции и перемещается в новую позицию списка, над которой находится указатель мыши.

Если разрешить копирование данных, задавая свойству `dragMoveEnabled` значение `false` (или оставляя его значение по умолчанию), при попытках выделения элементов, задействованных в операциях перетаскивания из нашего примера, проявляются некоторые странности. Это объясняется тем, что в классах из базового API коллекций операции с данными выполняются с использованием уникального идентификатора (UID), назначаемого элементам коллекции.

При включении элемента в данные `dataProvider` спискового компонента вызывается защищенный метод `ListBase.copyItemWithUID`. Но если источник данных представляет собой простой список объектов `String` (например, `Array` или `ArrayCollection`, как в нашем примере), внутренняя реализация не считает нужным присваивать скопированным элементам новые идентификаторы. Из-за этого при выполнении любых операций выделения с компонентом `List` из приведенного примера выделяется элемент с наибольшим индексом для заданного UID. Иначе говоря, если скопировать первый элемент списка на пятую позицию, то при любых попытках повторного выделении первого элемента список отобразит как выделенный пятый элемент.

Чтобы скопированные данные были уникальными, а при добавлении в коллекцию им назначался идентификатор UID, следует добавить обработчики операций перетаскивания и копировать данные так, как требует ваше приложение. В следующем примере предыдущий фрагмент дополнен обработчиком события `dragComplete`, передаваемого компонентом `List`:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectUtil;
            import mx.events.DragEvent;
            import mx.collections.ArrayCollection;

            private function creationHandler():void
            {
                var collection:ArrayCollection =
                    new ArrayCollection( ['Josh', 'Todd', 'Abey'] );
                contactList.dataProvider = collection;
            }
```

```
            private function dropHandler( evt:DragEvent ):void
            {
                var listItem:Object = evt.dragSource.dataForFormat(
                    "items" );
                var index:int = contactList.calculateDropIndex( evt );
                ArrayCollection( contactList.dataProvider ).setItemAt(
                    ObjectUtil.copy( listItem ), index );
            }

        ]]>
    </mx:Script>

    <mx:Panel title="Contact List:"
        width="200" height="200">
        <mx:List id="contactList"
            width="100%" height="100%"
            dragEnabled="true"
            dropEnabled="true"
            dragMoveEnabled="false"
            dragComplete="dropHandler(event);"
            />
    </mx:Panel>

</mx:Application>
```

Задание свойству `dragMoveEnabled` значения `false` включает режим копирования, а экземпляр `List` принимает новые элементы по завершении жеста перетаскивания. Событие `dragComplete` регистрирует метод `dropHandler` как обработчик события завершения операции перетаскивания. В обработчике события текущий перетаскиваемый элемент извлекается из `DragSource` в формате `items` – внутреннем формате, назначенном в начале операции перетаскивания. Индекс элемента в списке извлекается методом `List.calculateDropIndex` и используется для внутреннего обновления элемента, включенного в коллекцию. Происходит глубокое копирование элемента с назначением ему нового уникального идентификатора вызовом метода `ObjectUtil.copy`. Так как провайдер данных представляет собой объект `ArrayCollection`, при любых изменениях в элементах коллекции активизируется привязка данных к экземпляру `List`, а все изменения немедленно отражаются в изображении компонента.

См. также

Рецепт 10.1.

10.4. Перетаскивание между списками

Задача

Требуется переместить данные из одного компонента `List` в другой компонент `List`.

Решение

Воспользуйтесь встроенной поддержкой перетаскивания в компонентах `List`.

Обсуждение

Встроенная поддержка перетаскивания для списковых компонентов Flex позволяет легко передать данные из одного компонента `List` в другой без прямого взаимодействия с объектом `DragManager`. Задавая свойства `dragEnabled` и `dropEnabled` компонентов, производных от `ListBase`, вы по умолчанию поручаете всю работу внутренним операциям перемещения и копирования источника.

При выполнении двустороннего перетаскивания между компонентами `List` у обоих компонентов включаются соответствующие свойства. Если источник и приемник относятся к одному базовому типу (например, если это два компонента `List`), структура объекта данных источника несущественна для успешного завершения операции, так как оба компонента одинаково отображают данные при использовании одного типа рендерера. При включении одно- и двустороннего перетаскивания между списковыми компонентами разных типов (например, компонентом `List` и компонентом `DataGrid`) приходится учитывать структуру данных источника, так как компоненты обладают разными рендерерами элементов и по-разному отображают данные.

Следующий пример позволяет перемещать элементы из компонента `List` в компонент `DataGrid` с полной информацией об элементах списка:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            private function creationHandler():void
            {
                contactList.dataProvider = new ArrayCollection([
                {label:'Josh Noble', phone:'555.111.2222'},
                {label:'Todd Anderson', phone:'555.333.4444'},
                {label:'Abey George', phone:'555.777.8888'}
                ]);
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Contact List:"
        width="200" height="200">
        <mx:List id="contactList"
```

```
            width="100%" height="100%"
            dragEnabled="true"
            dropEnabled="true"
            dragMoveEnabled="false"
            />

    </mx:Panel>
    <mx:Panel title="Contact Info:"
        width="300" height="200">
        <mx:DataGrid id="contactGrid"
            width="100%" height="100%"
            dragEnabled="true"
            dropEnabled="true"
            dragMoveEnabled="true">
            <mx:columns>
                <mx:DataGridColumn dataField="label" headerText="Name"/>
                <mx:DataGridColumn dataField="phone" headerText="Phone"/>
            </mx:columns>
        </mx:DataGrid>
    </mx:Panel>

</mx:Application>
```

В объект `ArrayCollection` включаются обобщенные объекты со свойствами `label` и `phone`. При назначении `ArrayCollection` провайдером данных экземпляра `List` список заполняется значениями свойства `label`. При перетаскивании элементов из компонента `List` в `DataGrid` происходит копирование, и в таблице отображаются оба свойства объекта источника: `label` и `phone`.

Так как свойство `dropEnabled` обоих списковых компонентов задано равным `true`, этот пример демонстрирует систему с двусторонним перетаскиванием.

См. также

Рецепт 10.1.

10.5. Разрешение и запрет операций перетаскивания

Задача

Требуется динамически разрешать и запрещать операции перетаскивания для списковых компонентов во время выполнения.

Решение

Используйте события перетаскивания для управления значениями свойств.

Обсуждение

Списковые компоненты Flex Framework обладают встроенной поддержкой взаимодействия с DragManager и предоставляют удобный механизм, активизирующий реакцию компонентов на жесты перетаскивания: свойства dragEnabled и dropEnabled. Обработчики событий для них назначаются так же, как и для других компонентов, производных от UIComponent: при помощи событийных свойств dragStart, dragEnter, dragOver, dragExit, dragDrop и dragComplete.

Чтобы экземпляр спискового компонента мог инициировать операции перетаскивания, установите для него логическое свойство dragEnabled. Чтобы компонент мог выполнять функции приемника перетаскивания, установите свойство dropEnabled. Списковые компоненты могут поддерживать одно- или двустороннее перетаскивание. В односторонней схеме экземпляр компонента выполняет функции либо источника, либо приемника. В двусторонней схеме компонент поддерживает оба вида операций.

Назначение обработчиков событий, передаваемых DragManager, позволяет управлять обработкой событий и данных источника в приложении. В следующем примере обработчик события dragEvent резрешает или запрещает операции перетаскивания для двух компонентов List:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.events.DragEvent;
            import mx.collections.ArrayCollection;

            [Bindable]
            public var isEnabled:Boolean = true;
            private static const DIS_LABEL:String = "disable drag and drop";
            private static const EN_LABEL:String = "enable drag and drop";

            private function creationHandler():void
            {
                list1.dataProvider = new ArrayCollection([
                    'Spider Monkey', 'Orangutan',  'Gorilla'
                ]);
                list2.dataProvider = new ArrayCollection([
                    'Lion', 'Cheetah', 'Puma'
                ])
            }

            private function clickHandler():void
            {
                enableBtn.label = ( enableBtn.label == DIS_LABEL )
```

```
                    ? EN_LABEL
                    : DIS_LABEL;
                isEnabled = !isEnabled;
            }

            private function dragEnterHandler( evt:DragEvent ):void
            {
                evt.target.dropEnabled = (
                    evt.target != evt.dragInitiator );
            }

        ]]>
    </mx:Script>

    <mx:VBox width="100%" height="100%">
        <mx:Button id="enableBtn"
            label="disable drag and drop"
            click="clickHandler();"
            />
        <mx:HBox width="100%" height="100%">
            <mx:List id="list1"
                width="200" height="200"
                dragEnabled="{isEnabled}"
                dragMoveEnabled="true"
                dragEnter="dragEnterHandler(event);"
                />
            <mx:List id="list2"
                width="200" height="200"
                dragEnabled="{isEnabled}"
                dragMoveEnabled="true"
                dragEnter="dragEnterHandler(event);"
                />
        </mx:HBox>
    </mx:VBox>

</mx:Application>
```

Каждый список разрешает инициировать операции перетаскивания в зависимости от состояния локального привязываемого свойства `isEnabled`, значение которого обновляется в обработчике события `click` экземпляра `Button`. Когда спискам разрешается инициировать операции перетаскивания, событие `dragEnter` выдается при выделении элемента и входе указателя мыши в границы компонента.

Когда свойству `dragMoveEnabled` задается значение `true`, завершение жеста перетаскивания приводит к выполнению перемещения. По умолчанию свойство `dragMoveEnabled` равно `false`; это означает, что при завершении операции перетаскивания объект данных копируется из источника в приемник перетаскивания.

Чтобы запретить перемещение данных в пределах экземпляра `List`, свойство `dropEnabled` каждого компонента обновляется в обработчике

события dragEnter. Обработчик dragEnterHandler проверяет, совпадает ли приемник перетаскивания с источником, и изменяет dropEnabled. Чтобы компонент, инициировавший жест перетаскивания, не мог стать приемником, его свойству dropEnabled задается значение false.

См. также

Рецепты 10.3 и 10.4.

10.6. Настройка посредника перетаскивания в списковых компонентах

Задача

Требуется изменить внешний вид перетаскиваемого изображения при выполнении операций перетаскивания внутри списковых компонентов.

Решение

Создайте пользовательский экземпляр UIComponent, отображаемый во время перетаскивания. Переопределите защищенный get-метод dragImage в пользовательском компоненте List.

Обсуждение

Списковые компоненты Flex Framework содержат встроенную поддержку перетаскивания. Это означает, что разработчик не обязан специально назначать слушателей для событий, передаваемых DragManager. Но так как вы не взаимодействуете с DragManager напрямую, в списковых компонентах по умолчанию во время перетаскивания используется изображение рендерера элемента с альфа-прозрачностью. Это изображение невозможно задать в свойстве спискового компонента, как это делается вручную при использовании метода DragManager.doDrag.

Чтобы настроить изображение, выводимое во время жеста перетаскивания для компонента List, необходимо создать пользовательский компонент List, который расширяет приемный компонент и переопределяет защищенный get-метод dragImage. Переопределение возвращает пользовательское изображение перетаскивания, расширяющее класс UIComponent.

В следующем примере представлен пользовательский класс, который расширяет класс UIComponent и переопределяет защищенный метод createChildren для включения изображения в посредника:

```
package oreilly.cookbook
{
    import flash.display.Bitmap;
    import flash.display.Loader;
```

```
        import flash.display.LoaderInfo;
        import flash.events.Event;
        import flash.net.URLRequest;

        import mx.controls.List;
        import mx.controls.listClasses.IListItemRenderer;
        import mx.core.UIComponent;

        public class CustomDragProxy extends UIComponent
        {
            public function CustomDragProxy()
            {
                super();
            }

            override protected function createChildren():void
            {
                super.createChildren();

                var list:List = List( owner );
                var items:Array = list.selectedIndices;
                items.sort();

                for( var i:int = 0; i < items.length; i++ )
                {
                    var item:Object = list.dataProvider[items[i]];
                    var loader:Loader = new Loader();
                    loader.contentLoaderInfo.addEventListener(
                        Event.COMPLETE, completeHandler );
                    addChild( loader );

                    loader.load( new URLRequest( item.image ) );

                    var source:IListItemRenderer =
                        list.indexToItemRenderer(items[i]);
                    loader.x = source.x;
                    loader.y = source.y - 20 + ( i * 45 );
                }
            }

            private function completeHandler( evt:Event ):void
            {
                var info:LoaderInfo = LoaderInfo( evt.target );
                var image:Bitmap = Bitmap( info.content );
                image.width = image.height = 40;
            }
        }
    }
```

Во внутренней реализации защищенный метод `createChildren` **вызывается при создании экземпляра компонента. Переопределяя метод** `createChildren` **по аналогии с этим примером (файл** `CustomDragProxy.as`**), можно**

задать собственные составляющие, включаемые в список отображения компонента. Считается, что создание посредника было инициировано компонентом List, а свойство owner преобразуется к экземпляру класса List. Преобразование owner дает возможность обращаться к свойствам родительского экземпляра List и использовать их для модификации изображения.

В зависимости от количества элементов, выделенных в List, в изображение включается соответствующее количество экземпляров Loader, а свойству image задается графический файл, заданный URL-адресом. Далее метод получает экземпляр рендерера элемента при помощи метода List.indexToItemRenderer и использует его для позиционирования родительского объекта Loader.

Чтобы назначить пользовательского посредника перетаскивания для спискового компонента, необходимо расширить компонент и переопределить get-метод dragImage. В следующем примере представлен пользовательский класс List, который возвращает нестандартного посредника при вызове метода dragImage:

```
package oreilly.cookbook
{
    import mx.controls.List;
    import mx.core.IUIComponent;

    public class CustomList extends List
    {
        public var dragProxy:Class;
        public function CustomList()
        {
            super();
        }

        override protected function get dragImage():IUIComponent
        {
            if( dragProxy == null )
                return super.dragImage;

            var proxy:IUIComponent = new dragProxy();
            proxy.owner = this;
            return proxy
        }
    }
}
```

В этом примере при обращении к свойству dragImage создается и возвращается новый экземпляр посредника. Как упоминалось в описании предыдущего примера, класс CustomDragProxy обращается к родительскому экземпляру List через унаследованное от класса UIComponent свойство owner. В классе CustomList свойство owner задается в переопределении dragImage до возвращения экземпляра dragProxy. Задайте свойству drag-

Proxy полное имя класса посредника, экземпляр которого будет создаваться при каждом обращении к свойству dragImage.

Следующий пример включает в приложение компонент CustomList и задает класс CustomDragProxy свойству dragProxy пользовательской версии List:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexcookbook="oreilly.cookbook.*"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[

            import mx.collections.ArrayCollection;
            private function creationHandler():void
            {
                contactList.dataProvider = new ArrayCollection([
                    {label:'Josh', image:'assets/bigshakey.png'},
                    {label:'Todd', image:'assets/smiley.png'}
                ]);
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Contact List:"
        width="200" height="200">
        <flexcookbook:CustomList id="contactList"
            width="100%" height="100%"
            allowMultipleSelection="true"
            dragEnabled="true"
            dropEnabled="true"
            dragMoveEnabled="true"
            dragProxy="com.oreilly.flexcookbook.CustomDragProxy"
            />
    </mx:Panel>

</mx:Application>
```

В свойстве dataProvider компоненту CustomList назначается массив объектов со свойствами label и image. CustomDragProxy объявляется во встроенном формате, для чего свойству dragProxy списка задается полное имя класса. В начале жеста перетаскивания создается экземпляр CustomDragProxy, а свойство image, связанное с объектом данных элемента списка, используется для загрузки графического файла, отображаемого во время операции перетаскивания.

См. также

Рецепты 10.1–10.3.

10.7. Настройка индикатора сброса для списковых компонентов

Задача

Требуется настроить для компонента `List` графический признак сброса, отображаемый во время операций перетаскивания.

Решение

Создайте пользовательский программный скин и задайте свойство `dropIndicatorSkin` компонента `List`.

Обсуждение

Программные скины по умолчанию, содержащиеся в списковых компонентах Flex Framework, отображают индикатор (признак использования текущего компонента как приемника) во время перетаскивания. При внутреннем вызове метода `ListBase.showDropFeedback` экземпляр класса индикатора создается и размещается на один пиксел выше или левее рендерера элемента, в зависимости от структуры элементов в компоненте `List`. Чтобы настроить индикатор сброса, расширьте класс `mx.skins.ProgrammaticSkin` и задайте стилевое свойство `dropIndicatorSkin` компонента.

В следующем примере определяется пользовательский индикатор, переопределяющий метод `updateDisplayList` класса `ProgrammaticSkin`; он использует графический API для рисования стрелки с учетом текущего значения свойства `direction`:

```
package oreilly.cookbook
{
    import mx.skins.ProgrammaticSkin;

    public class CustomDropIndicator extends
        ProgrammaticSkin
    {

        public var direction:String = "horizontal";
        public function CustomDropIndicator()
        {
            super();
        }

        override protected function updateDisplayList(
            unscaledWidth:Number, unscaledHeight:Number ):void
        {
            super.updateDisplayList( unscaledWidth, unscaledHeight );

            graphics.clear();
            graphics.beginFill( 0x000000 );
```

```
            if( direction == "horizontal" )
            {
                graphics.moveTo( 4, -10 );
                graphics.lineTo( 6, -10 );
                graphics.lineTo( 6, -4 );
                graphics.lineTo( 10, -4 );
                graphics.lineTo( 5, 0 );
                graphics.lineTo( 0, -4 );
                graphics.lineTo( 4, -4 );
                graphics.lineTo( 4, -10 );
            }
            else
            {
                graphics.moveTo( 10, 4 );
                graphics.lineTo( 10, 6 );
                graphics.lineTo( 5, 6 );
                graphics.lineTo( 5, 10 );
                graphics.lineTo( 0, 5 );
                graphics.lineTo( 5, 0 );
                graphics.lineTo( 5, 4 );
                graphics.lineTo( 10, 4 );
            }
            graphics.endFill();
        }
    }
}
```

Значение `direction` определяется способом упорядочения элементов коллекции в родительском компоненте `List`. Если свойство `direction` равно `horizontal`, над изображением элемента рисуется стрелка, направленная вниз. Если свойство `direction` отлично от `horizontal`, считается, что используется вертикальная ориентация, и рисуется стрелка, направленная на запад.

В следующем приложении в список отображения включаются элементы `List` и `TileList`, а свойству `dropIndicatorSkin` каждого компонента назначается пользовательский индикатор из предыдущего примера:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            private function creationHandler():void
            {
                contactList.dataProvider = new ArrayCollection([
                    'Josh', 'Abey', 'Todd'
                ]);
            }
```

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

    <mx:List id="contactList"
        width="200" height="200"
        allowMultipleSelection="true"
        dragEnabled="true"
        dropEnabled="true"
        dropIndicatorSkin="com.oreilly.flexcookbook.CustomDropIndicator"
        />
    <mx:TileList id="tileList"
        width="180" height="200"
        dropEnabled="true"
        dropIndicatorSkin="com.oreilly.flexcookbook.CustomDropIndicator"
        />

</mx:Application>
```

Во время перетаскивания элемента из компонента `List` над `TileList` отображается стрелка, направленная на экземпляр рендерера элемента в позиции, над которой находится указатель мыши. При перетаскивании над компонентом `List` индикатор принимает вид стрелки, направленной вниз к визуальному индексу для размещения исходного объекта.

Свойство `dragIndicatorSkin` является стилевым свойством списковых компонентов и задается так же, как любое другое стилевое свойство, принимающее полное имя класса или класс программного скина для прорисовки нестандартных изображений.

См. также

Рецепты 10.3 и 10.4.

11

Состояния

Состояния – мощный новый механизм, инкапсулирующий большую часть работы по созданию *компонентов с состояниями*, т. е. компонентов, обладающих несколькими представлениями. Это могут быть элементы управления, совмещающие функции редактирования и вывода информации, многостраничные диалоговые окна или компоненты, отображаемые в режиме меню или в режиме подробного вывода. Альтернативные представления, содержащиеся в одном компоненте, называются *состояниями* (states). Flex Framework определяет в пакете `mx.state` класс `State`, который позволяет определять свойства для конкретного представления компонента. В любом компоненте `UIComponent` присутствует массив состояний, в который можно включить один или несколько `mx.state.State`; при помощи этого массива можно легко добавлять и удалять любые дочерние компоненты и стили, а также назначать эффекты и переходы для входа или выхода из состояний. Довольно часто, когда нет реальной необходимости создавать отдельный механизм для хранения состояний и связанных с ними изменений, использование `x.states.State` является гораздо более простым и логичным механизмом реализации нескольких состояний или представлений в одном компоненте.

Состояния могут добавлять дочерние компоненты, удаляемые компонентом сразу же при выходе из состояния. Также возможно определение переходов, воспроизводимых при изменении текущего состояния `currentState` компонента, и применение эффектов к выбранным дочерним компонентам при каждом изменении свойства `state`. Любые свойства компонента, которые являются временными или относятся только к определенному состоянию компонента, легко и эффективно реализуются в состояниях.

Для разработчиков с опытом **Flash-программирования использование состояний** для добавления/удаления дочерних компонентов и изменения

эффектов выглядит довольно странно. Казалось бы, разработчику необходимы более точные средства управления тем, когда и как применяются переходы, эффекты и изменения в списках отображения. Но хотя в отдельных ситуациях повышенный уровень контроля действительно необходим, разумное применение состояний часто экономит время и помогает избежать простых ошибок; процессы добавления/удаления компонентов упрощаются, а код MXML становится более стройным и логичным.

11.1. Назначение стилей и свойств в состояниях

Задача

Требуется задать стиль или свойство, когда некоторое состояние становится текущим, а затем удалить их при выходе из этого состояния.

Решение

Используйте тег `SetStyle` для изменения стилей при входе в состояние и автоматического возврата к предыдущему стилю при выходе из него.

Обсуждение

Теги `SetStyle` и `SetProperty` позволяют задать для любого компонента атрибут стиля или свойства соответственно, активизируемые при входе в состояние. При выходе из состояния стили и свойства, установленные в этом состоянии (а также добавленные в нем дочерние компоненты), возвращаются к базовому состоянию всех свойств за пределами всех состояний.

В тегах `SetStyle` и `SetProperty` задается экземпляр, обладающий устанавливаемым свойством или стилем, а также имя и значение свойства или стиля:

```
<mx:SetProperty target="{this}" name="height" value="500"/>
<mx:SetStyle target="{this}" name="backgroundColor" value="#ccccff"/>
```

Исходное состояние компонента задается при помощи свойства `currentState`:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
currentState="initialState">
```

Все изменения, вносимые в состояние, отменяются при выходе из него (включая и состояние `initialState`). Следующий листинг демонстрирует, что при изменении свойства `currentState` компонента удаляются все компоненты, добавленные в предыдущем состоянии, а также добавляются все компоненты, определенные в новом состоянии, которое задается в свойстве `currentState`:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
currentState="initialState">
```

```
    <mx:Script>
        <![CDATA[
            private var i:int = 1;
            private function cycleStates():void
            {
                switch(i){
                    case 0:
                        currentState = "initialState";
                    break;
                    case 1:
                        currentState = "addImg";
                    break;
                    case 2:
                        currentState = "changeHolderBG";
                    break;
                }
                if(i == 2){i=0;}else{i++;}
            }

        ]]>
    </mx:Script>
    <mx:states>
        <mx:State name="initialState"/>
        <mx:State name="addImg">
            <mx:SetProperty target="{this}" name="height" value="500"/>
            <mx:SetStyle target="{this}"
                name="backgroundColor" value="#ccccff"/>
            <mx:AddChild relativeTo="{mainHolder}">
                <mx:Image source="../assets/image.jpg"/>
            </mx:AddChild>
        </mx:State>
        <mx:State name="changeHolderBG">
            <mx:SetProperty target="{mainHolder}"
                name="height" value="500"/>
            <mx:SetStyle target="{this}"
                name="backgroundColor" value="#ffcccc"/>
            <mx:SetProperty target="{mainHolder}" name="alpha" value="0.5"/>
        </mx:State>
    </mx:states>
    <mx:Button click="cycleStates()" label="change"/>
    <mx:HBox id="mainHolder"/>
</mx:VBox>
```

11.2. Создание переходов для входа и выхода из состояний

Задача

Требуется создать эффекты, воспроизводимые при входе или выходе из состояния.

Решение

Используйте объект Transition и задайте свойства fromState и toState, определяющие время воспроизведения эффекта.

Обсуждение

Переход (*transition*) представляет собой эффект или серию эффектов, воспроизводимых при входе в состояние. Объект Transition содержит свойства fromState и toState, определяющие, в каких состояниях будет воспроизводиться переход. В свойствах fromState и toState могут быть как конкретные состояния, так и метасимвол *, представляющий произвольное состояние.

Существует несколько способов создания объектов Transition, добавления эффектов в компоненты и привязке свойства effect объекта Transition к этому эффекту. В следующем фрагменте кода определяется объект Transition, а его свойство effect связывается с эффектом glow.

```
<mx:Transition id="thirdTrans" fromState="edit" toState="show"
effect="{glow}"/>
```

Обратите внимание: объект Transition, как и State, определяет выполняемые действия. У этих действий могут быть свои приемники, не являющиеся частью Transition. В объекте Transition также могут содержаться теги setPropertyStyle и setPropertyAction, изменяющие стили и свойства (например, высоту и ширину какого-либо из дочерних компонентов или самого текущего компонента):

```
<mx:Transition id="firstTrans" fromState="show" toState="edit">
            <mx:SetPropertyAction target="{holder}"
                name="alpha" value="0"/>
        </mx:Transition>
```

Используйте тег SetStyleAction в тех случаях, когда объект Transition задает стилевое свойство своего компонента-приемника:

```
<mx:Transition id="secondTrans" fromState="*" toState="upload">
            <mx:SetStyleAction target="{holder}"
                name="backgroundColor" value="#ff0000
"/>
</mx:Transition>
```

Следующий фрагмент создает три состояния и определяет объекты Transition для переключения между ними:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
currentState="show">
    <mx:Script>
        <![CDATA[

            [Bindable]
            private var _imgURL:String;
```

```
        [Bindable]
        private var _title:String;

    ]]>
</mx:Script>
<mx:Glow blurXTo="20" blurYTo="20" duration="1000"
            color="0xffff00" id="glow" target="{holder}"/>
<mx:Glow blurXTo="25" blurYTo="25" duration="1000"
            color="0xffffff" id="fade" target="{holder}"/>
```

А в этом примере определяются несколько объектов `Transition` **с заданием свойств** `fromState` **и** `toState`**, определяющими время воспроизведения перехода. В объектах** `Transition` **на программном уровне определяются изменения, вносимые при воспроизведении перехода:**

```
<mx:transitions>
    <mx:Transition id="firstTrans" fromState="show"
        toState="edit">
        <mx:SetPropertyAction target="{holder}"
            name="alpha" value="0"/>
    </mx:Transition>
    <mx:Transition id="secondTrans" fromState="*"
        toState="upload">
        <mx:SetStyleAction target="{holder}"
            name="backgroundColor" value="#ff0000"/>
    </mx:Transition>
    <mx:Transition id="thirdTrans" fromState="edit"
        toState="show" effect="{glow}"/>
    <mx:Transition id="fifthTrans" fromState="upload"
        toState="*" effect="{fade}"/>
</mx:transitions>
<mx:states>
    <mx:State/>
    <mx:State name="show">
        <mx:AddChild relativeTo="{holder}">
            <mx:HBox>
                <mx:Label text="{_title}"/>
                <mx:Image source="{_imgURL}"/>
            </mx:HBox>
        </mx:AddChild>
    </mx:State>
    <mx:State name="edit" exitState="_title =
        input.text;">
        <mx:AddChild relativeTo="{holder}">
            <mx:HBox>
                <mx:TextInput id="input"
                    text="{_title}"/>
            </mx:HBox>
        </mx:AddChild>
    </mx:State>
    <mx:State name="upload">
        <mx:AddChild relativeTo="{holder}">
```

```
                <mx:HBox>
                    <mx:Button label="start upload"/>
                </mx:HBox>
            </mx:AddChild>
        </mx:State>
    </mx:states>
    <mx:ComboBox dataProvider="{['show', 'edit', 'upload']}" change=
        "{this.currentState = cb.selectedItem as String}" id="cb"/>
    <mx:Canvas id="holder"/>
</mx:HBox>
```

11.3. Теги AddChildAction и RemoveChildAction

Задача

Требуется управлять добавлением и удалением дочерних компонентов в серии эффектов, воспроизводимых в процессе перехода.

Решение

Используйте теги `AddChildAction` и `RemoveChildAction` для управления соответственно добавлением и удалением дочерних компонентов в `Sequence`.

Обсуждение

Объекты `AddChildAction` и `RemoveChildAction` по своим функциям напоминают объекты `SetPropertyAction` и `SetPropertyStyle` — они тоже позволяют получить доступ к функциональности, обычно инкапсулированной в объектах `State`, и переместить ее в `Transition` для интеграции с объектами `Parallel` или `Sequence`, содержащимися в `Transition`.

Использование тега `AddChild` в `State` по умолчанию приводит к добавлению дочернего компонента сразу же после входа в состояние. Чтобы управлять временем добавления (или воспроизвести некоторые эффекты до момента добавления/удаления дочернего компонента), вместо включения тега `AddChild` в `State` можно воспользоваться тегом `AddChildAction` в `Sequence`. Например, следующий фрагмент описывает порядок выполнения объекта `Transition`:

```
<mx:Transition fromState="view" toState="edit">
    <mx:Sequence>
        <mx:Fade alphaFrom="1" alphaTo="0" duration="1000"
            target="{viewCanvas}"/>
        <mx:RemoveChildAction  target="{viewCanvas}"/>
        <mx:AddChildAction relativeTo="{this}">
            <mx:target>
                <mx:Canvas id="editCanvas" addedToStage=
                    "editCanvas.includeInLayout = true" removedFromStage=
                    "editCanvas.includeInLayout = false">
                    <mx:TextInput text="SAMPLE"/>
```

```
                </mx:Canvas>
            </mx:target>
        </mx:AddChildAction>
    </mx:Sequence>
</mx:Transition>
```

В этом примере сначала выполняется эффект Fade, затем удаляется дочерний компонент, после чего добавляется новый дочерний компонент. При использовании тега AddChild в State дочерний компонент будет добавлен до завершения эффекта Fade, а это не то, что требуется в нашем примере. Использование AddChildAction дает возможность разработчику управлять добавлением дочерних компонентов.

Обратите внимание: тег RemoveChild управляет тем, *когда* будет удален дочерний компонент, но сам удаления не выполняет. Механизм AddChildAction не обеспечивает автоматического вызова remove. Чтобы удалить дочерний компонент, необходимо добавить объект RemoveChild в State следующим образом:

```
<mx:State name="edit">
    <mx:RemoveChild target="{viewCanvas}"/>
</mx:State>
```

Полный код примера выглядит так:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300"
currentState="view">
    <mx:transitions>
        <mx:Transition fromState="view" toState="edit">
            <mx:Sequence>
                <mx:Fade alphaFrom="1" alphaTo="0"
                    duration="1000" target="{viewCanvas}"/>
                <mx:RemoveChildAction  target="{viewCanvas}"
                    effectStart="trace('removing')"/>
                <mx:AddChildAction relativeTo="{this}">
                    <mx:target>
                        <mx:Canvas id="editCanvas"
                            addedToStage="editCanvas.
                            includeInLayout = true"
                            removedFromStage=
                            "editCanvas.includeInLayout = false">
                            <mx:TextInput text="SAMPLE"/>
                        </mx:Canvas>
                    </mx:target>
                </mx:AddChildAction>
                <mx:SetPropertyAction target="{editCanvas}"
                    name="includeInLayout" value="true"/>
                <mx:SetPropertyAction target="{editCanvas}"
                    name="alpha" value="1"/>
                <mx:Glow color="0xffff00" blurXTo="30"
                    blurYTo="30" blurXFrom="0"
                    blurYFrom="0" duration="1000" target="{this}"/>
```

```
                <mx:Glow color="0xffff00" blurXTo="30"
                blurYTo="30" blurXFrom="0" blurYFrom="0"
                duration="1000" target="{editCanvas}"/>
            </mx:Sequence>
        </mx:Transition>
        <mx:Transition fromState="edit" toState="view">
            <mx:Sequence>
                <mx:Fade alphaFrom="1" alphaTo="0"
                duration="1000" target="{editCanvas}"/>
                <mx:RemoveChildAction
                target="{editCanvas}"/>
                <mx:AddChildAction relativeTo="{this}"
                effectStart="trace('removing')">
                    <mx:target>
                        <mx:Canvas id="viewCanvas"
                        addedToStage="viewCanvas.
                        includeInLayout = true"
                        removedFromStage=
                        "viewCanvas.includeInLayout =
                        false">
                            <mx:Text text="DIFFERENT TEXT"/>
                        </mx:Canvas>
                    </mx:target>
                </mx:AddChildAction>
                <mx:SetPropertyAction target="{viewCanvas}"
                    name="includeInLayout" value="true"/>
                <mx:SetPropertyAction target="{viewCanvas}"
                    name="alpha" value="1"/>
                <mx:Glow color="0xffff00" blurXTo="30"
                    blurYTo="30" blurXFrom="0"
                    blurYFrom="0" duration="1000"
                    target="{this}"/>
                <mx:Glow color="0xffff00" blurXTo="30"
                    blurYTo="30" blurXFrom="0"
                    blurYFrom="0" duration="1000"
                    target="{viewCanvas}"/>
            </mx:Sequence>
        </mx:Transition>
    </mx:transitions>
    <mx:states>
        <mx:State name="view">
            <mx:RemoveChild target="{editCanvas}"/>
        </mx:State>
        <mx:State name="edit">
            <mx:RemoveChild target="{viewCanvas}"/>
        </mx:State>
    </mx:states>
    <mx:ComboBox dataProvider="{['view', 'edit']}"
        change="currentState =
        cb.selectedItem as String" id="cb"/>
</mx:HBox>
```

11.4. Фильтрация переходов по типам дочерних компонентов

Задача

Требуется, чтобы переход применялся только к определенным типам дочерних компонентов в списке приемников.

Решение

Используйте объект `EffectTargetFilter` для определения функции `filter`, которая определяет, к каким типам приемников должен (или не должен) применяться тот или иной эффект в составе перехода.

Обсуждение

Объект `EffectTargetFilter` позволяет задавать фильтры, определяющие приемники для воспроизведения переходов. Для работы объекта `EffectTargetFilter` необходима функция фильтрации, которая возвращает `true` или `false` для каждого объекта, переданного `Transition`. Функция фильтрации назначается свойству `filterFunction` объекта `EffectTargetFilter` следующим образом:

```
filter.filterFunction = func;
private function func(propChanges:Array,
    instanceTarget:Object):Boolean
{
    if(instanceTarget is HBox)

    {
        return true;
    }
    return false;
}
```

В функцию можно включить команду условной проверки, определяющую, должна ли функция возвращать `true` в случае применения к объекту `Sequence` или `Parallel`. Учтите, что функция в общем случае не может применяться к отдельным эффектам в объекте `Transition` (за дополнительной информацией обращайтесь к рецепту 11.5).

Объект `EffectTargetFilter` передается в теге `Parallel` или `Sequence` следующим образом:

```
<mx:Sequence filter="resize" targets="{[one, two, three]}"
customFilter="{filter}">
```

При активизации `Sequence` объект `customFilter` возвращает все объекты, на которые должно распространяться действие фильтра. Следует помнить, что функция фильтрации в случае необходимости может неоднократно меняться на протяжении жизненного цикла компонента.

Полный код примера:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="comp()" xmlns:cookbook="oreilly.cookbook.*">
    <mx:Script>
        <![CDATA[
            import mx.containers.HBox;
            import mx.effects.EffectTargetFilter;

            [Bindable]
            private var filter:EffectTargetFilter;

            private function comp():void {
                filter = new EffectTargetFilter();
                filter.filterFunction = func;
            }
            private function func(propChanges:Array,
                instanceTarget:Object):Boolean {
                if(instanceTarget is HBox) {
                    return true;
                }
                return false;
            }

        ]]>
    </mx:Script>
    <mx:transitions>
        <mx:Transition toState="closeState"
             fromState="openState">
             <mx:Sequence filter="resize" targets="{[one, two,
                    three]}" customFilter="{filter}">
                 <mx:Move xTo="200" xFrom="0"/>
                 <mx:Glow color="0xffff00" blurXTo="20" blurYTo="20"/>
             </mx:Sequence>
        </mx:Transition>
    </mx:transitions>
    <mx:states>
        <mx:State name="closeState"/>
        <mx:State name="openState"/>
    </mx:states>
    <mx:HBox id="one">
        <mx:Text text="one"/>
        <mx:Text text="two"/>
    </mx:HBox>
    <mx:HBox id="two">
        <mx:Text text="one"/>
        <mx:Text text="two"/>
    </mx:HBox>
    <mx:Canvas id="three">
        <mx:Text text="one"/>
        <mx:Text text="two" y="10"/>
    </mx:Canvas>
```

```
        <mx:Button click="(currentState == 'openState') ?
            currentState = 'closeState' : currentState = 'openState'"/>
    </mx:VBox>
```

См. также

Рецепт 11.6.

11.5. Частичное применение перехода к некоторым дочерним компонентам

Задача

Требуется применить отдельные части объекта `Transition`, `Sequence` или `Parallel` только к определенным дочерним компонентам.

Решение

Отфильтруйте приемники каждого эффекта при помощи функции-фильтра, возвращающей массив всех дочерних компонентов, удовлетворяющих критерию.

Обсуждение

Как упоминалось в рецепте 11.4, фильтрация объекта `EffectTargetFilter` применяется либо ко всей последовательности, либо ко всему составному эффекту `Parallel`. Чтобы отфильтровать приемники отдельных эффектов, составляющих переход, необходимо написать пользовательскую функцию, которая возвращает массив, используемый для задания свойства `target` каждого эффекта. Поскольку эффект будет иметь собственные приемники, заданные независимо от приемников `Transition`, функции фильтрации приходится в цикле перебирать все дочерние компоненты. Эта операция может оказаться весьма затратной, поэтому для определения дочерних компонентов, на которые должен воздействовать эффект, лучше воспользоваться независимым массивом.

Функция фильтрации в этом примере перебирает все дочерние компоненты и в зависимости от переданного параметра возвращает список всех объектов `HBox` или всех объектов `Canvas`, содержащихся в компоненте:

```
private function returnArray(state:*):Array
{
    var arr:Array = new Array();
    var i:int;
    if(state == "foo") {
        for(i = 0; i<this.numChildren; i++) {
            if(getChildAt(i) is HBox) {
                arr.push(getChildAt(i));
            }
        }
```

```
                    } else {
                        for(i = 0; i<this.numChildren; i++) {
                            if(getChildAt(i) is Canvas) {
                                arr.push(getChildAt(i));
                            }
                        }
                    }
                    return arr;
                }
```

Эффект с вызовом метода выглядит так:

```
<mx:Move xTo="200" xFrom="0" targets="{returnArray('foo')}"/>
<mx:Glow color="0xffff00" blurXTo="20" blurYTo="20"
    targets="{returnArray('bar')}"/>
```

При каждой активизации эффекта массив создается заново; это означает, что один объект `Effect` может распространяться на разные типы дочерних компонентов в зависимости от текущего значения `currentState` компонента. Таким образом, вы можете создать один объект `Effect` и использовать его разными способами в зависимости от функции фильтрации, возвращающей все приемники для этого эффекта. Пример:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
        <mx:Script>
            <![CDATA[
                import mx.core.UIComponent;

                [Bindable]
                private function returnArray(state:*):Array
                {
                    var arr:Array = new Array();
                    var i:int;
                    if(state == "foo") {
                        for(i = 0; i<this.numChildren; i++) {
                            if(getChildAt(i) is HBox) {
                                arr.push(getChildAt(i));
                            }
                        }
                    } else {
                        for(i = 0; i<this.numChildren; i++) {
                            if(getChildAt(i) is Canvas) {
                                arr.push(getChildAt(i));
                            }
                        }
                    }
                    return arr;
                }
            ]]>

        </mx:Script>
        <mx:transitions>
```

```
            <mx:Transition toState="closeState" fromState="openState">
                <mx:Sequence>
                    <mx:Move xTo="200" xFrom="0"
                    targets="{returnArray('foo')}"/>
                    <mx:Glow color="0xffff00" blurXTo="20"
                    blurYTo="20" targets="{returnArray('bar')}"/>
                </mx:Sequence>
            </mx:Transition>
        </mx:transitions>
        <mx:states>
            <mx:State name="closeState"/>
            <mx:State name="openState"/>
        </mx:states>
        <mx:HBox id="one">
            <mx:Text text="one"/>
            <mx:Text text="two"/>
        </mx:HBox>
        <mx:HBox id="two">
            <mx:Text text="one"/>
            <mx:Text text="two"/>
        </mx:HBox>
        <mx:Canvas id="three">
            <mx:Text text="one"/>
            <mx:Text text="two" y="10"/>
        </mx:Canvas>
        <mx:Canvas id="four">
            <mx:Text text="three"/>
            <mx:Text text="four" x="10" y="10"/>
        </mx:Canvas>
        <mx:Button click="(currentState == 'openState') ? currentState =
            'closeState': currentState = 'openState'"/>
</mx:VBox>
```

См. также

Рецепт 11.4.

11.6. Определение состояния на базе другого состояния

Задача

Требуется создать состояние, наследующее все свойства другого состояния, с переопределением некоторых свойств.

Решение

Задайте в свойстве `basedOn` объект `State`, от которого создаваемый объект `State` наследует свои свойства.

Обсуждение

Создание состояний на базе других состояний – удобный способ реализации некоего аналога наследования. Состояние, базирующееся на другом состоянии, наследует все его переопределения, дополняя их своими собственными. Если состояние определяет метод `AddChild`, а затем другое состояние, базирующееся на нем, определяет собственный метод `AddChild`, все операции метода `AddChild` исходного состояния будут присутствовать в новом состоянии, как и все прочие переопределения нового состояния.

Создать состояние на базе другого состояния несложно:

```
<mx:State name="secondaryState2" basedOn="primaryState">
```

В примере ниже объект `secondaryState2` добавляет свойство с именем, уже присутствующим в `primaryState`, и для обоих свойств активируются назначения `SetProperty`. `SetProperty` для `secondaryState2` активируется последним, и `title` будет содержать строку «Third Title», а не «Super New Title».

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="500"
title="Initial Title">
    <mx:states>
        <mx:State name="primaryState">
            <mx:AddChild>
                <mx:VBox>
                    <mx:Text fontSize="18" text="NEW TEXT 1"/>
                    <mx:Text fontSize="18" text="NEW TEXT 2"/>
                </mx:VBox>
            </mx:AddChild>
            <mx:SetProperty target="{this}"
                name="title" value="'Super New Title'"/>
        </mx:State>
        <mx:State name="secondaryState1">
            <mx:AddChild>
                <mx:RichTextEditor height="300" width="250"/>
            </mx:AddChild>
            <mx:SetProperty target="{this}" name="title" value="'Lame Old Title'"/>
        </mx:State>
        <mx:State name="secondaryState2" basedOn="primaryState">
            <mx:SetProperty target="{this}" name="title" value="'Third Title'"/>
        </mx:State>
    </mx:states>
    <mx:ComboBox dataProvider="{['primaryState', 'secondaryState1',
'secondaryState2']}" change="currentState=cb.selectedItem as String" id="cb"/>
</mx:Panel>
```

11.7. Интеграция состояний с HistoryManagement

Задача

Требуется интегрировать состояния с механизмом Flex Framework `HistoryManagement`.

Решение

Создайте приложение или компонент, реализующие интерфейс IHistoryManagerClient. Используя HistoryManagement, зарегистрируйте приложение в HistoryManager. Используйте метод HistoryManager.save для сохранения текущего состояния при внесении изменений.

Обсуждение

Клиент IHistoryManager определяет следующие методы:

`loadState(state:Object):void`

Загружает состояние объекта.

`saveState():Object`

Сохраняет состояние объекта.

`toString():String`

Преобразует объект в уникальную строку.

Эти методы позволяют компоненту сохранить в State всю информацию, которая позднее может потребоваться для восстановления состояния. Метод loadState загружает информацию State по URL-адресу. В следующем методе saveState сохраняется как State, так и текущее содержимое TextInput:

```
        public function saveState():Object {
            trace(" save state ");
            var state:Object = {};
            state.lastSearch = lastSearch;
            state.currentState = currentState;
            return state;
}
```

Метод loadState компонента читает State из объекта, полученного от HistoryManager, и задает currentState компонента:

```
public function loadState(state:Object):void {
            if (state) {
                trace(" last search "+state.lastSearch);
                lastSearch = searchInput.text = state.lastSearch;
                currentState = state.currentState;
            }
}
```

Полный код примера:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="mx.managers.IHistoryManagerClient"
    creationComplete=" HistoryManager.register(this)"
    currentState="search">
    <mx:Script>
        <![CDATA[
        import mx.managers.HistoryManager;
```

```
        public function loadState(state:Object):void {
            if (state) {
                trace(" last search "+state.lastSearch);
                lastSearch = searchInput.text = state.lastSearch;
                currentState = state.currentState;
            }
        }
        // Сохранение текущего состояния и значения searchString.
        public function saveState():Object {
            trace(" save state ");
            var state:Object = {};
            state.lastSearch = lastSearch;
            state.currentState = currentState;
            return state;
        }
        // Текущая строка поиска
        [Bindable]
        public var lastSearch:String;
        public function search():void {
            currentState = "display";
            lastSearch = searchInput.text;
            HistoryManager.save();
        }
        public function reset():void {
            trace(" reset ");
            currentState = 'search';
            searchInput.text = "";
            lastSearch = "";
            HistoryManager.save();
        }
        ]]>
    </mx:Script>
    <mx:states>
        <mx:State name="display">
            <mx:SetProperty target="{panel}" name="title" value="Results"/>
            <mx:AddChild relativeTo="{panel}">
                  <mx:VBox id="results">
                      <mx:Text text="Getting Results"/>
                      <mx:Button label="Reset" click="reset()"/>
                  </mx:VBox>
            </mx:AddChild>
        </mx:State>
        <mx:State name="search">
            <mx:SetProperty target="{panel}" name="title" value="Search"/>
            <mx:AddChild relativeTo="{panel}">
                <mx:HBox id="searchFields" defaultButton="{btn}">
                    <mx:TextInput id="searchInput" />
                    <mx:Button id="btn" label="Find" click="search();" />
                </mx:HBox>
            </mx:AddChild>
        </mx:State>
```

```
    </mx:states>
    <mx:Panel id="panel" title="Results" resizeEffect="Resize">
    </mx:Panel>
</mx:Application>
```

11.8. Фабрики экземпляров для состояний

Задача

Требуется создать объект, способный создавать экземпляры разных типов объектов для `AddChild`.

Решение

Создайте класс-фабрику и назначьте его свойству `targetFactory` объекта `AddChild`.

Обсуждение

Свойство `targetFactory` объекта `AddChild` должно получать объект, реализующий интерфейс `IDeferredInstance`. Интерфейс `IDeferredInstance` состоит из единственного метода `getInstance():Object`. Метод возвращает экземпляр объекта, который создается, когда объект `AddChild` запрашивает новый объект для включения в список отображения компонента.

Рассмотрим относительно простой класс, который возвращает разные типы `UIComponent` в зависимости от свойства `type`:

```
package oreilly.cookbook
{
    import mx.containers.HBox;
    import mx.containers.VBox;
    import mx.controls.Button;
    import mx.controls.Text;
    import mx.controls.TextInput;
    import mx.core.IDeferredInstance;
    import mx.core.UIComponent;

    public class SpecialDeferredInstance implements
        IDeferredInstance
    {
        private var comp:UIComponent;
        private var _type:String;

        public function set type(str:String):void {
            _type = str;
        }
        public function get type():String{
            return _type;
        }
        public function getInstance():Object
        {
```

```
            var text:Object;
            if(_type == "TextVBox"){
                comp = new VBox();
                text = new Text();
                text.text = "TEXT";
                comp.addChild(text as Text);
                var btn:Button = new Button();
                btn.label = "LABEL";
                comp.addChild(btn);
                comp.height = 160;
                comp.width = 320;
            } else {
                comp = new HBox();
                text = new TextInput();
                text.text = "TEXT";
                comp.addChild(text as TextInput);
                var btn:Button = new Button();
                btn.label = "LABEL";
                comp.addChild(btn);
                comp.height = 160;
                comp.width = 320;
            }
            return comp;
        }
    }
}
```

С заданным свойством `targetFactory` **метод** `AddChild` **может использовать разные типы объектов, определяемые параметром** `type` **объекта** `SpecialDeferredInstance`**. Пример:**

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
currentState="empty">
    <mx:Script>
        <![CDATA[
            import oreilly.cookbook.SpecialDeferredInstance;
            [Bindable]
            private var defInst:SpecialDeferredInstance =
                new SpecialDeferredInstance();
        ]]>
    </mx:Script>
    <mx:states>
        <mx:State name="defInst">
            <mx:AddChild relativeTo="{mainHolder}"
                targetFactory="{defInst}"/>
        </mx:State>
        <mx:State name="empty"/>
    </mx:states>
    <mx:Button click="currentState == 'defInst' ? currentState = 'empty'
         : currentState = 'defInst'" label="change"/>
    <mx:HBox id="mainHolder"/>
</mx:VBox>
```

11.9. Привязка данных для объектов, добавленных в состоянии

Задача

Требуется организовать привязку к свойству объекта, который будет создан только при входе в определенное состояние.

Решение

Используйте метод `bindProperty` класса `mx.binding.utils.BindingUtils` для динамического создания привязки по нужному свойству.

Обсуждение

Привязка может создаваться как во время компиляции (запись с фигурными скобками в MXML), так и во время выполнения, с использованием метода `bindProperty`. Метод `bindProperty` обладает следующей сигнатурой:

```
public static function bindProperty(site:Object, prop:String,
    host:Object, chain:Object, commitOnly:Boolean =
    false):ChangeWatcher
```

Параметры метода:

`site`

Объект, который определяет свойство, привязываемое к `chain`. Например, если привязка должна использоваться для изменения свойства `text` компонента `TextField`, параметр `site` должен содержать значение `TextField`.

`prop`

Имя открытого свойства, определенного в объекте `site`. Свойство получает текущее значение `chain` при изменении значения `chain`. Например, если привязка должна использоваться для изменения свойства `text` компонента `TextField`, параметр `prop` должен содержать значение `text`.

`host`

Объект, содержащий отслеживаемое свойство или цепочку свойств. Например, в случае привязки к значению `text` компонента `TextInput` параметр должен содержать `TextInput`.

`chain`

Значение, определяющее отслеживаемое свойство или цепочку. Параметр может содержать строку открытого привязываемого свойства объекта `host`. В случае привязки к значению `text` компонента `TextInput` параметр должен содержать `text`.

commitOnly

Задается равным true, если обработчик должен вызываться только для закрепления событий изменения.

Привязка данных более подробно рассматривается в главе 14. В этом рецепте мы ограничимся использованием bindProperty для привязки вновь созданного компонента RichTextEditor к TextArea:

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    width="450" height="650" title="Initial Title" layout="vertical">
    <mx:Script>
        <![CDATA[
            import mx.binding.utils.*;

        ]]>
    </mx:Script>
    <mx:states>
        <mx:State name="primaryState">
            <mx:AddChild>
                <mx:VBox id="vbox">
                    <mx:Text fontSize="18" text="NEW TEXT 1"/>
                    <mx:Text fontSize="18" text="NEW TEXT 2"/>
                </mx:VBox>
            </mx:AddChild>
            <mx:SetProperty target="{this}" name="title"
                value="'Super New Title'"/>
        </mx:State>
        <mx:State name="secondaryState1">
            <mx:AddChild>
```

В этом примере свойство htmlText компонента TextArea привязывается к свойству htmlText только что созданного компонента RichTextEditor при помощи метода bindProperty. Если вы попытаетесь привязать свойство htmlText компонента TextArea к компоненту, который еще не был создан, произойдет ошибка. Описанный механизм позволяет осуществлять привязку к вновь создаваемым компонентам только после того, как они будут созданы:

```
                <mx:RichTextEditor id="richText" height="200" width="250"
                creationComplete="BindingUtils.bindProperty(
                area, 'htmlText', richText, 'htmlText')"/>
            </mx:AddChild>
            <mx:SetProperty target="{this}" name="title"
                value="'Lame Old Title'"/>
        </mx:State>
        <mx:State name="secondaryState2"
            basedOn="primaryState">
            <mx:SetProperty target="{this}" name="title"
                value="'Third Title'"/>
        </mx:State>
    </mx:states>
    <mx:ComboBox dataProvider="{['primaryState',
```

```
            'secondaryState1', 'secondaryState2']}"
            change="currentState=cb.selectedItem as String" id="cb"/>
        <mx:TextArea height="100" width="450"
            id="area" htmlText="foo bar baz"/>
    </mx:Panel>
```

11.10. Добавление и удаление слушателей событий при изменении состояний

Задача

Требуется добавить слушателей событий к компонентам, создаваемым и добавляемым при изменении состояния, и удалить их при переходе в следующее состояние.

Решение

Добавьте слушателей событий в событии `addedToStage` компонента и удалите их в событии `removedFromStage`. Другой способ: воспользуйтесь объектом `SetEventHandler` для создания слушателя события, который будет удален при выходе из состояния.

Обсуждение

Своевременное удаление обработчиков событий в ActionScript 3 – один из лучших способов предотвратить утечку памяти в приложениях, когда Flash Player захватывает все больше памяти, не освобождая ее. Следовательно, добавление обработчиков событий в компонентах всегда должно сопровождаться их организованным удалением при уничтожении компонента. Например, компоненты можно просто добавлять и удалять при добавлении компонента на сцену и его последующем удалении:

```
<mx:AddChild relativeTo="{holder}">
    <mx:TextInput text="TEXT" id="textInput1" width="200"
        addedToStage="{textInput1.
        addEventListener(TextEvent.TEXT_INPUT,
        checkNewTextInput)}" removedFromStage="
        {textInput2.addEventListener(
        TextEvent.TEXT_INPUT, checkNewTextInput) }" />
</mx:AddChild>
```

Того же эффекта можно добиться при помощи объекта `SetEventHandler`, которому передается приемник с присоединенным обработчиком, имя прослушиваемого события и функция `handlerFunction`, обрабатывающая переданное событие:

```
<mx:SetEventHandler handlerFunction="checkNewTextInput"
    name="{TextEvent.TEXT_INPUT}" target="{textInput2}"/>
```

Смена состояний приводит к добавлению и удалению обработчика. Чтобы убедиться в этом, приведите прослушиваемое событие к типу `Event`.

ENTER_FRAME; это позволит точно узнать, когда именно происходит добавление и удаление слушателя. Полный код примера:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Script>
        <![CDATA[
            private function
            checkNewTextInput(event:Event):void
            {
                trace(" event "+event.target);
            }
        ]]>
    </mx:Script>
    <mx:states>
        <mx:State id="openState" name="openState">
            <mx:AddChild relativeTo="{holder}">
                <mx:TextInput text="TEXT"
                id="textInput1" width="200" addedToStage=
                "{textInput1.addEventListener(
                TextEvent.TEXT_INPUT, checkNewTextInput)}"
                     removedFromStage="{
                     textInput2.addEventListener
                     (TextEvent.TEXT_INPUT, checkNewTextInput)}"/>
            </mx:AddChild>
        </mx:State>
        <mx:State id="closedState" name="closedState">
            <mx:SetEventHandler handlerFunction=
                 "checkNewTextInput" name=
                 "{TextEvent.TEXT_INPUT}"
                 target="{textInput2}"/>
            <mx:AddChild relativeTo="{holder}">
                <mx:TextInput id="textInput2" width="200"
                     text="MORE TEXT"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>
    <mx:VBox id="holder">
        <mx:Button click="(this.currentState == 'openState')
             ? currentState = 'closedState'
             : currentState = 'openState'" label="change"/>
    </mx:VBox>
</mx:Canvas>
```

11.11. Добавление состояний в компоненты Flash

Задача

Компонент импортируется из среды разработки Flash в приложение Flex. Требуется использовать определенные кадры компонента Flash как состояния.

Решение

Назначьте метки тем кадрам, которые должны использоваться как состояния, в экземплярах `UIMovieClip` или `ContainerMovieClip` в приложении Flex.

Обсуждение

Сначала создайте класс, экземпляр которого будет создаваться в приложении Flex. Класс должен расширять либо `UIMovieClip`, если в приложении Flex в него не включаются дочерние компоненты, либо `ContainerMovieClip` в противном случае. Пример:

```
package
{
    import flash.text.TextField;
    import mx.flash.ContainerMovieClip;

    public class FlashAssetClass extends ContainerMovieClip{
        private var txt:TextField;
        public function FlashAssetClass() {
                txt = new TextField();
                addChild(txt);
                txt.text = "INIT";
                super();
        }

        override public function set currentState(value:String):void {
            trace(" set current state ");
            super.currentState = value;
            txt.text = value;
        }
        override public function gotoAndStop(frame:Object,
                scene:String=null):void {
            trace(" go to and stop ");
            txt.text = String(frame);
            super.gotoAndStop(frame, scene);
        }
        }
    }
```

Переопределенные версии методов `currentState` и `gotoStop` выводят трассировочную информацию и изменяют добавленный компонент `TextField`. На рис. 11.1 изображен клип Flash с несколькими помеченными кадрами (First, Second и Third), распознаваемыми как состояния в приложениях Flex.

Чтобы использовать кадры в компоненте Flash, вам понадобится следующий код MXML:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="400" creationComplete="createComp()">
```

```
    <mx:Script>
        <![CDATA[
            import mx.controls.Label;
            import FlashAssetClass;
            private var classInst:FlashAssetClass;
            private function createComp():void {
                classInst = new FlashAssetClass();
                rawChildren.addChild(classInst);
                invalidateDisplayList();
            }
        ]]>
    </mx:Script>
    <mx:Button click="classInst.currentState = 'First';"
        label="First" y="300"/>
    <mx:Button click="classInst.currentState = 'Second';"
        label="First" y="330"/>
    <mx:Button click="classInst.currentState = 'Third';"
        label="First" y="360"/>
</mx:Canvas>
```

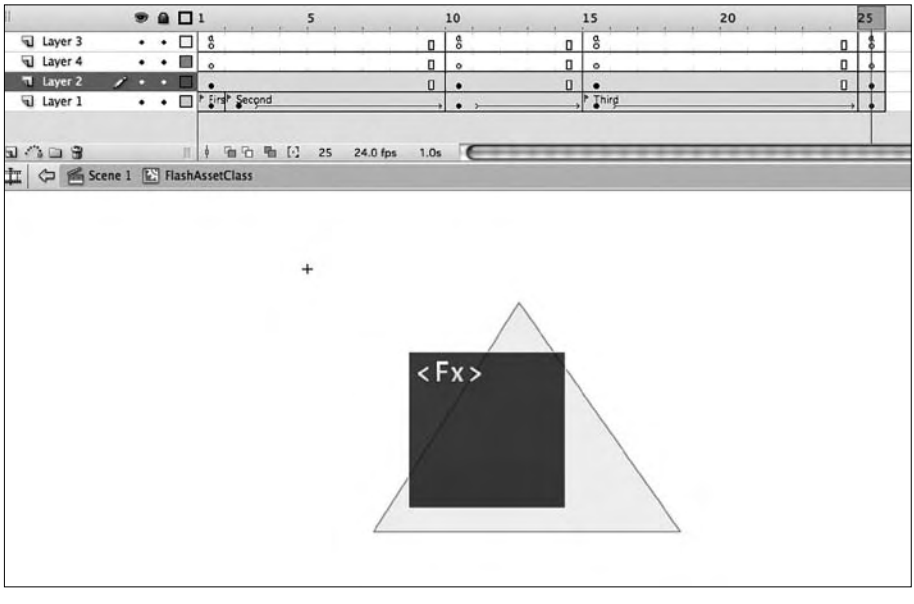

***Рис. 11.1.** Создание кадров в Flash IDE*

Обратите внимание: для изменения состояния объекта `FlashAssetClass` достаточно просто задать свойство `currentState`, как и для любого другого компонента.

При изменении состояния класс `ContainerMovieClip` по умолчанию вызывает метод `gotoAndStop` для остановки на кадрах, созданных в Flash IDE. Если вы хотите настроить `currentState` для воспроизведения анимации, созданной в Flash IDE, переопределите set-метод `currentState` и включите

в него вызов метода gotoAndPlay, который не останавливает воспроизведение и отображает анимацию, подготовленную в Flash IDE. Пример:

```
override public function set currentState(value:String):void
{
    trace(" set current state ");
    gotoAndPlay(value);
    txt.text = value;
}
```

Если вы используете метод gotoAndPlay, не забудьте включить вызовы метода stop класса ContainerMovieClip, чтобы воспроизведение своевременно остановилось; в противном случае клип начнет снова воспроизводиться с первого кадра.

11.12. Работа с событиями изменения состояния

Задача

Необходимо понять смысл событий, сопровождающих изменение состояния, и научиться пользоваться ими.

Решение

Включите трассировочные команды в события ENTER_STATE и CREATION_COMPLETE дочерних компонентов. Полученная информация покажет, что дочерние компоненты были созданы когда состояние вводилось или перемещалось, но не уничтожалось.

Обсуждение

Серии событий, сопровождающие смену состояния, довольно сложны. События, передаваемые самим объектом State, смешиваются с событиями creation, addedToStage и removedFromStage от дочерних компонентов. Разобравшись во взаимосвязи этих событий, вы получите очень четкое представление о том, как в Flex Framework организовано изменение состояния компонента. События, передаваемые при создании объектов State, входе и выходе в состояния:

mx.events.StateChangeEvent.CURRENT_STATE_CHANGE

Передается после изменения состояния.

mx.events.StateChangeEvent.CURRENT_STATE_CHANGING

Передается после изменения свойства currentState, но перед изменением состояния.

События State:

mx.events.FlexEvent.ENTER_STATE

Передается при входе в состояние. Событие передается после внесения изменений в базовое состояние.

`mx.events.FlexEvent.EXIT_STATE`

Передается перед входом в состояние. Событие передается перед удалением изменений в базовом состоянии.

События дочерних компонентов:

`mx.events.FlexEvent.ADD`

Передается при добавлении компонента в контейнер методом `addChild` или `addChildAt`.

`mx.events.FlexEvent.REMOVE`

Передается при удалении компонента из контейнера методом `removeChild` или `removeChildAt`.

`mx.events.FlexEvent.PREINITIALIZE`

Передается в начале инициализации компонента.

`mx.events.FlexEvent.INITIALIZE`

Передается при завершении конструирования компонента и задания всех свойств инициализации.

`mx.events.FlexEvent.CREATION_COMPLETE`

Передается при завершении конструирования компонента, обработки свойств, вычисления размеров, формирования раскладки и прорисовки.

При переходе компонента из базового состояния (того состояния, в котором компонент находится при создании) во второе состояние происходит следующая последовательность событий:

```
[child] constructor()
[component] CURRENT_STATE_CHANGING;
[child] ADD;
[child] PREINITIALIZE;
[child] createChildren();
[child] INITIALIZE;
[state] ENTER_STATE; (second state)
[component] CURRENT_STATE_CHANGE;
[child] commitProperties();
[child] updateDisplayList();
[child] CREATION_COMPLETE;
```

При возвращении компонента из второго состояния к базовому события и вызовы методов следуют в следующем порядке:

```
[component] CURRENT_STATE_CHANGING;
[state] EXIT_STATE; (second state)
[child] REMOVE;
[component] CURRENT_STATE_CHANGE;
```

На этот раз событий не так уж много. Мы видим событие `EXIT_STATE` при выходе компонента из второго состояния, но событие `ENTER_STATE` при возврате к базовому состоянию не передается. Обратите внимание: хотя

в первой последовательности событие ENTER_STATE передается при входе компонента во второе состояние, аналогичное событие EXIT_STATE для базового состояния компонента отсутствует. При возврате компонента ко второму состоянию будет получен следующий результат:

```
[component] CURRENT_STATE_CHANGING;
[child] ADD;
[state] ENTER_STATE; (second state)
[component] CURRENT_STATE_CHANGE;
[child] updateDisplayList();
```

Очень важно понять различия между первым и вторым входом компонента во второе состояние; дочерние компоненты при этом не создаются заново, а только повторно добавляются в родительский компонент.

11.13. Динамическое построение и использование новых состояний и переходов

Задача

Требуется динамически генерировать новые состояния и переходы, связанные с этими состояниями.

Решение

Создайте новые объекты State и Transition, добавьте свойства и включите их в массивы состояний и переходов, определяемые всеми объектами UIComponent.

Обсуждение

Создавать новые состояния и переходы приходится нечасто. Впрочем, в некоторых ситуациях эта возможность может оказаться полезной, например, при создании компонентов-шаблонов. Поскольку каждый объект UIComponent содержит массивы состояний и переходов со всеми объектами State и Transition для данного компонента, создание новых состояний и переходов сводится к простому определению объектов State и Transition и их занесению в соответствующий массив:

```
var state:State = new State();
var button:Button = new Button();
button.label = "LABEL";
var addChild:AddChild = new AddChild(vbox, button);
state.overrides = [addChild];
state.name = "buttonState";
states.push(state);
```

Использованный в этом примере массив overrides содержит список переопределений, которые будут использоваться State, т. е. описывает определяемую функциональность. Другие свойства, определяемые

конкретным объектом State, **например действия** SetProperty **и** SetStyle, **также добавляются в массив переопределений. Все переопределения, определяемые в** State**, представляют собой объекты, реализующие интерфейс** IOverride**; обычно к их числу относятся:**

- AddChild
- RemoveChild
- SetEventHandler
- SetProperty
- SetStyle

За дополнительной информацией об интерфейсе IOverride **обращайтесь к рецепту 11.15.**

В приведенном далее листинге обратите особое внимание на методы создания объектов Transition **и добавления свойств (конкретнее, свойств** toState **и** fromState**):**

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.controls.Button;
            import mx.states.AddChild;
            import mx.states.State;
            import mx.states.Transition;
            import mx.effects.Move;

            public function addTransitions():void {
                var transition:Transition = new Transition();
                var move:Move = new Move();
                move.duration=400;
                move.target = vbox;
                transition.fromState = "buttonState";
                transition.toState = "*";
                transition.effect = move;
                transitions.push(transition);
            }

            public function addState():void {
                var state:State = new State();
                var button:Button = new Button();
                button.label = "LABEL";
                var addChild:AddChild = new AddChild(vbox, button);
                state.overrides = [addChild];
                state.name = "buttonState";
                states.push(state);
            }
        ]]>
    </mx:Script>
    <mx:VBox id="vbox"/>
    <mx:Button click="addTransitions()" label="new transition"/>
```

```
        <mx:Button click="addState()" label="new state"/>
        <mx:Button click="currentState = 'buttonState'" label="change state"/>
    </mx:VBox>
```

См. также

Рецепт 11.15.

11.14. Создание пользовательских действий для состояний

Задача

Требуется создать пользовательское действие для включения в список переопределений, вызываемых объектом `State` при входе в состояние.

Решение

Создайте класс, реализующий интерфейс `IOverride`. Определите все действия, которые должны использоваться пользовательским действием.

Обсуждение

Для реализации условной логики в действиях состояний необходимо создать пользовательский объект `IOverride`, выполняющий условную логику при входе в состояние. В массив переопределений состояния может быть включен любой объект, реализующий этот интерфейс, иначе говоря, любой компонент, обладающий следующими методами:

`apply(parent:UIComponent):void`

Метод применяет переопределение, выполняя его фактическое действие. Изменяемое значение `parent` должно быть сохранено в этом методе, чтобы метод `remove` мог отменить все изменения при выходе из состояния.

`initialize():void`

Метод инициализирует переопределение; может быть пустым.

`remove(parent:UIComponent):void`

Метод вызывается механизмом управления состояниями при выходе из состояния, в которое был включен объект `IOverride`. Все значения, изменяемые методом `apply`, должны быть сохранены для их восстановления.

В следующем примере интерфейс `IOverride` реализуется классом `CustomOverride`, а условная логика применяется в методе `apply`:

```
package oreilly.cookbook
{
    import flash.display.DisplayObject;
    import mx.core.UIComponent;
```

```
    import mx.states.IOverride;
    public class CustomOverride implements IOverride {
        private var widthValue:Number;
        private var _target:DisplayObject;

        public function CustomOverride(target:DisplayObject = null) {
            _target = target;
        }
        public function get target():DisplayObject {
            return _target;
        }
        public function set target(value:DisplayObject):void {
            _target = value;
        }
        // Пустая функция
        public function initialize():void {}

        // В этом методе необходимо сохранить родительское
        // значение перед его изменением, чтобы
        // затем восстановить его при вызове remove.
        public function apply(parent:UIComponent):void {
            widthValue = _target.width;
            if(_target.width > 500) {
                _target.width = 500;
            }
        }

        // Восстановление сохраненного значения
        public function remove(parent:UIComponent):void {
            _target.width = widthValue;
        }
    }
}
```

В следующем примере класс `CustomOverride` используется для изменения размера `Box`. Конечно, на практике этот класс следовало бы расширить для поддержки разных видов приемников и делать что-то более содержательное, чем изменять ширину приемника, но для демонстрационных целей этого достаточно.

```
<mx:states>
    <mx:State name="openState">
        <cookbook:CustomOverride target="{box1}"/>
    </mx:State>
</mx:states>
```

12

Эффекты

Эффекты играют чрезвычайно важную роль в приложениях Flex и являются одним из важнейших признаков RIA-приложения. Хорошее понимание эффектов и их инфраструктуры в Flex важно не только в контексте проектирования и реализации (т. е. того, что непосредственно видит пользователь), но и в тех аспектах, которые остаются невидимыми в случае правильной реализации эффектов, но мгновенно становятся заметными при задержках в работе приложения или некорректной уборке мусора. Центральное место в механизме эффектов занимает система таймеров и функций обратного вызова, которые на концептуальном уровне выглядят примерно так:

```
var timer:Timer = new Timer(100, 0);
timer.addEventListener(TimerEvent.TIMER, performEffect);
timer.start();

private function performEffect(event:Event):void {
    // Реализация эффекта
}
```

Конечно, на практике инфраструктура эффектов не сводится к простому созданию экземпляра класса `Effect` и вызову метода `play`. Класс `EffectManager` управляет всеми экземплярами эффектов, предотвращая чрезмерную загрузку процессора обработкой лишних таймеров и вызовов функций. Можно считать, что эффект складывается из двух составляющих: `EffectInstance` (информация о самом эффекте, о том, что он должен делать, и о тех элементах, на которые распространяется его действие) и `Effect` (этот класс выполняет функции фабрики, генерирует эффект, запускает его и удаляет после завершения).

Воспроизведение эффекта состоит из четырех фаз. Сначала для каждого целевого компонента эффекта создается экземпляр класса `EffectInstance`.

Таким образом, если эффект воздействует на четыре приемника, для него будут созданы четыре объекта `EffectInstance`. Затем инфраструктура копирует все конфигурационные данные из объекта-фабрики в экземпляр: продолжительность, количество повторений, задержка и т. д. Все эти данные задаются в свойствах нового экземпляра. В третьей фазе эффект воспроизводится на приемнике с использованием объекта экземпляра. Наконец, инфраструктура (и в первую очередь класс EffectManager) удаляет экземпляр объекта после его завершения.

Как правило, при работе с готовыми эффектами разработчик имеет дело только с классом-фабрикой, отвечающим за генерирование эффекта. Но при реализации пользовательских эффектов необходимо создать как объект `Effect`, выполняющий функции фабрики для данного типа эффекта, так и объект `EffectInstance`, относящийся к непосредственному воспроизведению эффекта. Использование эффекта всегда (независимо от того, знает об этом разработчик или нет) сопряжено с созданием фабрики, генерирующей экземпляры, а любая настройка производится на уровне объекта-фабрики, который затем передает заданные значения генерируемым экземплярам. Заглянув в исходный код Framework, вы найдете в нем класс `Glow` и класс `GlowInstance`. Для создания собственных эффектов вам понадобится аналогичная пара классов.

12.1. Вызов эффектов в MXML и ActionScript

Задача

Требуется создать и активизировать экземпляр эффекта в приложении.

Решение

Чтобы определить эффект в MXML, включите тег `Effect` в теги верхнего уровня компонента. Чтобы определить эффект в ActionScript, импортируйте класс эффекта, создайте экземпляр, укажите `UIComponent` в качестве приемника и вызовите метод `play` для воспроизведения эффекта.

Обсуждение

Классу `Effect` должен быть назначен приемный компонент `UIComponent`. При создании экземпляра `Effect` в ActionScript приемник передается в конструкторе:

```
var blur:Blur = new Blur(component);
```

Приемник также может быть назначен после создания экземпляра `Effect`, в свойстве `target` класса `Effect`. Приемник представляет собой компонент `UIComponent`, к которому будет применен эффект при вызове метода `play` класса `Effect`. Если эффект определяется в MXML, приемный компонент `UIComponent` назначается в теге:

```
<mx:Glow id="glowEffect" duration="1000" color="#ff0f0f"
target="{glowingTI}"/>
```

В следующем примере эффект `Glow` в MXML активизируется при щелчке на кнопке:

```
<mx:Button click="glowEffect.play()"/>
```

Другой пример: `glowintTI` назначается приемником эффекта `Blur` при вызове конструктора в методе `applyBlur`. После задания всех необходимых свойств `Effect` вызывается метод `play`.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="600">
    <mx:Script>
        <![CDATA[

            import mx.effects.Blur;
            private var blur:Blur;
            private function applyBlur():void {
                blur = new Blur(glowingTI);
                blur.blurXFrom = 0;
                blur.blurXTo = 20; // Величина размывки в пикселах
                blur.blurYFrom = 0;
                blur.blurYTo = 20; // Величина размывки в пикселах
                blur.duration = 1000;
                blur.play();
            }

        ]]>
    </mx:Script>
    <!-- В свойствах эффекта Glow задается цвет
    и продолжительность отображения эффекта -->
    <mx:Glow id="glowEffect" duration="1000" color="#ff0f0f"
        target="{glowingTI}"/>
    <mx:TextInput id="glowingTI"/>
    <mx:Button click="applyBlur()" toggle="true" id="glowToggle"
        label="Play the BlurEffect"/>
    <mx:Button click="glowEffect.play()" label="Play the Glow Effect"/>
</mx:VBox >
```

12.2. Создание пользовательского эффекта

Задача

Требуется создать пользовательский эффект, который может использоваться как в MXML, так и в ActionScript.

Решение

Создайте класс, расширяющий `Effect`; определите в нем пользовательские get- и set-методы для всех создаваемых экземпляров эффекта. Затем

создайте класс экземпляра, расширяющий EffectInstance, который будет использоваться для непосредственного внесения изменений в эффект.

Обсуждение

В Flex Framework эффекты образуются из двух составляющих: Effect и EffectInstance. Effect создает экземпляры EffectInstance и передает им значения свойств. Подобное «разделение ответственности» позволяет легко создавать эффекты, воспроизводимые на разных приемниках.

Начните с определения класса TestEffect (фабрика, генерирующая объекты EffectInstance), задайте их свойства и вызовите метод play для каждого экземпляра:

```
package oreilly.cookbook
{
    import mx.effects.Effect;
    import mx.effects.IEffectInstance;
    import mx.events.EffectEvent;

    import oreilly.cookbook.TestInstance;

    public class TestEffect extends Effect
    {
        public var color:uint;
        public var alpha:Number;

        public function TestEffect(target:Object=null) {
            // Непременный вызов базового конструктора
            super(target);
            // В свойстве instanceClass задается
            // нужный тип экземпляров.
            instanceClass = TestInstance;
        }

        override protected function
            initInstance(instance:IEffectInstance):void {
            trace(" instance initialized ");
            super.initInstance(instance);
            // После создания экземпляра можно
            // задать значения его свойств.
            TestInstance(instance).color = color;
            TestInstance(instance).alpha = alpha;
        }

        override public function getAffectedProperties():Array {
            trace(" return all the target properties ");
            return [];
        }

        override protected function
            effectEndHandler(event:EffectEvent):void {
```

```
            trace(" effect ended ");
        }

        override protected function
        effectStartHandler(event:EffectEvent):void {
            trace(" effect started ");
        }
    }
}
```

Обратите внимание на создание экземпляра `EffectInstance` **в методе** `InitInstance`**.** `EffectInstance` **преобразуется к типу** `TestInstance`**, а свойства** `TestInstance` **задаются из фабрики** `TestEffect`**. Таким образом, свойства каждого экземпляра** `TestInstance` **задаются всего один раз – в фабрике** `TestEffect`**.**

Класс `TestInstance`**, экземпляры которого создаются фабрикой** `TestEffect`**, выглядит так:**

```
package oreilly.cookbook
{
    import flash.display.DisplayObject;
    import mx.core.Container;
    import mx.core.FlexShape;
    import mx.core.UIComponent;
    import mx.effects.EffectInstance;

    public class TestInstance extends EffectInstance
    {

        public var alpha:Number;
        public var color:uint;

        public function TestInstance(target:Object) {
            super(target);
        }

        override public function play():void {
            super.play();
            (target as DisplayObject).alpha = alpha;
            var shape:FlexShape = new FlexShape();
            shape.graphics.beginFill(color, 1.0);
            shape.graphics.drawRect(0, 0, (
                target as DisplayObject).width,
                (target as DisplayObject).height);
            shape.graphics.endFill();
            var uiComp:UIComponent = new UIComponent();
            uiComp.addChild(shape);
            UIComponent(target).addChild(uiComp);
        }
    }
}
```

Свойство target каждого экземпляра TestInstance задается классом-фабрикой TestEffect при создании экземпляра. Это гарантирует, что в случае передачи нескольких приемников свойству targets класса Effect для каждого из них будет создан и воспроизведен экземпляр TestInstance. Свойства color и alpha экземпляра TestInstance задаются при создании в методе InitInstance класса TestEffect.

Переопределенный метод play класса TestInstance содержит логику модификации приемника UIComponent, назначенного TestInstance.

12.3. Создание параллельных или последовательных серий эффектов

Задача

Требуется создать несколько эффектов, воспроизводящихся параллельно (одновременно) или последовательно, т. е. друг за другом.

Решение

Используйте тег Parallel для объединения эффектов, воспроизводимых одновременно, или тег Sequence для последовательного воспроизведения.

Обсуждение

Тег Sequence воспроизводит следующий эффект последовательности в тот момент, когда предыдущий объект Effect выдает свое событие effectComplete. Конечно, последовательности могут состоять из нескольких тегов Parallel, потому что тег Parallel интерпретируется точно так же, как Effect, и содержит метод play, вызываемый Sequence при завершении воспроизведения предыдущего тега Effect или Parallel.

```
<mx:Sequence id=" sequencee" target="{this}">
    <mx:Blur duration="3000" blurXTo="10" blurYTo="10" blurXFrom="0"
        blurYFrom="0"/>
    <mx:Glow duration="3000" color="#ffff00"/>
</mx:Sequence>
```

Тег Parallel передает все приемники каждому объекту Effect или Sequence, заключенному в тег, и вызывает метод play для каждого внутреннего эффекта:

```
<mx:Parallel id=" parallel" targets="{[bar, foo]}">
    <mx:Blur duration="3000" blurXTo="10"
        blurYTo="10" blurXFrom="0" blurYFrom="0"/>
    <mx:Glow duration="3000" color="#ffff00"/>
</mx:Parallel>
<mx:ComboBox id="bar" dataProvider="{['one', 'two', 'three']}"/>
<mx:ComboBox id="foo" dataProvider="{['one', 'two', 'three']}"/>
```

12.4. Пауза, инверсия и перезапуск эффектов

Задача

Требуется приостановить воспроизводимый эффект, а затем запустить его с текущей позиции или с начала.

Решение

Используйте метод `pause` для приостановки воспроизводимого эффекта, а метод `resume` – для возобновления эффекта с места остановки.

Обсуждение

На первый взгляд метод `stop` класса `Effect` делает то же, что метод `pause`: оба метода останавливают эффект во время воспроизведения. Однако метод `stop` сбрасывает базовый таймер эффекта, что делает его возобновление невозможным. Метод `pause` просто приостанавливает таймер (а следовательно, и эффект), позволяя продолжить его выполнение с точки остановки. Во время приостановки эффект можно инвертировать методом `reverse`; в случае остановки инверсия невозможна.

Серии `Parallel` и `Sequence` также поддерживают приостановку и возобновление. Код выглядит примерно так:

```
<mx:VBoxxmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300">
    <mx:Parallel id="parallel" target="{this}">
        <mx:Blur duration="3000" blurXTo="10" blurYTo="10"
            blurXFrom="0" blurYFrom="0"/>
        <mx:Glow duration="3000" color="#ffff00"/>
    </mx:Parallel>
    <mx:Button click="parallel.play();" label="play()"/>
    <mx:Button click="parallel.pause();" label="pause()"/>
    <mx:Button click="parallel.reverse()"
        label="reverse()"/>
    <mx:Button click="parallel.resume()" label="resume ()"/>
</mx:VBox >
```

Если после вызова метода `pause` для серии `Sequence`, `Parallel` или отдельного эффекта вызывается метод `reverse`, то для начала воспроизведения эффекта в обратном направлении необходимо вызвать метод `resume`.

12.5. Создание пользовательских триггеров эффектов

Задача

Требуется создать пользовательские триггеры эффектов для компонентов.

Решение

Используйте тег `Effect` для определения имени триггера и события, которое воспроизводит эффект, связанный с триггером в компоненте.

Обсуждение

Триггер (*trigger*) определяет событие, инициирующее воспроизведение эффекта. Триггеры широко используются в Flex Framework, например, при определении эффекта `mouseDownEffect` для компонента `ComboBox`:

```
<mx:ComboBox mouseDownEffect="{glowEffect}"/>
```

Когда компонент `ComboBox` передает событие `mouseDown`, воспроизводится эффект, связанный с событием `mouseDownEffect` экземпляра `ComboBox`. В этом случае воспроизводится эффект `glowEffect`. Чтобы определить пользовательский триггер, определите тег `Event` с именем и типом события, а также тег `Effect` с именем триггера и именем события, которое будет инициировать эффект, связанный с триггером:

```
[Event(name="darken", type="flash.events.Event")]
[Effect(name="darkenEffect", event="darken")]
```

В предыдущем примере определяется триггер с именем `darkenEffect`, с которым можно связать эффект. Эффект, связанный с триггером, инициируется при передаче компонентом события с именем `darken`. Более полный пример кода:

```
<mx:HBoxxmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Metadata>
        [Event(name="darken", type="flash.events.Event")]
        [Effect(name="darkenEffect", event="darken")]

        [Event(name="lighten", type="flash.events.Event")]
        [Effect(name="lightenEffect", event="lighten")]
    </mx:Metadata>
    <mx:Script>
        <![CDATA[
            import flash.events.Event;

            private function dispatchDarken():void {
                // Обеспечивает воспроизведение эффекта,
                // связанного с триггером darkenEffect
                dispatchEvent(new Event("darken"));
            }

            private function dispatchLighten():void {
                // Обеспечивает воспроизведение эффекта,
                // связанного с триггером lightenEffect
                dispatchEvent(new Event("lighten"));
            }

        ]]>
```

```
    </mx:Script>
    <mx:Button click="dispatchDarken()" label="darken"/>
    <mx:Button click="dispatchLighten()" label="lighten"/>
</mx:HBox>
```

12.6. Создание tween-эффектов

Задача

Требуется создать пользовательский tween-эффект, изменяющий значения свойств в течение заданного промежутка времени.

Решение

Расширьте классы TweenEffect и TweenEffectInstance для создания фабрики и класса, генерируемого фабрикой для каждого приемника, передаваемого в определении эффекта.

Обсуждение

Самое заметное отличие обычных эффектов Effect от TweenEffect заключается в том, что TweenEffect происходит во времени. Начальные и конечные значения TweenEffect передаются экземпляру EffectInstance, который затем использует их либо для построения новых экземпляров фильтра, добавляемых в приемник, либо для изменения свойств приемника. Изменяющиеся значения свойств генерируются на протяжении интервала эффекта посредством использования объекта mx.effects.Tween в классе TweenInstance.

Этот рецепт демонстрирует построение простого tween-эффекта, медленно растворяющего альфа-канал приемника на протяжении периода времени, назначенного классу TweenEffect. Tween-эффекты, как и обычные эффекты, строятся из двух классов; в данном случае это класс-фабрика TweenEffect, генерирующий экземпляры TweenInstance для каждого приемника, переданного TweenEffect, и класс TweenInstance, создающий объект Tween и использующий значения, генерируемые объектом Tween в течение промежутка эффекта.

Начнем с класса TweenEffect:

```
package oreilly.cookbook
{
    import mx.effects.TweenEffect;

    public class CustomTweenEffect extends TweenEffect
    {

        public var finalAlpha:Number = 1.0;

        public function CustomTweenEffect (target:Object=null) {
```

```
            super(target);
        }

        public function CustomDisplacementEffect(target:Object=null){
            super(target);
            this.instanceClass = CustomTweenInstance;
        }

        // Создание нового экземпляра
        override protected function initInstance(
        instance:IEffectInstance):void {
            super.initInstance(instance);
            // Экземпляр создан, можно задать значения его свойств.
            CustomTweenInstance(instance).finalAlpha = this.finalAlpha;
        }

        override public function getAffectedProperties():Array {
            trace(" return all the target properties ");
            return [];
        }
    }
}
```

Свойство `finalAlpha` **каждого объекта** `CustomTweenInstance` **, передаваемого методу** `initInstance`**, задается при создании экземпляра** `TweenInstance`**.**

Класс `CustomTweenInstance` **расширяет класс** `TweenEffectInstance` **и переопределяет методы** `play` **и** `onTweenUpdate` **этого класса. Переопределенный метод** `play` **содержит логику создания объекта** `Tween`**, генерирующего изменяющиеся значения на протяжении действия** `TweenEffect`**:**

```
        override public function play():void
        {
            super.play();
            this.tween = new Tween(this, 0, finalAlpha, duration);
            (target as DisplayObject).alpha = 0;
        }
```

Свойства `finalAlpha` **и** `duration` **передаются из** `CustomTweenEffect`**, а** `mx.effects.Tween` **генерирует для каждого кадра клипа SWF значения, плавно изменяющиеся от начального значения (в данном случае 0) до конечного значения (в данном случае переменная** `finalAlpha`**). Если потребуется, объекту** `Tween` **можно передать несколько значений в массиве, при условии, что массивы начальных и конечных значений имеют одинаковое количество элементов. Метод** `play` **класса** `TweenEffectInstance`**, вызываемый здесь в форме** `super.play`**, добавляет слушателя события в** `Tween` **для метода** `onTweenUpdate`**. Переопределяя этот метод, можно включить в** `TweenEffectInstance` **любую пользовательскую логику:**

```
override public function onTweenUpdate(value:Object):void {
    (target as DisplayObject).alpha = value as Number;
}
```

В данном случае свойству alpha приемника задается значение, возвращаемое переменной Tween. В результате свойство alpha приемника постепенно изменяется до значения переменной finalValue:

```
package oreilly.cookbook
{

    import flash.display.DisplayObject;
    import mx.effects.effectClasses.TweenEffectInstance;

    public class CustomTweenInstance extends TweenEffectInstance
    {
        public var finalAlpha:Number;

        public function NewTweenInstance(target:Object) {
            super(target);
        }

        override public function play():void {
            super.play();
            this.tween = new Tween(this, 0, finalAlpha, duration);
            (target as DisplayObject).alpha = 0;
        }

        override public function onTweenUpdate(value:Object):void {
            (target as DisplayObject).alpha = value as Number;
        }
    }
}
```

При каждом вызове метода onTweenUpdate значение alpha вычисляется заново и используется для обновления приемника.

12.7. Использование фильтра DisplacementMapFilter в эффектах Flex

Задача

Требуется создать tween-эффект, обеспечивающий трансформацию одного изображения в другое.

Решение

Расширьте классы TweenEffect и TweenEffectInstance. В результате создается экземпляр TweenEffect со значениями смещения, передаваемыми каждому создаваемому им классу TweenEffectInstance. В пользовательском классе TweenEffectInstance создайте объект DisplacementMapFilter и воспользуйтесь механизмом tween-преобразований Flex Framework для достижения желаемых смещений, генерируя новые фильтры по каждому событию onTweenUpdate.

Обсуждение

Фильтр DisplacementMapFilter замещает или деформирует пикселы одного изображения, используя пикселы другого изображения для определения позиции и величины деформации. Он часто используется для создания иллюзии изображения, выходящего из-под другого изображения (словно накрытого листом бумаги).

Позиция и величина смещения, применяемого к каждому пикселу, определяется цветовым значением изображения, используемого в качестве карты смещения. Конструктор DisplacementMapFilter имеет следующую сигнатуру:

```
public function DisplacementMapFilter(mapBitmap:BitmapData = null,
    mapPoint:Point = null, componentX:uint = 0, componentY:uint = 0,
    scaleX:Number = 0.0, scaleY:Number = 0.0, mode:String = "wrap",
    color:uint = 0, alpha:Number = 0.0)
```

Обычно в такой длинной строке кода бывает проще разобраться, если разбить ее на фрагменты:

BitmapData (*default = null*)

Объект BitmapData, используемый для замещения изображения или компонента, к которому применяется фильтр.

mapPoint

Позиция фильтруемого изображения, в которой будет применен левый верхний угол фильтра. Используется в том случае, если фильтр применяется к части изображения.

componentX

Определяет цветовой канал карты, изменяющий **x-позицию пикселов**. В BitmapDataChannel все допустимые значения определяются в виде констант со значениями: BitmapDataChannel.BLUE или 4, BitmapDataChannel.RED или 1, BitmapDataChannel.GREEN или 2, BitmapDataChannel.ALPHA или 8.

componentY

Определяет цветовой канал карты, изменяющий **x-позицию пикселов**. Допустимые значения те же, что у componentX.

scaleX

Коэффициент, определяющий величину смещения по оси x.

scaleY

Коэффициент, определяющий величину смещения по оси y.

mode

Строка, которая определяет, как следует поступать с пустыми местами, образовавшимися на месте сдвинутых пикселов. Допустимые значения, определяемые в виде констант в классе DisplacementMap-

FilterMode: отображение исходных пикселов (mode = IGNORE), перенос пикселов с другой стороны изображения (mode = WRAP, используется по умолчанию), использование ближайшего сдвинутого пиксела (mode = CLAMP) или заполнение пустых мест цветом (mode = COLOR).

Класс CustomDisplacementEffect создает экземпляры CustomDisplacement-Instance:

```
package oreilly.cookbook
{
    import mx.effects.IEffectInstance;
    import mx.effects.TweenEffect;
    import mx.events.EffectEvent;

    public class CustomDisplacementEffect extends TweenEffect
    {

        public var image:Class;
        public var yToDisplace:Number;
        public var xToDisplace:Number;
        public function CustomDisplacementEffect(target:Object=null)
        {
            super(target);
            this.instanceClass = CustomDisplacementInstance;
        }

        override protected function initInstance(
        instance:IEffectInstance):void {
            trace(" instance initialized ");
            super.initInstance(instance);
            // Теперь, когда мы привели пример экземпляра,
            // можно задать его свойства
            CustomDisplacementInstance(instance).image = image;
            CustomDisplacementInstance(instance).xToDisplace =
                this.xToDisplace;
            CustomDisplacementInstance(instance).yToDisplace =
                this.yToDisplace;
        }
        override public function  getAffectedProperties():Array {
            trace(" return all the target properties ");
            return [];
        }
    }
}
```

CustomDisplacementInstance обеспечивает фактическое создание объекта DisplacementEffect, применяемого к приемнику. Объект растровых данных, фильтр, величины смещений x и y CustomDisplacementTween применяются к экземпляру и передаются DisplacementEffect.

CustomTweenEffect генерирует экземпляры класса CustomDisplacement-Instance:

```
package oreilly.cookbook
{
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;
    import flash.display.DisplayObject;
    import flash.filters.DisplacementMapFilter;
    import flash.filters.DisplacementMapFilterMode;
    import flash.geom.Point;

    import mx.effects.Tween;
    import mx.effects.effectClasses.TweenEffectInstance;

    public class CustomDisplacementInstance extends
        TweenEffectInstance
    {
        public var image:Class;
        public var xToDisplace:Number;
        public var yToDisplace:Number;
        public var filterMode:String = DisplacementMapFilterMode.WRAP;

        private var filter:DisplacementMapFilter;
        private var img:DisplayObject;
        private var bmd:BitmapData;

        public function CustomDisplacementInstance(target:Object)
        {
            super(target);
        }

        override public function play():void {
            super.play();
            // Встроенное изображение готовится к использованию
            img = new image();
            bmd = new BitmapData(img.width, img.height, true);
            // Вывод графических данных в изображении
            bmd.draw(img);
```

При создании нового фильтра все значения задаются в соответствии с исходным состоянием фильтра:

```
            filter = new DisplacementMapFilter(bmd, new Point(DisplayObject(
            target).width/2 - (img.width/2), DisplayObject(target).height/2 -
            (img.height/2))), BitmapDataChannel.RED, BitmapDataChannel.RED, 0, 0,
            filterMode, 0.0, 1.0);
                // Скопировать все фильтры, уже существующие в приемнике,
                // чтобы они не были уничтожены при добавлении нового фильтра
                var targetFilters:Array = (target as DisplayObject).filters;
                targetFilters.push(filter);
                // Назначение фильтра в приемнике
                (target as DisplayObject).filters = targetFilters;
                // Создание tween-преобразования, которое генерирует
                // новые значения для каждого кадра нашего эффекта
```

```
            this.tween = new Tween(this, [0, 0], [xToDisplace,
            yToDisplace], duration);
        }
```

Основная работа этого класса выполняется в методе `setDisplacementFilter`**. Так как фильтры действуют кумулятивно (т. е. применяются поверх друг друга), предыдущие вхождения** `DisplacementMapFilter` **необходимо удалить. Задача решается циклическим перебором массива** `filters` **приемника** `target` **и удалением всех найденных экземпляров** `DisplacementMapFilter`**. Далее на основе данных, полученных от** `Tween`**, создается новый фильтр, который применяется к приемнику. Обратите внимание: для правильного отображения фильтра необходим сброс** `Array`**. Простое включение** `filter` **методом** `Array.push` **не обеспечит перерисовки** `DisplayObject` **с новым фильтром.**

```
    private function setDisplacementFilter(displacement:Object):void {
        var filters:Array = target.filters;
      // Удаление существующих фильтров смещения гарантирует,
      // что добавляемый фильтр будет единственным
        var n:int = filters.length;
        for (var i:int = 0; i < n; i++) {
            if (filters[i] is DisplacementMapFilter)
                filters.splice(i, 1);
      }

        // Создание нового фильтра на основе данных,
        // полученных от Tween
        filter = new DisplacementMapFilter(bmd, new Point(0, 0),
            BitmapDataChannel.RED, BitmapDataChannel.RED,
            displacement[0] as Number, displacement[1] as Number,
            filterMode, 0.0, 0);
        // Включение фильтра в список фильтров приемника
        filters.push(filter);
        target.filters = filters;
      }
      // Каждый раз при готовности к обновлению
      // фильтр создается заново
      override public function onTweenUpdate(value:Object):void
      {
          setDisplacementFilter(value);
      }
      // Фильтр применяется в последний раз, после чего
      // передается событие завершения tween-эффекта.
      override public function onTweenEnd(value:Object):void
      {
          setDisplacementFilter(value);

             super.onTweenEnd(value);
      }
    }
}
```

При завершении tween-эффекта последние значения DisplacementMapFilter определяют итоговый внешний вид приемника DisplayObject, и в конце вызывается метод onTweenEnd класса TweenEffectInstance.

12.8. Создание эффекта анимации цвета

Материал предоставлен Дарроном Шеллом (Darron Schall).

Задача

Требуется обеспечить плавный переход между двумя цветами.

Решение

Используйте пользовательский эффект AnimateColor для достижения плавного цветового перехода.

Обсуждение

Реализация цветовых переходов с использованием свойства AnimateProperty вызывает мерцание и создает проблемы, связанные со специфической природой самих цветовых значений. Эффект AnimateProperty обеспечивает плавный переход между fromValue и toValue, но он применим только для числовых значений. Например, AnimateProperty отлично подходит для анимации перемещения компонента из точки с координатой x = 10 в точку x = 100.

Цвета же не являются обычными числами. Они представляются числовыми кодами (например, 0xFFCC33 или 16 763 955 в десятичном представлении), но каждое число фактически состоит из трех отдельных чисел (*каналов*): составляющих красного, синего и зеленого цветов. Значения каждого канала лежат в интервале от 0 до 255 (0xFF). Чистому красному цвету соответствует код 0xFF0000, зеленому – 0x00FF00, а синему – 0x0000FF. Таким образом, цветовое значение 0x990099 представляет собой смесь красного и синего цветов.

Чтобы обеспечить плавную анимацию цвета, необходимо рассматривать все части цветового значения по отдельности. Например, при переходе от 0x000000 к 0xFF0000 изменяться должен только красный канал. Однако AnimateProperty рассматривает его как переход от 0 к 16 711 680. Число 255 лежит между этими границами и с высокой вероятностью появится при анимации, но 255 соответствует 0x0000FF, т. е. чистому синему цвету. Таким образом, в ходе анимации появится синий цвет, хотя изменяться должен только красный канал.

Итак, анимация цвета должна осуществляться на уровне отдельных каналов цветовых значений. Руководствуясь этим основополагающим фактом, я создал новый эффект с именем AnimateColor.

Пример использования AnimateColor:

```
<ds:AnimateColor xmlns:ds="com.darronschall.effects.*"
        id="fadeColor"
        target="{someTarget}"
        property="backgroundColor" isStyle="true"
        fromValue="0xFF0000"
        toValue="0x00FF00"
        duration="4000" />
```

Эффект AnimateColor вычисляет разность между fromValue и toValue отдельно по каждому цветовому каналу. На каждом шаге анимации создается новый цвет, учитывающий величину изменений по каждому отдельному каналу. Общим результатом является плавная анимация цвета. В предыдущем примере свойство backgroundColor приемника someTarget переходит от чистого красного к чистому синему цвету за 4 секунды, с промежуточными оттенками фиолетового.

12.9. Использование фильтра свертки для создания tween-эффекта

Задача

Требуется создать эффект TweenEffect для компонента MXML с использованием ConvolutionFilter.

Решение

Создайте класс TweenEffectInstance, который создает новые экземпляры ConvolutionFilter в обработчике события onTweenUpdate и заносит эти экземпляры ConvolutionFilter в массив фильтров приемника DisplayObject.

Обсуждение

Фильтр ConvolutionFilter позволяет выполнять чрезвычайно разнообразные модификации объекта-приемника DisplayObject или BitmapImage, в том числе размывку, выделение границ, повышение контраста, рельефное выделение и т. д. Каждый пиксел изображения-источника изменяется в соответствии со значениями окружающих пикселов. Изменения пиксела определяются массивом Matrix, передаваемым в конструкторе ConvolutionFilter. Конструктор имеет следующую сигнатуру:

```
public function ConvolutionFilter(matrixX:Number = 0, matrixY:Number = 0,
matrix:Array = null, divisor:Number = 1.0, bias:Number = 0.0,
preserveAlpha:Boolean = true, clamp:Boolean = true, color:uint = 0,
alpha:Number = 0.0)
```

Параметры конструктора:

matrixX:Number (*по умолчанию = 0*)

Количество столбцов в матрице.

`matrixY:Number` (*по умолчанию = 0*)

Количество строк в матрице.

`matrix:Array` (*по умолчанию = null*)

Массив значений, определяющих преобразование каждого пиксела. Количество элементов в массиве должно быть равно `matrixX * matrixY`.

`divisor:Number` (*по умолчанию = 1.0*)

Делитель, используемый в ходе преобразования (по умолчанию 1). Делитель, равный сумме всех значений в матрице, нормализует общую интенсивность цвета. Значение 0 игнорируется и вместо него используется значение по умолчанию.

`bias:Number` (*по умолчанию = 0.0*)

Смещение, прибавляемое к результату матричного преобразования.

`preserveAlpha:Boolean` (*по умолчанию = true*)

Значение `false` означает, что величина альфа-канала не сохраняется и свертка применяется ко всем каналам. Значение `true` означает, что свертка применяется только к цветовым каналам.

`clamp:Boolean` (*по умолчанию = true*)

Для пикселов, выходящих за границы исходного изображения, при значении `true` изображение продлевается за границы посредством дублирования цветов граничных пикселов у соответствующего края. Значение `false` означает, что вместо них должен использоваться другой цвет, определяемый свойствами `color` и `alpha`. По умолчанию используется значение `true`.

`color:uint` (*по умолчанию = 0*)

Шестнадцатеричный код цвета пикселов, выходящих за границу исходного изображения.

`alpha:Number` (*по умолчанию = 0.0*)

Альфа-канал цвета-заменителя.

Класс `TweenEffect`, использующий экземпляры `TweenEffectInstance` на базе `ConvolutionFilter`, очень похож на класс `TweenEffect` из рецепта 12.8:

```
package oreilly.cookbook
{
    import mx.effects.IEffectInstance;
    import mx.effects.TweenEffect;

    public class ConvolutionTween extends TweenEffect
    {
```

В этом разделе задаются значения, передаваемые каждому новому экземпляру `EffectInstance`:

```
        public var toAlpha:Number;
        public var fromAlpha:Number;

        public var toColor:uint;
        public var fromColor:uint;

        public var fromMatrix:Array;
        public var toMatrix:Array;

        public var toDivisor:Number;
        public var fromDivisor:Number;

        public var toBias:Number;
        public var fromBias:Number;
        public function ConvolutionTween(target:Object=null)
        {
            super(target);
            this.instanceClass = ConvolutionTweenInstance;
        }
```

Свойства вновь созданных экземпляров класса `ConvolutionTweenInstance` **задаются в методе** `initInstance`**:**

```
        override protected function initInstance(instance:IEffectInstance):
void {
            trace(" instance initialized ");super.initInstance(instance);
            // Экземпляр создан, можно задавать свойства
            ConvolutionTweenInstance(instance).toAlpha = toAlpha;
            ConvolutionTweenInstance(instance).fromAlpha = fromAlpha;

            ConvolutionTweenInstance(instance).toColor = toColor;
            ConvolutionTweenInstance(instance).fromColor = fromColor;

            ConvolutionTweenInstance(instance).fromMatrix = fromMatrix;
            ConvolutionTweenInstance(instance).toMatrix = toMatrix;

            ConvolutionTweenInstance(instance).toDivisor = toDivisor;
            ConvolutionTweenInstance(instance).fromDivisor = fromDivisor;

            ConvolutionTweenInstance(instance).toBias = toBias;
            ConvolutionTweenInstance(instance).fromBias = fromBias;
        }

        override public function getAffectedProperties():Array {
            trace(" return all the target properties ");
            return [];
        }
    }
}
```

`ConvolutionTweenInstance` **получает свой объект-приемник и значения от фабрики** `ConvolutionTweenEffect`**:**

```
package oreilly.cookbook
{
    import flash.filters.ConvolutionFilter;
    import mx.effects.Tween;
    import mx.effects.effectClasses.TweenEffectInstance;
    public class ConvolutionTweenInstance extends
        TweenEffectInstance
    {
        private var convolutionFilter:ConvolutionFilter;
        public var toAlpha:Number;
        public var fromAlpha:Number;

        public var toColor:uint;
        public var fromColor:uint;

        public var fromMatrix:Array;
        public var toMatrix:Array;

        public var toDivisor:Number;
        public var fromDivisor:Number;

        public var toBias:Number;
        public var fromBias:Number;

        public function ConvolutionTweenInstance(
            target:Object) {
            super(target);
        }
```

В переопределенном методе `play` **экземпляру** `Tween` **передаются значения** `from` **и** `to`**, представляющие соответственно исходные и конечные значения** `ConvolutionFilter`**. Поскольку класс** `Tween` **не может сгенерировать промежуточные значения для** `Array`**, промежуточные значения генерируются для элементов массива, а затем используются для создания нового массива** `Matrix`**:**

```
        override public function play():void {

            this.tween = new Tween(this, [fromMatrix[0], fromMatrix[1],
                fromMatrix[2], fromMatrix[3], fromMatrix[4], fromMatrix[5],
                fromMatrix[6], fromMatrix[7], fromDivisor, fromBias,
                fromAlpha, fromColor], [toMatrix[0], toMatrix[1],
                toMatrix[2], toMatrix[3], toMatrix[4], toMatrix[5],
                toMatrix[6], toMatrix[7], , toDivisor, toBias, toAlpha,
                toColor], duration);
            convolutionFilter = new ConvolutionFilter(fromMatrixX,
                fromMatrixY, fromMatrix, 1.0, 0, true, true, fromAlpha,
                fromColor);
        }
```

Каждое новое значение, полученное от `Tween`**, передается** `onTweenUpdate` **в виде объекта. Объект содержит массив, в котором хранятся новые**

значения для предоставления перехода от исходного состояния к конечному для каждого промежуточного момента времени. Поскольку ConvolutionFilter должен передаваться массив, каждый элемент массива подвергается tween-преобразованию и передается новому массиву в параметре matrix конструктора ConvolutionFilter:

```
        override public function onTweenUpdate(value:Object):void {
            // Получение фильтров от приемника
            var filters:Array = target.filters;
            // Удаление существующих фильтров гарантирует,
            // что добавляемый фильтр будет единственным
            var n:int = filters.length;
            for (var i:int = 0; i < n; i++) {
                if (filters[i] is ConvolutionFilter)
                    filters.splice(i, 1);
            }
            // Создание нового фильтра
            convolutionFilter = new ConvolutionFilter(3, 3, [value[0],
                value[1], value[2], value[3], value[4], value[5], value[6],
                value[7]], value[8], value[9], true, true, value[10],
                value[11]);
            // Добавление фильтра
            filters.push(convolutionFilter);
            target.filters = filters;
        }
    }
}
```

Обратите внимание на удаление всех фильтров ConvolutionFilter, примененных к приемнику. Если этого не сделать, к приемнику будут применены несколько фильтров свертки с накоплением изменений, а результат получится совершенно не таким, как предполагалось.

13

Коллекции

Коллекции являются мощным дополнением к компоненту индексируемого массива ActionScript – базового класса ActionScript `Array`. В коллекциях добавлена функциональность сортировки содержимого массива, поддержания текущей позиции чтения и создания представлений, в которых может отображаться отсортированная версия массива. Коллекции также могут оповещать слушателей событий об изменении своего содержимого и применять пользовательскую логику к элементам, включаемым в массив-источник. Именно поддержка оповещения слушателей делает возможной привязку данных, а поддержка сортировки обеспечивает сортировку и фильтрацию в компонентах `DataGrid` и `List`. Коллекции являются неотъемлемым аспектом работы с элементами, управляемыми данными.

Из всех типов коллекций чаще всего используются классы `ArrayCollection` и `XMLListCollection`. `ArrayCollection` инкапсулирует объект `Array` и предоставляет вспомогательные методы для добавления и удаления элементов, а также создания курсоров, упрощающих сохранение текущей позиции чтения в `Array`. `XMLListCollection` инкапсулирует объект данных XML и предоставляет аналогичные функции: доступ к объектам по индексу, вспомогательные методы для добавления новых объектов и поддержку курсоров. Мощь `XMLListCollection` особенно хорошо проявляется при работе с массивами объектов XML; нередко `XMLListCollection` избавляет разработчика от необходимости разбирать код XML в массивы объектов данных.

13.1. Добавление, сортировка и выборка данных из ArrayCollection

Задача

Требуется сохранить новые данные в коллекции ArrayCollection, а потом прочитать их из той же коллекции.

Решение

Создайте экземпляр ArrayCollection. Методы addItemAt и addItem используются для вставки объектов, методы getItemIndex и contains – для проверки присутствия элемента в массиве, а свойство sort – для сортировки ArrayCollection и получения первых или последних элементов в порядке значений определенного поля.

Обсуждение

Чтобы увидеть, как работают различные методы добавления, проверки и сортировки элементов ArrayCollection, сначала необходимо создать коллекцию. Код создания коллекции выглядит примерно так:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import mx.collections.SortField;
            import mx.collections.Sort;
            import mx.collections.ArrayCollection;

            private var coll:ArrayCollection;
            private function init():void {
                coll = new ArrayCollection([{name:"Martin Foo", age:25},
                    {name:"Joe Bar", age:15}, {name:"John Baz", age:23}]);
            }
```

Вставка элемента в определенную позицию источника ArrayCollection осуществляется методом addtemAt:

```
private function addItem():void {
    coll.addItemAt({name:"James Fez", age:40}, 0);
}
```

Чтобы узнать, присутствует ли сложный объект в ArrayCollection, необходимо сравнить значения двух объектов. Возникает заманчивая идея действовать примерно так:

```
private function checkExistence():void {
    trace(coll.contains({name:nameTI.text, age:Number(ageTI.text)}));
    trace(coll.getItemIndex({name:nameTI.text, age:ageTI.text}));
    // При отсутствии выводится -1
}
```

Однако подобное решение не работает, потому что методы contains и getItemIndex сравнивают указатели на объекты, а не их содержимое. Так как сравниваемые объекты занимают два разных блока памяти с уникальными идентификаторами, Flash Player не считает их равными. Метод getItemIndex не вернет индекс элемента или хотя бы информацию о наличии элемента в ArrayCollection. Чтобы определить, присутствует ли элемент с таким же значением в коллекции, необходимо выполнить сравнение для каждого элемента массива-источника. Задача решается функцией следующего вида:

```
private function checkExistence():int {
    var i:int;
    var arr:Array = coll.souce;
    while(i < arr.length) {
           if(arr[i].name == nameTI.text && arr[i].age == ageTI.text) {
               return i;
          }
          i++;
    }
    return -1;
}
```

Объект Sort содержит метод findItem, который выполняет подобный и более гибкий поиск по всем объектам ArrayCollection с использованием свойства source ArrayCollection. Метод findItem имеет следующую сигнатуру:

```
public function findItem(items:Array, values:Object, mode:String,
returnInsertionIndex:Boolean = false, compareFunction:Function = null):int
```

В параметре value передается любой объект, содержащий все свойства и необходимые значения. Допустимые значения строки режима mode: Sort.ANY_INDEX_MODE (индекс любого вхождения); Sort.FIRST_INDEX_MODE (индекс первого вхождения); Sort.LAST_INDEX_MODE (индекс последнего вхождения). Параметр returnInsertionIndex указывает, должна ли функция findItem вернуть позицию отсортированного массива, в которой размещался бы элемент при отсутствии объекта, равного параметру value. Параметр compareFunction задает функцию, которая должна использоваться объектом Sort для определения равенства двух элементов.

Вместо этого метода также можно воспользоваться методом findItem объекта Sort:

```
private function checkExistence():int {
    var sort:Sort = new Sort();
    return sort.findItem(coll.source, {name:nameTI.text,
    age:Number(ageTI.text)}, Sort.ANY_INDEX_MODE);
}
```

Чтобы отсортировать коллекцию ArrayCollection, создайте объект Sort и передайте ему массив объектов SortField. Объекты SortField содержат строку с именем свойства объектов, содержащихся в ArrayCollection,

которое должно использоваться для определения порядка сортировки. Например, чтобы отсортировать коллекцию по свойству age каждого объекта в коллекции, создайте объект Sort и передайте ему объект SortField с полем сортировки age:

```
private function getOldest():void {
    var sort:Sort = new Sort();
    sort.fields = [new SortField("age", false)];
    coll.sort = sort;
    coll.refresh();
    trace(coll.getItemAt(0).age+" "+coll.getItemAt(0).name);
}
```

Функция сортирует ArrayCollection от минимального значения age к максимальному.

13.2. Фильтрация коллекции ArrayCollection

Задача

Требуется отфильтровать ArrayCollection с исключением всех элементов, не соответствующих критерию фильтрации.

Решение

Задайте свойству filterFunction объекта ArrayCollection функцию-фильтр с сигнатурой function(item:Object):Boolean. Функция-фильтр возвращает true, если элемент должен остаться в отфильтрованной коллекции, или false для исключаемых элементов.

Обсуждение

Свойство filterFunction определяется классом ListCollectionView, расширяемым классом ArrayCollection. После передачи filterFunction любому классу, расширяющему ListCollectionView (в данном случае ArrayCollection), необходимо вызвать метод refresh, чтобы фильтр был применен к ArrayCollection.

```
import mx.collections.ArrayCollection;
private var coll:ArrayCollection;

private function init():void {
    coll = new ArrayCollection([{name:"Martin Foo", age:25},
        {name:"Joe Bar",age:15}, {name:"John Baz", age:23},
        {name:"Matt Baz", age:21}]);
    coll.filterFunction = filterFunc;
    coll.refresh();
    for(var i:int =  0; i<coll.length; i++) {
        trace(coll.getItemAt(i).name);
    }
}
```

```
private function filterFunc(value:Object):Object {
    if(Number(value.age) > 21) {
        return true;
    }
    return false;
}
```

Обратите внимание: функция `filterFunction` не должна изменять массив-источник `ArrayCollection`. В предыдущем примере после вызова `refresh` массив-источник по-прежнему содержит 4 элемента. Он всегда остается неизменным, что позволяет передать несколько функций `filterFunction`; при этом предыдущий фильтр удаляется, а исходный массив-источник фильтруется заново.

13.3. Проверка модификации элементов в ArrayCollection

Задача

Требуется определить, когда элементы `ArrayCollection` добавляются или удаляются из коллекции процессом, находящимся вне области видимости.

Решение

Прослушивайте события типа `collectionChange` или `CollectionEvent.COLLECTION_CHANGE`. События передаются классом `ArrayCollection`, расширяющим `EventDispatcher`.

Обсуждение

При каждом добавлении и удалении элемента из коллекции передается событие `CollectionEvent` типа `collectionChange`. Если элемент привязан к коллекции, механизм привязки при помощи этого события сообщает об изменении коллекции. Включение для коллекции слушателя события `COLLECTION_CHANGE` позволяет запрограммировать обработку любых изменений коллекции:

```
private var coll:ArrayCollection = new ArrayCollection();
    coll.addEventListener(CollectionEvent.COLLECTION_CHANGE,
    collChangeHandler);
```

Класс `CollectionEvent` определяет следующие дополнительные свойства:

`items:Array`

При передаче события, обусловленного добавлением элементов в `ArrayCollection`, свойство `items` содержит массив добавленных элементов. При удалении элементов из коллекции в массиве содержатся все удаленные элементы.

`kind:String`

Строка, описывающая тип произошедшего события. Допустимые значения — `add`, `remove`, `replace` и `move`.

`location:int`

Свойство содержит индекс элемента(-ов) из свойства `items`. Индексация элементов начинается с нуля.

`oldLocation:int`

Если свойство `kind` равно `move`, свойство `oldLocation` содержит индекс предыдущей позиции элемента(-ов), заданного свойством `items`. Индексация элементов начинается с нуля.

Использование `CollectionEvent` позволяет определить состояние `ArrayCollection` или `XMLListCollection` до или после изменения. В частности, данная возможность чрезвычайно полезна, когда вы хотите убедиться в том, что внесенные в приложении Flex изменения привели к обновлениям на сервере.

13.4. Создание объекта GroupingCollection

Задача

Требуется сгруппировать элементы коллекции по значениям свойств элементов.

Решение

Передайте `Array` конструктору `GroupingCollection` или задайте свойство `source` для уже существующего объекта `GroupingCollection`.

Обсуждение

Любой объект `GroupingCollection` может содержать объект `Grouping` с объектом `GroupingField`, который определяет свойство объекта данных, используемое для генерирования группы. Объект `GroupingCollection` может использоваться для группировки объектов данных по общему для них всех свойству. Например, для заполнения `GroupingCollection` объектами данных, обладающими свойствами `state` и `region`, используется код следующего вида:

```
var groupingColl:GroupingCollection = new GroupingCollection();
groupingColl.source = [{city:"Columbus", state:"Ohio", region:"East"},
{city:"Cleveland", state:"Ohio", region:"East"}, {city:"Sacramento",
state:"California", region:"West"}, {city:"Atlanta", state:"Georgia",
region:"South"}];
```

Чтобы сгруппировать объекты по свойству `state` (т. е. создать группы с одинаковыми значениями `state`), создайте экземпляр `Grouping` и передайте ему массив объектов `GroupingField`:

```
var groupingInst:Grouping = new Grouping();
groupingInst.fields = [new GroupingField("state")];
groupingColl.grouping = groupingInst;
groupingColl.refresh(false);
```

Создав экземпляр `Grouping`**, задайте свойству** `grouping` **объекта** `GroupingCollection` **значение** `groupingInst` **и обновите коллекцию. Все объекты данных коллекции группируются по значению свойства** `state`**:**

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
width="400" height="300"
creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import mx.collections.Grouping;
            import mx.collections.GroupingField;
            import mx.collections.GroupingCollection;

            [Bindable]
            private var groupingColl:GroupingCollection;
            private function init():void {
                groupingColl = new GroupingCollection();
                groupingColl.source = [{city:"Columbus", state:"Ohio",
                    region:"East"}, {city:"Cleveland", state:"Ohio",
                    region:"East"}, {city:"Sacramento", state:"California",
                    region:"West"}, {city:"Atlanta", state:"Georgia",
                    region:"South"}];
                var groupingInst:Grouping = new Grouping();
                groupingInst.fields = [new GroupingField("state")];
                groupingColl.grouping = groupingInst;
                groupingColl.refresh(false);
            }
```

Назначение другой группировки `GroupingCollection` **приводит к замене текущей группировки при вызове метода** `refresh`**:**

```
            private function createRegionGrouping():void {
                var groupingInst:Grouping = new Grouping();
                       groupingInst.fields = [new GroupingField("region")];
                    groupingColl.grouping = groupingInst;
                    groupingColl.refresh(false);
            }
        ]]>
    </mx:Script>
<mx:AdvancedDataGrid dataProvider="{groupingColl}">
<mx:columns>
            <mx:AdvancedDataGridColumn dataField="city"/>
</mx:columns>

</mx:AdvancedDataGrid>
    <mx:Button click="createRegionGrouping()"/>
</mx:VBox>
```

Следовательно, многоуровневая группировка должна определяться иначе – передайте несколько объектов `GroupingField` в свойстве `fields` объекта `Grouping`:

```
groupingInst.fields = [new GroupingField("region"),
    new GroupingField("state")];
```

Объекты данных группируются сначала по полю `region`, а затем по полю `state`.

13.5. Создание иерархического провайдера данных

Задача

Требуется использовать *плоский объект* (т. е. объект, не содержащий отношений «родитель-потомок»), представляющий иерархические данные, в качестве значения свойства `dataProvider` компонента `DataGrid`.

Решение

Создайте пользовательский класс данных, реализующий интерфейс `IHierarchicalData`. Создайте методы, определяющие, имеет ли узел или объект данных родительские и дочерние узлы.

Обсуждение

Интерфейс `IHierarchicalData` определяет все методы, необходимые компонентам `DataGrid` и `AdvancedDataGrid` для отображения иерархических данных. Термином «*иерархические данные*» обозначаются данные, описывающие совокупность отношений «родитель-потомок». Для примера представьте иерархию различных транспортных средств – легковых и грузовых машин, лодок и т. д.; каждая категория делится на более конкретные виды транспорта. На рис. 13.1 изображена иерархия транспортных средств от седана до самого абстрактного верхнего уровня.

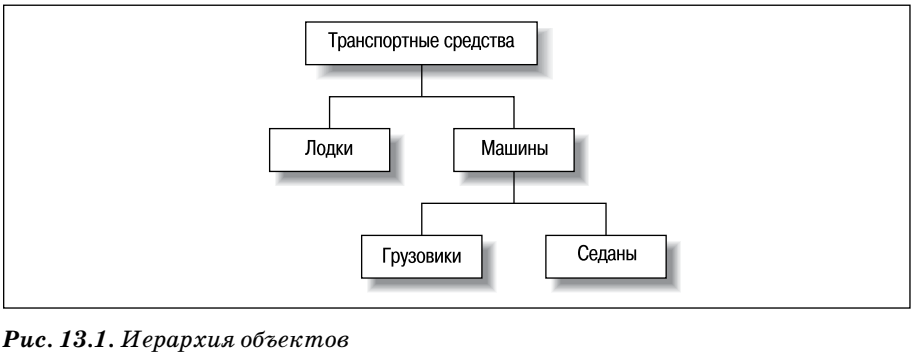

***Рис. 13.1.** Иерархия объектов*

Одно из возможных представлений этих данных может выглядеть так:

```
private var data:Object =
    [{name:"Vehicles", id:1, parentId:0, type:"parent"},
    {name:"Automobiles", id:2, parentId:1, type:"parent"},
    {name:"Boats", id:3, parentId:0, type:"parent"},
    {name:"Trucks", id:4, parentId:1, type:"parent"},
    {name:"Sedans", id:5, parentId:2, type:"parent"}];
```

В каждом узле хранится идентификатор `id` и идентификатор родительского узла `parentId`. Подобные структуры данных быстро разрастаются, становятся громоздкими и усложняют отображение. Другой подход основан на использовании интерфейса `IHierarchicalData`; при таком подходе компонент `AdvancedDataGrid` может группировать данные, а компонент `Tree` отображает их в виде дерева. Интерфейс `IHierarchicalData` требует определения следующих методов:

`canHaveChildren(node:Object):Boolean`

Метод определяет, имеются у заданного узла дочерние узлы.

`dispatchEvent(event:Event):Boolean`

Метод передает событие.

`getChildren(node:Object):Object`

Метод возвращает объект с перечислением всех дочерних узлов заданного узла.

`getData(node:Object):Object`

Метод возвращает всю информацию об узле, включая его дочерние узлы, в виде объекта.

`getParent(node:Object):*`

Метод возвращает родительский узел для заданного узла.

`getRoot():Object`

Метод возвращает корневой узел для объекта с иерархическими данными.

`hasChildren(node:Object):Boolean`

Метод возвращает `true`, если у узла имеются дочерние узлы, или `false` в противном случае.

Класс `ObjectHierarchicalData` реализует все перечисленные методы для иерархической структуры, показанной на рис. 13.1:

```
package oreilly.cookbook
{
    import flash.events.EventDispatcher;
    import mx.collections.IHierarchicalData;

    public class ObjectHierarchicalData extends EventDispatcher
        implements IHierarchicalData
```

```
{
    private var source:Object;

    public function ObjectHierarchicalData(value:Object)
    {
        super();
        source = value;
    }
    /* В этой простой иерархии только узлы с типом 'parent'
       могут иметь дочерние узлы; у узлов с другими типами
       дочерних узлов быть не может */
    public function canHaveChildren(node:Object):Boolean
    {
        if (node.type == "parent") {
            return true;
        }
        return false;
    }
    /* Чтобы узнать, имеет ли заданный узел дочерние узлы,
       необходимо перебрать все остальные узлы и попытаться
       найти среди них те, у которых идентификатор родителя
       совпадает с идентификатором данного узла */
    public function hasChildren(node:Object):Boolean
    {
       trace(node.name);
        var children:Array = new Array();
            // = source.parentTask == parentId);
        for each(var obj in source) {
            if(obj.parentTask == node.id) {children.push(obj);
            }
        }
        if (children.length > 0)
            return true;
        return false;
    }
    /* Для любого заданного узла возвращает массив
       со всеми узлами, которые являются дочерними
       по отношению к нему */
    public function getChildren(node:Object):Object
    {
        var parentId:String = node.id;
        var children:Array = new Array();
        for each(var obj in source) {
            if(obj.parentTask == parentId) {
                children.push(obj);
            }
        }
        return children;
    }
    public function getData(node:Object):Object
    {
```

```
                for each(var obj in source) {
                    for(var prop in node) {
                        if(obj[prop] == node[prop]) {
                            return obj;
                        } else {
                            break;
                        }
                    }
                }
                return null;
            }
            /* Метод должен вернуть все корневые объекты,
               в данном случае это все узлы с родительским узлом '0' */
            public function getRoot():Object
            {
                var rootsArr:Array = new Array();
                for each(var obj in source) {
                    if(obj.parentTask == "0") {
                         rootsArr.push(obj);
                    }
                }
                return rootsArr;
            }

            public function getParent(node:Object):*
            {
                for each(var obj in source) {
                    if(obj.parentTask == node.parentTask) {
                        return obj;
                    }
                }
                return null;
            }
        }
    }
```

Теперь, когда готовы все необходимые методы для определения отношений между узлами объекта данных, новый класс иерархических данных может быть назначен свойству `dataProvider` **компонента** `AdvancedDataGrid`**. Компонент сможет правильно отобразить отношения между иерархическими данными. Объект данных с отношениями между узлами, описываемыми при помощи свойств** `parentTask` **и** `id`**, передается новому объекту** `ObjectHierarchicalData`**:**

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300"
creationComplete="init()">
    <mx:RemoteObject>
    </mx:RemoteObject>
    <mx:Script>
        <![CDATA[
            import mx.collections.*;
```

```
            [Bindable]
            private var ohd:ObjectHierarchicalData;
            /* Огромный объект, который будет использован
               для заполнения объекта ObjectHierarchicalData */
            private var largeObject:Object =
            [
            {"id":"1", "name":"Misc", "type":"parent",
                "parentTask":"0"},
            {"id":"2", "name":"Clean the kitchen",
                "type":"parent", "parentTask":"0"},
            {"id":"3", "name":"Pay the bills",
                "type":"parent", "parentTask":"0"},
            {"id":"4", "name":"Paint the shed",
                "type":"parent", "parentTask":"1"},
            {"id":"5", "name":"Get ready for party",
                "type":"parent", "parentTask":"1"},
            {"id":"6", "name":"Do the dishes",
                "type":"child", "parentTask":"2"},
            {"id":"7", "name":"Take out trash",
                "type":"child", "parentTask":"2"},
            {"id":"8", "name":"Gas Bill",
                "type":"child", "parentTask":"3"},
            {"id":"9", "name":"Registration",
                "type":"child", "parentTask":"3"},
            {"id":"10", "name":"Fix the car",
                "type":"parent", "parentTask":"0"},
            {"id":"11", "name":"New tires",
                "type":"child", "parentTask":"10"},
            {"id":"12", "name":"Emissions test",
                "type":"child", "parentTask":"10"},
            {"id":"13", "name":"Get new paint",
                "type":"child", "parentTask":"4"},
            {"id":"14", "name":"Buy brushes",
                "type":"child", "parentTask":"4"},
            {"id":"15", "name":"Buy Drinks",
                "type":"child", "parentTask":"5"},
            {"id":"16", "name":"clean living room",
                "type":"child", "parentTask":"5"},
            {"id":"16", "name":"finish invitations",
                "type":"child", "parentTask":"5"} ];
            private function init():void
            {
                ohd = new ObjectHierarchicalData(largeObject);
            }
        ]]>
    </mx:Script>
    <mx:AdvancedDataGrid dataProvider="{ohd}" width="300" height="200">
<mx:columns>
            <!-- Отображаться будет только поле name,
                 а компонент AdvancedDataGrid обеспечит
                 отображение связей "родитель-потомок" -->
```

```
            <mx:AdvancedDataGridColumn dataField="name"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
</mx:Canvas>
```

13.6. Перемещение по коллекции и сохранение текущей позиции

Задача

Требуется организовать двустороннее перемещение по коллекции и сохранение позиции, в которой была прекращена обработка элементов.

Решение

Используйте метод `createCursor` класса `ListViewCollection` для создания *курсора*, способного перемещаться между элементами коллекции и сохранять текущую позицию. Позднее по сохраненной позиции можно определить, где завершилась обработка.

Обсуждение

Курсоры используются для перебора элементов в представлении данных коллекции, обращения и модификации данных в коллекции. *Курсор* фактически является индикатором позиции; он указывается на некоторый элемент коллекции. Для получения курсора используется метод `createCursor`. Методы и свойства курсоров определяются в интерфейсе `IViewCursor`.

Методы `IViewCursor` позволяют перемещать курсор в прямом и обратном направлении, искать в коллекции элементы, удовлетворяющие некоторому критерию, получать элементы в заданной позиции, сохранять позицию последнего обращения к коллекции, а также добавлять, удалять и изменять значения элементов.

При использовании стандартных классов коллекций Flex – `ArrayCollection` и `XMLListCollection` – вы работаете с интерфейсом `IViewCursor` напрямую, без создания специального экземпляра `IViewCursor`. Пример:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import mx.collections.SortField;
            import mx.collections.Sort;
            import mx.collections.IViewCursor;
            import mx.collections.CursorBookmark;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var coll:ArrayCollection;
```

```
            [Bindable]
            private var cursor:IViewCursor;

            private function init():void {
                coll = new ArrayCollection([{city:"Columbus", state:"Ohio",
                    region:"East"}, {city:"Cleveland", state:"Ohio",
                    region:"East"}, {city:"Sacramento", state:"California",
                    region:"West"}, {city:"Atlanta", state:"Georgia",
                    region:"South"}]);
                cursor = coll.createCursor();
            }
```

В следующем примере метод `findFirst` **объекта** `IViewCursor` **используется для поиска первого объекта коллекции, у которого любое свойство содержит текст, введенный пользователем в компоненте** `TextInput`**:**

```
            private function findRegion():void {
                var sort:Sort = new Sort();
                sort.fields = [new SortField("region")];
                coll.sort = sort;
                coll.refresh();
                cursor.findFirst({region:regionInput.text});
            }
            private function findState():void {
                var sort:Sort = new Sort();
                sort.fields = [new SortField("state")];
                coll.sort = sort;
                coll.refresh();
                cursor.findFirst({region:stateInput.text});
            }

        ]]>
    </mx:Script>
    <mx:Label text="{cursor.current.city}"/>
    <mx:Button click="cursor.moveNext()" label="Next"/>
    <mx:Button click="cursor.movePrevious()" label="Previous"/>
    <mx:HBox>
        <mx:TextInput id="regionInput"/>
        <mx:Button click="findRegion()" label="find region"/>
    </mx:HBox>
    <mx:HBox>
        <mx:TextInput id="stateInput"/>
        <mx:Button click="findRegion()" label="find state"/>
    </mx:HBox>
</mx:VBox>
```

В интерфейсе `IViewCursor` **определены три метода поиска в коллекциях:**

`findFirst(values:Object):Boolean`

Метод устанавливает курсор в позицию первого элемента, соответствующего критерию.

`findLast(values:Object):Boolean`

Метод устанавливает курсор в позицию последнего элемента, соответствующего критерию.

`findAny(values:Object):Boolean`

Метод устанавливает курсор в позицию любого элемента, соответствующего критерию. Этот метод работает быстрее других; используйте его, если вам не нужен именно первый или последний элемент.

Ни один из этих методов не работает с неотсортированными коллекциями `ArrayCollection` или `XMLListCollection`.

13.7. Создание объекта HierarchicalViewCollection

Задача

Требуется создать коллекцию, которая позволяет работать с объектом `IHierarchicalData` в коллекции.

Решение

Создайте класс, реализующий интерфейс `IHierarchicalData` для определения родительского и дочерних узлов каждого узла. Создайте объект `HierarchicalViewCollection` и передайте объект `IHierarchicalData` конструктору класса `HierarchicalViewCollection`.

Обсуждение

По умолчанию для работы с иерархическими данными `HierarchicalData` компонент `AdvancedDataGrid` создает объект `HierarchicalCollectionView`. Этот объект дает возможность `AdvancedDataGrid` получить объект `ArrayCollection` и применить все его методы к `HierarchicalData`. Данная возможность полезна не только при работе с `AdvancedDataGrid`, но и при работе с пользовательскими компонентами, отображающими иерархические данные. Класс `ObjectHierarchicalData` из рецепта 13.5 реализует интерфейс `IHierarchicalData` и предоставляет методы для определения отношений «родитель-потомок» между разными узлами. Класс `HierarchicalViewCollection` использует эти методы для визуального открытия/закрытия узлов, а также для проверки присутствия некоторого значения в объекте данных. В этом рецепте `ObjectHierarchicalData` используется для создания экземпляра `HierarchicalViewCollection`.

Класс `HierarchicalViewCollection` содержит следующие методы:

`addChild(parent:Object, newChild:Object):Boolean`

Добавляет дочерний узел к узлу данных.

`addChildAt(parent:Object, newChild:Object, index:int):Boolean`

Добавляет дочерний узел в заданной позиции.

`closeNode(node:Object):void`

Закрывает узел, скрывая все его дочерние узлы.

`contains(item:Object):Boolean`

Ищет в коллекции объект данных, чтобы определить, существует ли этот объект в коллекции. Таким образом, при передаче сложного объекта с тем же содержимым, что у объекта в коллекции, но находящегося в другом блоке памяти, метод не вернет `true`.

`createCursor():IViewCursor`

Возвращает новый экземпляр итератора для элементов представления данных.

`getParentItem(node:Object):*`

Возвращает родительский узел.

`openNode(node:Object):void`

Открывает узел с отображением всех его дочерних узлов.

`removeChild(parent:Object, child:Object):Boolean`

Удаляет дочерний узел из родительского узла.

`removeChildAt(parent:Object, index:int):Boolean`

Удаляет дочерний узел в заданной позиции.

Определение узла, с которым должны выполняться операции, зависит от качественной реализации метода `getData` интерфейса `IHierarchicalData`. Метод `getData` получает объект с парой «ключ-значение» и возвращает узел, содержащий эту же пару; это позволяет `HierarchicalViewCollection` определить, с каким объектом в источнике данных должна выполняться операция. В следующем фрагменте большой иерархический объект данных определяется и передается объекту `HierarchicalData`, после чего создается объект `HierarchicalCollectionView`:

```
var largeObject:Object =
    [{ id:"1", name:"Misc", type:"parent", parentTask:"0"},
    {id:"2", name:"Clean the kitchen", type:"parent", parentTask:"0"},
    {id:"3", name:"Pay the bills", type:"parent", parentTask:"0"},
    {id:"4", name:"Paint the shed", type:"parent", parentTask:"1"},
    {id:"5", name:"Get ready for party", type:"parent", parentTask:"1"},
    {id:"6", name:"Do the dishes", type:"child", parentTask:"2"},
    {id:"7", name:"Take out trash", type:"child", parentTask:"2"},
    {id:"8", name:"Registration", type:"child", parentTask:"3"},
    {id:"9", name:"Fix the car", type:"parent", parentTask:"0"},
    {id:"10", name:"New tires", type:"child", parentTask:"9"},
    {id:"11", name:"Get new paint", type:"child", parentTask:"4"},
    {id:"12", name:"Buy Drinks", type:"child", parentTask:"5"},
    {id:"13", name:"finish invitations", type:"child", parentTask:"5"}];

    /* Создание класса, реализующего интерфейс
       IHierarchicalData */
```

```
var dataObj:ObjectHierarchicalData =
    new ObjectHierarchicalData(largeObject);

/* Созданный класс передается классу
   HierarchicalCollectionView */
var hCollView:HierarchicalCollectionView =
    new HierarchicalCollectionView(dataObj);
hCollView.openNode(largeObject[2]);

var ac:ArrayCollection  = hCollView.getChildren(
    hCollView.source.getData( {id:"3"}));
hCollView.closeNode(hCollView.source.getData(
    {name:"Pay the bills"})
```

`HierarchicalViewCollection` **инкапсулирует объект** `IHierarchicalData` **и предоставляет методы для создания представлений на основе объектов коллекции с использованием метода** `getChildren`**.**

13.8. Фильтрация и сортировка XMLListCollection

Задача

Требуется отфильтровать и отсортировать `XMLListCollection`**.**

Решение

Используйте свойства `filterFunction` **и** `sortFunction` **класса** `ListViewCollection`**, расширяемого классом** `XMLListCollection`**, или просто задайте объект типа** `Sort` **свойству** `sort` **класса** `XMLListCollections`**.**

Обсуждение

Класс `XMLListCollection` **описывает данные XML с несколькими узлами, содержащимися в корневом узле. Например, набор элементов** `food` **в узле** `nutrition` **преобразуется в объект** `XMLListCollection`**, что позволяет рассматривать узлы** `food` **как коллекцию:**

```
<nutrition>
 <food>
    <name>Avocado Dip</name>
    <calories>110</calories>
    <total-fat>11</total-fat>
    <saturated-fat>3</saturated-fat>
    <cholesterol>5</cholesterol>
    <sodium>210</sodium>
    <carb>2</carb>
    <fiber>0</fiber>
    <protein>1</protein>
</food>
...
</nutrition>
```

Коллекция `XMLListCollection` **фильтруется таким же способом, как** `ArrayCollection`**: передачей ссылки на функцию, которая получает объект и возвращает логический признак его присутствия в отфильтрованном представлении. Пример:**

```
coll.filterFunction = lowCalFilter;
private function lowCalFilter(value:Object):Boolean {
    if(Number(value.calories) < 200) {
        return true;
    }
    return false;
}
```

Для сортировки `XMLListCollection` **понадобится объект** `Sort`**, массив** `fields` **которого заполнен объектами** `SortField`**:**

```
var sort:Sort = new Sort();
  sort.fields = [new SortField("calories",
      false, false, true)];
  coll.sort = sort;
  coll.refresh();
```

В приведенном далее листинге продемонстрировано создание `XMLListCollection` **по результатам вызова** `HTTPService`**, сортировка и фильтрация полученных данных:**

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" creationComplete="xmlService.send()">
    <mx:HTTPService url="assets/data.xml" resultFormat="xml"
    id="xmlService" result="createXMLCollection(event)"/>
    <mx:Script>
        <![CDATA[
            import mx.collections.SortField;
            import mx.collections.Sort;
            import mx.rpc.events.ResultEvent;
            import mx.collections.XMLListCollection;

            [Bindable]
            private var coll:XMLListCollection;
            private function createXMLCollection(
                event:ResultEvent):void {
                var list:XMLList = new XMLList(event.result);
                coll = new XMLListCollection(list.food);
                var sort:Sort = new Sort();
                sort.fields = [new SortField("calories", false,
                    false, true)];
                coll.sort = sort;
                coll.refresh();
              }
           private function applyFilter():void {
               coll.filterFunction = lowCalFilter;
               coll.refresh();
            }
```

```
            private function
                lowCalFilter(value:Object):Boolean {
                if(Number(value.calories) < 200) {
                    return true;
                }
                return false;
            }

        ]]>
    </mx:Script>
    <mx:DataGrid dataProvider="{coll}">
            <mx:columns>
                  <mx:DataGridColumn dataField="calories"/>
                  <mx:DataGridColumn dataField="name"/>
            </mx:columns>
    </mx:DataGrid>
    <mx:Button click="applyFilter()" label="filter"/>
</mx:VBox>
```

Для выполнения сложной фильтрации применяются конструкции E4X с различными узлами коллекции XML. Например, для обращения к атрибутам используется синтаксис `@`:

```
private function lowFatFilter(value:Object):Boolean {
    if(value.calories(@fat) < Number(value.calories)/5) {
          return true;
    }
    return false;
}
```

13.9. Сортировка коллекции по нескольким полям

Задача

Требуется отсортировать коллекцию по нескольким полям.

Решение

Передайте объекту `Sort` несколько объектов `SortField` и задайте его свойству `sort` коллекции.

Обсуждение

Так как массивы могут сортироваться по нескольким полям, поле `fields` объекта `Sort` содержит массив. Множественные поля образуют иерархию сортировки, в которой все объекты сортируются по группам: сначала по свойству `field` первого объекта `SortField`, затем по второму и т. д. В следующем примере коллекция сортируется сначала по свойству `region`, а затем по свойству `state`:

```
coll = new ArrayCollection([{city:"Cleveland", state:"Ohio", region:"East"},
{city:"Sacramento", state:"California", region:"West"}, {city:"Atlanta",
state:"Georgia", region:"South"}, {city:"Columbus", state:"Ohio",
region:"East"}]);
var sort:Sort = new Sort();
sort.fields = [new SortField("region"), new SortField("state")];
coll.sort = sort;
coll.refresh();
```

Состояние коллекции после сортировки показано на рис. 13.2.

city	region	state
Cleveland	East	Ohio
Columbus	East	Ohio
Atlanta	South	Georgia
Sacramento	West	California

Рис. 13.2. *Сортировка данных по нескольким полям*

13.10. Хронологическая сортировка в коллекциях

Задача

Требуется отсортировать коллекцию по датам, хранящимся в строковых свойствах объектов данных.

Решение

Создайте объекты `Date` на основе свойств `date` элементов данных. Воспользуйтесь методом `dateCompare` класса `mx.utils.ObjectUtil` для сравнения дат.

Обсуждение

Класс `ObjectUtil` содержит метод `dateCompare`, который определяет, какой из двух объектов `Date` соответствует более ранней дате. Он может использоваться для сортировки коллекции объектов `Date`; создайте функцию `sortFunction`, возвращающую результат вызова метода `ObjectUtil.dateCompare`. Метод `dateCompare` возвращает 1, 0 или –1: 0 возвращается, если объекты имеют равные или неопределенные значения; 1, если первое значение не определено или предшествует второму в порядке сортировки; –1, если второе значение не определено или предшествует первому в порядке сортировки.

```
import mx.collections.Sort;
import mx.collections.ArrayCollection;
import mx.utils.ObjectUtil;
```

```
// Функция сортировки должна иметь следующую сигнатуру:
// function [name](a:Object, b:Object, fields:Array = null):int
private function sortFunction(a:Object, b:Object, fields:Array = null):int {
    var tempDateA:Date = new Date(Date.parse(a.dob));
    var tempDateB:Date = new Date(Date.parse(b.dob));
    return ObjectUtil.dateCompare(tempDateA, tempDateB);
}
private var arrColl:ArrayCollection;
private function init():void {
    arrColl = new ArrayCollection([{name:"Josh", dob:"08/17/1983"},
        {name:"John", dob:"07/30/1946"}, {name:"John", dob:"07/30/1990"},
        {name:"John", dob:"07/30/1986"}]);
    var sort:Sort = new Sort();
    sort.compareFunction = sortFunction;
    arrColl.sort = sort;
    arrColl.refresh();
    trace(arrColl);
}
```

13.11. Глубокое копирование ArrayCollection

Задача

Требуется скопировать все элементы индексируемого массива или объекта в новый объект.

Решение

Используйте метод `mx.utils.ObjectUtil.copy`.

Обсуждение

Как нетрудно убедиться, копирование объекта сводится к простому созданию указателя на новый объект; это означает, что любые изменения в значениях первого объекта будут автоматически отражаться на втором объекте:

```
var objOne:Object =
    {name:"foo", data:{first:"1", second:"2"}};
var objTwo = objOne;
objOne.data.first = "4";
trace(objTwo.data.first);//Выводит 4
```

Чтобы полностью скопировать все данные объекта в другой объект, воспользуйтесь методом `copy` класса `mx.utils.ObjectUtil`. Метод получает объект и возвращает *глубокую* копию объекта в новом блоке памяти. Это означает, что в ходе копирования создаются дубликаты всех свойств объекта, никак не связанные со свойствами оригинала. Пример использования метода:

```
var objTwo = mx.utils.ObjectUtil.copy(objOne);
```

Работа метода copy основана на создании объекта ByteArray на основе переданного объекта и последующей записи ByteArray в новый объект:

```
var ba:ByteArray = new ByteArray();
ba.writeObject(objToCopy);
ba.position = 0;
var objToCopyInto:Object = ba.readObject();
return objToCopyInto;
```

Теперь предыдущий пример работает так, как и предполагалось:

```
var objOne:Object = {name:"foo", data:{first:"1", second:"2"}};
var objTwo = objOne;
var objThree = mx.utils.ObjectUtil.copy(objOne);
objOne.data.first = "4";
trace(objTwo.data.first); //Выводит 4
trace(objThree.data.first);
    //Выводит 1, т. е. исходное значение 1
```

Копирование объекта конкретного типа в новый объект того же типа создает некоторые трудности. При попытке выполнения следующего кода происходит ошибка:

```
var newFoo:Foo = ObjectUtil.copy(oldFoo) as Foo;
```

Flash Player не знает, как преобразовать ByteArray в тип, запрашиваемый в преобразовании. В результате использования ByteArray объект сериализуется в двоичные данные AMF (ActionScript Message Format) так же, как при пересылке сериализованных объектов посредством Flash Remoting. Чтобы десериализовать объект данных, необходимо зарегистрировать тип в Flash Player методом flash.net.registerClassAlias. Метод регистрирует класс, после чего двоичное представление любого объекта этого типа может быть десериализовано в объект. Метод registerClassAlias получает два параметра:

```
public function registerClassAlias(aliasName:String,
    classObject:Class):void
```

В первом параметре передается полное имя класса, а во втором – объект типа Class. Полное имя класса имеет вид mx.containers.Canvas или com.oreilly.cookbook.Foo. В приведенном примере ни имя класса, ни ссылка на класс не будут известны при копировании объекта. К счастью, метод flash.utils.getQualifiedClassName возвращает полное имя класса для переданного ему объекта, а метод flash.utils.getDefinitionByName возвращает ссылку на класс переданного объекта. В совокупности эти два метода позволяют зарегистрировать класс любого объекта:

```
private function copyOverObject(objToCopy:Object,
    registerAlias:Boolean = false):Object
 {
          if(registerAlias) {
          var className:String =
              flash.utils.getQualifiedClassName(objToCopy);
```

```
            flash.net.registerClassAlias(className,
                (flash.utils.getDefinitionByName(className) as Class));
            }
        return mx.utils.ObjectUtil.copy(objToCopy);
    }
```

Теперь коллекция `ArrayCollection` объектов с сильной типизацией может быть корректно скопирована с передачей каждого объекта из `ArrayCollection` методу `copyOverObject`:

```
private function copyOverArray(arr:Array):Array {
    var newArray:Array = new Array();
    for(var i:int; i<arr.length; i++) {
    newArray.push(copyOverObject(arr[i], true));
    }
    return newArray;
}
var ac:ArrayCollection = new ArrayCollection([{name:'Joseph', id:21},
    foo, {name:'Josef', id:81}, {name:'Jose', id:214}]);
var newAC:ArrayCollection =
    new ArrayCollection(copyOverArray(ac.source));
```

Все данные, содержащиеся в объектах исходной коллекции `ArrayCollection`, будут присутствовать в копии `ArrayCollection` и при простом копировании методом `mx.utils.ObjectUtil.copy`. Однако в этом случае информация о классе объектов будет отсутствовать, и любые попытки преобразования объекта из коллекции к другому типу приведут к ошибке или неопределенному значению.

13.12. Использование объектов данных с уникальными идентификаторами

Задача

Несколько объектов данных используются в разных местах приложения. Требуется сделать так, чтобы всем объектам назначались уникальные идентификаторы, которые могли бы использоваться для сравнения объектов и принятия решений о том, представляют ли они один блок данных.

Решение

Реализуйте в объектах данных интерфейс `IUID`. Используйте метод `mx.core.UIDUtil.createUID` для генерирования идентификаторов объектов, уникальных в масштабах приложения.

Обсуждение

Описанная ситуация особенно важна при обмене сообщениями через Adobe LiveCycle или другие службы, так как при использовании про-

стого (==) и сложного (===) операторов сравнения объекты сравниваются по ссылке. Чтобы проверить, представляют ли два объекта одни и те же данные, часто приходится сравнивать свойства всех их полей. Для больших и сложных объектов это приведет к неэффективным затратам ресурсов. Но при реализации интерфейса IUID в класс включается свойство-идентификатор, при помощи которого можно узнать, представляют ли объекты одинаковые данные. Даже если один объект был получен в результате глубокого копирования другого, свойство uid останется прежним, и объекты будут идентифицированы как представляющие одинаковые данные.

Идентификатор, генерируемый методом createUID класса UIDUtil, представляет собой 32-разрядное шестнадцатеричное число в следующем формате:

```
E4509FFA-3E61-A17B-E08A-705DA2C25D1C
```

В следующем примере метод createUID используется для создания нового экземпляра класса Message, реализующего IUID. Методы get uid и set uid интерфейса IUID предоставляют доступ к сгенерированному идентификатору объекта:

```
package
{
    import mx.core.IUID;
    import mx.utils.UIDUtil;

    [Bindable]
    public class Message implements IUID {
        public var messageStr:String;
        public var fromID:String;
        private var _uid:String;
        public function Message() {
            _uid = UIDUtil.createUID();
        }
        public function get uid():String {
            return _uid;
        }

        public function set uid(value:String):void {
            // Идентификатор уже создан, поэтому этот метод
            // ничего полезного не делает, но его
            // присутствия требует интерфейс IUID.
        }
    }
}
```

14

Привязка данных

Flex Framework предоставляет мощную организационную структуру для построения приложений, управляемых компонентами. В эту мощную структуру входит событийный механизм *привязки данных*, при помощи которого объекты могут подписываться на оповещения об изменениях свойств других объектов.

Привязка данных предоставляет удобный способ передачи данных между разными уровнями приложение; свойство-источник связывается со свойством-приемником. Изменение в свойствах приемного объекта происходит после передачи исходным объектом события, оповещающего всех приемников об изменении. Чтобы разрешить привязку данных для свойства, следует определить тег метаданных `[Bindable]` одним из трех способов.

- Перед определением класса:

```
package com.oreilly.flexcookbook
{
    import flash.events.EventDispatcher;

    [Bindable]
    public class DataObject extends EventDispatcher{}
}
```

  Добавление тега `[Bindable]` перед определением класса создает выражение привязки для всех открытых атрибутов этого класса. Классы, использующие привязку, должны реализовать интерфейс `IEventDispatcher`, потому что привязка данных представляет собой событийную систему оповещений для копирования свойств источника в свойства приемника.

- Перед открытой, защищенной или приватной переменной:

```
[Bindable] private var _lastName:String;
[Bindable] protected var _age:Number;
[Bindable] public var firstName:String;
```

Привязываемые переменные с уровнем доступа `private` доступны для привязки только в пределах класса. Защищенные переменные могут использоваться для привязки в классе, в котором объявлена переменная, и во всех его субклассах. Открытые переменные доступны для привязки в самом классе, в его субклассах и во всех внешних классах.

- Перед определением открытого, защищенного или приватного атрибута с использованием методов `get/set`:

```
private var _lastName:String;
...
[Bindable]
public function get lastName():String
{
    return _lastName;
}
public function set lastName( str:String ):void
{
    _lastName = str;
}
```

Когда привязка включается для методов `get/set` добавлением тега метаданных `[Bindable]` над объявлением get-метода, привязка к свойству может осуществляться в синтаксисе «точечной записи». Иначе говоря, обращение к свойству выглядит так же, как обращение к обычной переменной без привязки, например `Owner.property` для задания источника привязки.

Во внутренней реализации привязываемые свойства объектов Framework передают событие `propertyChange` при изменении своих значений. Тег метаданных `[Bindable]` принимает атрибут `event`, который может определяться с пользовательским типом события:

```
[Bindable(event="myValueChanged")]
```

По умолчанию атрибуту `event` задается тип события `propertyChange`. Если оставить ему значение по умолчанию, приемные свойства будут оповещаться об изменениях источника событиями этого типа. Если для оповещения объектов об обновлениях будет задан пользовательский тип события, вы должны обеспечить явную передачу этого события из класса.

Привязка посредством оповещающих событий происходит при инициализации объекта, содержащего свойство-источник, и далее при любых изменениях этого свойства. Чтобы выполнить принудительную обработ-

ку привязки для объекта-приемника, являющегося субклассом `mx.core.UIComponent`, следует использовать метод `executeBindings`.

Привязка образует уровень синхронизации данных между объектами, упрощающий создание приложений с расширенной функциональностью. В этой главе рассматриваются различные приемы интеграции привязки данных в архитектуру приложения.

14.1. Привязка к свойству

Задача

Требуется привязать свойство одного объекта к свойству другого объекта.

Решение

Используйте либо фигурные скобки (`{}`) в объявлении компонента MXML, либо тег `<mx:Binding>`.

Обсуждение

Когда вы привязываете свойство одного объекта (*приемника*) к свойству другого объекта (*источника*), объект-источник передает событие, оповещающее объект-приемник об изменении значения свойства. Во внутренней реализации значение свойства объекта-источника копируется в свойство объекта-приемника. Для создания привязки в объявлении MXML используются фигурные скобки (`{}`) или тег `<mx:Binding>`. Пример назначения привязки в объявлении компонента:

```
<mx:Application
       xmlns:mx="http://www.adobe.com/2006/mxml"
       layout="vertical">
       <mx:Panel
              paddingLeft="5" paddingRight="5"
              paddingTop="5" paddingBottom="5">
              <mx:Label text="Enter name:" />
              <mx:TextInput id="nameInput" maxChars="20" />
              <mx:HRule width="100%" />
              <mx:Label text="You've typed:" />
              <mx:Text text="{nameInput.text}" />
      </mx:Panel>
</mx:Application>
```

В этом примере свойство `text` компонента `Text` привязывается к свойству `text` компонента `TextInput`. С изменением свойства `text` экземпляра `TextInput` автоматически изменяется значение свойства `text` экземпляра `Text`. В фигурных скобках точечная запись используется для ссылки на атрибут `text` экземпляра `TextInput`, которому присвоен идентификатор `nameInput`.

Для определения выражения привязки данных также может использоваться тег <mx:Binding> в коде MXML; результат будет таким же, как при включении фигурных скобок в объявление компонента. Какой из способов использовать в конкретном случае? Ответ зависит от компонента. В контексте архитектуры MVC (Model-View-Controller) при определении тега <mx:Binding> вы создаете контроллер для своего представления. При использовании фигурных скобок представление не отделяется от контроллера, потому что компонент представления выполняет функции контроллера.

Запись с фигурными скобками проста, ускоряет разработку и приводит к тому же результату, однако тег <mx:Binding> тоже может пригодиться в процессе разработки, поскольку он обладает удобочитаемым синтаксисом и позволяет определить несколько свойств-источников для одного приемника.

При использовании тега <mx:Binding> необходимо определить атрибут source и атрибут destination:

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

       <mx:Binding source="nameInput.text" destination="nameOutput.text" />
       <mx:Panel
           paddingLeft="5" paddingRight="5"
           paddingTop="5" paddingBottom="5">
           <mx:Label text="Enter name:" />
           <mx:TextInput id="nameInput" maxChars="20" />
           <mx:HRule width="100%" />
           <mx:Label text="You've typed:" />
           <mx:Text id="nameOutput" />
    </mx:Panel>

</mx:Application>
```

Результат ничем не отличается от предыдущего примера, но на этот раз свойствам id компонентов TextInput и Text задаются значения, которые используются при определении свойств source и destination в объявлении <mx:Binding>.

Обратите внимание: в атрибутах source и destination фигурные скобки не нужны, тогда как при встроенном объявлении привязки их присутствие обязательно. Это объясняется тем, что значения атрибутов source и destination обрабатываются как выражения ActionScript. Это позволяет включить в выражение любые дополнительные данные. Например, если вы хотите, чтобы в компоненте Text в этих примерах выводилась длина входного текста с присоединенной строкой «letters.», определите значение атрибута source следующим образом:

```
<mx:Binding source="nameInput.text.length + ' letters.'"
       destination="nameOutput.text" />
```

14.2. Привязка к функции

Задача

Требуется использовать функцию в качестве источника для привязки свойства.

Решение

Включите фигурные скобки в объявление компонента, чтобы передать привязанное свойство в аргументе функции, или определите функцию, вызываемую по событию привязки.

Обсуждение

Обновление свойства объекта-приемника на основании свойства объекта-источника – простой и быстрый механизм синхронизации данных. Но если привязка ограничивается только значениями свойств, свойство приемника должно относиться к тому же типу, что и свойство источника. Однако может возникнуть ситуация, когда свойство приемника должно относиться к другому типу для отображения разнородных, хотя и логически связанных значений. В таких ситуациях помогает привязка с использованием функций.

Существует два способа привязки на базе функций: передача привязанного свойства в аргументе и определение функции, привязываемой к свойству.

В следующем примере привязанное свойство объекта-источника передается функции, обновляющей значение свойства объекта-приемника:

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

       <mx:CurrencyFormatter id="formatter" precision="2" />

       <mx:Form>
              <mx:FormItem label="Enter the withdrawl amount:">
                     <mx:TextInput id="amtInput" />
              </mx:FormItem>
              <mx:FormItem label="Formatted amount:">
                     <mx:TextInput editable="false" restrict="1234567890"
                         text="{formatter.format( amtInput.text )}" />
              </mx:FormItem>
       </mx:Form>

</mx:Application>
```

Свойство `text` первого экземпляра `TextInput` используется для форматирования данных, отображаемых вторым экземпляром `TextInput` с применением тега `<mx:CurrencyFormatter>`. В результате привязки метод `format`

будет вызываться при каждом обновлении свойства `text` экземпляра `amtInput`. Передача привязанного свойства в аргументе функции является удобным механизмом синхронизации данных без установления однозначного соответствия между источником и приемником.

Чтобы создать привязку к функции без передачи привязанного свойства в аргументе, воспользуйтесь атрибутом `event` тега метаданных `[Binding]` и определите функцию, привязанную к событию. При перехвате указанного события вызывается функция, которая выполняет обновление привязанных свойств. Пример:

```
<mx:Application
      xmlns:mx="http://www.adobe.com/2006/mxml"
      layout="vertical"
      creationComplete="initHandler();">

      <mx:Script>
        <![CDATA[
            private var _fruit:String;
            private var _fruits:Array = ["Apple", "Banana", "Orange"];

            private function initHandler():void
            {
                fruitCB.dataProvider = _fruits;
            }
            [Bindable(event="fruitChanged")]
            private function isOrangeChosen():Boolean
            {
                return _fruit == "Orange";
            }

            public function get fruit():String
            {
                return _fruit;
            }
            public function set fruit( str:String ):void
            {
                _fruit = str;
                dispatchEvent( new Event( "fruitChanged" ) );
            }

        ]]>
    </mx:Script>

    <mx:Label text="select a fruit:" />
    <mx:HBox>
        <mx:ComboBox id="fruitCB"
               change="{fruit = fruitCB.selectedLabel}" />
        <mx:Button label="Eat the orange."
               enabled="{isOrangeChosen()}" />
    </mx:HBox>
</mx:Application>
```

В этом примере атрибут enabled экземпляра Button привязывается к логическому значению, возвращаемому методом isOrangeChosen. Возвращаемое значение определяется значением переменной _fruit, обновляемой при изменении выбора в ComboBox. При любых изменениях атрибута fruit передается событие fruitChanged и вызывается метод isOrangeChosen, который, в свою очередь, обеспечивает обновление атрибута enabled экземпляра Button.

Фактически состояние блокировки кнопки привязывается к выбору строки в компоненте ComboBox. Определение функции для привязки опредоставляет удобный механизм обновления значений в объекте-приемнике, принадлежащих к другому типу, нежели свойства объекта-источника.

См. также

Рецепт 14.1.

14.3. Создание двусторонней привязки

Задача

Требуется связать свойства двух компонентов так, чтобы каждое из них было одновременно и источником, и приемником привязки.

Решение

Назначьте свойство каждого компонента источником в выражении привязки.

Обсуждение

Термин *двусторонняя привязка* означает, что каждый из двух компонентов является источником для приемного свойства другого компонента. Flex Framework поддерживает двустороннюю привязку и следит за тем, чтобы обновления свойств не привели к зацикливанию. Пример:

```
<mx:VBox

    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Label text="From Input 2:" />
    <mx:TextInput id="input1" text="{input2.text}" />
    <mx:HRule />
    <mx:Label text="From Input 1:" />
    <mx:TextInput id="input2" text="{input1.text}" />

</mx:VBox>
```

Каждый экземпляр TextInput является и источником, и приемником, а его изменение приводит к обновлению свойства text другого компонента.

Текст, вводимый в одном компоненте TextInput, копируется в другое поле TextInput.

См. также

Рецепт 14.1.

14.4. Привязка свойств в коде ActionScript

Задача

Требуется создать выражение привязки данных в коде ActionScript (вместо объявления в MXML).

Решение

Используйте класс mx.utils.binding.BindingUtils для создания объектов mx.utils.binding.ChangeWatcher.

Обсуждение

Создание выражений привязки данных в коде ActionScript позволяет более точно управлять тем, когда и как обновляются значения приемных свойств. Чтобы создать привязку в ActionScript, необходимо использовать класс BindingUtils для создания объекта ChangeWatcher. В классе BindingUtils определены два статических метода, которые могут использоваться для создания привязки: bindProperty и bindSetter.

Метод bindProperty используется почти так же, как тег <mx:Binding> в MXML:

```
var watcher:ChangeWatcher =
    BindingUtils.bindProperty( destination, "property",
    source, "property" );
```

Метод BindingUtils.bindSetter позволяет назначить функцию для обработки обновлений свойства источника:

```
var watcher:ChangeWatcher =
    BindingUtils.bindSetter( invalidateProperty, source, "property" );
...
private function invalidateProperty( arg:* ):void
{
    // Все необходимые действия
}
```

Вообще говоря, определять переменную ChangeWatcher при вызове статических методов bindProperty и bindSetter не обязательно. Однако в некоторых ситуациях возвращаемый объект ChangeWatcher может быть полезен – он предоставляет методы, которые могут использоваться во время выполнения для изменения источника данных, изменения приемного свойства и прерывания привязки.

В следующем примере метод `BindingUtils.bindProperty` **используется для создания привязки данных между свойством** `text` **компонента** `TextInput` **и свойством** `text` **компонента** `Text`**:**

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
        <![CDATA[
            import mx.binding.utils.ChangeWatcher;
            import mx.binding.utils.BindingUtils;

            private var _nameWatcher:ChangeWatcher;

            private function initHandler():void
            {
                nameWatcher = BindingUtils.bindProperty(
                    nameField, "text", nameInput, "text" );
            }

            private function clickHandler():void
            {
                if( _nameWatcher.isWatching() )
                {
                    _nameWatcher.unwatch();
                    btn.label = "watch";
                }
                else
                {
                    _nameWatcher.reset( nameInput );
                    btn.label = "unwatch";
                }
            }
        ]]>
    </mx:Script>

    <mx:Panel title="User Entry."
      paddingLeft="5" paddingRight="5"
          paddingTop="5" paddingBottom="5">

          <mx:Form>
                <mx:FormItem label="Name:">
                      <mx:TextInput id="nameInput" />
                </mx:FormItem>
          </mx:Form>

          <mx:HRule width="100%" />
          <mx:Label text="You Entered:" fontWeight="bold" />
          <mx:HBox>
               <mx:Label text="First Name:" />
```

```
            <mx:Text id="nameField" />
        </mx:HBox>
        <mx:Button id="btn" label="unwatch" click="clickHandler();" />
    </mx:Panel>
</mx:Application>
```

Привязка определяется методом `BindingUtils.bindProperty` как отношение «один к одному» между свойствами источника и приемника. В данном приведенном примере все изменения свойства `text` экземпляра `TextInput` отражаются в свойстве `text` экземпляра `Text`. Экземпляр `Button` прерывает жизненный цикл выражения привязки и сбрасывает его при помощи объекта `ChangeWatcher`.

Чтобы организовать более точное управление обновлением приемного свойства или обновлением нескольких приемников по одному источнику, воспользуйтесь методом `BindingUtils.bindSetter` для назначения функции, которая будет выполнять функции промежуточного звена привязки:

```
<mx:Application
       xmlns:mx="http://www.adobe.com/2006/mxml"
       layout="vertical"
       creationComplete="initHandler();">

       <mx:Script>
           <![CDATA[
                  import mx.binding.utils.ChangeWatcher;
                  import mx.binding.utils.BindingUtils;

                  private var _nameWatcher:ChangeWatcher;

                  private function initHandler():void
                  {
                       _nameWatcher = BindingUtils.bindSetter(
                           invalidateName, nameInput, "text" );
                  }

                  private function invalidateName( arg:* ):void
                  {
                      if( btn.label == "unwatch" )
                          nameField.text = nameInput.text;
                  }

                  private function clickHandler():void
                  {
                       if( _nameWatcher.isWatching() )
                       {
                           _nameWatcher.unwatch();
                           btn.label = "watch";
                       }
                       else
                       {
```

```
                        _nameWatcher.reset( nameInput );
                        btn.label = "unwatch";
                    }
                }
            ]]>
        </mx:Script>

        <mx:Panel title="User Entry."
                paddingLeft="5" paddingRight="5"
                paddingTop="5" paddingBottom="5">
                <mx:Form>
                    <mx:FormItem label="Name:">
                        <mx:TextInput id="nameInput" />
                    </mx:FormItem>
                </mx:Form>
                <mx:HRule width="100%" />
                <mx:Label text="You Entered:" fontWeight="bold" />
                <mx:HBox>
                    <mx:Label text="First Name:" />
                    <mx:Text id="nameField" />
                </mx:HBox>
                <mx:Button id="btn" label="unwatch"
                    click="clickHandler();" />
        </mx:Panel>

    </mx:Application>
```

Обновления приемника определяются действиями set-метода, передаваемого в первом параметре `BindingUtils.bindSet`. Этот set-метод используется в качестве обработчика события каждый раз, когда объект-приемник передает событие, оповещающее о его изменении. В приведенном примере свойство `text` обновляется на основании состояния свойства `label` экземпляра `Button`.

Хотя при каждом изменении свойства `text` экземпляра `nameInput` будет вызываться метод `invalidateName`, изменения приемного свойства определяются действиями объекта `ChangeWatcher`, а именно проверкой `if` в зависимости от состояния кнопки.

Помните, что для всех объектов `ChangeWatcher`, связываемых с экземпляром, следует вызывать метод `unwatch()`, завершающий отслеживание изменений; это необходимо для нормального уничтожения объектов уборщиком мусора. При создании объекта `ChangeWatcher` так, как это было сделано в предыдущих примерах с использованием класса `BindingUtils`, в памяти хранятся ссылки как на источник, так и на приемник привязки. Чтобы эти ссылки освободились, а память объекта была нормально освобождена, необходимо вызвать метод `unwatch`.

14.5. Привязка в цепочках свойств

Задача

Требуется определить свойство-источник привязки, которое является частью цепочки свойств.

Решение

Используйте точечную запись для обращения к свойствам в цепочке в теге `<mx:Binding>` или в фигурных скобках (`{}`) или же передайте массив строк в аргументе `chain` методов `BindingUtils.bindProperty` или `BindingUtils.bindSetter`.

Обсуждение

При определении свойства-источника в выражении привязки данных отслеживаются изменения во всех свойствах, ведущих к нему. Например, при создании привязки к свойству `text` компонента `TextInput` экземпляр `TextInput` является частью привязываемой цепочки свойств:

```
<mx:TextInput id="myInput" />
<mx:Label text="{myInput.text}" />
```

Формально класс, содержащий компонент `myInput`, тоже входит в цепочку свойств, но присутствие директивы `this` в определении выражения привязки данных не обязательно. Framework проверяет, что значение `myInput` отлично от `null`, а затем привязка перемещается вниз по цепочке до источника: свойства `text` экземпляра `TextInput`. Чтобы обновление было успешно инициировано, а значение из объекта-источника было скопировано в объект-приемник, привязка должна быть включена только для свойства-источника.

Обращение к свойству-источнику в цепочке свойств происходит по той же модели, которая используется при обращении к свойству-источнику компонента (как в предыдущем примере этого рецепта). В коде MXML привязываемые цепочки свойств определяются с использованием синтаксиса точечной записи:

```
<!-- Привязка цепочки свойств с использованием <mx:Binding> -->
<mx:Binding source="usermodel.name.firstName" destination="fNameField.text"
/>
<mx:Label id="fNameField" />
<!-- Привязка цепочки свойств с использованием фигурных скобок ({}).
<mx:Label text="{usermodel.name.firstName}" />
```

Чтобы определить привязываемую цепочку свойств в ActionScript 3, следует передать массив строковых значений при вызове метода `BindingUtils.bindProperty` или `BindingUtils.bindSetter`:

```
BindingUtils.bindProperty( nameField, "text",
    usermodel, ["name", "firstName"] );
```

```
BindingUtils.bindSetter( invalidateProperties,
    this, ["usermodel", "name", "firstName"] );
```

В аргументе `chain` **этих двух методов передается массив строк, определяющих привязываемую цепочку свойств относительно владельца (хоста).**

В следующем примере фигурные скобки, тег `<mx:Binding>` **и метод** `BindingUtils.bindProperty` **используются для определения выражений привязки данных, использующих цепочки свойств:**

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
        <![CDATA[

            import mx.binding.utils.BindingUtils;
            // Создание привязки данных
            // с использованием BindingUtils.

            private function initHandler():void
            {
                BindingUtils.bindProperty( lastNameField,
                    "text",usermodel, ["name", "lastName"] );
            }
            private function clickHandler():void
            {
                usermodel.name.firstName = fNameInput.text;
                usermodel.name.lastName = fNameInput.text;
                usermodel.birth.date = dateInput.text;
            }
        ]]>
    </mx:Script>

    <!-- Определение модели -->
    <mx:Model id="usermodel">
        <user>
            <name>
                  <firstName>Ted</firstName>
                  <lastName>Henderson</lastName>
           </name>
           <birth>
                  <date>February 29th, 1967</date>
              </birth>
        </user>
    </mx:Model>

    <!-- Создание привязки данных
         с использованием <mx:Binding> -->
    <mx:Binding source="usermodel.birth.date"
```

```
            destination="dateField.text" />
        <mx:Form>
            <mx:FormItem label="First Name:">
                <!-- create data binding using curly braces -->
                <mx:Text text="{usermodel.name.firstName}" />
         </mx:FormItem>
            <mx:FormItem label="Last Name:">
                <mx:Text id="lastNameField" />
            </mx:FormItem>
            <mx:FormItem label="Birthday:">
                <mx:Text id="dateField" />
            </mx:FormItem>
        </mx:Form>
        <mx:HRule />

        <mx:Form>
            <mx:FormItem label="First Name:">
                 <mx:TextInput id="fNameInput" />
            </mx:FormItem>
            <mx:FormItem label="Last Name:">
                <mx:TextInput id="lNameInput" />
            </mx:FormItem>
            <mx:FormItem label="Birthday:">
                <mx:TextInput id="dateInput" />
            </mx:FormItem>
            <mx:FormItem label="Submit Changes">
                 <mx:Button label="ok" click="clickHandler();" />
            </mx:FormItem>

        </mx:Form>
    </mx:Application>
```

См. также

Рецепты 14.1 и 14.3.

14.6. Привязка к свойствам XML с использованием E4X

Задача

Требуется привязать свойства объекта-приемника к источнику в формате XML.

Решение

Используйте ECMAScript for XML (E4X) при определении выражения привязки данных с использованием фигурных скобок или тега `<mx:Bindable>`.

Обсуждение

Поддержка E4X в ActionScript 3 обеспечивает фильтрацию данных из формата XML (Extensible Markup Language); используемые при этом выражения имеют синтаксис, сходный с синтаксисом выражений ActionScript. Ограниченный объем рецепта не позволяет описать все нюансы записи выражений E4X; важно понимать, что язык может использоваться для создания привязок между компонентами и XML.

Выражения E4X определяются в фигурных скобках при объявлении компонента и в тегах `<mx:Binding>`. E4X не может использоваться с классом `BindingUtils`. Чтобы вы лучше поняли, как работать с E4X, рассмотрим пример на базе следующего фрагмента XML:

```
<item>
    <name>Moe</name>
    <type>The brains.</type>
    <description>Has bowl cut.</description>
</item>
```

Выражение E4X можно включить в атрибут в фигурных скобках:

```
<mx:Label text="{_data..item.(name == 'Moe').description}" />
```

Привязка также может создаваться с использованием тега `<mx:Binding>`:

```
<mx:Binding source="_data..item.(name == 'Moe').description"
    destination="desc.text" />
<mx:Label id="desc" />
```

Оба способа приводят к одному результату. Фигурные скобки в атрибуте `source` тега привязки `<mx:Binding>` не обязательны, потому что значение обрабатывается как выражение ActionScript.

В следующем примере E4X используется для создания привязки свойства `dataProvider` компонентов `List` и `DataGrid`:

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

    <mx:Script>
        <![CDATA[

            [Bindable] private var _data:XML =
                   <items>
                       <item id='1'>
                           <name>Larry</name>
                           <type>The foil.</type>
                           <description>Has curly hair.</description>
                       </item>
                       <item id='2'>
                           <name>Moe</name>
                           <type>The brains.</type>
```

```
                        <description>Has bowl cut.</description>
                    </item>
                    <item id='3'>
                        <name>Curly</name>
                        <type>The braun.</type>
                        <description>Has bowl cut.</description>
                    </item>
                </items>;
        ]]>
    </mx:Script>

    <mx:Binding source="{_data..item.(@id == '1').name}
        {_data..item.(@id == '1').description.toLowerCase()}"
        destination="lab.text" />
    <mx:Label id="lab" />
    <mx:List width="200" dataProvider="{_data..item.name}" />
    <mx:DataGrid width="200" dataProvider="{_data..item}">
        <mx:columns>
            <mx:DataGridColumn dataField="name" />
            <mx:DataGridColumn dataField="type" />
        </mx:columns>
    </mx:DataGrid>
</mx:Application>
```

После инициализации компонентов обрабатывается привязка, и значения свойств обновляются в зависимости от переданных выражений E4X.

См. также

Рецепт 14.1.

14.7. Нестандартная привязка

Задача

Привязка данных должна происходить по пользовательскому событию (вместо стандартного события `propertyChange`).

Решение

Задайте атрибут `event` тега метаданных `[Bindable]` и передайте событие с указанием строки события в аргументе `type`.

Обсуждение

Инфраструктура привязки данных Flex Framework основана на событиях. По умолчанию привязка реализуется передачей события `propertyChange`. Во внутренней реализации значение приемного свойства обновляется без необходимости явной передачи этого события источником. Однако разработчик может назначить для привязки данных

пользовательский тип события, задавая свойство `event` тега метаданных `[Bindable]`. Пример:

```
Bindable(event="myValueChanged")]
```

При переопределении атрибута `event` по умолчанию в определении тега `[Bindable]` привязка будет работать только при явной передаче указанного события. В следующем примере пользовательское событие привязки используется для обновления свойств приемника:

```
<mx:Application
       xmlns:mx="http://www.adobe.com/2006/mxml"
       layout="vertical">
       <mx:Script>
          <![CDATA[

            private var _firstName:String;
            private var _lastName:String;
            public static const FIRST_NAME_CHANGED:String =
                "firstNameChanged";
            public static const LAST_NAME_CHANGED:String =
                "lastNameChanged";
            private function clickHandler():void
            {
                firstName = fnInput.text;
                lastName = lnInput.text;
            }

            [Bindable(event="firstNameChanged")]
            public function get firstName():String
            {
                return _firstName;
            }

            public function set firstName( str:String ):void
            {
                 firstName = str;
                 dispatchEvent( new Event(
                 FIRST_NAME_CHANGED ) );
            }

            [Bindable(event="lastNameChanged")]
            public function get lastName():String
            {
                return _lastName;
            }

            public function set lastName( str:String ):void
            {
                lastName = str;
                dispatchEvent( new Event( LAST_NAME_CHANGED ) );
            }
```

```
        ]]>
    </mx:Script>
    <mx:Panel title="User Entry."
        paddingLeft="5" paddingRight="5"
        paddingTop="5" paddingBottom="5">
        <mx:HBox>
            <mx:Label text="First Name:" />
            <mx:TextInput id="fnInput" />
        </mx:HBox>
        <mx:HBox>
            <mx:Label text="Last Name:" />
            <mx:TextInput id="lnInput" />
        </mx:HBox>
        <mx:Button label="submit" click="clickHandler();" />
        <mx:HRule width="100%" />
        <mx:Label text="You Entered:" fontWeight="bold" />
        <mx:HBox>
            <mx:Label text="First Name:" />
            <mx:Text text="{firstName}" />
        </mx:HBox>
        <mx:HBox>
            <mx:Label text="Last Name:" />
            <mx:Text text="{lastName}" />
        </mx:HBox>
    </mx:Panel>
</mx:Application>
```

Когда пользователь вводит свое имя и фамилию, обновляются свойства `firstName` и `lastName`. В каждом set-методе передается соответствующее событие, определенное в теге `[Bindable]`; оно инициирует обновление приемного свойства.

Нестандартная привязка обладает рядом полезных особенностей. В частности, она позволяет определить, когда именно обновляется приемное свойство в выражениях привязки данных. Так как работа привязки основана на модели событий, нестандартная привязка позволяет управлять тем, когда активизируется привязка данных (и активизируется ли она вообще).

В следующем примере запускается таймер, задерживающий передачу события привязки:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
        <![CDATA[
            private var _timer:Timer;
            private var _firstName:String;
            private var _lastName:String;
```

```
        public static const FIRST_NAME_CHANGED:String =
           "firstNameChanged";
        public static const LAST_NAME_CHANGED:String =
            "lastNameChanged";

        private function initHandler():void
        {
            _timer = new Timer( 2000, 1 );
            _timer.addEventListener( TimerEvent.TIMER_COMPLETE,
                 timerHandler );
        }

        private function clickHandler():void
        {
            firstName = fnInput.text;
            lastName = lnInput.text;
        }
        private function timerHandler( evt:TimerEvent ):void
        {
            dispatchEvent( new Event( FIRST_NAME_CHANGED ) );
        }

    [Bindable(event="firstNameChanged")]
    public function get firstName():String
    {
            return _firstName;
        }
        public function set firstName( str:String ):void
        {
             _firstName = str;
             _timer.reset();
             _timer.start();
        }

        [Bindable(event="lastNameChanged")]
        public function get lastName():String
        {
            return _lastName;
        }
        public function set lastName( str:String ):void
        {
            _lastName = str;
           dispatchEvent( new Event( LAST_NAME_CHANGED ) );
        }

    ]]>
</mx:Script>

<mx:Panel title="User Entry."
      paddingLeft="5" paddingRight="5"
      paddingTop="5" paddingBottom="5">
```

```
            <mx:HBox>
               <mx:Label text="First Name:" />
               <mx:TextInput id="fnInput" />
            </mx:HBox>
            <mx:HBox>
                <mx:Label text="Last Name:" />
                <mx:TextInput id="lnInput" />
                </mx:HBox>
            <mx:Button label="submit" click="clickHandler();" />
            <mx:HRule width="100%" />
            <mx:Label text="You Entered:" fontWeight="bold" />
            <mx:HBox>
                <mx:Label text="First Name:" />
                <mx:Text text="{firstName}" />
            </mx:HBox>
            <mx:HBox>
                <mx:Label text="Last Name:" />
                <mx:Text text="{lastName}" />
            </mx:HBox>
        </mx:Panel>
    </mx:Application>
```

Передача события откладывается до завершения экземпляра `Timer`. Если запустить эту программу, привязка данных для свойства `lastName` будет происходить немедленно, так как нестандартное событие передается в set-методе атрибута. С другой стороны, обновление приемника привязки для свойства `firstName` происходит на две секунды позже, потому что событие `firstNameChanged` передается по таймеру `Timer`.

См. также

Рецепт 14.1.

14.8. Привязка к обобщенному объекту

Задача

Требуется создать привязку свойств, используя в качестве источника экземпляр `Object` верхнего уровня.

Решение

Используйте `mx.utils.ObjectProxy` для инкапсуляции `Object` и передачи событий привязки.

Обсуждение

При создании привязки к обобщенному объекту `Object` обновление обычно напрямую активизируется только по завершении инициализации объекта-приемника. Чтобы свойства объекта-приемника обновлялись с изменением свойств `Object`, используйте класс `ObjectProxy`. При созда-

нии экземпляра ObjectProxy экземпляр Object передается в конструкторе. Пример:

```
var obj:Object = {name:'Tom Waits', album:'Rain Dogs', genre:'Rock'};
var proxy:ObjectProxy = new ObjectProxy( obj );
```

Изменения свойств исходного объекта обрабатываются объектом ObjectProxy, который при обновлении передает событие propertyChange (стандартное событие, передаваемое по умолчанию для привязки). При передаче события по умолчанию значение свойства объекта-источника копируется в заданное свойство объекта-приемника. В следующем примере обобщенный объект передается экземпляру класса ObjectProxy в аргументе конструктора:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectProxy;

            private var obj:Object = {name:'Tom Waits',
                album:'Rain Dogs',
                genre:'Rock'};
            [Bindable]
            private var proxy:ObjectProxy =
                new ObjectProxy( obj );
            private function clickHandler():void
            {
                proxy.name = nameField.text;
                proxy.album = albumField.text;
                proxy.genre = genreField.text;
            }

        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="Name:">
             <mx:TextInput id="nameField" />
        </mx:FormItem>
        <mx:FormItem label="Album:">
            <mx:TextInput id="albumField" />
          </mx:FormItem>
        <mx:FormItem label="Genre:">
           <mx:TextInput id="genreField" />
        </mx:FormItem>
        <mx:FormItem label="Submit Changes">
             <mx:Button label="ok" click="clickHandler();" />
         </mx:FormItem>
    </mx:Form>
    <mx:HRule width="100%" />
```

```
    <mx:Form>
        <mx:FormItem label="Name:">
            <mx:Text text="{proxy.name}" />
        </mx:FormItem>
        <mx:FormItem label="Album:">
            <mx:Text text="{proxy.album}" />
        </mx:FormItem>
        <mx:FormItem label="Genre:">
            <mx:Text text="{proxy.genre}" />
        </mx:FormItem>
    </mx:Form>

</mx:Application>
```

В этом примере передача информации об обновлении приводит к изменению свойств `ObjectProxy`, а изменения отражаются в привязанных к нему компонентах. Возможности разработчика не ограничиваются обновлением заранее определенных свойств объекта-посредника; выражения привязки могут определяться и для свойств, назначаемых посреднику в произвольный момент времени.

Вместо использования обобщенных экземпляров `Object` рекомендуется создать пользовательский класс и опубликовать привязываемые свойства. Но если вследствие особенностей архитектуры приложения этот вариант невозможен, следует использовать `ObjectProxy`.

См. также

Рецепт 14.1.

14.9. Привязка к свойствам в динамических классах

Задача

Требуется привязать свойства объекта-приемника к свойствам, не определенным явно в динамических классах.

Решение

Создайте субкласс `mx.utils.Proxy`, реализующий интерфейс `mx.events.IEventDispatcher` и передающий событие `propertyChange` в переопределении `setProperty` из пространства имен `flash_proxy`.

Обсуждение

Класс `Proxy` позволяет обращаться и изменять значения свойств с использованием точечной записи. Чтобы эффективно работать со ссылками на динамические свойства, переопределите методы `getProperty` и `setProperty` из пространства имен `flash_proxy` в своей реализации субкласса. При

определении пользовательского поведения в этих методах вы получаете доступ к свойствам так, как если бы они непосредственно публиковались классом. Тем не менее ссылок на динамические свойства недостаточно для установления привязки, потому что механизм привязки данных базируется на передаче событий.

Так как привязка инициируется событиями, для создания класса-посредника `Proxy`, подходящего для привязки, необходимо также реализовать интерфейс `IEventDispatcher` и его методы. Чтобы ссылки на динамические свойства могли использоваться в привязке, класс объявляется с ключевым словом `dynamic` и определяется с использованием тега метаданных `[Bindable]`, у которого атрибуту `event` задается значение `propertyChange`:

```
[Bindable(event="propertyChange")]
dynamic public class Properties extends Proxy implements IEventDispatcher {}
```

Превосходный пример ситуации, в которой уместно создание пользовательского класса `Proxy`, — обращение к данным, загруженным из внешнего источника с установлением правил обработки в переопределениях методов `setProperty` и `getProperty` (вместо написания парсера, заполняющего свойства пользовательского объекта по загруженным данным).

Предположим, приложение загружает следующий код XML, для которого необходимо обеспечить возможность обращения к свойствам и их модификации:

```
<properties>
    <property id="name"><![CDATA[Tom Waits]]></property>
    <property id="album"><![CDATA[Rain Dogs]]></property>
    <property id="genre"><![CDATA[Rock]]></property>
</properties>
```

Мы можем создать субкласс `mx.utils.Proxy` и использовать E4X в переопределениях `setProperty` и `getProperty`, давая возможность клиенту обращаться к свойствам данных XML и изменять их:

```
override flash_proxy function getProperty( name:* ):*
{
    return xml..property.(@id == String( name ) );
}
override flash_proxy function setProperty( name:*, value:* ):void
{
    var index:Number = xml..property.(@id == String( name ) ).childIndex();
    xml.replace( index, '<property id="' + name + '">' + value +
   '</property>' );
}
```

Привязки данных активизируются событиями по обновлению свойства. Переопределение `setProperty` в этом примере хотя и обновляет значение свойства, не передает оповещение об изменении. Чтобы активизировать привязку для ссылок на динамические свойства, необходимо передать событие `PropertyChangeEvent` из субкласса `Proxy`:

```
override flash_proxy function setProperty(
    name:*, value:* ):void
{
    var oldVal:String = xml..property.(@id ==
        String( name ) );
    var index:Number = xml..property.(@id ==
        String( name ) ).childIndex();
    xml.replace( index, '<property id="' + name + '">' +
        value + '</property>' );
    var evt:Event = PropertyChangeEvent.createUpdateEvent(
        this, name, oldVal, value );
    dispatchEvent( evt );
}
```

Статический метод `createUpdateEvent` **класса** `PropertyChangeEvent` **возвращает экземпляр** `PropertyChangeEvent` **со свойством** `type`**, которому задано значение** `propertyChange`**; это событие используется по умолчанию для привязок и назначается в теге метаданных** `[Bindable]` **для класса.**

В следующем примере представлена полная реализация субкласса `Proxy`**, пригодная для привязки данных:**

```
package com.oreilly.flexcookbook
{
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.utils.Proxy;
    import flash.utils.flash_proxy;

    import mx.events.PropertyChangeEvent;
    [Event(name="complete", type="flash.events.Event")]
    [Bindable(event="propertyChange")]
    dynamic public class Properties extends Proxy
        implements IEventDispatcher
    {
        private var _evtDispatcher:EventDispatcher;
        private var _data:XML;
        private var _loader:URLLoader;

        public static const COMPLETE:String = "complete";

        public function Properties()
        {
            _evtDispatcher = new EventDispatcher();
        }
        // Загрузка внешнего кода xml
        public function loadProperties( fnm:String ):void
        {
            _loader = new URLLoader();
```

```
        _loader.addEventListener( Event.COMPLETE,
            loadHandler );
        _loader.load( new URLRequest( fnm ) );
    }
    // Задание свойства data и передача оповещения 'complete'
    private function loadHandler( evt:Event ):void
    {
        data = XML( _loader.data );
        dispatchEvent( new Event( Properties.COMPLETE ) );
    }
    public function get data():XML
    {
        return _data;
    }
    public function set data( xml:XML ):void
    {
        _data = xml;
    }
    // Использование E4X для получения свойства из xml.
    override flash_proxy function getProperty( name:* ):*
    {
        if( _data == null ) return "";
        else return _data..property.(@id == String( name ) );
    }
    // Использование E4X для изменения свойства из xml.
    // Передача события 'propertyChange'
    override flash_proxy function setProperty( name:*, value:* ):void
    {
        var oldVal:String = _data..property.(@id == String( name ) );
        var index:Number =
            _data..property.(@id == String( name ) ).childIndex();
        _data.replace( index, '<property id="' + name + '">'
            + value + '</property>' );
        var evt:Event = PropertyChangeEvent.createUpdateEvent(
            this, name, oldVal, value );
        dispatchEvent( evt );
    }
    // Реализация IEventDispatcher.
    public function addEventListener( type:String,
        listener:Function,
        useCapture:Boolean = false,
        priority:int = 0,
        useWeakReference:Boolean = false):void
    {
        _evtDispatcher.addEventListener( type, listener,
             useCapture, priority, useWeakReference );
    }
    // Реализация IEventDispatcher.
    public function removeEventListener( type:String,
             listener:Function,
             useCapture:Boolean = false ):void
```

```
        {
           _evtDispatcher.removeEventListener(
                type, listener, useCapture );
        }
        // Реализация IEventDispatcher.
        public function dispatchEvent( evt:Event ):Boolean
        {
            return _evtDispatcher.dispatchEvent( evt );
        }
        // Реализация IEventDispatcher.
        public function hasEventListener( type:String ):Boolean
        {
            return _evtDispatcher.hasEventListener( type );
        }
        // Реализация IEventDispatcher.
        public function willTrigger( type:String ):Boolean
        {
            return _evtDispatcher.willTrigger( type );
        }
    }
}
```

Обращение и модификация элементов в загруженном коде XML осуществляется через посредника `Properties` в синтаксисе точечной записи:

```
var myProxy:Properties = new Properties();
myProxy.load('properties.xml' );
..
var name:String = myProxy.name;
myProxy.album = "Blue Valentine";
```

Хотя со ссылками на динамические свойства можно работать в точечной записи, этот синтаксис не может использоваться в фигурных скобках или тегах `<mx:Binding>` для создания выражений привязки данных в MXML. Если вы попытаетесь использовать точечную запись, то в процессе компиляции будет выдано предупреждение.

Так как данные XML загружаются на стадии выполнения, вполне логично, что привязка создается после того, как они будут загружены. Для этой цели используется класс `mx.utils.BindingUtils`, который инициирует обновление и обеспечивает привязку данных к посреднику.

В следующем примере экземпляр класса-посредника `Properties` создает привязку данных к свойствам компонента:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="initHandler();">

    <mx:Script>
        <![CDATA[
            import mx.binding.utils.BindingUtils;
```

```
            import com.oreilly.flexcookbook.Properties;
            private var _properties:Properties;
            // Создание посредника и загрузка xml.
            private function initHandler():void
            {
                _properties = new Properties();
                _properties.addEventListener( Event.COMPLETE,
                    propertiesHandler );
                _properties.loadProperties("data/properties.xml" );
            }
            // Данные xml загружены.
            // Установить привязку данных.
            private function propertiesHandler(
                evt:Event ):void
            {
                BindingUtils.bindProperty( nameOutput,
                    "text", _properties, "name" );
                BindingUtils.bindProperty( albumOutput,
                    "text", _properties, "album" );
                BindingUtils.bindProperty( genreOutput,
                    "text", _properties, "genre" );
            }
            // Изменение свойств опосредованных данных
           private function changeHandler():void
            {
               _properties.name = nameField.text;
               _properties.album = albumField.text;
               _properties.genre = genreField.text;
            }
        ]]>

    </mx:Script>
    <mx:Label text="Data Loaded." />
    <mx:Form>
        <mx:FormItem label="Name:">
            <mx:Text id="nameOutput" />
         </mx:FormItem>
        <mx:FormItem label="Album:">
            <mx:Text id="albumOutput" />
        </mx:FormItem>
        <mx:FormItem label="Genre:">
        <mx:Text id="genreOutput" />
        </mx:FormItem>
    </mx:Form>
    <mx:HRule width="100%" />
    <mx:Form>
        <mx:FormItem label="Name:">
             <mx:TextInput id="nameField" />
           </mx:FormItem>
        <mx:FormItem label="Album:">
           <mx:TextInput id="albumField" />
```

```
            </mx:FormItem>
            <mx:FormItem label="Genre:">
                <mx:TextInput id="genreField" />
            </mx:FormItem>

            <mx:FormItem label="Submit Changes">
                 <mx:Button label="ok" click="changeHandler();" />
            </mx:FormItem>
        </mx:Form>
    </mx:Application>
```

В обработчике события `propertiesHandler` привязка данных реализуется вызовом метода `BindingUtils.bindProperty` после успешной загрузки данных XML экземпляром `Properties`. Свойство `text` каждого компонента `Text` с первого экземпляра `Form` привязывается к соответствующему элементу XML на основании атрибута `id`. Вследствие использования E4X в переопределении метода `getProperty` класса `Properties` выполняется обновление привязки и копирование значений.

Значения свойств изменяются с использованием точечной записи в обработчике события `changeHandler`, что, в свою очередь, приводит к вызову метода `setProperty` экземпляра `Properties` и передаче оповещения об активизации привязки через объект `PropertyChangeEvent`.

См. также

Рецепты 14.4–14.6.

15

Проверка данных, форматирование и регулярные выражения

На первый взгляд объединение проверки данных с форматированием и регулярными выражениями выглядит довольно странно, однако все эти операции используются для решения сходных задач в повседневной работе программиста: разборе строк по заранее определенному шаблону, их преобразованию к некоторому формату при обнаружении некоторых закономерностей (или их отсутствии), возвращению пользователям сообщений об ошибках при отсутствии некоторых свойств: телефонные номера, имена с правильным регистром символов, денежные суммы, почтовые индексы, коды ISBN и другие данные, получаемые от третьих сторон, не всегда имеют правильный формат, необходимый для наших приложений. Flex Framework предоставляет два мощных инструмента для интеграции подобных задач разбора и форматирования с элементами пользовательского интерфейса Framework – классы `Validator` и `Formatter`. В основу обоих классов заложены *регулярные выражения*. Поддержка регулярных выражений лишь недавно появилась в языке ActionScript и на платформе **Flex Player, однако это проверенный временем и мощ**ный инструмент, который используется почти во всех языках программирования. Программисты в равной степени любят и ненавидят его за невероятную мощь и сложный синтаксис.

Объект `Validator` проверяет поле произвольного компонента **Flex и убеж**дается в том, что содержащееся в нем значение соответствует заданным условиям: поле не осталось пустым, содержащиеся в нем данные имеют определенный формат, определенную длину и т. д. Интеграция `Validator` с компонентами Flex означает, что для вывода ошибок при проверке достаточно задать в источнике `Validator` компонент, в котором будет проверяться пользовательский ввод, и указать проверяемое свойство. `Validator` передаст событие компоненту, а компонент выведет пользо-

вательское сообщение, заданное в `Validator`. Flex Framework содержит много заранее определенных валидаторов для проверки номеров кредитных карт, телефонных номеров, адресов электронной почты и номеров социального страхования, но в этой главе основное внимание уделяется построению пользовательских валидаторов, их интеграции и событиям проверки.

Класс `Formatter` выполняет простую, но чрезвычайно важную функцию: он берет произвольное значение и преобразует его к заданному формату. Например, это может означать преобразование серии цифр в стандартный формат телефонного номера (например, (555)555-5555), правильное форматирование даты или почтовых индексов для разных стран. Сам класс `Formatter` определяет простой, но чрезвычайно важный для нас метод: `format`. Именно этот метод получает входные данные и возвращает нужную строку.

Оба класса выполняют операции со строками, которые могут быть выполнены с использованием регулярных выражений, хотя в базовых классах регулярные выражения обычно не используются. Бесспорно, регулярные выражения – один из самых мощных, элегантных и сложных инструментов, присутствующих в большинстве современных языков программирования. Они позволяют программисту создать сложный набор правил, выполняемых для произвольной строки. Практически во всех основных языках программирования имеется встроенная поддержка регулярных выражений. Несмотря на некоторые различия в функциональности, эти механизмы поддерживают одинаковые синтаксические конструкции, поэтому регулярные выражения станут полезным дополнением к вашему арсеналу. В ActionScript регулярные выражения представлены классом `RegExp`, который определяет два основных метода: метод `test` проверяет, совпало ли регулярное выражение в какой-либо позиции строки, а метод `exec` возвращает массив всех совпадений и позицию первого совпадения. Для проверки совпадений регулярного выражения также можно воспользоваться методами `match`, `search` и `replace` класса `String`. На мой взгляд, методы класса `String` более удобны, потому что они позволяют выполнять операции с символами строки. Регулярные выражения – весьма обширная тема, и об их правильном использовании написаны целые книги. В этой главе я постарался представить некоторые специфические аспекты их использования и решения некоторых стандартных задач, не пытаясь описать более общий набор типовых сценариев использования.

15.1. Использование объектов Validator и Formatter с компонентами TextInput и TextArea

Задача

Требуется организовать проверку и форматирование нескольких компонентов `TextInput` и `TextArea`.

Решение

Для проверки каждого типа входных данных (даты, телефоны, денежные суммы) используйте объект Validator, а для задания текста TextInput — объект Formatter.

Обсуждение

Чтобы совместно использовать объекты Validator и Formatter в компонентах, просто создайте отдельный валидатор для каждого типа проверяемых данных. Когда в компоненте TextInput происходит событие focusOut, вызовите метод validate для соответствующего валидатора. Чтобы связать валидатор с компонентом TextInput, укажите в свойстве source валидатора экземпляр TextInput, а свойству property задайте значение text:

```
<mx:NumberValidator id="numValidator" source="{inputCurrency}"
    property="text"/>
```

Объект Formatter вызывается после проверки данных. Базовый класс Formatter получает форматную строку со знаками #, которые заменяются цифрами или символами форматируемой строки. Например, для телефонного номера форматная строка будет выглядеть так:

```
(###) ###-####
```

Тег форматирования для телефонного номера выглядит так:

```
<mx:PhoneFormatter id="phoneFormatter" formatString="(###) ###-####"
    validPatternChars="#-() "/>
```

Чтобы использовать этот объект Formatter, вызовите метод format и передайте ему свойство text нужного компонента TextInput:

```
inputPhone.text = phoneFormatter.format(inputPhone.text);
```

Далее приводится полный код примера. Обратите внимание: если данные не прошли проверку, приложение стирает текст, введенный пользователем, и выводит сообщение об ошибке. Вероятно, в реальном приложении это не лучшее решение.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600" height="400">
   <mx:Script>
      <![CDATA[

            import mx.events.ValidationResultEvent;
            private var vResult:ValidationResultEvent;
// Обработчик события для проверки
// и форматирования входных данных.
            private function dateFormat():void
            {
               vResult = dateVal.validate();
                if (vResult.type==ValidationResultEvent.VALID) {
                    inputDate.text = dateFormatter.format(inputDate.text);
```

```
                } else {
                    inputDate.text= "";
                }
            }

            private function phoneFormat():void {
               vResult = phoneValidator.validate();
                if (vResult.type==ValidationResultEvent.VALID) {
                    inputPhone.text =
                    phoneFormatter.format(inputPhone.text);
                } else {
                    inputPhone.text= "";
                }
            }

            private function currencyFormat():void {
               vResult = numValidator.validate();
                if (vResult.type==ValidationResultEvent.VALID) {
                    inputCurrency.text =
                    currencyFormatter.format(inputCurrency.text);
                } else {
                    inputCurrency.text= "";
                }
            }
        ]]>
    </mx:Script>
    <mx:DateFormatter id="dateFormatter" formatString="day:
    DD, month: MM, year: YYYY"/
>
     <mx:DateValidator id="dateVal" source="{inputDate}"
     property="text" inputFormat="mm/dd/yyyy"/>
     <mx:PhoneNumberValidator id="phoneValidator"
     property="text" source="{inputPhone}"/>
     <mx:PhoneFormatter id="phoneFormatter"
     formatString="(###) ###-####"
     validPatternChars="#-() "/>
     <mx:CurrencyFormatter id="currencyFormatter"
     currencySymbol="-" thousandsSeparator
         From="." decimalSeparatorFrom=","/>
     <mx:NumberValidator id="numValidator"
     source="{inputCurrency}" property="text"/>
     <mx:Form>
       <mx:FormItem label="Currency Input">
          <mx:TextInput id="inputCurrency"
          focusOut="currencyFormat()" width="300"/>
       </mx:FormItem>
       <mx:FormItem label="Phone Number Input">
          <mx:TextInput id="inputPhone"
              focusOut="phoneFormat()" width="300"/>
       </mx:FormItem>
       <mx:FormItem label="Date Input">
```

```
            <mx:TextInput id="inputDate"
                focusOut="dateFormat();" width="300"/>
        </mx:FormItem>
    </mx:Form>
</mx:VBox>
```

15.2. Создание пользовательского форматера

Задача

Требуется создать пользовательский форматер, который получает любую строку, соответствующую его требованиям, и возвращает ее в отформатированном виде.

Решение

Расширьте класс `Formatter` и переопределите метод `format`.

Обсуждение

В методе `format` создайте экземпляр `SwitchSymbolFormatter` и передайте методу `formatValue` этого экземпляра форматную строку с символами `#`, представляющими символы исходной строки. Например, если передать методу `formatValue` форматную строку `###-###` и исходную строку `123456`, он вернет `123-456`. Верните это значение из метода `format` пользовательского форматера.

Класс `Formatter` использует форматную строку со знаками `#`, которые заменяются всеми символами строки, переданные методу `format`. Замена этих символов сводится к простому перебору, посимвольному построению отформатированной строки и последующей замене.

```
package oreilly.cookbook
{
    import mx.formatters.Formatter;
    import mx.formatters.SwitchSymbolFormatter;

    public class ISBNFormatter extends Formatter
    {

    public var formatString : String = "####-##-####";

        public function ISBNFormatter()
        {
            super();
        }

        override public function format(value:Object):String {
            // Необходимо проверить длину строки.
            // Коды ISBN состоят из 10 или 13 символов.
                if( ! (value.toString().length == 10 ||
```

```
                value.toString().length == 13) ) {
                 error="Invalid String Length";
               return ""
            }

            // Подсчет знаков # в форматной строке
            var numCharCnt:int = 0;
            for( var i:int = 0; i<formatString.length; i++ ) {
                if( formatString.charAt(i) == "#" ) {
                   numCharCnt++;
                }
            }

            // Если форматная строка содержит неверное
            // количество метасимволов, вернуть ошибку.
            if( ! (numCharCnt == 10 || numCharCnt == 13) ) {
                error="Invalid Format String";
                return ""
            }
            // Если форматная и исходная строки
            // прошли проверку,отформатировать число.
            var dataFormatter:SwitchSymbolFormatter = new
                SwitchSymbolFormatter();
            return dataFormatter.formatValue(formatString, value );
        }
    }
}
```

15.3. Создание универсального валидатора с использованием регулярных выражений

Задача

Требуется проверить все форматы южноамериканских почтовых индексов для тех стран, в которых они используются.

Решение

Создайте серию регулярных выражений для всех стран с проверяемыми почтовыми индексами. Создайте пользовательский класс `Validator`, которому передается код страны; в зависимости от этого кода используйте соответствующий объект `RegExp` в методе `doValidation`. Если регулярное выражение совпадает со строкой, переданной методу `doValidation`, или указанная страна не использует проверку почтового индекса, верните `true`; в противном случае верните `false`.

Обсуждение

Использование регулярных выражений в пользовательских валидаторах позволяет создавать более универсальные и гибкие методы проверки. Без

регулярных выражений объект Validator ограничивается одной строкой, по которой он может проверить введенные данные. Использование нескольких регулярных выражений в валидаторе позволяет создать класс, способный проверять данные в нескольких форматах.

В следующем примере создается хеш-таблица с почтовыми индексами нескольких стран. Когда пользователь выбирает страну и передает ее валидатору, из хеша извлекается правильное регулярное выражение:

```
private var countryHash:Object = {"Argentina":/[a-zA-Z]\d{4}[a-zA-Z]
{3}/, "Brazil":/\d{5}-\d{3}/, "Mexico":/\d{5}/, "Bolivia":/\d{4}/,
"Chile":/\d{7}/, "Paraguay":/\d{4}/,"Uruguay":/\d{5}/};
```

Свойство country валидатора используется в методе doValidation класса Validator, переопределяемого в этом примере:

```
// Проверить, что название страны задано
  if(countryHash[_country] != null) {
     // Прочитать данные из хеша
     // и получить правильный объект RegExp
     var regEx:RegExp = countryHash[_country];
     if(regEx.test(value as String)) {
        return results;
     } else { // Если почтовый индекс не прошел
              // проверку, вернуть ошибку.
        var err:ValidationResult = new
        ValidationResult(true, "", "",
        "Please Enter A Correct Postal Code");
        results.push(err);
     }
  } else {
     return results;
  }
```

Полный код валидатора:

```
package oreilly.cookbook
{
   import mx.validators.ValidationResult;
   import mx.validators.Validator;

   public class SouthAmericanValidator extends Validator
   {
      // Сохранение всех стран и их почтовых индексов
      // в хеш-таблице.
      private var countryHash:Object =
          {"Argentina":/[a-zA-Z]\d{4}[a-zA-Z]{3}/,
          "Brazil":/\d{5}-\d{3}/, "Mexico":/\d{5}/,
          "Bolivia":/\d{4}/, "Chile":/\d{7}/,
          "Paraguay":/\d{4}/, "Uruguay":/\d{5}/};

      private var results:Array;
      private var _country:String;
```

```
        public function SouthAmericanValidator() {
           super();
        }

        public function set country(str:String):void {
           _country = str;
           trace(_country);
        }

        // Определение метода doValidation().
        override protected function
        doValidation(value:Object):Array {
              // Очистка массива результатов
              results = [];
        // Если страна не задана, вернуть ошибку.
              if(_country == "") {
                 var err:ValidationResult = new
                 ValidationResult(true, "", "",
                 "Please Select a Country");
                 results.push(err);
                 return results;
              } else {
                 // Если страна не имеет почтового индекса,
                 // вернуть управление без выдачи ошибки.
                 if(countryHash[_country] != null) {
                    // Чтение из хеш-таблицы и получение
                    // правильного объекта RegExp.
                    var regEx:RegExp = countryHash[_country];
                    if(regEx.test(value as String)) {
                       return results;
                    } else {
                       // Если почтовый индекс не проходит
                       // проверку, вернуть ошибку.
                       var err:ValidationResult = new
                       ValidationResult(true, "", "",
                       "Please Enter  A Correct Postal Code");
                       results.push(err);
                    }
                 } else {
                    return results;
                 }
              }
              return results;
          }
      }
   }
```

При реализации пользовательского валидатора проследите за тем, чтобы страна была задана перед вызовом метода `doValidation`**. В следующем примере компонент** `ComboBox` **используется для задания свойства** `country` **объекта** `SouthAmericanValidator`**:**

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
xmlns:cookb
ook="oreilly.cookbook.*">
   <cookbook:SouthAmericanValidator property="text" source="{zip}"
required="true" id=
"validator" invalid="showInvalid(event)"/>
   <mx:Script>
      <![CDATA[
         import mx.events.ValidationResultEvent;

         private function
            showInvalid(event:ValidationResultEvent):void {
            trace( " event " + event.message );
            zip.errorString = event.message;
         }

      ]]>
   </mx:Script>
   <mx:Form>
      <mx:FormItem label="Select country ">
            <mx:ComboBox dataProvider="{['Argentina',
            'Brazil', 'Mexico', 'Bolivia', 'Ecuador',
            'Colombia', 'Chile','Paraguay','Uruguay']}"
            id="cb" change="validator.country =
                cb.selectedItem as String"/>
      </mx:FormItem>
      <mx:FormItem label="Enter zip ">
         <mx:TextInput id="zip"/>
      </mx:FormItem>
   </mx:Form>
</mx:HBox>
```

15.4. Создание валидатора для проверки кодов UPC

Задача

Требуется организовать проверку кодов UPC на форме.

Решение

Создайте пользовательский валидатор, который проверяет наличие и правильность контрольной суммы UPC и возвращает ошибку в случае неудачи.

Обсуждение

Код UPC (Universal Product Code) состоит из 12 цифр и используется кассовыми аппаратами и многими торговыми фирмами. Коды UPC проверяются при помощи загадочных на первый взгляд контрольных сумм,

вычисляемых суммированием утроенных цифр кода в позициях 1, 3, 5... с последующим прибавлением цифр в позициях 2, 4, 6... Программу понять гораздо проще, чем объяснения:

```
var sum:Number = 0;
for ( var i:Number=0; i < UPC.length; i += 2){
   sum += Number(UPC.charAt(i)) * 3;
}
for ( i = 1; i < UPC.length-1; i += 2) {
   sum += Number(UPC.charAt(i));
}

var checkSum:Number = ( 10 - (sum % 10) ) % 10;
// Если контрольная сумма не совпадает, выдать ошибку проверки.
if ( Number(UPC.charAt(11)) != checkSum ) {
   results.push(new ValidationResult(true, null,
   "invalidUPC", "Invalid UPC Number."));
    return results;
}
```

Проверка гарантирует правильность кода UPC. Остальная реализация весьма прямолинейна:

```
package com.passalong.utils
{
   import mx.validators.Validator;
   import mx.validators.ValidationResult;
   import mx.controls.Alert;

   public class UPCValidator extends Validator
   {
      private var results:Array;
      public function UPCValidator()
      {
         super();
      }

      override protected function doValidation(value:Object):Array
      {
         // Преобразование в String для просмотра всех цифр.
            var UPC:String = String(value);
            // Отсечение точки от начала строки (добавляется
            // для принудительного распознавания начальных нулей).
            UPC = UPC.substring(1);
         var UPCnum:Number = Number(UPC);
            // Очистка массива результатов.
            results = [];
            // Вызов реализации doValidation() базового класса
            results = super.doValidation(value);
            // Вернуть управление при наличии ошибок.
            if (results.length > 0)
                return results;
```

```
            // Если входные данные не являются числом
            // или не содержат значения, выдать ошибку проверки.
            if (isNaN(UPCnum) || !value )
            {
                results.push(new ValidationResult(true,
                    null, "NaN","UPC required."));
                return results;
            }

        if ( UPC.length != 12 )
        {
            results.push(new ValidationResult(true, null,
               "invalidUPCLength",
               "Please enter a full 12-digit UPC."));
            return results;
        } else {
            var sum:Number = 0;

            for ( var i:Number=0; i < UPC.length; i += 2)
               sum += Number(UPC.charAt(i)) * 3;

            for ( i = 1; i < UPC.length-1; i += 2) {
               sum += Number(UPC.charAt(i));
            }
            var checkSum:Number = ( 10 - (sum % 10) ) % 10;

               // Если контрольная сумма не совпадает,
               // выдать ошибку проверки.
               if ( Number(UPC.charAt(11)) != checkSum )
               {
                  results.push(new ValidationResult(true,
                       null, "invalidUPC", "Invalid UPC Number."));
                   return results;
               }
               return results;
          }
      }
   }
}
```

Спасибо Майку Орту (Mike Orth) за идею и код этого рецепта.

15.5. Проверка компонентов ComboBox и групп переключателей

Задача

Требуется проверить компонент `ComboBox` или группу переключателей. В компоненте `ComboBox` должна быть выбрана строка, отличная от исходной подсказки, а в группе переключателей должен присутствовать установленный переключатель.

Решение

Используйте NumberValidator для проверки переключателей и пользовательскую реализацию Validator для проверки ComboBox.

Обсуждение

Чтобы вернуть ValidationResultEvent для группы переключателей, используйте NumberValidator для проверки того, что свойство selectedIndex объекта RadioButtonGroup отлично от −1 (это значение указывает на отсутствие установленного переключателя). Чтобы проверить ComboBox, создайте пользовательский валидатор и убедитесь в том, что свойство selectedItem экземпляра ComboBox отлично от null и не совпадает со строкой подсказки или другим недопустимым значением (если оно задано).

Код пользовательского валидатора для ComboBox вполне тривиален; он представлен ниже с комментариями:

```
package oreilly.cookbook
{
   import mx.validators.ValidationResult;
   import mx.validators.Validator;

   public class ComboValidator extends Validator
   {
      // Сообщение об ошибке; возвращается в том случае,
      // если в ComboBox не выбрана строка.
      public var error:String;
      // Если разработчик назначает подсказку вручную,
      // а потом добавляет элемент в массив ComboBox
      // (я много раз видел, как это делалось по разным
      // причинам), выбранную в CB строку необходимо
      // проверить дополнительно.
      public var prompt:String;
      public function ComboValidator() {
         super();
      }
      // Метод проверяет, что в ComboBox не выбрано значение null
      // или подсказка, заданная разработчиком;
      // в этих случаях возвращается ошибка.
      override protected function
         doValidation(value:Object):Array {
         var results:Array = [];
         if(value as String == prompt || value == null) {
            var res:ValidationResult = new
            ValidationResult(true, "", "", error);
            results.push(res);
         }
         return results;
      }
   }
}
```

Одна из стратегий выполнения нескольких проверок основана на использовании массива: в массив добавляются все валидаторы компонента, а затем открытый статический метод `Validator.validateAll` используется для проверки по всем валидаторам. Это особенно полезно в тех случаях, когда несколько полей необходимо проверить одновременно. Если какие-либо валидаторы возвращают ошибки, то сообщения об ошибках объединяются и отображаются в компоненте `Alert`:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="600" height="400"
xmlns:cookb
ook="oreilly.cookbook.*" creationComplete="init()">
   <mx:Script>
      <![CDATA[

        import mx.events.ValidationResultEvent;
        import mx.validators.Validator;
        import mx.controls.Alert;

        [Bindable]
         private var validatorArr:Array;
       // Создание массива всех валидаторов, проверяемых
       // одним вызовом метода.
         private function init():void {
              validatorArr = new Array();
          // Занесение всех валидаторов в один массив.
              validatorArr.push(rbgValidator);
              validatorArr.push(toggleValidator);
              validatorArr.push(comboValidator);
         }
       // Проверка по всем элементам массива валидаторов
       // и отображение Alert при наличии ошибок
            private function validateForm():void {
            // Метод validateAll проверяет данные
            // по всем валидаторам в массиве, переданном
            // при вызове validateAll.
                var validatorErrorArray:Array =
                    Validator.validateAll(validatorArr);;
                var isValidForm:Boolean =
                    validatorErrorArray.length == 0;
                if (!isValidForm) {
                    var err:ValidationResultEvent;
                    var errorMessageArray:Array = [];
                    for each (err in validatorErrorArray) {
                        errorMessageArray.push(err.message);
                    }
                    Alert.show(errorMessageArray.join("\n"),
                        "Invalid form...", Alert.OK);
                }
            }
      ]]>
   </mx:Script>
```

```
<mx:StringValidator id="rbgValidator" source="{rbg}"
   property="selectedValue"/>
<mx:NumberValidator id="toggleValidator" source="{toggleButton}"
   property="selectedIndex" allowNegative="false"
   negativeError="Please select a Radio Button"/>
<cookbook:ComboValidator id="comboValidator"
   error="Please Select A State" prompt="
   {stateCB.prompt}" source="{stateCB}"
   property="selectedItem"/>
<mx:Form id="form">
   <mx:FormItem>
      <mx:ComboBox id="stateCB" dataProvider="{someDataProvider}"
         prompt="Select A State"/>
   </mx:FormItem>
   <mx:FormItem>
      <mx:RadioButtonGroup id="rbg"/>
      <mx:RadioButton group="{rbg}" label="first" id="first"/>
      <mx:RadioButton group="{rbg}" id="second" label="second"/>
      <mx:RadioButton id="third" label="third" group="{rbg}"/>
   </mx:FormItem>
   <mx:FormItem>
      <mx:ToggleButtonBar id="toggleButton">
         <mx:dataProvider>
            <mx:Array>
               <mx:String> First Option </mx:String>
               <mx:String> Second Option </mx:String>
               <mx:String> Third Option </mx:String>
               <mx:String> Fourth Option </mx:String>
               <mx:String> Fifth Option </mx:String>
            </mx:Array>
         </mx:dataProvider>
      </mx:ToggleButtonBar>
   </mx:FormItem>
</mx:Form>
<mx:Button label="validate" click="validateForm()"/>
</mx:VBox>
```

15.6. Отображение ошибок проверки с использованием подсказок

Задача

Требуется создать и отобразить результаты нескольких ошибок проверки независимо от того, принадлежит ли фокус `TextInput` или другому компоненту.

Решение

Используйте объект `ToolTipManager` для создания нового класса `ToolTip` и его позиционирования над компонентом. Создайте объект `Style` и свя-

жите его с ToolTip, чтобы подсказка выводилась на красном фоне и была оформлена нужным шрифтом.

Обсуждение

Подсказка, отображаемая в том случае, когда валидатор возвращает ошибку, представляет собой обычный компонент ToolTip. Вы можете использовать стили для представления всей необходимой визуальной информации в ToolTip: backgroundColor, fontColor, fontType и т. д. Для применения стиля к новым подсказкам, создаваемым для каждой ошибки проверки, используется метод setStyle компонента ToolTip; например:

```
errorTip.setStyle("styleName", "errorToolTip");
```

Чтобы вывести сразу несколько подсказок, расположите их на основании позиции соответствующего компонента на сцене. Пример:

```
var pt:Point = this.stage.getBounds(err.currentTarget.source);
var yPos:Number = pt.y * -1;
var xPos:Number = pt.x * -1;
// Создание подсказки с сообщением об ошибке
var errorTip:ToolTip =
ToolTipManager.createToolTip(err.message, x
Pos + err.currentTarget.source.width, yPos) as ToolTip;
```

Когда проверка формы завершается успешно, все подсказки удаляются методом destroyToolTip класса ToolTipManager с перебором всех добавленных объектов ToolTip. Код выглядит примерно так:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="500" height="400"
xmlns:cookb
ook="oreilly.cookbook.*" creationComplete="init();">

   <mx:Style>
      /* Класс CSS, который используется для оформления
         подсказки как сообщения об ошибке. */
      .errorToolTip {
          color: #FFFFFF;
          fontSize: 9;
          fontWeight: "bold";
          shadowColor: #000000;
          borderColor: #CE2929;
          borderStyle: "errorTipRight";
          paddingBottom: 4;
          paddingLeft: 4;
          paddingRight: 4;
          paddingTop: 4;
      }

   </mx:Style>
   <mx:Script>
      <![CDATA[
```

```
import mx.controls.ToolTip;
import mx.managers.ToolTipManager;
import mx.events.ValidationResultEvent;
import mx.validators.Validator;
import mx.controls.Alert;

  [Bindable]
   private var validatorArr:Array;
   private var allErrorTips:Array;

   private function init():void {
       validatorArr = new Array();
       validatorArr.push(comboValidator1);
       validatorArr.push(comboValidator2);
   }
```

А вот как происходит сама проверка:

```
   private function validateForm():void {
      // Если подсказки с ошибками уже существуют,
      // их необходимо удалить.
      if(!allErrorTips) {
         allErrorTips = new Array();
      } else {
         for(var i:int = 0; i<allErrorTips.length; i++) {
            // Удаление подсказки
            ToolTipManager.destroyToolTip(allErrorTips[i]);
         }
         // Очистка массива
         allErrorTips.length = 0;
      }
       var validatorErrorArray:Array =
         Validator.validateAll(validatorArr);
```

Если в массив `validatorErrorArray` **не были добавлены новые элементы, мы знаем, что ошибки в ходе проверки не возникали; в противном случае мы создаем эти подсказки и размещаем их в нужных местах:**

```
       var isValidForm:Boolean = validatorErrorArray.length == 0;
       if (!isValidForm) {
           var err:ValidationResultEvent;
           for each (err in validatorErrorArray) {
                // Позиция подсказки определяется
                // координатами x и y компонента.
                // Нам нужны их фактические значения
                // на случай, если на форме применяется
                // управление раскладкой, поэтому
                // для получения координат
                // используется метод getBounds.
```

Так как в свойстве `target` **объекта** `ErrorEvent` **передается компонент, инициировавший передачу события, это свойство используется для позиционирования подсказки:**

```
                        var pt:Rectangle =
                           this.stage.getBounds(
                           err.currentTarget.source);
                        var yPos:Number = pt.y * -1;
                        var xPos:Number = pt.x * -1;
                        // Создание подсказки
                     var errorTip:ToolTip =
                        ToolTipManager.createToolTip(err.message, xPos +
                        err.currentTarget.source.width, yPos) as ToolTip;
                     // Назначение селектора класса для errorTip.
                     errorTip.setStyle("styleName", "errorToolTip");
                     // Сохранить объект подсказки, чтобы его можно было
                     // удалить позднее при успешном прохождении проверки.
                     allErrorTips.push(errorTip);
                     }
                  }
               }
         ]]>
      </mx:Script>
      <!-- Наши два валидатора -->
      <cookbook:ComboValidator id="comboValidator1"
         error="Please Select A State" prompt=
         "{stateCB1.prompt}" source="{stateCB1}"
         property="selectedItem"/>
      <cookbook:ComboValidator id="comboValidator2"
         error="Please Select A State" prompt=
         "{stateCB2.prompt}" source="{stateCB2}"
         property="selectedItem"/>
      <mx:Form id="form">
         <mx:FormItem>
            <mx:ComboBox id="stateCB1"
               dataProvider="{someDataProvider}"
               prompt="Select A State"/>
         </mx:FormItem>
         <mx:FormItem>
            <mx:ComboBox id="stateCB2"
               dataProvider="{someDataProvider}"
               prompt="Select A State"/>
         </mx:FormItem>
      </mx:Form>
      <mx:Button label="validate" click="validateForm()"/>
   </mx:VBox>
```

15.7. Использование регулярных выражений для поиска адресов электронной почты

Задача

Требуется идентифицировать адреса электронной почты, вводимые в компоненте или присутствующие в тексте.

Решение

Создайте регулярное выражение для адресов электронной почты в формате *имя@хост.com*; включите флаг глобального поиска, указывающий, что выражение может совпадать многократно.

Обсуждение

Регулярное выражение для адресов электронной почты выглядит примерно так:

```
var reg:RegExp = /\w+?@\w+?\.\w{3}/g;
```

Для поиска всех адресов электронной почты в большом блоке текста используйте метод `String match`, который возвращает массив всех совпадений. Метод `match` получает либо искомую подстроку, либо регулярное выражение для выполнения поиска.

15.8. Использование регулярных выражений для поиска номеров кредитных карт

Задача

Требуется построить регулярное выражение для идентификации основных типов кредитных карт: Visa, MasterCard, American Express и Discover.

Решение

Создайте регулярное выражение, содержащее начальные цифры всех основных типов кредитных карт и совпадающее с ожидаемым количеством цифр для каждого типа.

Обсуждение

При построении регулярного выражения можно воспользоваться тем фактом, что номера всех основных типов кредитных карт начинаются с определенных цифр, определяющих действительность карты. Например, номера всех карт MasterCard начинаются с цифры 5, номера всех карт Visa – с цифры 4, номера всех карт American Express – с цифр 30, а номера карт Discover – с цифр 6011. Соответствующее регулярное выражение выглядит так:

```
(5[1-5]\d{14})|(4\d{12}(\d{3})?)|(3[47]\d{13})|(6011\d{14})
```

Для карт MasterCard выражение `(5[1-5]\d{14})` совпадает только с действительными номерами, не содержащими пробелов. Обычно рекомендуется удалять пробелы из номеров кредитных карт, прежде чем отправлять их службе обработки.

Следующий сегмент выражения относится к картам Visa, далее следуют сегменты для карт American Express и Discover.

Конструкции альтернативы (|), разделяющие четыре части регулярного выражения, указывают на то, что совпадение будет возвращено при совпадении хотя бы одного из четырех шаблонов.

15.9. Использование регулярных выражений для проверки ISBN

Задача

Требуется создать регулярное выражение для проверки кодов ISBN (International Standard Book Numbers).

Решение

Создайте регулярное выражение, совпадающее с кодами ISBN, которые могут содержать дефисы, имеют длину 10 или 13 цифр и могут завершаться необязательным суффиксом X.

Обсуждение

В следующем регулярном выражении используются метасимволы ^ и $, которые означают, что строка содержит только код ISBN без каких-либо посторонних символов. Если удалить эти метасимволы, выражение будет совпадать с кодами ISBN в блоке текста:

```
private var isbnReg:RegExp =
    /^(?=.{13}$)\d{1,5}([- ])\d{1,7}\1\d{1,6}\1(\d|X)$/;
private function testISBN():void {
     var s:String ="ISBN 1-56389-016-X";
     trace(s.match(isbnReg));
}
```

Метасимвол ^ означает, что совпадение должно начинаться от начала строки, а метасимвол $ – что оно завершается в конце строки. Между этими двумя символами располагаются группы цифр, которые могут разделяться дефисами (-).

15.10. Создание регулярных выражений с символьными классами

Задача

Требуется создать регулярное выражение с явным перечислением символов, которые могут совпадать в тексте, например для поиска слов, состоящих из одних гласных букв.

Решение

Заключите все символы, которые могут совпадать в некоторой позиции регулярного выражения, в квадратные скобки, например конструкция `[aeiou]` совпадает с любой (одной) гласной буквой английского языка.

Обсуждение

В регулярных выражениях используются различные метасимволы (служебные символы). Некоторые часто встречающиеся метасимволы:

`[ ]` (квадратные скобки)

Символьный класс, т. е. список возможных совпадений для одного символа; например, конструкция `/[aeiou]/` совпадает с одним из указанных символов.

`-` (дефис)

В символьных классах дефисом обозначаются диапазоны символов; например, `/[A-Z0-9]/` совпадает с одним символом, который является буквой от A до Z или цифрой от 0 до 9.

`\` (обратная косая черта)

Символ `\` используется в символьных классах для *экранирования* таких символов, как `]` и `-`; например, конструкция `/[+\-]\d+/` совпадает с одной или несколькими цифрами, перед которыми стоит знак + или -.

В символьных классах другие символы, которые обычно используются в качестве метасимволов, интерпретируются как обычные символы, т. е. литералы; скажем, без префикса `\` конструкция `/[$£]/` совпадает с одним из двух символов $ или £.

За дополнительной информацией обращайтесь к описанию символьных классов в документации Flex.

`|` (вертикальная черта)

Конструкция альтернативы совпадает или с частью выражения слева от символа `|`, или с частью выражения справа от него; например, выражение `/abc|xyz/` совпадает либо со строкой *abc*, либо со строкой *xyz*.

Символьный класс, совпадающий только с нечетными цифрами, выглядит так:

```
var reg:RegExp = /[13579]/;
```

Класс, совпадающий с гласными буквами:

```
var vowels:RegExp = /[aeiou]/;
```

Класс, совпадающий с любым символом, кроме гласных букв:

```
var notVowels:RegExp = /[^aeiou]/;
```

Символ ^ из последнего примера в квадратных скобках означает *инверсию*. Вне квадратных скобок, как говорилось ранее, символ ^ обозначает позицию в начале строки.

15.11. Символьные типы в регулярных выражениях

Задача

Требуется использовать регулярное выражение для поиска символов различных типов (цифры, буквы, пробелы, их инверсия и т. д.).

Решение

Используйте символьные типы.

Обсуждение

Символьные типы состоят из префикса \, который сообщает механизму обработки регулярных выражений, что следующий символ обозначает символьный тип (а не буквальный символ, т. е. литерал), и обозначения категории символов. Многие символьные типы также существуют в инвертированном варианте, например:

- `\d` – одна десятеричная цифра, эквивалент `[0-9]`.
- `\D` – любой символ, кроме десятичных цифр; эквивалент `[^0-9]`.
- `\b` – позиция между символом слова и символом, не являющимся символом слова. Если первый или последний символ строки является символом слова, `\b` также совпадает в начале или конце строки.
- `\B` – позиция между двумя символами слова. Также совпадает с позицией между двумя символами, не являющимися символами слов.
- `\f` – символ перевода страницы.
- `\n` – символ новой строки.
- `\r` – символ возврата.
- `\s` – любой символ из категории пропусков (пробел, табуляция, новая строка, возврат).
- `\S` – любой символ, не являющийся пропуском.
- `\t` – символ табуляции.
- `\unnnn` – символ Юникода с кодом, заданным шестнадцатеричным числом *nnnn*. Например, `\u263a` соответствует изображению улыбающейся рожицы.
- `\v` – символ вертикальной подачи.
- `\w` – символ слова (A–Z, a–z, 0–9 или _). Учтите, что `\w` не совпадает с символами, не входящими в английский алфавит.

- `\W` – любой символ, не являющийся символом слова.
- `\xnn` – символ с заданным ASCII-кодом, заданным шестнадцатеричным числом *nn*.
- `\` – экранирование, т. е. отключение метасимвольной интерпретации специальных символов.
- `.` – один произвольный символ. Точка совпадает с символом новой строки (`\n`) только при установленном флаге `s(dotall)`. За дополнительной информацией обращайтесь к описанию флага `s(dotall)` в документации Flex.

Рассмотрим несколько несложных примеров практического использования этих метасимволов. Следующее регулярное выражение совпадает с символом 1, за которым следуют две буквы:

```
/1\w\w/;
```

Символ 1, за которым следуют два символа, не являющихся буквами:

```
/1\W\W/;
```

Пять цифр, следующих друг за другом:

```
/\d\d\d\d\d/;
```

То же в упрощенной записи:

```
/\d{5}/;
```

Две цифры, разделенные пробелом:

```
/\d\b\d/;
```

Три цифры, разделенные произвольными символами:

```
/\d.\d.\d/;
```

Метасимволы можно группировать в выражения, совпадающие с определенными последовательностями цифр, алфавитных символов или пробелов, а также с их инверсиями. Это позволяет создавать более мощные и компактные регулярные выражения.

15.12. Поиск действительных IP-адресов с использованием подвыражений

Задача

Требуется найти несколько действительных IP-адресов в блоке текста.

Решение

Используйте подвыражения для создания шаблонов, совпадающих с компонентами IP-адресов, состоящими из трех цифр.

Обсуждение

Для определения последовательности, содержащей от 1 до 3 цифр, используется символьный тип \d:

```
\d{1,3}
```

Три группы, содержащие от 1 до 4 цифр, описываются следующим выражением:

```
(\d{1,4}){3}
```

Подвыражение представляет собой отдельный элемент регулярного выражения, который интерпретируется по тем же правилам, что и любой другой шаблон. Таким образом, для идентификации IP-адреса необходимо составить выражение, состоящее из четырех групп, каждая из которых содержит от одной до трех цифр; группы разделяются точками. Эту последовательность символов также можно описать как три группы, состоящие из трех цифр и точки, за которыми следует одна группа из трех цифр. Такое представление значительно упрощает регулярное выражение:

```
(\d{1,3}\.){3}\d{1,3}
```

Это немного приближает нас к поставленной цели, но признать его абсолютно работоспособным нельзя; такое выражение совпадет со строкой 838.381.28.99, которая не является действительным IP-адресом. Необходимо каким-то образом учесть, что максимальное значение каждой группы цифр в IP-адресе ограничивается числом 255. Используя подвыражения, можно прийти к следующему результату:

```
(((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5]))\.){3}((\d{1,2})|(1\d{2})|(2[0-
4]\d)|(25[0-5]))
```

Начнем с рассмотрения первой секции:

```
(((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5]))\.){3}
```

В переводе на естественный язык это означает следующее: одна или две цифры (`(\d{1,2})`), либо 1 с двумя произвольными цифрами (`(1\d{2})`), либо 2 с двумя произвольными цифрами от 0 до 4 (`(2[0-4]\d)`), либо 2 и 5 с любой цифрой от 0 до 5 (`(25[0-5])`). За любой из перечисленных комбинаций следует символ «.».

Завершающая группа представляется следующим образом:

```
((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5]))
```

Эта часть выражения полностью совпадает с предыдущей, за одним исключением: точки в конце. **IP-адрес (например, 192.168.0.1) не содержит завершающей точки.**

Синтаксис подвыражений (интервальных квантификаторов):

- `{n}` – входит не менее *n* раз.
- `{n,m}` – входит не менее *n*, но не более *m* раз.

15.13. Использование регулярных выражений для поиска совпадений переменной длины

Задача

Требуется найти совпадение, состоящее из неизвестного заранее количества повторений шаблона.

Решение

Используйте синтаксис группировки (.) или (+) для поиска совпадений, повторяющихся неизвестное заранее количество раз.

Обсуждение

Как было показано в рецепте 15.12, синтаксическая конструкция { } позволяет задать минимальное и максимальное количество совпадений подвыражения в результате. Допустим, мы хотим найти совпадение для всех символов в интервале от 0 до 4 в двух строках:

```
var firstString:String = "12430";
var secondString:String = "603323";
```

Рассмотрим различные режимы поиска совпадений в этих двух строках. При поиске могут использоваться следующие модификаторы:

- ?? совпадает ноль или один раз.
- *? совпадает ноль или более раз.
- +? совпадает один или более раз.

Не забывайте, что проверка наличия совпадений и возвращение результатов поиска – не одно и то же. Например, если вы хотите узнать, состоит ли строка только из цифр от 0 до 4, воспользуйтесь методом `test` класса `RegExp`, возвращающим логический признак `true` или `false`. Если же вам нужны все символы строки, совпавшие до обнаружения первого несовпадающего символа, воспользуйтесь методом `match` класса `String`. Если же вас интересуют все совпадающие символы строки (т. е. строка после удаления всех несовпадающих символов), воспользуйтесь флагом глобального поиска `/[0-4]+g/` в сочетании с методом `match` класса `String`.

Например, выражение `/[abc]+/` совпадет с подстрокой *abbbca* или *abba* и вернет символы *abc* из строки `abcss`.

Конструкция `\w+@\w+\.\w+` совпадает с последовательностью символов, сходной с адресом электронной почты. Обратите внимание на экранирование точки; оно означает, что точка является обычным литералом, а не частью синтаксиса регулярного выражения. Метасимвол + обозначает любое количество символов; за символами следует знак `@`, за которым идет произвольное количество дополнительных символов.

В следующем фрагменте продемонстрировано использование различных квантификаторов:

```
var atLeastOne:RegExp = /[0-4]+/g;
var zeroOrOne:RegExp = /[0-4]*/g;
var atLeastOne2:RegExp = /[0-4]+?/g;
var zeroOrOne2:RegExp = /[0-4]*?/g;
var firstString:String = "12430";
var secondString:String = "663323";

firstString.match(atLeastOne));// Возвращает "1243"
secondString.match(atLeastOne));
    // Возвращает совпадение максимальной длины "3323"
firstString.match(zeroOrOne));
    // Возвращает "1243", первые совпавшие символы
secondString.match(zeroOrOne));
    // Возвращает ""; так как начальные символы
    // не совпали, поиск прекращается.
firstString.match(atLeastOne2));
    // Возвращает "1,2,4,3", так как используется
    // глобальный поиск совпадения для одного символа
secondString.match(atLeastOne2));// Возвращает "3,3,2,3"
firstString.match(zeroOrOne2));// Возвращает ""
secondString.match(zeroOrOne2));// Возвращает ""
zeroOrOne2.test(firstString));// Возвращает true
zeroOrOne2.test(secondString));// Возвращает false
```

15.14. Привязка совпадения к началу или концу логической строки

Задача

Требуется создать регулярное выражение, совпадения которого начинаются в начале или завершаются в конце строки (или занимают всю строку без каких-либо посторонних символов).

Решение

Используйте в регулярном выражении метасимволы ^ и $.

Обсуждение

При поиске совпадения, занимающего всю строку или привязанного к ее началу или концу, поставьте в начало регулярного выражения метасимвол ^ и/или завершите регулярное выражение метасимволом $.

Например, чтобы найти имя файла произвольной длины с расширением jpg или jpeg, занимающего всю строку без посторонних символов, используйте следующее регулярное выражение:

```
/^.+?\.jpe?g$/i
```

Следующее выражение находит слово, завершающее строку в текстовом поле:

```
/\w+?$/;
```

Слово в начале строки:

```
/^\w+?/;
```

15.15. Обратные ссылки

Задача

Требуется найти совпадение, а затем использовать его для поиска следующего потенциального совпадения, например, при поиске парных тегов HTML.

Решение

Используйте обратные ссылки в регулярном выражении.

Обсуждение

Механизм регулярных выражений Flash Player позволяет сохранять до 99 обратных ссылок (по сути, это просто список всех совпадений). Метасимвол `\1` всегда означает последнее совпадение, а метасимвол `\2` – предыдущее совпадение. В методе `replace` класса `String`, использующем совпадения от другого регулярного выражения, последнее совпадение обозначается конструкцией `$1`.

Для проверки наличия парных тегов HTML (например, что за тегом `<h2>` следует тег `</h2>`) в следующем примере используется обратная ссылка `\1`:

```
private var headerBackreference:RegExp = /<H([1-6])>.*?<\/H\1>/g;

private function init():void {
   var s:String = "<BODY> <H2>Valid Chocolate</H2> <H2>Valid
      Vanilla</H2> <H2>This is not valid HTML</H3></BODY>";
   var a:Array = s.match(headerBackreference);
   if(a != null) {
      for(var i:int = 0; i<a.length; i++) {
         trace(a[i]);
      }
   }
}
```

Обратные ссылки также могут использоваться для замены всех действительных URL-адресов тегами `<a>` (т. е. преобразования их в гиперссылки). В этом варианте использования обратные ссылки выглядят несколько иначе. Сначала просмотрите код, а затем прочитайте дальнейшие пояснения:

```
private var domainName:RegExp =
   /(ftp|http|https|file):\/\/[\S]+(\b|$)/gim;
private function matchDomain():void
{
   var s:String = "Hello my domain is http://www.bar.com,
      but I also like http://foo.net as well as www.baz.org";
      var replacedString = (s.replace(domainName,
         '<a href="$&">$&</a>').replace
         (/([^\/])(www[\S]+(\b|$))/gim,
         '$1<a href="http://$2">$2</a>'));
}
```

Сначала в исходной строке ищется совпадение для действительного URL-адреса:

```
/(ftp|http|https|file):\/\/[\S]+(\b|$)/gim;
```

Затем действительные URL-адреса заключаются в теги `<a>`:

```
s.replace(domainName, '<a href="$&">$&</a>')
```

На этой стадии мы получаем строку следующего вида:

```
Hello my domain is <a href="http://www.bar.com">http://www.bar.com</a>,
but I also like <a href="http://foo.net">http://foo.net</a>
as well as www.baz.org
```

Это не совсем то, что требовалось. Так как исходное регулярное выражение ищет совпадения, начинающиеся с префиксов `ftp`, `http`, `https` или `file`, для `www.baz.org` совпадение не обнаружено. Второй вызов `replace` ищет в строке вхождения `www` без префикса `/`, свидетельствующего об уже найденном совпадении:

```
replace(/([^\/])(www[\S]+(\b|$))/gim, '$1<a href="http://$2">$2</a>'))
```

`$1` и `$2` обозначают соответственно первое и второе совпадения; во втором совпадении хранится нужный URL-адрес.

15.16. Опережение и ретроспектива

Задача

Требуется найти совпадение для шаблона, который не начинается или заканчивается определенными символами.

Решение

Используйте конструкцию отрицательной опережающей `(?!)` или отрицательной ретроспективной `(?<!)` проверки для обозначения символов, которые не могут предшествовать совпадению или следовать за ним. Положительная опережающая `(?=)` и положительная ретроспективная `(?<=)` проверки используются для обозначения символов, которые должны находиться перед совпадением или после него.

Обсуждение

Положительная ретроспективная проверка определяет начальные символы, которые используются для определения совпадения, но *не включаются* в само совпадение. Например, чтобы найти все числа, следующие за знаком $ (но так, чтобы совпадения не включали сам знак $), в строке

```
400 boxes at $100 per unit and 300 boxes at $50 per unit.
```

используется регулярное выражение с положительной ретроспективной проверкой:

```
/(?<=\$)\d+/
```

С другой стороны, если вас интересуют все символы, перед которыми нет знака $, следует использовать отрицательную ретроспективную проверку:

```
/\b(?<!\$)\d+\b/
```

Обратите внимание: при отрицательной ретроспективной проверке знак = из положительной ретроспективной проверки заменяется знаком !; это означает, что для совпадения знака $ быть не должно. Чтобы выбрать из строки только цены, без количества единиц товара, можно воспользоваться положительной опережающей проверкой:

```
private var lookBehindPrice:RegExp = /(?<=[\$|€])[0-9.]+/g;
private function matchPrice():void {
   var s:String = "dfsf24ds: €23.45 ds2e4d: $5.31 CFMX1:
      $899.00 d3923: €69";
   trace(s.match(this.lookBehindPrice));
}
```

В следующем примере положительная опережающая проверка используется для поиска объявлений переменных в строке:

```
private var lookBehindVariables:RegExp = /(?<=var )[0-9_a-zA-Z]+/g;
private function matchVars():void {
   var s:String = " private var lookAheadVariables:RegExp =
      /blah/  private var str:String = 'foo'";
        trace(s.match(lookBehindVariables));
```

А если, например, потребуется найти все строки с подстрокой `pic`, за которой не следует суффикс `.jpg`, используйте отрицательную опережающую проверку:

```
var reg:RegExp = /pic(?!\.jpg)/;
```

16

Работа со службами и взаимодействие с сервером

Одним из важнейших аспектов приложений Flex является взаимодействие с сервером и базами данных. Рецепты этой главы посвящены в основном настройке приложений Flex для работы с серверами и обработке данных, полученных приложением от сервера, с использованием трех основных механизмов взаимодействия между сервером и приложением.

Flex предоставляет три класса для взаимодействия с серверами: `HTTPService`, `RemoteObject` и `WebService`. Класс `HTTPService` обеспечивает взаимодействие с сервером на основе протокола HTTP (Hypertext Transfer Protocol). Приложение Flex использует запросы GET или POST для отправки данных серверу и обрабатывает код XML или символьные строки, полученные в результате запроса. При помощи класса `HTTPService` можно взаимодействовать со страницами PHP, страницами ColdFusion, JSP (JavaServer Pages), сервлетами Java, Ruby on Rails и Microsoft ASP (Active Server Pages). Класс `RemoteObject` используется для взаимодействия с сервером с использованием объектов ActionScript Message Format. Класс `RemoteObject` также позволяет взаимодействовать со шлюзами Java и ColdFusion, .NET и PHP, для чего используются проекты с открытым исходным кодом AMFPHP, SabreAMF и WebORB. Класс `WebService` предназначен для взаимодействия с веб-службами, интерфейс которых определяется на языке WSDL (Web Services Description Language); при взаимодействиях используется традиционный формат XML или SOAP.

Чтобы создать компонент для работы со службой, необходимо задать свойства службы, задать URL-адрес сервера, который будет использоваться для отправки запросов и получения данных, а также информацию о предполагаемом типе данных. Для объекта `HTTPService` необходимо

выбрать метод передачи параметров серверу (GET или POST) и формат результата (resultFormat). Для компонента WebService задается URL-адрес документа WSDL с описанием службы, все используемые операции описываются тегами <mx:Operation>, для них задается результат и обработчики ошибок. Для класса RemoteObject UTL-адрес службы описывается в файле services-config.xml, компилируемом при создании файла SWF. В файле перечисляются все методы, определяемые службой, с указанием результата и обработчиков ошибок.

Данные, полученные от службы в результате вызова через HTTPService, размещаются в объекте lastResult, содержащемся в компоненте службы. Свойство resultFormat компонента службы по умолчанию является объектом ActionScript Object. Все данные, возвращаемые службой, представляются свойствами Object. Данные XML, возвращаемые WebService или HTTPService, Flex преобразует в соответствующие базовые типы: Number, String, Boolean или Date. Если же потребуется объект с сильной типизацией, экземпляры этого типа создаются и заполняются с использованием объектов, хранящихся в lastResult. Классы WebService и RemoteObject используют обработчик события, вызываемый при возвращении результатов службой. Обработчик ошибок вызывается для обработки всех сбоев и признаков ошибок, возвращаемых службой. Вся обработка полученных результатов выполняется в теле обработчика результата.

16.1. Настройка HTTPService

Задача

Требуется создать и настроить компонент HTTPService, чтобы приложение могло взаимодействовать со службами на базе HTTP.

Решение

Добавьте в приложение компонент HTTPService; задайте его свойству url URL-адрес, по которому приложение будет запрашивать данные. Если служба возвращает код XML, требующий дополнительной обработки, задайте свойству xmlDecode метод, получающий объект XML.

Обсуждение

Объект HTTPService обеспечивает все взаимодействия по протоколу HTTP. К их числу относится передача информации командами GET и POST, а также ее прием (в том числе и статических файлов) с URL-адресов. Объекту HTTPService могут быть назначены обработчики результатов и ошибок, получающие соответственно объекты mx.event.ResultsEvent и mx.event.FaultEvent:

```
<mx:HTTPService url="http://192.168.1.101/service.php" id="service"
result="serviceResult(event)" fault="serviceFault(event)">
```

Так организуется обработка результатов запросов HTTP. Свойство `lastResult` объекта `HTTPService` может использоваться для создания привязки:

```
<mx:Image source="{service.lastResult as String}"/>
```

Обратите внимание: свойство `lastResult` компонента `HTTPService` представляет собой объект, поэтому его необходимо преобразовать в строку.

Объект `HTTPService` также может использоваться для передачи информации сценариям в переменных `GET` и `POST` посредством задания свойства `request` объекта `HTTPService`:

```
<mx:HTTPService>
    <mx:request xmlns="">
        <id>{requestedId}</id>
    </mx:request>
</mx:HTTPService>
```

Этот фрагмент передает свойство `requestedId`, упакованное в тег `<id>`, по URL-адресу, заданному в объекте `HTTPService`.

В следующем примере объект `HTTPService` получает код XML от сценария PHP:

```
<mx: Application xmlns:mx="http://www.adobe.com/2006/mxml"
    width="400" height="300">

    <mx:HTTPService url="http://localhost/service.php"
         id="service"
         result="serviceResult(event)"
         fault="serviceFault(event)"
         method="GET" contentType="application/xml"
         useProxy="false">
         <mx:request xmlns="">
             <id>{requestedId}</id>
         </mx:request>
     </mx:HTTPService>
     <mx:Script>
         <![CDATA[

             import mx.rpc.events.FaultEvent;
             import mx.rpc.events.ResultEvent;

             [Bindable]
             private var requestedId:Number;
            // Трассировка полученных результатов
            private function serviceResult(event:Event):void {
                trace(service.lastResult.name);
            }
            // В случае сбоя или тайм-аута службы
            private function serviceFault(event:Event):void {
                trace('broken service');
            }
```

```
            private function callService():void {
                requestedId = input.text as Number;
                service.send()
            }
        ]]>
    </mx:Script>
    <mx:TextInput id="input"/>
    <mx:Button label="get user name" click="callService()"/>
    <mx:Text text="{service.lastResult.name}"/>
    <mx:Text text="{service.lastResult.age}"/>
</mx:Application>
```

Сценарий PHP, который получает данные из переменных `GET`, отправляемых приложением Flex, и возвращает отформатированные данные XML, выглядит так:

```
<?php
$id_number = $_GET["id"];
echo('<id>'.$id_number.'</id><name>Todd Anderson</name><age>30</age>')
?>
```

16.2. REST-взаимодействия в приложениях Flex

Задача

Требуется интегрировать приложение Flex с сервером, использующим коммуникации в стиле **REST (Representational State Transfer), например** Rails или другим сервером.

Решение

Создайте объект `HTTPService` для взаимодействия с сервером; используйте соответствующие пути в сочетании с методами POST и GET при вызове методов сервера.

Обсуждение

Термин «REST-служба» обозначает службы, использующие четыре основных заголовка (метода) HTTP: PUT, POST, DELETE и GET. Эти четыре заголовка обычно соответствуют четырем основным операциям с данными: созданию, чтению, обновлению и удалению. На практике один перегруженный серверный метод обеспечивает выполнение всех четырех операций в зависимости от передаваемого заголовка. В REST-приложениях эти методы часто соответствуют ресурсам, а четыре заголовка обеспечивают создание, удаление, обновление и чтение ресурсов. Ресурсом может быть простой блок данных, таблица базы данных или сложный объект.

Flash Player ограничивается использованием методов GET и POST; это означает, что в любых взаимодействиях между приложением Flex и службой методы **DELETE и PUT должны обозначаться другим спосо-**

бом, например присоединением идентификатора метода к сообщению GET или POST.

Отправка команды PUT приложению Rails выглядит примерно так:

```
var request:URLRequest = new URLRequest();
var loader:URLLoader = new URLLoader();
loader.addEventListener(Event.COMPLETE, resultHandler);
loader.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
loader.addEventListener(HTTPStatusEvent.HTTP_STATUS,httpStatusHandler);
request.url = "http://rails/view/resource";
// Выдать запрос типа POST с указанием команды DELETE
// в виде переменной в данных запроса
request.method = URLRequestMethod.POST;
request.data._method = "DELETE";
loader.load(request);
```

Ruby on Rails позволяет указать метод в переменной `_method` даже в том случае, когда использование правильного метода HTTP невозможно. Для других типов REST-служб могут использоваться аналогичные приемы.

Чтобы обойти это ограничение для объектов `HTTPService`, можно воспользоваться серверами BlazeDS или Adobe LiveCycle. `HTTPService` определяет свойство `useProxy`; если оно равно `true`, Flash Player взаимодействует только с сервером, определенным в файле services-config.xml. Запрос на построение и передачу опосредованных запросов PUT/DELETE/OPTIONS (и т. д.) отправляется серверу Adobe LiveCycle или BlazeDS; сервер строит и отправляет фактический запрос HTTP, а затем возвращает ответ проигрывателю. Прокси-сервер также обрабатывает коды ошибок в диапазоне HTTP 500, представляющие сообщения об ошибках сервера, и возвращает их в форме, пригодной для содержательного использования (Flash Player не обрабатывает коды ответов в диапазоне 500).

После того как объект `HTTPService` будет настроен для использования прокси-сервера BlazeDS или LiveCycle, вы можете использовать все типы заголовков с объектом `HTTPService`, задавая значение свойства `method`:

```
<mx:HTTPService id="proxyService" destination="http://localhost/app/url"/>
<mx:Script>
    <![CDATA[
        private function sendPut():void {
            proxyService.method = "DELETE";
            proxyService.send("id=2");
        }
    ]]>
</mx:Script>
```

Наконец, существует библиотека as3httpclient, написанная Гэбриелом Хандфордом (Gabriel Handford); она использует двоичный сокет Flash для чтения потока данных HTTP и декодирования ответа. Она позволяет передавать и читать HTTP-ответы GET, POST, PUT и DELETE, но для ее

использования вам придется создать файл crossdomain.xml и разрешить Flash Player подключение к серверу через порт 80. Хотя взаимодействовать с сервером через `Socket` относительно сложно, если вам потребуется работать с сервером, использующим REST-ответы и жесткие ограничения HTTP, эта библиотека становится достойной альтернативой для стандартного компонента `HTTPService`. За дополнительной информацией и примерами кода обращайтесь по адресу *http://code.google.com/p/as3httpclientlib/*.

16.3. Настройка и подключение к RemoteObject

Задача

Требуется настроить компонент `RemoteObject` в приложении Flex для подключения к объекту ColdFusion, AMFPHP или Java.

Решение

Создайте в приложении экземпляр `RemoteObject` и задайте идентификатор службы, например URL-адрес, по которому должно производиться обращение к службе.

Обсуждение

Компонент `RemoteObject` используется для определения взаимодействий между приложением и объектами на сервере. В этом отношении он существенно отличается от компонента `WebService` (который берет информацию из WSDL-файла) или `HTTPService` (который просто использует URL-адрес для отправки и получения информации HTTP). Компонент `RemoteObject` может использоваться для вызова методов компонентов ColdFusion **CFC или классов Java, опубликованных для обслуживания** внешних клиентов. `RemoteObject` также способен взаимодействовать с объектами и ресурсами, определяемыми в проектах с открытым исходным кодом – таких как AMFPHP, SabreAMF и WebORB. Класс `RemoteObject` определяет следующие свойства:

`channelSet : ChannelSet`

Предоставляет доступ к `ChannelSet`, используемому службой.

`concurrency : String`

Значение, которое определяет способ обработки нескольких вызовов к одной службе.

`constructor : Object`

Ссылка на объект класса или функцию-конструктор для заданного экземпляра.

`destination : String`

Приемный адрес службы.

`endpoint : String`

Позволяет быстро задать конечную точку для приемного адреса `RemoteObject` без обращения к конфигурационному файлу службы на стадии компиляции или программного создания `ChannelSet`.

`makeObjectsBindable : Boolean`

Если свойство равно `true`, для возвращаемых анонимных объектов принудительно включается возможность привязки.

`operations : Object`

Задает методы, определяемые службой; используется при определении `RemoteObject` в коде MXML. В ActionScript обычно не используется.

`requestTimeout : int`

Определяет тайм-аут запроса (в секундах) для отправленных сообщений.

`showBusyCursor : Boolean`

Если свойство равно `true`, во время обращения к службе отображается признак ожидания.

`source : String`

Позволяет задать источник для клиента; не поддерживается для приемников, использующих JavaAdapter для сериализации обмена данными между файлом SWF и объектом Java.

Так как методы `RemoteObject` могут возвращать объекты, которые не нуждаются в дополнительной обработке или десериализации из XML, результат вызова `RemoteObject` из `ResultEvent` может быть преобразован в `ArrayCollection` или типизованный объект. В следующем фрагменте `RemoteObject` настраивается для использования службы Java, доступной по адресу `http://localhost:8400`:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:RemoteObject id="local_service" concurrency="single"
            destination="http://localhost:8400/app"
        showBusyCursor="true" source="LocalService.Namespace.Service.
            ServiceName">
            <mx:method name="getNames" fault="getNamesFault(event)"
                result="getNamesResult(event)"/>
            <mx:method name="getAges" fault="getAgesFault(event)"
                result="getAgesResult(event)"/>
    </mx:RemoteObject>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;
            import mx.controls.Alert;
            import mx.rpc.events.FaultEvent;
```

```
            private function getNamesFault(event:FaultEvent):void {
                mx.controls.Alert.show(event.message as String,
                "Service Error");
            }

            private function getNamesResult(event:ResultEvent):void {
                var namesColl:ArrayCollection = event.result as
                ArrayCollection;
            }

            private function getAgesFault(event:FaultEvent):void {
                mx.controls.Alert.show(event.message as String,
                "Service Error");
            }

            private function getAgesResult(event:ResultEvent):void {
                   var agesColl:ArrayCollection = event.result as
                   ArrayCollection;
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Событие `result` каждого метода привязывается к отдельному методу-обработчику. В коде ActionScript эта задача решалась бы добавлением к `RemoteObject` метода-слушателя события:

```
import mx.collections.ArrayCollection;
import mx.rpc.events.ResultEvent;
import mx.controls.Alert;
import mx.rpc.events.FaultEvent;
import mx.rpc.AbstractService;
import mx.rpc.AsyncToken;
import mx.rpc.Responder;

private function init():void {
    var responder:Responder = new Responder( getNamesResult,
        getNamesFault );
    var call:AsyncToken = ( local_service as AbstractService).getNames();
    call.addResponder(responder);
}

private function getNamesFault(event:FaultEvent):void {
    Alert.show(event.message as String, "Service Error");
}

private function getNamesResult(event:ResultEvent):void {
    var namesColl:ArrayCollection = event.result as ArrayCollection;
}
```

В этом примере класс `mx.rpc.Responder` используется для хранения методов, обрабатывающих серверные события `result` и `fault`. Он включается

в класс AsyncToken, который активизируется при возвращении сервером результата или ошибки.

16.4. Использование удаленных взаимодействий Flex с AMFPHP 1.9

Материал предоставлен Санкаром Панирселвамом (Sankar Paneerselvam).

Задача

Требуется организовать удаленное взаимодействие приложения Flex с сервером, на котором установлена реализация AMFPHP.

Решение

Установите AMFPHP и настройте его для подключения к источнику данных. Используйте RemoteObject для обращения к службе AMFPHP и вызова ее методов.

Обсуждение

Для демонстрации использования AMFPHP с Oracle Database Express Edition (XE) в этом примере таблица из базы данных Oracle отображается с использованием компонентов удаленных взаимодействий. Таблица EMPLOYEES из схемы HR в Oracle Database XE обладает следующей структурой:

```
EMPLOYEE_ID         PLS_INTEGER
FIRST_NAME          VARCHAR2
LAST_NAME           VARCHAR2
EMAIL               VARCHAR2
PHONE_NUMBER        VARCHAR2
HIRE_DATE           DATE
JOB_ID              PLS_INTEGER
SALARY              NUMBER
COMMISSION_PCT      NUMBER
MANAGER_ID          PLS_INTEGER
DEPARTMENT_ID       PLS_INTEGER
```

Соответствующий класс Employee в PHP создается следующим образом:

```
<?php
class Employee
{
    var $EMPLOYEE_ID;
```

```
    var $FIRST_NAME;
    var $LAST_NAME;
    var $EMAIL;
    var $PHONE_NUMBER;
    var $HIRE_DATE;
    var $JOB_ID;
    var $SALARY;
    var $COMMISSION_PCT;
    var $MANAGER_ID;
    var $DEPARTMENT_ID;

    var $_explicitType = "project.Employee";
}
?>
```

Обозначение `$_explicitType` **подразумевает, что в ActionScript определен класс, соответствующий классу** `Employee`**.**

В этом рецепте AMFPHP – инфраструктура PHP с открытым исходным кодом – используется для потоковой передачи данных через PHP. Службы AMFPHP могут вызываться Flex `RemoteObject` **для получения данных в формате AMF. В этом рецепте используется служба с именем** `EmployeeService`**, возвращающая информацию о работниках. Файл службы необходимо поместить в папку services установочной папки AMFPHP. Предполагается, что разработчик использует каталог верхнего уровня Project, в котором размещаются все службы.**

```
<?php
require_once('./Employee.php');
class EmployeeService {
    var $myconnection=null;
    var $statement=null;

function getEmployees(){
    $myconnection = oci_connect('hr','hr', "//localhost/xe");
        # Check Oracle connection"
        if (!myconnection) {
            # Не использовать die (фатальная ошибка),
            # вернуть клиенту полезную информацию
            trigger_error("AMFPHP Remoting 'EmployeeService'
                class could not connect: " . oci_error());
        }
        $query="SELECT * FROM EMPLOYEES";
        # Вернуть список всех работников
        $statement=oci_parse($myconnection,$query);
        if (!$statement) {
            oci_close($myconnection);
            trigger_error("AMFPHP Remoting 'EmployeeService'
               class database SELECT query error: " .
               oci_error());
        }
        oci_execute($statement);
```

```
        while ($row = oci_fetch_array($statement,OCI_RETURN_NULLS)) {
            $data_array[] = $row;
        }
        return($data_array);
    }
}
?>
```

Работу службы можно проверить при помощи программы из папки browser в установочном каталоге AMFPHP.

Чтобы приложение Flex могло участвовать в удаленных взаимодействиях, необходимо предоставить файл service-config.xml. Файл размещается в проекте Flex, а его примерный вид показан ниже. Вероятно, путь и номер порта придется изменить в соответствии с данными вашего сервера, а вся остальная информация должна остаться неизменной.

```
<services-config>
    <services>
        <service id="amfphp-flashremoting-service"
            class="flex.messaging.services.RemotingService"
            messageTypes=
            "flex.messaging.messages.RemotingMessage">
            <destination id="amfphp">
                <channels>
                    <channel ref="my-amfphp"/>
                </channels>
                <properties>
                    <source>*</source>
                </properties>
            </destination>
        </service>
    </services>
    <channels>
        <channel-definition id="my-amfphp"
            class="mx.messaging.channels.AMFChannel">
            <endpoint uri="http://localhost:9999/
            amfphp2/amfphp/gateway.php"
            class="flex.messaging.endpoints.AMFEndpoint"/>
        </channel-definition>
    </channels>
</services-config>
```

Созданный файл service-config.xml необходимо зарегистрировать:

1. Выполните в Flex Builder команду Project→Properties.
2. В секции Flex Compiler введите `-services service-config.xml` после строки `locale en_US`. Если ранее содержимое этого поля не изменялось, оно должно выглядеть так:

```
-locale en_US -services service-config.xml
```

3. Чтобы изменения вступили в силу, щелкните на кнопке Apply.

Создайте класс ActionScript для хранения объектов удаленного взаимодействия:

```
package project
{
    [RemoteClass(alias="project.Employee")]
    public class Employee
    {
        public var EMPLOYEE_ID:Number;
        public var FIRST_NAME:String;
        public var LAST_NAME:String;
        public var EMAIL:String;
        public var PHONE_NUMBER:Number;
        public var HIRE_DATE:Date;
        public var JOB_ID:Number;
        public var SALARY:Number;
        public var COMMISSION_PCT:Number;
        public var MANAGER_ID:Number;
        public var DEPARTMENT_ID:Number;
    }
}
```

Код MXML с вызовом удаленного метода, получением результатов и их выводом:

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

А здесь у объекта `RemoteObject`**, который обращается к службе** `Employee`**, задаются свойства** `destination` **и** `source`**, а также назначаются обработчики событий:**

```
<mx:RemoteObject id="myservice" source="Project.EmployeeService"
destination="amfphp" fault="faultHandler(event)"
    showBusyCursor="true">
    <mx:method name="getEmployees" result="resultHandler(event)"
fault="faultHandler(event)">
    </mx:method>
</mx:RemoteObject>
<mx:Script>
    <![CDATA[

        import Project.Employee;
        import mx.utils.ArrayUtil;
        import mx.collections.ArrayCollection;
        import mx.rpc.events.FaultEvent;
        import mx.rpc.events.ResultEvent;
        import mx.controls.Alert;
```

При возвращении данных в `dp:ArrayCollection` **сохраняется значение свойства** `result` **объекта** `ResultEvent`**, полученного от сервера:**

```
        [Bindable]
        private var dp:ArrayCollection;
        private function faultHandler(event:FaultEvent):void {
```

```
            Alert.show(event.fault.faultString,
            event.fault.faultCode.toString());
        }
        private function
            resultHandler(event:ResultEvent):void {
            dp=new ArrayCollection(
            ArrayUtil.toArray(event.result));
        }
    ]]>
</mx:Script>
    <mx:Canvas x="0" y="0" width="100%" height="100%">
        <mx:Button x="10" y="10" label="Get data"
click="myservice.getOperation('getEmployees').send()"/>
```

Каждый столбец соответствует некоторому свойству в объекте данных, возвращенном службой:

```
        <mx:DataGrid x="10" y="40" width="100%" height="100%"
dataProvider="{dp}">
            <mx:columns>
                <mx:DataGridColumn headerText="EMPLOYEE_ID"
                    dataField="EMPLOYEE_ID"/>
                <mx:DataGridColumn headerText="FIRST_NAME"
                    dataField="FIRST_NAME"/>
                <mx:DataGridColumn headerText="LAST_NAME"
                    dataField="LAST_NAME"/>
                <mx:DataGridColumn headerText="EMAIL"
                    dataField="EMAIL"/>
                <mx:DataGridColumn headerText="PHONE_NUMBER"
                    dataField="PHONE_NUMBER"/>
                <mx:DataGridColumn headerText="HIRE_DATE"
                    dataField="HIRE_DATE"/>
                <mx:DataGridColumn headerText="JOB_ID"
                    dataField="JOB_ID"/>
                <mx:DataGridColumn headerText="SALARY"
                    dataField="SALARY"/>
                <mx:DataGridColumn
                    headerText="COMMISSION_PCT"
                    dataField="COMMISSION_PCT"/>
                <mx:DataGridColumn headerText="MANAGER_ID"
                    dataField="MANAGER_ID"/>
                <mx:DataGridColumn
                    headerText="DEPARTMENT_ID"
                    dataField="DEPARTMENT_ID"/>
            </mx:columns>
        </mx:DataGrid>
    </mx:Canvas>
</mx:Application>
```

Когда пользователь щелкает на кнопке Get Data, программа отправляет удаленный запрос, получает результаты и заполняет компонент `DataGrid`**.**

16.5. Использование интерфейса IExternalizable для пользовательской сериализации

Материал предоставлен Питером Фарлендом (Peter Farland).

Задача

Требуется настроить состав свойств, передаваемых по каналу связи при передаче данных с сильной типизацией через объекты `RemoteObject` и `DataService`.

Решение

Используйте ActionScript 3 API `flash.utils.IExternalizable`, совместимый с Java API `java.io.IExternalizable`.

Обсуждение

В одном из типовых сценариев внешнего доступа к классам в сериализацию включаются только свойства, доступные только для чтения. Хотя в серверном коде эта цель может быть достигнута разными способами, для клиентского кода таких способов не так уж много. Одно из элегантных решений, работающее как на стороне клиента, так и на стороне сервера, заключается в обеспечении двусторонней пользовательской сериализации.

Идея, в общем, тривиальна: клиентский класс ActionScript просто реализует интерфейс `flash.utils.IExternalizable`. API требует определения двух методов `readExternal` и `writeExternal`, получающих потоки `flash.utils.IDataInput` и `flash.utils.IDataOutput` соответственно. Реализации этих методов повторяют серверный класс Java, реализующий интерфейс `java.io.Externalizable`, который тоже состоит из двух методов `readExternal` и `writeExternal` и получает потоки `java.io.ObjectInput` и `java.io.ObjectOutput` соответственно.

В этом примере основное внимание уделяется сериализации свойств, доступных только для чтения, однако пользовательская сериализация находит немало других полезных применений, например пропуск отдельных свойств, предотвращение сериализации неиспользуемой информации или включения свойств из пользовательских пространств имен.

Хотя классы `IDataInput` и `IDataOutput` позволяют проектировать собственные протоколы и записывать такие фундаментальные типы данных, как `byte`, `int` и `String` в кодировке UTF-8, большинство реализаций использует методы `readObject` и `writeObject`, которые обеспечивают эффективную сериализацию и десериализацию объектов ActionScript средствами AMF 3. Вы даже можете опустить имена свойств в своей пользователь-

ской реализации и положиться на фиксированный порядок их следования, чтобы передавать только значения свойств.

Обратите внимание на сходство метода Java `writeExternal`

```
public void writeExternal(ObjectOutput out)
    throws IOException
{
    out.writeObject(id);
    out.writeObject(name);
    out.writeObject(description);
    out.writeInt(price);
}
```

и клиентского метода `readExternal` в ActionScript:

```
public function readExternal(input:IDataInput):void
{
    _id = input.readObject() as String;
    name = input.readObject() as String;
    description = input.readObject() as String;
    price = input.readInt();
}
```

Аналогичная связь существует и для обратной ситуации, т. е. отправки экземпляров от клиента к серверу.

16.6. Отслеживание результатов нескольких параллельных вызовов службы

Материал предоставлен Эндрю Элдерсоном (Andrew Alderson).

Задача

Требуется определить, к какому из нескольких параллельных вызовов службы относятся возвращенные данные.

Решение

Используйте `ASyncToken` для добавления к каждому вызову идентифицирующей его переменной.

Обсуждение

Класс `mx.rpc.ASyncToken` является динамическим, т. е. может дополняться новыми свойствами и методами во время выполнения. В документации Flex говорится, что «здесь добавляются дополнительные данные для асинхронных операций `rpc`».

В качестве примера рассмотрим приложение с компонентом `DateChooser`. Каждый раз, когда пользователь прокручивает компонент к новому месяцу, с сервера должен загружаться файл XML для этого месяца. Так

как порядок поступления этих файлов не гарантирован, необходимо как-то идентифицировать их. Использование ASyncToken позволяет добавить в событие получения результата, возвращаемое службой, свойство-идентификатор. Пример:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"> <mx:Script>
<![CDATA[
import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;
import mx.rpc.AsyncToken;
import mx.events.DateChooserEvent;
private function
    scrollHandler(event:DateChooserEvent):void {
    var month:int = event.currentTarget.displayedMonth;
    var monthName:String =
        event.currentTarget.monthNames[month];
    service.url = "xml/"+monthName+".xml";
    var token:AsyncToken = service.send();
    token.resultHandler = onResult;
    token.faultHandler = onFault;
    token.month = monthName;
}
private function onResult(event:ResultEvent):void {
    resultText.text = "MonthName: "+
        event.token.month+"\n\n";
    resultText.text += "Result: "+
        event.result.data.month;
}
private function onFault(event:FaultEvent):void {
    resultText.text = event.fault.faultString;
}
]]> </mx:Script>
<mx:HTTPService id="service" result="event.token.resultHandler(event)"
fault="event.token.faultHandler(event)"/>
<mx:DateChooser id="dateChooser" scroll="scrollHandler(event)"/>
<mx:TextArea id="resultText" width="300" height="200"/>
</mx:Application>
```

В этом фрагменте обработчик события scrollHandler вызывается для получения файла XML с сервера. Если пользователь достаточно быстро работает с мышью, возможна одновременная отправка нескольких запросов. В HTTPService метод send возвращает ASyncToken; вы можете обратиться к нему и добавить свойство для идентификации месяца, к которому относятся возвращенные данные. Свойство token объекта ResultEvent используется для обращения к свойству month в обработчике result.

Описанный способ также может использоваться с вызовами WebService и RemoteObject. В этом случае вызываемая операция или метод возвращает ASyncToken:

```
var token : AsyncToken = service.login( loginVO );
```

16.7. Публикация и подписка

Задача

Требуется оповестить клиентское приложение Flex об изменении данных на стороне сервера или обеспечить групповую рассылку для всех слушателей на сервере обмена сообщениями.

Решение

Воспользуйтесь тегами `mx.messaging.Producer` и `mx.messaging.Consumer` и настройте приемный канал так, чтобы он мог использоваться для обмена данными и назначения обработчика события для передачи сообщений. Чтобы правильно выполнить такую настройку, придется использовать серверы Adobe LiveCycle или BlazeDS.

Обсуждение

Модель публикации/подписки состоит из двух компонентов: `mx.messaging.Producer` и `mx.messaging.Consumer`. Поставщик (`Producer`) отправляет сообщения *приемнику* на сервере, где эти сообщения обрабатываются. Потребитель (`Consumer`) подписывается на сообщения через приемник и обрабатывает сообщения при их поступлении.

Поддержка обмена сообщениями в Flex позволяет использовать механизмы ActionScript Messaging и Java Message Service (JMS). Механизм ActionScript Messaging совместим только с клиентами, работающими в формате AMF, а для его правильной работы необходимы соответствующие классы. Механизм JMS позволяет LiveCycle или BlazeDS участвовать в обмене сообщениями JMS и взаимодействовать с любыми клиентами JMS. Любое приложение, способное обмениваться сообщениями JMS, может быть напрямую вызвано из клиента Flex, а любое приложение Java может публиковать события для Flex.

`Consumer` получает сообщения в форме `mx.messaging.events.MessageEvent`:

```
private function receiveChatMessage(
    msgEvent:MessageEvent):void

{
    var msg:AsyncMessage = AsyncMessage(msgEvent.message);
    trace("msg.body "+msg.body);
}
```

`Producer` отправляет сообщения методом `send`, при вызове которого передается параметр `mx.messaging.AsyncMessage`. Тело `AsyncMessage` содержит значение, отправляемое всем подписчикам канала:

```
var msg:AsyncMessage = new AsyncMessage();
msg.body = "test message";
producer.send(msg);
```

Полный код примера:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
    pageTitle="Chat" creationComplete=
        "chatSubscriber.subscribe()">
    <!-- Обьявления, необходимые для обмена сообщениями -->
    <mx:Producer id="producer" destination="http://localhost:8400/
        chatDestination"/>
    <mx:Consumer id="subscriber" destination="http://localhost:8400/
        chatDestination"
message="receiveChatMessage(event)" />
    <mx:Script>
        <![CDATA[

        import mx.messaging.events.MessageEvent;
        import mx.messaging.messages.AsyncMessage;
        private function sendChatMessage():void
        {
            var msg:AsyncMessage = new AsyncMessage();
            msg.body = "test message";
            producer.send(msg);
        }
        private function receiveChatMessage(msgEvent:MessageEvent):void
        {
            var msg:AsyncMessage = AsyncMessage(msgEvent.message);
            trace("msg.body "+msg.body);
        }

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

16.8. Регистрация серверного типа данных в приложении Flex

Задача

Требуется зарегистрировать тип данных со стороны сервера в приложении, чтобы объекты, получаемые из `RemoteObject`, могли быть преобразованы в экземпляры удаленного класса в приложениях Flex.

Решение

Используйте метод `flash.net.RegisterClass` или включите пометку `RemoteClass` в объявление класса.

Обсуждение

Чтобы объект можно было десериализовать из данных AMF в объект `Class`, необходимо зарегистрировать сигнатуру класса в Flash Player. Допустим, в C# определяется следующий тип:

```
using System;
using System.Collections;

namespace oreilly.cookbook.vo
{
    public class RecipeVO {
           public string title;
           public ArrayList ingredients;
           public ArrayList instructions
               public RecipeVO(){}
    }
}
```

Соответствующий тип ActionScript может выглядеть так:

```
package oreilly.cookbook.vo
{
    public class RecipeVO

    public var ingredients:Array;
    public var instructions:Array;
    public var title:String;

    public function RecipeVO(){}
}
```

Служба, возвращающая объект `RecipeVO`, может создать новый объект C# и вернуть его:

```
using System;
using System.Web;
using oreilly.cookbook.vo;
namespace oreilly.cookbook.service {
    public class RecipeService {
        public RecipeService() { }

        public RecipeVO getRecipe() {
            RecipeVO rec = new RecipeVO();
            rec.title = "Apple Pie";
            string[] ingredients = {"flour", "sugar",
                "apples", "eggs", "water"};
            rec.ingredients = new ArrayList(ingredients);
            string[] instructions = {"instructions are long",
                "baking is hard", "maybe I'll just buy it at the store"};
            rec.instruction = new ArrayList(instructions);
            return rec;
        }
    }
}
```

Хотя служба и объект (в данном случае `RecipeVO`) написаны на C#, реализация службы и сериализации на Java и других серверных языках будет выглядеть практически так же.

При возвращении объекта службой приложение может обратиться к RecipeVO следующим образом:

```
<mx:RemoteObject id="recipeService" destination="fluorine"
source="oreilly.cookbook.FlexService" showBusyCursor="true"
       result="roResult(event)" fault="roFault(event)" />
    <mx:Script>
        <![CDATA[
        private function initApp():void {
            // Необходимо зарегистрировать класс
            // для правильного преобразования результата
            // к типу RecipeVO
            flash.net.registerClassAlias("oreilly.cookbook.vo.RecipeVO",
                RecipeVO);
        }
        public function serviceResult(e:ResultEvent):void {
            var rec:RecipeVO = (e.result as RecipeVO)
        }
        public function serviceFault(e:FaultEvent):void {
            trace(" Error :: "+(e.message as String));
        }
        ]]>
    </mx:Script>
```

После того как класс будет зарегистрирован методом registerClassAlias, объекты с подходящей сигнатурой, полученные от сервера, могут быть преобразованы к классу RecipeVO.

16.9. Взаимодействие с веб-службами

Задача

Требуется обеспечить взаимодействие приложений Flex с сервером через веб-службы, которые отправляют сообщения WSDL для описания своих методов, а затем воспользоваться полученной информацией для вызова методов веб-службы.

Решение

Создайте объект mx.rpc.WebService и задайте свойству wsdl местонахождение документа WSDL с описанием службы.

Обсуждение

Компонент WebService обеспечивает взаимодействие приложения с веб-службами на основе информации из файла WSDL. Flash Player поддерживает следующие свойства в файлах WSDL:

<binding>

Протокол, который должен использоваться клиентами (в том числе приложениями Flex) для взаимодействия с веб-службой. Возмож-

ные варианты: SOAP, HTTP GET, HTTP POST и MIME (Multipurpose Internet Mail Extensions). Flex поддерживает только SOAP.

`<fault>`

Код ошибки, возникшей в результате обработки сообщения.

`<input>`

Сообщение, отправляемое клиентом (например, приложением Flex) веб-службе.

`<message>`

Данные, передаваемые операцией веб-службы.

`<operation>`

Комбинация тегов `<input>`, `<output>` и `<fault>`.

`<output>`

Сообщение, отправляемое веб-службой клиенту (например, приложению Flex).

`<port>`

Конечная точка веб-службы; устанавливает связь между протоколом и сетевым адресом.

`<portType>`

Определения одной или нескольких операций, предоставляемых веб-службой.

`<service>`

Набор тегов `<port>`. Каждая запись `service` соответствует одному тегу `<portType>` и определяет разные способы обращения к операциям этого тега.

`<types>`

Типы данных, используемые в сообщениях веб-службы.

Приложение Flex просматривает файл WSDL, определяет все методы, поддерживаемые службой, и возвращаемые типы данных. В типичном файле WSDL определяется имя службы, используемые ей типы данных и содержимое получаемых/возвращаемых сообщений.

При создании объекта `WebService` следует задать `id` службы и местонахождение файла WSDL с определением службы:

```
<mx:WebService id="userRequest" wsdl="http://localhost:8400/service/
    service?wsdl">
    <mx:operation name="getRecipes" result="getRecipeHandler()"
        fault="mx.controls.Alert.show(event.fault.faultString)"/>
</mx:WebService>
```

Компонент `WebService` передает событие `LoadEvent` типа `load` или `LoadEvent.LOAD`; оно означает, что компонент успешно загрузил и разобрал файл

WSDL, указанный в свойстве wsdl, и готов к вызову методов. До этого момента вызовы WebService невозможны, поэтому рекомендуется использовать указанное событие для установки флага возможности вызова. Компонент WebService также определяет готовый логический признак, при помощи которого можно проверить, что файл WSDL был загружен, а компонент WebService готов к работе. В следующем примере определяется метод и назначаются обработчики событий result и fault службы:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:WebService id="userRequest" wsdl="http://localhost:8500/service/
service?wsdl"
load="callService()">
        <mx:operation name="getRecipes" resultFormat="object"
            fault="createRecipeFault(event)"
            result="createRecipeHandler(event)"/>
    </mx:WebService>
    <mx:Script>
        <![CDATA[
            import mx.rpc.events.FaultEvent;
            import mx.collections.ArrayCollection;
            import mx.rpc.events.ResultEvent;
            import mx.controls.Alert;

            private function callService():void {
                userRequest.getRecipes();
            }
            private function createRecipeHandler(
                event:ResultEvent):void {
                var arrayCol:ArrayCollection =
                event.result as ArrayCollection;
            }
            private function createRecipeFault(
                event:FaultEvent):void {
                Alert.show(" error :: "+event.message);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

16.10. Включение заголовка SOAP в запрос

Задача

Требуется отправить заголовок SOAP с запросом к компоненту WebService.

Решение

Создайте объект SOAPHeader и укажите пространство имен, которое должно использоваться для передаваемых значений и контента, присоеди-

няемого к заголовку. Затем отправьте заголовок с запросом методом WebService.addHeader.

Обсуждение

Заголовки SOAP часто используются веб-службами для передачи регистрационных данных пользователя, информации о пользователе и других данных вместе с запросом. При создании объекта SOAPHeader задается пространство имен для содержащихся в нем данных и объект с определением данных, отправляемых в заголовке, в формате пар «ключ-значение»:

```
SOAPHeader(qname:QName, content:Object)
```

Пример создания двух объектов SOAPHeader:

```
// Создание объекта QName, который может использоваться с заголовком
var qname:QName=new QName("http://soapinterop.org/xsd", "CookbookHeaders");
var headerone:SOAPHeader = new SOAPHeader(qname,
    {string:"header_one",int:"1"});
var headertwo:SOAPHeader = new SOAPHeader(qname,
    {string:"header_two",int:"2"});
```

Чтобы заголовок включался во все запросы, проходящие через веб-службу, вызовите метод addHeader для самого объекта WebService:

```
// Вызов метода addHeader
service.addHeader(headerone);
```

Чтобы добавить объект SOAPHeader только для одного конкретного метода, определяемого службой, укажите имя метода при вызове addHeader:

```
// Добавление заголовка headertwo для операции getRecipe.
service.getRecipes.addHeader(headertwo);
```

Если надобность в заголовках SOAP отпала, вызовите метод clearHeaders для компонента WebService или того метода, для которого добавлялись заголовки:

```
service.clearHeaders();
service.getRecipes.clearHeaders();
```

16.11. Разбор полученного ответа SOAP

Задача

Требуется разобрать данные SOAP, полученные в ответ на запрос.

Решение

Используйте встроенную десериализацию Flash Player типов SOAP в типы ActionScript для кода XML в кодировке SOAP, возвращаемого WebService.

Обсуждение

Для разбора ответов в формате SOAP можно использовать выражения E4X. Самые распространенные типы перечислены в табл. 16.1 вместе с их представлениями в SOAP и ActionScript.

Таблица 16.1. Типы SOAP и соответствующие им типы ActionScript

Тип	SOAP	ActionScript 3
String	xsd:String	String
Integer	xsd:int	Int
Float	xsd:float	Number
Boolean	xsd:Boolean	Boolean
Date	xsd:date	Date
Array	xsd:string[], xsd:int[] **и т. д.**	ArrayCollection
Object	Element	Object
Binary	xsd:Base64Binary	flash.utils.ByteArray
Null	xsl:Nil	Null

Например, для файла WSDL со следующими определениями возвращаемых типов:

```
<wsdl:types>
  <schema elementFormDefault="qualified"
targetNamespace = "http://cookbook.webservices.com"
xmlns = "http://www.w3.org/2001/XMLSchema">

   <complexType name="Recipe">
    <sequence>
    <element name="title" nillable="true" type="xsd:string"/>
    <element name="ingredients" nillable="true" type="xsd:string[]"/>
<element name="instructions" nillable="true" type="xsd:string[]"/>
</sequence>
   </complexType>

  </schema>
 </wsdl:types>
```

запрос к `WebService` возвращает следующий ответ:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:ns="http://cookbook.oreilly.com/service">
    <ns:GetRecipes>
        <ns:Recipe>
            <ns:title>"Blueberry Pie"</ns:title>
            <SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">
                <ns:ingredient>"Blueberry"</ns:ingredient>
```

```
                <ns:ingredient>"Sugar"</ns:ingredient>
                <ns:ingredient>"Crust"</ns:ingredient>
            </SOAP-ENC:Array>
            <SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[3]">
                <ns:instruction>"Blueberry"</ns:instruction>
                <ns:instruction>"Sugar"</ns:instruction>
                <ns:instruction>"Crust"</ns:instruction>
            </SOAP-ENC:Array>
        </ns:Recipe>
    </ns:GetRecipes>
  </soap:Body>
</soap:Envelope>
```

Этот ответ легко разбирается на объекты с использованием точечной записи, как это делается для любых других объектов XML. За дополнительной информацией о различных типах SOAP и их эквивалентах в ActionScript обращайтесь к описанию всех возможных типов SOAP по адресу *http://www.adobe.com/go/kb402005*.

16.12. Защищенное взаимодействие с AMF

Задача

Требуется обеспечить удаленные взаимодействия Flash с использованием данных AMF и SSL (Secure Sockets Layer).

Решение

Определите свой канал с классом `SecureAMFChannel` в файле services-config.xml, используемом при компиляции приложения.

Обсуждение

`SecureAMFChannel` позволяет использовать `AMFChannel` для обмена данными SSL, обеспечивающего безопасность всех пересылаемых данных. Чтобы создать новый канал, использующий защищенные версии классов AMF, просто создайте файл services-config.xml с каналом, использующим класс `mx.messaging.channels.SecureAMFChannel`. Конечная точка также настраивается для использования класса `flex.messaging.endpoints.SecureAMFEnd`, как показано ниже:

```
<channels>
    <channel ref="secure-amf"/>
</channels>

<channel-definition id="secure-amf" class="mx.messaging.channels.
    SecureAMFChannel">
    <endpoint uri="https://{server.name}:{server.port}/gateway/"
    class="flex.messaging.endpoints.SecureAMFEndpoint"/>
    <properties>
        <add-no-cache-headers>false</add-no-cache-headers>
```

```
            <polling-enabled>false</polling-enabled>
            <serialization>
                <instantiate-types>false</instantiate-types>
            </serialization>
        </properties>
    </channel-definition>
```

Предыдущий фрагмент когда будет работать с ColdFusion, Adobe LiveCycle и BlazeDS:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300"
creationComplete="init()">
    <mx:RemoteObject id="channelRO"/>
    <mx:Script>
        <![CDATA[

            import mx.messaging.ChannelSet;
            import mx.messaging.channels.SecureAMFChannel;

            private var cs:ChannelSet

            private function init():void {

                cs = new ChannelSet();
                // Имя канала совпадает с именем,
                // указанным в файле services-config.xml

                var chan: SecureAMFChannel =
                    new SecureAMFChannel("secure-amf", "gateway")
                chan.pollingEnabled = true;
                chan.pollingInterval = 3000;

                cs.addChannel(chan);
                channelRO.channelSet = cs;
            }

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

Теперь канал может использоваться для выполнения защищенных вызовов AMF через `RemoteObject`.

16.13. Отправка и получение двоичных данных через двоичный сокет

Задача

Требуется принять двоичные данные и выдать ответ в аналогичном двоичном формате после обработки данных.

Решение

Используйте класс `flash.net.Socket`; откройте сокетное подключение к серверу и номеру порта, с которыми будет взаимодействовать ваше приложение.

Обсуждение

Из всех коммуникационных средств Flex Framework или ActionScript 3 класс `flash.net.Socket` работает на самом низком уровне. С его помощью создаются сокетные соединения для чтения и записи неструктурированных двоичных данных. Класс `Socket` позволяет отправлять и принимать сообщения в форматах POP3 (Post Office Protocol version 3), SMTP (Simple Mail Transfer Protocol) и IMAP (Internet Message Access Protocol), а также пользовательских двоичных форматах. Flash Player также может взаимодействовать с сервером на уровне прямого использования двоичного протокола этого сервера.

Чтобы создать новый сокет, создайте экземпляр `Socket` при помощи конструктора и вызовите метод `connect` с передачей IP-адреса или имени домена с номером порта:

```
var socket:Socket;
// Создание нового сокета и подключение
// к хосту 127.0.0.1 через порт 8080
private function init():void {

    socket = new Socket();
    socket.addEventListener(ProgressEvent.SOCKET_DATA,
        readSocketData);
    socket.connect("127.0.0.1", 8080);

}
// Отправка данных сокету
private function sendSocketData(string:String):void {
    // Отправка строковых данных с выбором кодировки строки.
    // В данном примере используется стандартная
    // западноевропейская кодировка iso-08859-1.
    socket.writeMultiByte(string, "iso-8859-1");
}

// Данные, переданные через сокет, читаются  в новый массив ByteArray
private function readSocketData(progressEvent:ProgressEvent):void {

    trace(progressEvent.bytesLoaded);
    var ba:ByteArray = new ByteArray();
    trace(socket.readBytes(bs));
}
```

В предыдущем методе `sendSocketData` метод `writeMultiByte` отправляет данные через сокет. Метод получает строковое значение, которое передается в виде блока двоичных данных, в кодировке, определяемой вто-

рым параметром. Метод `readSocketData` читает данные, отправленные через сокет, и заполняет байтами данных новый объект `ByteArray`. Для чтения данных из `ByteArray` следует использовать специализированные методы чтения целых чисел, строк и массивов. Объекты, переданные через `Socket` в двоичном виде, могут читаться методом `readObject`, при условии, что класс был зарегистрирован методом `flash.net.RegisterClass`.

Чтобы связать `Socket` с номером порта, меньшим 1024, необходимо разместить в корневом каталоге сайта файл cross-domain.xml с явным разрешением доступа к этому порту. Например, чтобы Flash Player мог взаимодействовать с веб-сервером через порт 80, этот файл должен выглядеть так:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="80" />
</cross-domain-policy>
```

При наличии правильного файла cross-domain.xml объект `Socket` сможет взаимодействовать с сервером через нужный порт.

16.14. Взаимодействие с использованием XMLSocket

Задача

Требуется создать подключение к серверу, который принимает данные XML, не запрашивая их.

Решение

Используйте класс `XMLSocket` для открытия подключения к серверу. Сервер сможет отправить через это подключение информацию, которая будет принята и обработана клиентом.

Обсуждение

Класс `XMLSocket` реализует клиентские сокеты, через которые Flash Player или приложение AIR взаимодействует с серверным компьютером, идентифицируемым IP-адресом или доменным именем. Для этого на серверном компьютере должен быть запущен демон с поддержкой протокола, используемого классом `XMLSocket`. Что касается протокола:

- Сообщения XML передаются через дуплексное потоковое сокетное подключение TCP/IP.
- Каждое сообщение XML представляет собой законченный документ XML, завершенный нуль-байтом.

Количество сообщений XML, передаваемых через одно подключение `XMLSocket`, не ограничено. Чтобы подключиться к объекту `XMLSocket`, создайте

новый объект XMLSocket и вызовите метод connect с указанием IP-адреса или доменного имени и номера порта:

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Номер порта обязателен, так как XMLSocket не работает с номерами портов меньше 1024. Чтобы принять данные от XMLSocket, добавьте слушателя события для события DataEvent.DATA:

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + XML(event.data));
}
```

Возвращаемая строка преобразуется в XML и разбирается средствами E4X.

17

Взаимодействие с броузером

Довольно часто в приложениях возникает необходимость во взаимодействии с броузером, содержащим приложение. Это позволяет разработчику создавать приложения, выходящие за рамки собственно приложения Flex; вы можете подключаться к существующим сайтам, обмениваться данными с другими приложениями через JavaScript, работать с журналом посещений броузера – и это лишь начало. Класс `ExternalInterface` позволяет обратиться с вызовом к броузеру, содержащему приложение Flash, получить информацию о странице и вызывать методы JavaScript, а также разрешить методам JavaScript обращаться с вызовами к приложению Flash. В этой главе основное внимание уделяется функциональности, содержащейся в базовом варианте Flex Framework, хотя существуют и другие инструменты, содействующие интеграции Flash Player с броузером, в том числе Adobe Flex Ajax Bridge (FABridge) и UrlKit Джо Берковица (Joe Berkowitz).

17.1. Подключение к внешнему URL-адресу

Задача

Требуется направить приложение по внешнему URL-адресу.

Решение

Воспользуйтесь методом `navigateToURL` для направления броузера по новому URL-адресу.

Обсуждение

Функция `navigateToURL` позволяет открыть в броузере новый URL-адрес в том же окне, в новом окне или в конкретном фрейме. Это один из самых

распространенных способов взаимодействия с броузером в приложениях Flex. Вызов функции navigateToURL в приложении Flex 3 выглядит примерно так:

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

    <mx:Script>
        <![CDATA[
            import flash.net.navigateToURL;
            private function goToURL() : void
            {
                navigateToURL( new URLRequest(
                newUrl.text ), target.selectedItem as String );
            }
        ]]>
    </mx:Script>

    <mx:TextInput
        id="newUrl"
        top="10" left="10" right="10"
        text="http://www.oreilly.com/" />
    <mx:ComboBox
        id="target"
        top="40" left="10"
        dataProvider="{ [ '_blank', '_self' ] }" />
    <mx:Button
        label="Go"
        left="10" top="70"
        click="goToURL()" />

</mx:Application>
```

Пользователь вводит любой URL-адрес и щелкает на кнопке Go. В первом параметре метода navigateToURL передается объект URLRequest для открываемого URL-адреса. Второй параметр определяет способ открытия URL: имя конкретного окна или фрейма или одно из специальных значений: _blank для нового окна, _self для текущей страницы, _top для контейнера фрейма верхнего уровня, _parent для родителя контейнера текущего фрейма.

17.2. Работа с FlashVars

Задача

Требуется передать данные из страницы HTML в приложение Flex 3.

Решение

Используйте FlashVars для включения параметров прямо в тег HTML <embed>, содержащий SWF-файл Flex 3.

Обсуждение

Внедрите данные непосредственно в код HTML, содержащий ваше приложение Flex 3; вы сможете легко прочитать их во время выполнения из переменных `FlashVars`. Существует два способа получения этих данных в приложении Flex.

Можно изменить код JavaScript, используемый для внедрения приложения Flex в страницу HTML, как это сделано в следующем примере. Обратите внимание на последнюю строку фрагмента: в ней заданы четыре переменные, используемые для передачи данных в приложение Flex через параметр `FlashVars`:

```
AC_FL_RunContent(
    "src", "${swf}",
    "width", "${width}",
    "height", "${height}",
    "align", "middle",
    "id", "${application}",
    "quality", "high",
    "bgcolor", "${bgcolor}",
    "name", "${application}",
    "allowScriptAccess","sameDomain",
    "type", "application/x-shockwave-flash",
    "pluginspage", "http://www.adobe.com/go/getflashplayer",
    "FlashVars", "param1=one&param2=2&param3=3&param4=four"
);
```

Также можно напрямую модифицировать теги HTML `<object>` и `<embed>`, если для внедрения откомпилированного SWF-файла Flex 3 не используется JavaScript:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="${application}" width="${width}" height="${height}"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/
swflash.cab">
    <param name="movie" value="${swf}.swf" />
    <param name="quality" value="high" />
    <param name="bgcolor" value="${bgcolor}" />
    <param name="allowScriptAccess" value="sameDomain" />
    <param name="FlashVars"
        value="param1=one&param2=2&param3=3&param4=four" />
    <embed src="${swf}.swf" quality="high"
        bgcolor="${bgcolor}"
        width="${width}" height="${height}"
        name="${application}" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer"
```

```
        FlashVars="param1=one&param2=2&param3=3&param4=four"
    </embed>
</object>
```

Приложение Flex может в любой момент обратиться к FlashVars через объект Application.application.parameters. Следующий пример ActionScript демонстрирует обращение к каждому из четырех параметров FlashVars в строковом виде и отображение их в текстовом поле TestArea:

```
private function onCreationComplete() : void
{
    var parameters : Object = Application.application.parameters;
    var param1 : String = parameters.param1;
    var param2 : int = parseInt( parameters.param2 );
    var param3 : int = parseInt( parameters.param3 );
    var param4 : String = parameters.param4;
    output.text = "param1: " + param1 + "\n" +
                  "param2: " + param2 + "\n" +
                  "param3: " + param3 + "\n" +
                  "param4: " + param4;
}
```

17.3. Вызов функций JavaScript из Flex

Задача

Требуется вызвать функции JavaScript из Flex.

Решение

Используйте класс ExternalInterface для вызова функций JavaScript из ActionScript.

Обсуждение

Класс ActionScript ExternalInterface инкапсулирует все, что необходимо для взаимодействия с JavaScript во время выполнения. Вам остается лишь вызвать метод ExternalInterface.call для выполнения функции JavaScript в странице HTML, содержащей приложение Flex.

Вызов тривиальной (не имеющей параметров) функции JavaScript в ActionScript выглядит примерно так:

```
ExternalInterface.call( "simpleJSFunction" );
```

В следующем фрагменте показана вызываемая функция JavaScript. Имя функции JavaScript передается методу call в строковом виде; над приложением Flex появляется окно JavaScript Alert:

```
function simpleJSFunction()
{
    alert("myJavaScriptFunction invoked");
}
```

Аналогичный способ используется для передачи параметров функции из ActionScript в JavaScript. Следующая команда вызывает функцию JavaScript с передачей параметров:

```
ExternalInterface.call( "simpleJSFunctionWithParameters", "myParameter" );
```

Подобным образом из ActionScript в JavaScript можно передать несколько параметров (как простых значений, так и сложных объектов).

В коде JavaScript такие функции ничем не отличаются от других функций с параметрами. Например, функция из следующего фрагмента выводит значение параметра `myParameter` в окне `Alert` над приложением Flex:

```
function simpleJSFunctionWithParameters( parameter )
{
    alert( parameter);
}
```

Часто вызываемая функция JavaScript должна возвращать значение в приложение Flex. Для этого используется конструкция следующего вида:

```
var result:String = ExternalInterface.call(
    "simpleJSFunctionWithReturn" );
```

Строка, возвращаемая функцией JavaScript, сохраняется в экземпляре строки `result` в классе ActionScript:

```
function simpleJSFunctionWithReturn()
{
    return "this is a sample return value: " + Math.random();
}
```

17.4. Вызов функций ActionScript из JavaScript

Задача

Требуется вызвать функцию ActionScript из JavaScript в коде HTML, содержащем приложение Flex.

Решение

Используйте класс `ExternalInterface` для организации обратных вызовов из JavaScript в Flex и активизации функций ActionScript из JavaScript.

Обсуждение

Класс ActionScript `ExternalInterface` не только содержит все необходимое для взаимодействия с JavaScript во время выполнения, но и включает поддержку вызова функций ActionScript из JavaScript.

Прежде чем вызывать функции ActionScript из JavaScript, необходимо зарегистрировать функции ActionScript, доступ к которым будет предоставлен коду JavaScript. Регистрация в ActionScript осуществляется методом `addCallback` класса `ExternalInterface`; ее задачей является установление связи между вызовами функций JavaScript и существующими функциями ActionScript.

В следующем примере регистрируются обратные вызовы трех функций ActionScript:

```
private function registerCallbacks() : void
{
    ExternalInterface.addCallback( "function1", callback1 );
    ExternalInterface.addCallback( "function2", callback2 );
    ExternalInterface.addCallback( "function3", callback3 );
}
```

Соответствующие функции ActionScript выглядят так:

```
private function callback1() : void
{
    Alert.show( "callback1 executed" );
}
private function callback2( parameter : * ) : void
{
    Alert.show( "callback2 executed: " + parameter.toString() );
}
private function callback3() : Number
{
    return Math.random()
}
```

`calllback1` – простая функция ActionScript, которая не получает параметров и не возвращает значения. Функция `callback2` получает один параметр, а функция `callback3` возвращает случайное число.

Чтобы вызвать эти функции из JavaScript, необходимо вызвать функцию JavaScript с псевдонимом JavaScript. Следующий фрагмент кода JavaScript показывает, как вызываются зарегистрированные функции:

```
function invokeFlexFunctions()
{
    var swf = "mySwf";
    var container;
    if (navigator.appName.indexOf("Microsoft") >= 0)
    {
        container = document;
    } else {
        container = window;
    }
    container[swf].function1();
    container[swf].function2( "myParameter" );
```

```
        var result = container[swf].function3();
        alert( result );
    }
```

Переменная `swf` содержит имя приложения Flex в том виде, в котором оно было внедрено в страницу HTML (`mySwf` в данном случае). Работа сценария начинается с получения ссылки на модель DOM в зависимости от типа броузера. Получив правильную ссылку, сценарий вызывает функции Flex по псевдонимам обратного вызова, заданным при регистрации.

Функция ActionScript `callback1` вызывается простым обращением по имени `function1` для экземпляра приложения Flex из JavaScript DOM:

```
container[swf].function1();
```

После вызова этой функции над приложением Flex появляется окно сообщения `Alert`.

Функция ActionScript `callback2` вызывается по имени `function2`, но с передачей значения:

```
container[swf].function2( "myParameter" );
```

Над приложением Flex появляется окно `Alert` со значением параметра, переданным при вызове из JavaScript.

Последний пример показывает, как вернуть значение из Flex в JavaScript. Псевдоним обратного вызова `function3` вызывает функцию ActionScript `callback3`. Эта функция возвращает JavaScript случайное число, которое отображается в окне `Alert`. Пример:

```
var result = container[swf].function3();
alert( result );
```

17.5. Изменение заголовка страницы HTML

Задача

Требуется изменить название страницы HTML для приложения Flex 3.

Решение

Используйте метод `setTitle` класса `BrowserManager`.

Обсуждение

Класс `BrowserManager` в Flex 3 упрощает операции с моделью DOM и страницей HTML, содержащей приложение Flex. В частности, он позволяет сменить заголовок страницы HTML. Пример:

```
private function changePageTitle( newTitle : String ) : void
{
    // Получение экземпляра BrowserManager
    var bm : IBrowserManager = BrowserManager.getInstance();
```

```
        // Инициализация BrowserManager
        bm.init();
        // Назначение заголовка страницы
        bm.setTitle( newTitle );
    }
```

17.6. Разбор URL-адреса с использованием BrowserManager

Задача

Требуется прочитать и разобрать текущий URL-адрес броузера.

Решение

Используйте классы `BrowserManager` и `URLUtil`.

Обсуждение

Следующий пример демонстрирует чтение и разбор URL-адреса текущей страницы с использованием классов `BrowserManager` и `URLUtil`, а также записи разобранных результатов в экземпляр `mx:TextArea`.

Функции класса `URLUtil` упрощают разбор различных частей текущего URL-адреса. При использовании *глубоких ссылок* (deep links) в Flex 3 URL-адрес разбивается на две части: базу и фрагмент. *Базой* (base) называется вся часть URL-адреса, расположенная слева от знака #, а *фрагментом* (fragment) – все, что находится справа от него. Фрагмент используется для передачи данных приложению Flex, а также используется в управлении журналом. Правильно сконструированный фрагмент преобразуется методом `URLUtil.stringToObject` в объект ActionScript со значениями из фрагмента, разбитыми на строки. Пары «имя-значение» в фрагменте URL-адреса должны быть разделены символом «;»:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="parseURL()">

    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectUtil;
            import mx.managers.IBrowserManager;
            import mx.managers.BrowserManager;
            import mx.utils.URLUtil;

            private function parseURL() : void
            {
                // Получение экземпляра BrowserManager
                var bm:IBrowserManager = BrowserManager.getInstance();
```

```
                // Инициализация BrowserManager
                bm.init();
                // Вывод параметров url
                output.text += "Full URL:\n" + bm.url + "\n\n";
                output.text += "Base URL:\n" + bm.base + "\n\n";
                output.text += "URL Fragment:\n" + bm.fragment + "\n\n";
                // Преобразование параметров url
                // в объекты ActionScript средствами URLUtil
                var o:Object = URLUtil.stringToObject(bm.fragment);
                output.text += "Object:\n" +
                    ObjectUtil.toString( o ) + "\n\n";
                output.text += "name:\n" + o.name + "\n\n";
                output.text += "index:\n" + o.index + "\n\n";
                output.text += "productId:\n" + o.productId + "\n\n";

                // Разбор URL-адреса средствами URLUtil
                output.text += "URL Port:\n" +
                    URLUtil.getPort( bm.url ) + "\n\n";
                output.text += "URL Protocol:\n" +
                    URLUtil.getProtocol( bm.url ) + "\n \n";
                output.text += "URL Server:\n" +
                    URLUtil.getServerName( bm.url ) + "\n\n";
                output.text += "URL Server with Port:\n" +
                    URLUtil.getServerNameWithPort( bm.url );
            }

        ]]>
    </mx:Script>

    <mx:TextArea id="output" left="10" top="10" bottom="10"
        right="10"/>

</mx:Application>
```

Если бы в предыдущем примере использовался URL-адрес *http://localhost:8501/flex3cookbook/main.html#name=Andrew;index=12345;productId=987*, то результат его выполнения выглядел бы так:

```
Full URL:
http://localhost:8501/flex3cookbook/main.html#name=Andrew;index=12345;
    productId=987
Base URL:
http://localhost:8501/flex3cookbook/main.html
URL Fragment:
name=Andrew%20Trice;index=12345;productId=987654
Object:
(Object)#0
  index = 12345
  name = "Andrew"
  productId = 987
name:
Andrew
```

```
index:
12345
productId:
987
URL Port:
8501
URL Protocol:
http
URL Server:
localhost
URL Server with Port:
localhost:8501
```

17.7. Глубокие ссылки на данные

Задача

Требуется передать данные из URL-адреса броузера в компоненты Flex, а также обновить URL-адрес на основании данных из приложения Flex. Решение должно работать и при использовании навигационных кнопок броузера Forward/Back.

Решение

Используйте классы `BrowserManager` и `BrowserChangeEvent` для операций чтения и записи данных с URL-адресом броузера.

Обсуждение

Каждый раз, когда URL-адрес броузера изменяется (вследствие ввода текста в адресной строке или использования навигационных кнопок Forward/Back), через экземпляр `BrowserManager` передается событие `BrowserChangeEvent.BROWSER_URL_CHANGE`. Каждый раз, когда в приложении встретится этот тип события, вы просто вызываете метод `updateValues` для обновления значений в компонентах Flex.

Следующий пример демонстрирует чтение данных из URL-адреса броузера и сохранение значений в компонентах Flex `mx:TextInput`. При загрузке пример читает данные из текущего URL-адреса и записывает значения параметров `firstName` и `lastName` в текстовые поля. При изменении содержимого полей `mx:TextInput` для значений `firstName` или `lastName` приложение вызывает функцию `setFragment` экземпляра `BroeserManager`, что приводит к обновлению URL-адреса броузера новыми значениями параметров `lastName` и `firstName`. Обновленный URL-адрес может использоваться в операциях копирования и вставки, что позволяет создать ссылку непосредственно на текущее состояние броузера, а все изменения фиксируются в журнале броузера.

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

```
layout="absolute"
creationComplete="onCreationComplete()" >

<mx:Script>
    <![CDATA[
        import mx.events.BrowserChangeEvent;
        import mx.managers.IBrowserManager;
        import mx.managers.BrowserManager;
        import mx.utils.URLUtil;

        private var bm:IBrowserManager

        private function onCreationComplete():void
        {
            // Получение экземпляра BrowserManager
            bm = BrowserManager.getInstance();

            // Инициализация BrowserManager
            bm.init();

            // Задание начальных значений на основании параметров url
            updateValues();

            // Добавление слушателей событий для обработки
            // кнопок броузера Forward/Back
            bm.addEventListener(BrowserChangeEvent.BROWSER_URL_CHANGE,
                onURLChange );
        }

        private function updateValues():void
        {
            // Обновление текстовых полей по данным фрагмента url
            var o:Object = URLUtil.stringToObject(bm.fragment);
            firstName.text = o.firstName;
            lastName.text = o.lastName;
        }

        private function updateURL():void
        {
            // Обновление фрагмента URL
            bm.setFragment( "firstName=" + firstName.text +
                ";lastName=" + lastName.text );
        }

        private function onURLChange( event : BrowserChangeEvent ):void
        {
            // Вызов функции обновления при изменении url
            updateValues();
        }
    ]]>
</mx:Script>
```

```
    <mx:TextInput x="10" y="10" id="firstName" change="updateURL()" />
    <mx:TextInput x="10" y="40" id="lastName" change="updateURL()" />

</mx:Application>
```

17.8. Управление контейнерами через BrowserManager

Задача

Требуется управлять видимым содержимым контейнеров Flex 3 на основании параметров из URL-адреса.

Решение

Используйте классы `BrowserManager` и `BrowserChangeEvents` для управления видимостью и историей компонентов Flex.

Обсуждение

В этом сценарии фрагмент URL-адреса используется для управления видимостью контейнеров и компонентов в приложении Flex. При загрузке приложения инициализируется экземпляр класса `BrowserManager`, который помогает в разборе и обработке URL-адреса. Метод `updateContainers` (приведенный в следующем фрагменте кода) определяет, какая из вкладок экземпляра `TabNavigator` является видимой. При любом изменении видимой вкладки `TabNavigator` свойство `selectedIndex` в фрагменте URL-адреса изменяется следующей командой:

```
bm.setFragment( "selectedIndex=" + tabNav.selectedIndex );
```

Команда обновляет URL-адрес в броузере и включает изменение в журнал. Скопировав текущий URL-адрес, пользователь получит прямую ссылку на текущую вкладку `TabNavigator`.

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="onCreationComplete()">

    <mx:Script>
        <![CDATA[
            import mx.events.BrowserChangeEvent;
            import mx.managers.IBrowserManager;
            import mx.managers.BrowserManager;
            import mx.utils.URLUtil;

            private var bm:IBrowserManager;

            private function onCreationComplete() : void
            {
```

```
            // Получение экземпляра BrowserManager
            bm = BrowserManager.getInstance();

            // Инициализация BrowserManager
            bm.init();

            // Обновление контейнеров на основании параметров url
            updateContainers();

            // Добавление слушателей событий для обработки
            // кнопок броузера Forward/Back
            bm.addEventListener(BrowserChangeEvent.BROWSER_URL_CHANGE,
                onURLChange );

            updateURL():
        }

        private function updateContainers():void
        {
            // Преобразование параметров url в объект actionscript
            var o:Object = URLUtil.stringToObject(bm.fragment);

            // Выбор выделенной вкладки
            if ( !isNaN(o.selectedIndex) )
            {
                var newIndex : Number = o.selectedIndex;
                if ( newIndex >= 0 && newIndex < tabNav.numChildren )
                    tabNav.selectedIndex = newIndex;
            }
        }

        private function onURLChange(event:BrowserChangeEvent ):void
        {
            // Вызвать updateContainers при изменении url
            updateContainers();
        }

        private function updateURL():void
        {
            bm.setFragment( "selectedIndex=" + tabNav.selectedIndex );
        }

    ]]>
</mx:Script>

<mx:TabNavigator
    bottom="10" top="10" right="10" left="10"
    id="tabNav"
    historyManagementEnabled="false">

    <mx:Canvas label="Tab 0" show="updateURL()" >
```

```
            <mx:Label text="Tab 0 Contents" />
        </mx:Canvas>

        <mx:Canvas label="Tab 1" show="updateURL()" >
            <mx:Label text="Tab 1 Contents" />
        </mx:Canvas>

        <mx:Canvas label="Tab 2" show="updateURL()" >
            <mx:Label text="Tab 2 Contents" />
        </mx:Canvas>

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

Возможно, вы обратили внимание на то, что параметру `historyManagementEnabled` компонента `TabNavigator` задано значение `false`. Дело в том, что для идентификации изменений URL-адреса в броузере и соответствующего обновления вкладок мы используем события класса `BrowserManager`. Каждая смена видимой вкладки фиксируется в журнале броузера; это позволяет перебирать видимые вкладки навигационными кнопками Forward/Back.

17.9. Реализация нестандартного управления журналом броузера

Задача

Операции с пользовательскими компонентами должны регистрироваться в журнале броузера, чтобы переход к ним мог осуществляться кнопками Forward/Back.

Решение

Организуйте нестандартное управление журналом броузера для своих компонентов Flex; для этого следует реализовать интерфейс `mx.managers.IHistoryManagerClient`.

Обсуждение

Приводимое далее решение работает только в том случае, если для проекта Flex включено управление журналом. Откройте диалоговое окно свойств проекта Flex, откройте секцию Flex Compiler и убедитесь в том, что в ней установлен флажок Enable Integration with Browser.

Далее представлен пример реализации интерфейса `IHistoryManagerClient` для пользовательского текстового поля. Все изменения в компоненте регистрируются в журнале броузера. Пользователи могут перемещаться между разными состояниями ввода в компоненте `TextInput` при помощи кнопок броузера Forward/Back.

```
<mx:TextInput
     xmlns:mx="http://www.adobe.com/2006/mxml"
     text="Change Me!"
     implements="mx.managers.IHistoryManagerClient"
     creationComplete="mx.managers.HistoryManager.register(this);"
     change="textChanged(event)">

     <mx:Script>
         <![CDATA[

            import mx.managers.HistoryManager;

            public function saveState():Object
            {
                return {text:text};
            }

            public function loadState(state:Object):void
            {
                var newState:String = state ? state.text : "";
                if (newState != text)
                {
                    text = unescape( newState );
                }
            }
            private function textChanged(e:Event):void
            {
                HistoryManager.save();
            }
         ]]>
     </mx:Script>

</mx:TextInput>
```

Созданный экземпляр класса компонента необходимо зарегистрировать в `HistoryManager`. Это происходит в обработчике события `creationComplete` компонента:

```
creationComplete="mx.managers.HistoryManager.register(this);"
```

Интерфейс `IHistoryManagerClient` требует, чтобы в пользовательском компоненте присуствовали функции `saveState` и `loadState`.

При каждом изменении пользовательского компонента `TextInput` вызывается метод `textChange`, что приводит к вызову функции сохранения состояния `HistoryManager`. При сохранении состояния `HistoryManager` вызывает метод `saveState`.

Метод `saveState` должен вернуть объект, который должен быть сохранен в журнале броузера. В данном случае метод возвращает объект со свойством `text`, которому задается значение свойства `text` компонента `TextInput`.

При изменении состояния броузера кнопками Forward и Back вызывается метод `loadState`. Он читает свойство `text` из переданного ему объекта `State`, после чего задает свойство `text` компонента `TextInput` на основании полученного значения.

Код включения этого компонента в приложение Flex выглядит примерно так:

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

    <local:MyTextInput />

</mx:Application>
```

18

Модули и общие библиотеки

Каждому, кто занимается построением RIA-приложений, рано или поздно приходится заняться проблемами размера приложения и времени загрузки. Flex Framework предлагает несколько вариантов разделения кода приложений по файлам SWF для удобства пользователя.

Общие библиотеки времени выполнения (RSL, Runtime Shared Librares) представляют собой файлы, загружаемые и кэшируемые на стороне клиента. После того как библиотека будет загружена и сохранена на клиентском компьютере, ее ресурсы могут использоваться несколькими приложениями. Приложения могут загружать два типа RSL-библиотек: неподписанные и подписанные. *Неподписанные библиотеки* (стандартные и междоменные SWF-файлы) хранятся в кэше броузера. *Подписанные библиотеки* проходят сертификацию Adobe, имеют расширение .swz и хранятся в кэше Flash Player.

Как следует из самого названия, RSL-библиотеки загружаются во время выполнения и относятся к категории библиотек с динамической компоновкой. Библиотеки со статической компоновкой представляют собой файлы SWC, которые компилируются в приложение при помощи параметров компилятора `library-path` и `include-libraries`. Файлы приложений SWF, использующих библиотеки со статической компоновкой, обычно имеют большие размеры и дольше загружаются; с другой стороны, они быстрее запускаются, поскольку весь код напрямую доступен для приложения. Приложения с использованием RSL быстрее загружаются, а их файлы более компактны, но их запуск занимает больше времени из-за необходимости загрузки RSL и увеличения затрат памяти. Преимущества RSL становятся очевидными при использовании одной кодовой базы несколькими приложениями. Так как RSL-библиотека загружается только один раз, другие приложения, динамически скомпонованные с этой библиотекой, могут работать с ресурсами, уже находящимися

в кэше клиента. Хотя кэширование на стороне клиента является огромным преимуществом, приложение всегда загружает всю библиотеку полностью, независимо от того, какие из ее классов реально используются в приложении.

Модули, как и RSL-библиотеки, помогают разделить код приложения на несколько файлов SWF с целью уменьшения времени загрузки и размера файла. Одно из преимуществ модулей заключается в том, что в отличие от RSL главный файл приложения не обязан загружать модули при запуске. Приложение может загружать и выгружать модули по мере надобности. Процесс модульной разработки тоже обладает своими преимуществами: работа над модулями может вестись отдельно от приложения, потому что модули существуют независимо от него. Если в модуль потребуется внести какие-либо изменения, достаточно перекомпилировать один модуль вместо всего приложения.

Модульные приложения создаются с использованием как ActionScript, так и MXML. Модули на базе Flex используют корневой тег `<mx:Module>`, а модули на базе ActionScript расширяют `mx.modules.Module` или `mx.modules.ModuleBase`. Классы модулей напоминают классы приложений. Модуль компилируется программой mxmlc (компилятор MXML); в результате компиляции создается файл SWF, динамически загружаемый и выгружаемый приложением во время выполнения. Для управления загрузкой и выгрузкой модулей в Flex используется тег `<mx:ModuleLoader>`, а в ActionScript – классы `mx.modules.ModuleLoader` и `mx.modules.ModuleManager`.

Возможность построения модульных приложений является одним из замечательных особенностей Flex; она открывает дополнительные возможности управления временем загрузки и размером файла приложения. Используя RSL-библиотеки и модули, вы можете выделить из приложения код, который будет загружаться и использоваться другими приложениями. У каждого способа есть свои преимущества. В этой главе вы узнаете, как эти способы используются в процессе разработки и развертывания.

18.1. Создание RSL-библиотеки

Задача

Требуется создать *общую библиотеку времени выполнения* (RSL), которая загружается и кэшируется приложением (или несколькими приложениями в границах домена).

Решение

Создайте библиотеку пользовательских классов, компонентов и других ресурсов, откомпилируйте ее в файл SWC. Затем извлеките из SWC файл library.swf и поместите его в каталог развертывания вашего приложения.

Обсуждение

Файловый формат SWC представляет собой архив, содержащий файл library.swf и файл catalog.xml. Первый файл содержит набор ресурсов, откомпилированных в формат SWF, а второй – описание иерархии зависимостей в библиотеке. Чтобы использовать библиотеку как RSL, необходимо извлечь файл library.swf из сгенерированного файла SWC и включить его в каталог развертывания приложения.

Хотя библиотека должна принадлежать одному домену с приложениями, которые будут обращаться к ней во время выполнения, присутствие библиотечного файла SWF во время компиляции не обязательно. Однако файл SWC должен быть доступен во время компиляции, поскольку он используется для динамической компоновки.

В следующем примере представлен компонент MXML, который будет упакован в архивный файл SWC. Класс включается в сгенерированный библиотечный SWF-файл, а его экземпляр будет добавлен в список отображения главного приложения:

```
<mx:Canvas
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="300" height="200">

    <mx:Metadata>
        [Event(name="submit", type="flash.events.Event")]
    </mx:Metadata>

    <mx:Script>
        <![CDATA[

            public static const SUBMIT:String = "submit";
            private function clickHandler():void
            {
                dispatchEvent( new Event(
                CustomEntryForm.SUBMIT ) );
            }
            public function get firstName():String
            {
                return firstNameField.text;
            }
            public function get lastName():String
            {
                return lastNameField.text;
            }
        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="First Name:">
            <mx:TextInput id="firstNameField" />
        </mx:FormItem>
```

```
        <mx:FormItem label="Last Name:">
            <mx:TextInput id="lastNameField" />
        </mx:FormItem>
        <mx:Button label="submit" click="clickHandler();" />
    </mx:Form>

</mx:Canvas>
```

Этот простой компонент дает возможность пользователю ввести информацию и передает событие `submit`. Чтобы упаковать класс в файл SWC, запустите программу compc с параметрами командной строки `source-path` и `include-classes`. Следующая команда строит файл SWC с именем CustomLibrary.swc (предполагается, что путь к каталогу /bin установки Flex SDK задан в системной переменной PATH):

```
> compc -source-path . -include-classes com.oreilly.flexcookbook.CustomEn-
tryForm -output CustomLibrary.swc
```

Компонент MXML хранится в файле CustomEntryForm.mxml подкаталога com/oreilly/flexcookbook текущего каталога разработки; в качестве значения `source-path` передается текущий каталог (точка). При вызове compc можно передать любое количество полных имен классов, разделенных пробелами.

Файл library.swf, архивированный в созданном файле SWC, будет использоваться в качестве RSL-библиотеки. Файл библиотеки извлекается из архива SWC любой стандартной программой распаковки. При компиляции приложения файл SWC используется для динамической компоновки. Вы можете переименовать библиотеку так, как считаете нужным; новое имя файла указывается в качестве имени RSL-библиотеки при построении приложения. В нашем примере извлекаемой из SWC библиотеке присваивается имя CustomLibrary.swf.

В следующем примере приложение использует компонент CustomEntryForm из загруженной RSL-библиотеки:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:flexcookbook="com.oreilly.flexcookbook.*"
    layout="vertical">

    <mx:Script>
        <![CDATA[

            private function onFormSubmit():void
            {
                greetingField.text = "Hello " +
                    entryForm.firstName +
                    " " + entryForm.lastName;
            }

        ]]>
```

```
        </mx:Script>

        <mx:Panel title="Enter Name:" width="400" height="400">
            <flexcookbook:CustomEntryForm id="entryForm"
                submit="onFormSubmit()" />
            <mx:HRule width="100%" />
            <mx:Label id="greetingField" />
        </mx:Panel>

    </mx:Application>
```

Приложение использует ресурсы из классов RSL и объявляет компоненты MXML так, как если бы оно использовало библиотеки со статической компоновкой и локальные классы из каталога разработки. В приведенном примере пространство имен `flexcookbook` объявляется в теге `<mx:Application>` и используется для включения компонента `CustomEntryForm` в список отображения.

Чтобы откомпилировать приложение для использования ранее созданной и RSL-библиотеки CustomLibrary.swf, запустите программу mxmlc с параметрами командной строки `external-library` и `runtime-shared-libraries`:

```
> mxmlc -external-library=
    CustomLibrary.swc -runtime-shared-libraries=CustomLibrary.swf
 RSLExample.mxml
```

В этой команде файл CustomLibrary.swc используется для проверки связей при компиляции, а URL-адрес RSL-библиотеки объявляется как предоставляемый в одном домене со сгенерированным SWF-файлом приложения. Когда дело доходит до развертывания, файлы RSLExample.swf и CustomLibrary.swf должны находиться на сервере в одном каталоге. При запуске приложение загружает RSL-библиотеку CustomLibrary и может использовать ее код для отображения формы, на которой пользователь вводит информацию.

18.2. Междоменные RSL–библиотеки

Задача

Требуется сохранить RSL-библиотеку на сервере отдельно от приложения, чтобы обеспечить доступ к ней со стороны других приложений, не принадлежащих тому же домену.

Решение

Укажите параметр `compute-digest` программы compc при создании RSL-библиотеки. Затем создайте файл междоменной политики, который будет передаваться вместе с путями RSL-библиотек в параметре `runtime-shared-library-paths` программы.

Обсуждение

Дайджест (digest) RSL-библиотеки представляет собой хеш-код, по которому Flash Player может убедиться в том, что загружаемая RSL-библиотека поступила от проверенного источника.

При создании RSL-библиотеки с параметром `compute-digest=true` дайджест записывается в файл catalog.xml архива SWC. При компоновке междоменной RSL-библиотеки с приложением в процессе компиляции дайджест сохраняется в SWF-файле приложения и используется для проверки аутентичности запрашиваемой RSL-библиотеки.

Следующая команда строит файл SWC с именем CustomLibrary.swc (предполагается, что путь к каталогу /bin установки Flex SDK задан в системной переменной PATH):

```
> compc -source-path . -include-classes com.oreilly.flexcookbook.
CustomEntryForm -output CustomLibrary.swc -compute-digest=true
```

По умолчанию параметр `compute-digest` равен `true` и его не обязательно указывать для создания дайджеста при компиляции библиотеки. Дайджест необходим при подключении к приложению междоменных RSL-библиотек при помощи параметра runtime-shared-library-paths компилятора MXML.

В предыдущем разделе был представлен пример *стандартной* RSL-библиотеки, которая находится в одном домене с приложением и при подключении которой использовался параметр компилятора `runtime-shared-libraries`. Стандартные RSL-библиотеки тоже могут использовать дайджесты, хотя это и не обязательно.

Файл SWC, созданный этой командой, представляет собой архив с файлами library.swf и catalog.xml. Для извлечения этих файлов из архива можно воспользоваться любой стандартной утилитой распаковки. Первый файл содержит набор ресурсов, откомпилированных в формат SWF и используемых в качестве RSL, а второй – описание информации в библиотеке вместе с дайджестом, созданным параметром `compute-digest`. Информация дайджеста RSL-библиотеки хранится в секции `<digests>` файла catalog.xml:

```
<digests>
    <digest type="SHA-256" signed="false"
       value="2630d7061c913b4cea8ef65240fb295b2797bf73a
          0db96ceec5c319e2c00f8a5"   />
</digests>
```

Атрибут `value` содержит хеш-код, сгенерированный компилятором по алгоритму SHA-256.

При компиляции приложения, в котором используется RSL-библиотека, дайджест сохраняется в приложении и используется для проверки RSL-библиотеки, запрашиваемой с сервера.

Наряду с дайджестом, гарантирующим, что RSL-библиотека поступила из доверенного источника, на сервере, на котором размещается библиотека, должен присутствовать файл междоменной политики. Он представляет собой файл в формате XML с перечислением доменов, которым предоставляется доступ к данным на удаленном сервере. Чтобы разрешить доступ к библиотеке приложениям из внешних доменов, перечислите доменные пути в отдельных элементах `<allow-access-from>`. Пример файла crossdomain.xml:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*.mydomain.com" />
    <allow-access-from domain="*.myotherdomain.com" />
</cross-domain-policy>
```

Этот файл разрешает любым SWF-файлам из субдоменов *http://mydomain.com* и *http://myotherdomain.com* обращаться к данным на сервере, включая междоменные RSL-библиотеки.

Чтобы разрешить любым файлам SWF из перечисленных доменов доступ к данным, хранящимся на сервере, разместите файл междоменной политики в корневом каталоге сервера. Такое решение работает, но возможно, вы захотите более точно управлять доступом к RSL-библиотекам на сервере со стороны приложений. Параметр `runtime-shared-library-path` компилятора MXML позволяет задать местонахождение файла междоменной политики со списком доменов, которым разрешен доступ к библиотекам.

Чтобы откомпилировать приложение, в котором используется динамическая компоновка с междоменной RSL-библиотекой, созданной ранее программой compc и переименованной в CustomLibrary.swf, выполните программу mxmlc с параметром командной строки `runtime-shared-library-path` и указанием полных URL-адресов RSL и файла междоменной политики на сервере:

```
> mxmlc RSLExample.mxml -runtime-shared-library-path=
    CustomLibrary.swc,
    http://www.mytargetdomain.com/libraries/CustomLibrary.swf,
    http://www.mytargetdomain.com/libraries/crossdomain.xml
```

В разделенных запятыми аргументах `runtime-shared-library-path` передается местонахождение файла SWC, используемого при компиляции, полный URL-адрес библиотеки на удаленном сервере и полный URL-адрес файла междоменной политики, разрешающего приложениям из других доменов загружать библиотеку.

Существование SWF-файла библиотеки не проверяется во время компиляции; URL-адрес сохраняется в приложении и проверяется во время выполнения.

Наряду с уменьшением размера файла и ускорением загрузки междоменные RSL-библиотеки обладают и другими преимуществами: в част-

ности, они позволяют приложениям из других доменов получить доступ к доверенным данным. Список доменов в файле междоменной политики обновляется по мере развертывания на серверах других приложений, использующих RSL-библиотеку.

См. также

Рецепт 18.1.

18.3. Использование Flex Framework как RSL–библиотеки

Задача

Требуется сократить размер файла и ускорить загрузку приложения за счет компоновки Flex Framework в формате RSL-библиотеки.

Решение

Откомпилируйте приложение с использованием файла framework.swc и подписанной или неподписанной RSL-версии из подкаталога /frameworks установочного каталога Flex 3 SDK.

Обсуждение

Если вы ранее уже занимались построением приложений с использованием Flex Framework, вам, вероятно, известно об увеличении размера SWF-файла приложений, а соответственно и времени их загрузки. Даже для относительно простого приложения (вроде приведенного ниже) SWF-файл, созданный в результате компиляции, оказывается неожиданно большим для объема полезного кода:

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

    <mx:Label text="Hello World" />

</mx:Application>
```

По сути, приложение просто включает в список отображения компонент `Label` для вывода надписи *Hello World*. Но размер SWF-файла, полученного в результате компиляции приложения программой mxmlc, составляет целых 149 килобайт! Дело в том, что в файл приложения также включается код компонентов и библиотек Flex. Впрочем, это понятно: в конце концов, эти ресурсы необходимы для эффективного выполнения автономного SWF-файла в Flash Player. И все же для такого простого приложения, загружаемого броузером, такой размер файла выглядит несколько чрезмерным. Если вспомнить основные преимущества RSL-

библиотек (не только размер файла и время загрузки, но и возможность использования ресурсов одной библиотеки из нескольких приложений), возникает естественное желание откомпилировать приложение с динамической компоновкой инфраструктуры Flex Framework, хранящейся в RSL-библиотеке.

В Flex 3 SDK и Flash Player третьего поколения (Flash Player 9.0.60 и выше) появилась возможность отделения **Flex Framework от кода приложения**. В каталоге /frameworks/libs Flex 3 SDK находится архив Flex Framework: файл framework.swc. В каталоге /frameworks/rsl Flex 3 SDK находится подписанная **RSL-библиотека Flex Framework; на момент написания** книги файл назывался framework_3.0.189825.swz.

Неподписанные RSL-библиотеки (вроде тех, которые создавались в предыдущих рецептах этой главы) хранятся в кэше броузера и могут загружаться всеми приложениями домена RSL-библиотеки (или доменов, перечисленных в файле междоменной политики). Подписанные RSL-библиотеки (с расширением .swz) хранятся в кэше Flash Player. Для удаления подписанных RSL-библиотек используется Settings Manager. Только фирма **Adobe может подписывать RSL-библиотеки; это обеспечивает** защиту от внедрения и выполнения постороннего кода.

Чтобы откомпилировать приложение с файлом framework.swc, включенным в Flex 3 SDK, с динамической компоновкой подписанной RSL-библиотеки Flex Framework, используйте параметр `runtime-shared-library-path` программы mxmlc:

```
> mxmlc HelloWorldApplication.mxml -target-player=9.0.60
        -runtime-shared-library-path=
        /<sdk-installation>/frameworks/framework.swc,
        framework_3.0.1.189825.swz
```

Первый аргумент `runtime-shared-library-path` определяет местонахождение файла framework.swc в подкаталоге /frameworks каталога установки Flex 3 SDK. Второй аргумент, отделенный от первого запятой, содержит имя подписанной RSL-библиотеки Flex Framework. На момент написания книги файл назывался framework_3.0.189825.swz и находился в подкаталоге /rsls установочного каталога Flex 3 SDK.

Так как файл **SWZ находится в одном домене с SWF-файлом приложения**, подписанная RSL-библиотека загружается в кэш Flash Player при первой загрузке приложения. Все остальные приложения, откомпилированные с теми же аргументами `runtime-shared-library-path`, будут обращаться к кэшированному файлу. Это также означает, что все приложения Flex, созданные другим разработчиком с теми же значениями аргументов, смогут воспользоваться тем фактом, что RSL-библиотека уже загружена и сохранена в кэше Flash Player.

Если сравнить сгенерированный файл SWF с 149-килобайтным файлом, полученным до компиляции с **framework.swc, можно заметить, что размер** файла значительно уменьшился. В результате динамической ком-

поновки подписанной RSL-библиотеки Flex Framework он составляет всего 49 Кбайт.

См. также

Рецепт 18.1 и 18.2.

18.4. Оптимизация RSL-библиотеки

Задача

Требуется уменьшить размер файла **RSL-библиотеки, загруженной приложением.**

Решение

Воспользуйтесь программой командной строки optimizer из Flex 3 SDK для удаления отладочного кода и метаданных, включенных в файл library.swf.

Обсуждение

По умолчанию файл library.swf, генерируемый при создании файла SWC, содержит отладочную информацию и метаданные. Эти данные приводят к лишним затратам ресурсов при загрузке приложением RSL-библиотеки с удаленного сервера или из кэша броузера. Чтобы оптимизировать RSL-библиотеку, следует сначала создать архивный файл SWC программой compc из Flex 3 SDK, а затем извлечь файл library любой стандартной программой распаковки данных. Файл **library.swf представ**ляет собой RSL-библиотеку, которая загружается приложением, откомпилированным со сгенерированным файлом SWC. RSL-библиотека обрабатывается программой командной строки optimizer.

Размер сгенерированной библиотеки в архиве SWC изменяется в зависимости от того, какие библиотеки классов были включены во время компиляции. Чтобы вы лучше поняли, какие возможности предоставляет программа optimizer, создайте следующий компонент MXML и сохраните файл под именем MyCustomComponent.mxml:

```
<mx:Canvas
    xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:TextArea text="Lorem ipsum dolor sit amet" />
</mx:Canvas>
```

Этот простой фрагмент отображает компонент `<mx:TextArea>` с текстом. Следующая команда строит файл SWC с именем library.swc в корневом каталоге проекта (предполагается, что путь к каталогу /bin установки Flex SDK задан в системной переменной PATH):

```
> compc -source-path . -include-classes MyCustomComponent
    -output library.swc
```

Извлеките файл library.swf из сгенерированного файла SWC любой стандартной программой распаковки. В приведенном примере размер извлеченной библиотеки составляет приблизительно 320 Кбайт.

Программа командной строки optimizer сокращает размер файла библиотеки за счет удаления всей отладочной информации и метаданных из файла SWF:

```
> optimizer -keep-as3-metadata=
"Bindable,Managed,ChangeEvent,NonCommittingChangeEvent,Transient"
 -input library.swf -output optimized.swf
```

Размер оптимизированной RSL-библиотеки составляет менее половины размера исходной библиотеки – около 135 Кбайт.

В аргументе `keep-as3-metadata` рекомендуется включить как минимум метаданные `Bindable`, `Managed`, `ChangeEvent`, `NonCommittingChangeEvent` и `Transient`, так как эти теги метаданных часто связываются с компонентами Flex Framework. Другие метаданные, такие как `RemoteClass`, могут добавляться в список аргументов на основании зависимостей классов из RSL-библиотеки.

См. также

Рецепт 18.1 и 18.2.

18.5. Создание модуля на базе MXML

Задача

Требуется создать модуль на базе MXML, загружаемый приложением во время выполнения.

Решение

Создайте класс MXML, расширяющий класс `mx.modules.Module`, при помощи корневого тега `<mx:Module>`. Откомпилируйте модуль программой командной строки mxmlc.

Обсуждение

Модуль (*module*), как и приложение, компилируется утилитой mxmlc; в результате компиляции создается файл SWF, который может загружаться приложением или другим модулем во время выполнения. Модули на базе MXML создаются расширением класса `mx.modules.Module`, для чего в файле MXML используется корневой тег `<mx:Module>`.

В следующем примере модуль выводит список контактов в компоненте `DataGrid`:

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%">
```

```
    <mx:XMLList id="contacts">
        <contact>
            <name>Josh Noble</name>
            <phone>555.111.2222</phone>
            <address>227 Jackee Lane</address>
        </contact>
        <contact>
            <name>Todd Anderson</name>
            <phone>555.333.4444</phone>
            <address>1642 Ocean Blvd</address>
        </contact>
        <contact>
            <name>Abey George</name>
            <phone>555.777.8888</phone>
            <address>1984 Winston Road</address>
        </contact>
    </mx:XMLList>

    <mx:DataGrid id="contactGrid"
        width="100%" height="100%"
        rowCount="4"
        dataProvider="{contacts}">
         <mx:columns>
                <mx:DataGridColumn dataField="name"
                    headerText="Name"/>
                <mx:DataGridColumn dataField="phone"
                    headerText="Phone"/>
                <mx:DataGridColumn dataField="address"
                    headerText="Address"/>
            </mx:columns>
    </mx:DataGrid>
</mx:Module>
```

Структура этого модуля близка к структуре приложения или пользовательского компонента. Когда приложению потребуется вывести список контактов, оно может загрузить этот модуль и включить его в список отображения.

Чтобы откомпилировать модуль в файл SWF, загружаемый приложением, воспользуйтесь программой mxmlc и передайте ей имя файла с классом модуля:

```
> mxmlc ContactList.mxml
```

Команда создает файл SWF с именем ContactList.swf, хотя при помощи параметра `output` можно задать другое имя, при условии, что оно будет известно приложению в момент загрузки. Размер сгенерированного файла SWF составляет примерно 245 Кбайт – немало, если учесть, что его содержимое включается в откомпилированное приложение. Выделение этого кода в отдельный модуль уменьшает время загрузки и сокращает размер файла любого приложения, которому понадобится загрузить этот модуль.

18.6. Создание модуля на базе ActionScript

Задача

Требуется создать модуль на базе ActionScript, загружаемый приложением во время выполнения.

Решение

Создайте класс ActionScript, расширяющий класс `mx.modules.Module` или `mx.modules.ModuleBase`. Откомпилируйте модуль программой командной строки mxmlc.

Обсуждение

Модули на базе ActionScript создаются расширением классов `Module` и `ModuleBase`. Выбор между `Module` и `ModuleBase` зависит от роли модуля в приложении и определяется необходимостью использования списка отображения. Класс `Module` представляет визуальный контейнер, расширяющий `FlexSprite` и включающий часть инфраструктурного кода. Класс `ModuleBase` расширяет `EventDispatcher` и может использоваться для выделения из приложения логического кода, не зависящего от визуальных элементов.

Модули на базе MXML расширяют `mx.modules.Module` и используют корневой тег `<mx:Module>`. Если создаваемый вами модуль содержит визуальные элементы, расширьте класс `Module` и переопределите защищенные методы по своему усмотрению, например метод `createChildren`, унаследованный от `UIComponent`. В следующем примере показан модуль, добавляющий компоненты для ввода информации:

```
package
{
    import mx.containers.Form;
    import mx.containers.FormItem;
    import mx.controls.TextInput;
    import mx.modules.Module;

    public class ASContactList extends Module
    {
        private var _form:Form;
        private var _firstNameItem:FormItem;
        private var _lastNameItem:FormItem;
        public function ASContactList()
        {
            super();
            this.percentWidth = 100;
            this.percentHeight = 100;
        }

        override protected function createChildren():void {
```

```
            super.createChildren();
            _form = new Form();
            _firstNameItem = createInputItem( "First Name:" );
            _lastNameItem = createInputItem( "Last Name:" );
            _form.addChild( _firstNameItem );
            _form.addChild( _lastNameItem );
            addChild( _form );
        }

        private function createInputItem(label:String):FormItem
        {
            var item:FormItem = new FormItem();
            item.label = label;
            item.addChild( new TextInput() );
            return item;
        }
    }
}
```

Модули на базе ActionScript компилируются так же, как и модули на базе MXML: программой mxmlc с указанием имени файла ActionScript:

```
> mxmlc ASContactList.as
```

Команда создает файл SWF с именем ASContactList.swf, хотя при помощи параметра `output` можно задать другое имя, при условии, что оно будет известно приложению в момент загрузки. Класс `Module` является расширением `mx.core.Container` и поэтому во внутренней реализации преобразует дочерние компоненты, добавляемые в список отображения, к типу `mx.core.UIComponent`. Добавление компонентов из Flex Framework в список отображения модуля на базе ActionScript осуществляется методом `addChild`. Чтобы добавить компоненты из ActionScript API (например, `flash.text.TextField` и `mx.media.Video`), «заверните» их в экземпляр `UIComponent`.

Класс `Module` содержит код Framework для взаимодействия и отображения объектов пользовательского интерфейса. Если ваш модуль не использует код Framework, можно создать класс, расширяющий `ModuleBase`.

Пример модуля, расширяющего класс `ModuleBase`:

```
package
{
    import mx.modules.ModuleBase;

    public class EntryStateModule extends ModuleBase
    {
        public function EntryStateModule() {}

        public function greet(first:String, last:String ):String
        {
            return "Hello, " + first + " " + last + ".";
```

```
        }
        public function welcomBack(
            first:String, last:String ):String
        {
            return "Nice to see you again, " + first + ".";
        }
    }
}
```

Этот простой модуль предоставляет открытые методы `greet` и `welcomeBack`. Он не содержит кода Framework, и как следствие, файл имеет значительно меньшие размеры по сравнению с модулем, созданным на базе класса `Module`.

Модули, расширяющие класс `ModuleBase`, компилируются так же, как и расширения `Module`:

```
> mxmlc EntryStateModule.as
```

Команда создает файл SWF с именем EntryStateModule.swf. Чтобы воспользоваться открытыми методами модуля из этого примера, родительское приложение или родительский модуль должны обратиться к свойству `child` экземпляра `ModuleLoader` или свойству `factory` реализации `IModuleInfo`, как описано в рецепте 18.8.

См. также

Рецепты 18.5 и 18.8.

18.7. Загрузка модуля с использованием ModuleLoader

Задача

Требуется загрузить модуль в приложение Flex.

Решение

Используйте контейнер `<mx:ModuleLoader>` для загрузки модулей в своем приложении.

Обсуждение

Класс `mx.modules.ModuleLoader` представляет собой контейнер, сходный с `mx.controls.SWFLoader`. Он загружает файлы SWF и добавляет модули в список отображения приложения. `ModuleLoader` отличается от `SWFLoader` тем, что его контракт требует от загружаемого файла SWF реализации интерфейса `IFlexModuleFactory`. В откомпилированные модули включается фабрика `IFlexModuleFactory`, позволяющая приложениям динамически загружать модульные файлы SWF без обязательного включения реализации классов в главный модуль приложения.

Хотя объект ModuleLoader является визуальным контейнером, он может загружать модули, расширяющие Module и ModuleBase, а его работа не требует, чтобы модуль содержал код Framework или отображал визуальные объекты. Свойство url класса ModuleLoader содержит ссылку на местонахождение модуля, выраженное в виде URL-адреса. Задание свойства url приводит к внутреннему вызову открытого метода loadModule класса ModuleLoader и началу процесса загрузки модуля.

В следующем примере загружаемый модуль принадлежит тому же домену, что и приложение:

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

    <mx:Panel title="Contacts:" width="350" height="180"
        horizontalAlign="center" verticalAlign="middle">
        <mx:ModuleLoader url="ContactList.swf" />
    </mx:Panel>

</mx:Application>
```

При запуске приложения компонент ModuleLoader загружает модуль ContactList.swf из того же домена. Успешно загруженный модуль включается в список отображения.

Компонент ModuleLoader также позволяет динамически загружать и выгружать различные модули. Задание свойства url экземпляра ModuleLoader приводит к внутреннему вызову открытого метода loadModule класса ModuleLoader и включению модуля в список отображения. Чтобы удалить модуль из списка отображения, вызовите открытый метод unloadModule класса ModuleLoader. Вызов unloadModule приводит к тому, что ссылка на модуль принимает значение null, но не изменяет значения свойства url.

В следующем приложении модули загружаются и выгружаются в зависимости от действий пользователя:

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

    <mx:Script>
        <![CDATA[

            private function displayModule(moduleUrl:String ):void
            {
                var url:String = moduleLoader.url;
                if( url == moduleUrl ) return;
                if( url != null )  moduleLoader.unloadModule();
                moduleLoader.url = moduleUrl;
            }
```

```
            private function showHandler():void
            {
                displayModule( "ContactList.swf" );
            }
            private function enterHandler():void
            {
                displayModule( "ContactEntry.swf" );
            }
        ]]>
    </mx:Script>

    <mx:Panel title="Contacts:" width="350" height="210"
        horizontalAlign="center" verticalAlign="middle">
        <mx:ModuleLoader id="moduleLoader" height="110" />
        <mx:HRule width="100%" />
        <mx:HBox width="100%">
            <mx:Button label="show list" click="showHandler();" />
            <mx:Button label="enter contact" click="enterHandler();" />
        </mx:HBox>
    </mx:Panel>

</mx:Application>
```

Обработчики события `click` компонентов `Button` обновляют модуль, загружаемый в `ModuleLoader`. Приложение переходит от вывода списка контактных данных к выводу формы для ввода новых контактов и обратно, для чего загружаются модули ContactList.swf и ContactEntry.swf соответственно.

Модуль, загружаемый приложением, включается в список модулей объекта `mx.modules.ModuleManager`. При удалении ссылки на него задается значение `null` для освобождения памяти и ресурсов. Компонент `ModuleLoader` является удобным инструментом для управления загрузкой и выгрузкой модулей в приложениях Flex.

См. также

Рецепты 18.5, 18.6 и 18.8.

18.8. Загрузка модуля с использованием ModuleManager

Задача

Требуется обеспечить более точное управление загрузкой и выгрузкой модулей в приложениях на базе Flex и ActionScript.

Решение

Вызывайте методы класса `ModuleManager` для прослушивания событий состояния загрузки модулей.

Обсуждение

Класс ModuleManager предназначен для управления загруженными модулями. Реализация компонента <mx:ModuleLoader> связывается с ним при вызове открытых методов ModuleLoader.loadModule и ModuleLoader.unloadModule. Вы можете напрямую обращаться к модулям, находящимся под управлением ModuleManager, из кода ActionScript. При передаче URL-адреса модуля открытому методу ModuleManager.getModule последний включает его местонахождение в список модулей и возвращает экземпляр mx.modules.IModuleInfo.

В сущности, модули представляют собой экземпляры приватного класса ModuleInfo из ModuleManager. Объекты ModuleInfo загружают файл SWF и упаковываются в промежуточную реализацию интерфейса IModuleInfo, возвращаемую методом ModuleManager.getModule. Прослушивая события состояний для класса-посредника, вы сможете организовать более точное управление взаимодействием приложения с загруженными модулями.

В следующем примере ModuleManager используется приложением для управления включением модуля в список отображения:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="creationHandler();">
    <mx:Script>
        <![CDATA[
            import mx.events.ModuleEvent;
            import mx.modules.ModuleManager;
            import mx.modules.IModuleInfo;
            private var _moduleInfo:IModuleInfo;
            private function creationHandler():void
            {
                _moduleInfo = ModuleManager.getModule(
                    'ContactList.swf' );
                _moduleInfo.addEventListener(
                    ModuleEvent.READY,
                    moduleLoadHandler );
                _moduleInfo.load();
            }
            private function moduleLoadHandler( evt:ModuleEvent ):void
            {
                canvas.addChild(
                    _moduleInfo.factory.create()
                    as DisplayObject );
            }
        ]]>
    </mx:Script>
    <mx:Canvas id="canvas" width="500" height="500" />
</mx:Application>
```

Когда приложение завершит начальные операции формирования раскладки, поступает команда на загрузку модуля `ContactList` с использованием объекта `IModuleInfo`, возвращенного методом `ModuleManager.getModule`. После успешной загрузки модуль включается в список отображения методом `IFlexModuleFactory.create`. Метод возвращает экземпляр модуля, преобразованный в `DisplayObject`, который включается в список отображения контейнера `Canvas`.

Для прослушивания состояний, связанных с состоянием загрузки модуля, используется объект-посредник, реализующий `IModuleInfo` и возвращаемый методом `getModule`.

В приведенном примере приложение ожидает, пока модуль будет полностью загружен, и только после этого включает его в список отображения. События – от нормального хода загрузки до кодов ошибок, связанных с состоянием загрузки модуля SWF, – передаются в виде экземпляров класса `ModuleEvent` (табл. 18.1).

Таблица 18.1. Класс mx.events.ModuleEvent

Константа	Строковое значение	Описание
`PROGRESS`	`"progress"`	Передается во время загрузки модуля. Обработчик события может обращаться к свойствам `bytesLoaded` и `bytesTotal` загружаемого модуля
`SETUP`	`"setup"`	Передается при наличии достаточной информации о загружаемом модуле
`READY`	`"ready"`	Передается при завершении загрузки модуля
`UNLOAD`	`"unload"`	Передается при завершении выгрузки модуля
`ERROR`	`"error"`	Передается при возникновении ошибки в ходе загрузки модуля

Метод `unload` реализации `IModuleInfo` удаляет ссылку на модуль из `ModuleManager`, но не исключает файл SWF из списка отображения, для чего необходимо явно вызвать метод `removeChild` родительского изображения.

По сравнению с классом `ModuleLoader`, который начинает загрузку модуля по изменению свойства `url`, при использовании реализации `IModuleInfo`, возвращаемой методом `getModule`, вы можете отложить загрузку и последующее отображение модуля в вашем приложении. Это позволяет заранее загрузить модуль, чтобы потом немедленно сделать его доступным для отображения; это сокращает время на запросы и отображение модулей в результате взаимодействия с пользователем.

См. также

Рецепты 18.5–18.7.

18.9. Загрузка модулей с другого сервера

Задача

Требуется организовать хранение модулей на сервере, отличном от того, который используется для развертывания приложений.

Решение

Используйте класс `flash.system.Security` для установления доверенных отношений между SWF-файлами главного приложения и загружаемого модуля.

Обсуждение

Flash Player использует доменную систему безопасности и разрешает файлам SWF из определенного домена обращаться к данным из того же домена без каких-либо ограничений. При загрузке файла SWF в Flash Player для домена создается изолированная среда безопасности («песочница»), а ко всем ресурсам в ней предоставляется неограниченный доступ. Файл SWF также может обращаться к внешним ресурсам и взаимодействовать с другими файлами SWF из доверенного источника.

Чтобы файл SWF мог обращаться к ресурсам из некоторого домена (в том числе и модулям из другого домена), необходимо разместить на удаленном сервере файл междоменной политики и использовать метод `Security.allowDomain` в главном приложении. Чтобы загруженный модуль мог взаимодействовать с родительским файлом SWF, он тоже должен вызвать метод `allowDomain`.

Допустим, на удаленном сервере хранится следующий модуль:

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    initialize="initHandler();">

    <mx:Script>
        <![CDATA[
            private function initHandler():void
            {
                Security.allowDomain( "appserver" );
            }
        ]]>
    </mx:Script>
    <mx:Text width="100%" text="{loaderInfo.url}" />

</mx:Module>
```

Когда родительский файл SWF загружает этот модуль и происходит событие `initialize`, модуль разрешает доступ загружающему файлу SWF и выводит URL-адрес, с которого выполняется загрузка.

При компиляции модуля и его размещении на удаленном сервере (для примера допустим, что домен называется moduleserver) в корневом каталоге домена размещается файл междоменной политики, который разрешает загрузку модуля родительскому файлу SWF, находящемуся на appserver:

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="appserver" to-ports="*" />
</cross-domain-policy>
```

Чтобы родительский файл SWF мог загрузить модуль и установить с ним сценарный обмен данными, вызовите метод Security.allowDomain с передачей доменного имени удаленного сервера и загрузите файл crossdomain.xml:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    preinitialize="initHandler();">

    <mx:Script>
        <![CDATA[
            private function initHandler():void
            {
                Security.allowDomain( "moduleserver" );
                Security.loadPolicyFile(
                    "http://moduleserver/crossdomain.xml" );
                var loader:URLLoader = new URLLoader();
                loader.addEventListener( Event.COMPLETE,
                    loadHandler );
                loader.load(new URLRequest(
                    "http://moduleserver/crossdomain.xml"));
            }
            private function loadHandler( evt:Event ):void
            {
                moduleLoader.url =
                    "http://moduleserver/modules/MyModule.swf";
            }
        ]]>
    </mx:Script>

    <mx:ModuleLoader id="moduleLoader" />
</mx:Application>
```

Обработчик события preinitialize в главном приложении устанавливает связь со всеми ресурсами, загруженными с сервера moduleserver, вызовом метода SecurityDoman.allowDomain. Приложение также вызывает метод Security.loadPolicyFile с указанием местонахождения файла междоменной политики на удаленном сервере. Flash Player читает файл политики и убеждается в том, что файлу SWF приложения с appserver можно доверять. Метод loadPolicyFile должен вызываться до загрузки

файла междоменной политики с использованием экземпляра URLLoader; в противном случае будут выдаваться исключения безопасности.

После того как загрузка файла политики будет завершена, файл SWF приложения задает свойству url экземпляра <mx:ModuleLoader> URL-адрес нужного модуля на удаленном сервере. Приложение и модуль разрешили доступ друг другу с соответствующих серверов, канал связи открыт для дальнейших взаимодействий.

См. также

Рецепты 18.7 и 18.8.

18.10. Обмен данными с модулем

Задача

Требуется организовать передачу данных между родительским файлом SWF и загруженным модулем.

Решение

Используйте свойство child экземпляра mx.modules.ModuleLoader и свойство factory экземпляра mx.modules.IModuleInfo для прослушивания событий, вызова открытых методов и обращения к открытым свойствам родительского файла SWF и загруженного модуля.

Обсуждение

Основное приложение может взаимодействовать с загруженным модулем через свойства ModuleLoader и ModuleManager. Взаимодействие не ограничивается экземпляром <mx:Application>, потому что модули могут загружать другие модули; в результате загрузивший модуль становится родителем по отношению к загруженному модулю с такими же точками доступа, как у приложения.

Чтобы обратиться к данным из загруженного модуля, можно преобразовать возвращаемые свойства из экземпляра загрузчика к классу загруженного модуля. При использовании объекта <mx:ModuleLoader> обращение к экземпляру модуля осуществляется через свойство child:

```
<mx:Script>
    <![CDATA[
        private var myModule:MyModule;
        private function moduleReadyHandler():void
        {
            myModule = moduleLoader.child as MyModule;
            myModule.doSomething();
        }
    ]]>
</mx:Script>
```

```
<mx:ModuleLoader id="moduleLoader"
    url="MyModule.swf"
    ready="moduleReadyHandler();" />
```

Когда данные загруженного модуля становятся доступными для родительского приложения, вызывается обработчик события `moduleReadyHandler`. Преобразование свойства `child` объекта `<mx:ModuleLoader>` к типу класса модуля позволяет обращаться к данным модуля и вызывать открытые методы.

При использовании класса `ModuleManager` в родительском приложении экземпляр модуля возвращается открытым методом `create` экземпляра `IFlexModuleFactory` из реализации `IModuleInfo`:

```
private var _moduleInfo:IModuleInfo;

private function creationHandler():void
{
    _moduleInfo = ModuleManager.getModule( 'MyModule.swf' );
    _moduleInfo.addEventListener( ModuleEvent.READY,
        moduleLoadHandler );
    _moduleInfo.load();
}

private function moduleLoadHandler( evt:ModuleEvent ):void
{
    var myModule:MyModule = _moduleInfo.factory.create() as MyModule;
    myModule.doSomething();
}
```

Преобразование свойства `child` экземпляра `ModuleLoader` или объекта `Object`, возвращаемого методом `IFlexModuleFactory.create`, к типу загружаемого модуля создает жесткую логическую привязку между модулем и загружающим приложением. В общем случае для сведения к минимуму зависимостей от преобразования модуля к экземпляру его класса следует использовать интерфейсы. Преобразование к типу интерфейса повышает гибкость кода и не ограничивает родительское приложение взаимодействием с одним экземпляром конкретного класса.

Следующий пример поможет вам лучше понять, почему преобразование к типу интерфейса повышает гибкость при разработке модульных приложений. Допустим, вы создали модуль, который загружается и выводит форму для ввода информации о пользователе. В процессе развития и изменения приложения может оказаться, что модуль должен выводить не одну, а несколько форм. Хотя на первый взгляд модули визуально различаются и могут выполнять разные операции с пользовательскими данными, доступ к данным модулей в сигнатурах методов остается прежним. Реализация одного интерфейса разными модулями, и как следствие, возможность преобразования к типу единого API, делает приложение более гибким.

В следующем примере интерфейс содержит свойства, относящиеся к информации о пользователе, которые могут быть реализованы разными модулями:

```
package {
    import flash.events.IEventDispatcher;
    public interface IUserEntry extends IEventDispatcher
    {
        function getFullName():String;
        function get firstName():String;
        function set firstName( str:String ):void;
        function get lastName():String;
        function set lastName( str:String ):void;
    }
}
```

Чтобы создать модуль, реализующий этот интерфейс, задайте свойству `implements` **узла** `<mx:Module>` **имя интерфейса** `IUserEntry`**:**

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="IUserEntry"
    layout="vertical"
    width="100%" height="100%">

    <mx:Metadata>
        [Event(name="submit", type="flash.events.Event")]
    </mx:Metadata>

    <mx:Script>
        <![CDATA[
            private var _firstName:String;
            private var _lastName:String;
            public static const SUBMIT:String = "submit";

            private function submitHandler():void
            {
                firstName = firstNameInput.text;
                lastName = lastNameInput.text;
                dispatchEvent( new Event( SUBMIT ) );
            }

            public function getFullName():String
            {
                return _firstName + " " + _lastName;
            }

            [Bindable]
            public function get firstName():String
            {
                return _firstName;
            }
```

```
            public function set firstName( str:String ):void
            {
                _firstName = str;
            }
            [Bindable]
            public function get lastName():String
            {
                return _lastName;
            }
            public function set lastName( str:String ):void
            {
                _lastName = str;
            }
        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="First Name:">
            <mx:TextInput id="firstNameInput" width="100%" />
        </mx:FormItem>
        <mx:FormItem label="Last Name:">
            <mx:TextInput id="lastNameInput" width="100%" />
        </mx:FormItem>
        <mx:Button label="submit" click="submitHandler();" />
    </mx:Form>

</mx:Module>
```

Этот модуль предоставляет визуальные компоненты для ввода и отправки имени и фамилии пользователя. Методы `get/set` **и открытый метод** `getFullName` **реализуются в теге** `<mx:Script>` **модуля. Привязка данных к атрибутам** `firstName` **и** `lastName` **устанавливается интерфейсом** `IUserEntry`**, который расширяет интерфейс** `IEventDispatcher`**, реализуемый классами** `mx.modules.Module` **и** `mx.modules.ModuleBase`**.**

Чтобы обратиться к данным этого или любого другого модуля, реализующего интерфейс `IUserEntry`**, родительское приложение преобразует значение соответствующего свойства с использованием** `ModuleLoader`**.**

В следующем примере свойство `child` **экземпляра** `<mx:ModuleLoader>` **используется для обращения к данным модуля, реализующего** `IUserEntry`**:**

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Script>
        <![CDATA[
            private var myModule:IUserEntry;

            private function moduleReadyHandler():void
            {
```

```
                myModule = moduleLoader.child as IUserEntry;
                myModule.addEventListener( "submit", submitHandler );
            }
            private function submitHandler( evt:Event ):void
            {
                welcomeField.text = 'Hello, ' + myModule.getFullName();
                trace( myModule.firstName + " " + myModule.lastName );
            }
        ]]>
    </mx:Script>

    <mx:ModuleLoader id="moduleLoader"
        url="ContactEntry.swf"
        ready="moduleReadyHandler();" />
    <mx:Label id="welcomeField" />

</mx:Application>
```

Обработчик события `ready` **экземпляра** `<mx:ModuleLoader>` **назначает обработчик для отправки информации о пользователе. При вызове метода** `submitHandler` **выводится строка, возвращаемая реализацией** `getFullName` **загруженного модуля. Преобразование свойства** `child` **экземпляра** `ModuleLoader` **к типу интерфейса** `IUserEntry` **обеспечивает слабую привязку в обмене данных родительского приложения с модулем. Это позволяет динамически взаимодействовать с модулями разных классов с одинаковой реализацией.**

Взаимодействие не ограничивается обращением к данным модуля со стороны родительского файла SWF. Модули также могут обращаться к данным родителя через свойство `parentApplication`**:**

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            private function creationHandler():void
            {
                infoField.text = parentApplication.getInformation();
            }
        ]]>
    </mx:Script>

    <mx:Text id="infoField" />
</mx:Module>
```

Когда модуль завершит формирование начальной раскладки, вызывается метод `creationHandler`**, а данные, возвращаемые методом** `getInformation` **родительского приложения, отображаются в дочернем компоненте** `Text`**.**

Свойство `parentApplication` экземпляра `Module` наследуется от суперкласса `UIComponent` и относится к типу `Object`. Динамический класс `Object` является корнем иерархии классов времени выполнения ActionScript. Соответственно к данным экземпляра `parentApplication` можно обращаться в точечной записи без учета реализации родительского класса; иначе говоря, модули могут вызывать свойства родительского приложения независимо от того, доступно такое свойство или нет, без выдачи исключений времени выполнения.

В общем случае обращения модулей к данным родительского приложения через свойство `parentApplication` нежелательны, потому что это создает жесткую логическую привязку модуля к родительскому приложению. Конечно, для устранения этой привязки можно назначить приложениям, загружающим модуль, определенный тип интерфейса, как это было сделано с модулями в предыдущих примерах этого рецепта. Но чтобы организовать единый механизм взаимодействий с одним модулем из разных приложений, рекомендуется использовать прямую передачу данных модулю из родительского приложения (вместо обращения к данным из динамического свойства `parentApplication`). Это позволит разрабатывать модульные приложения без обязательного наличия у модуля информации о его родительском приложении.

См. также

Рецепт 18.7–18.9.

18.11. Передача данных модулям с использованием строк запросов

Задача

Требуется передать данные модулю в процессе загрузки.

Решение

Присоедините к URL-адресу загружаемого модуля SWF строку запроса. После того как модуль будет успешно завершен, разберите строку URL-адреса из свойства `loaderInfo` модуля.

Обсуждение

К URL-адресу, используемому для загрузки модуля, можно присоединить строку с параметрами запроса. Когда модуль будет загружен, для обращения к URL-адресу используется свойство `mx.modules.Module`. Код ActionScript разбирает параметры из URL-адреса модуля, загруженного родительским приложением. Строка запроса следует за адресом модуля и отделяется от него вопросительным знаком (`?`), а параметры разделяются знаком `&`.

Пример присоединения строки запроса к URL-адресу модуля:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            private static const F_NAME:String = "Ted";
            private static const L_NAME:String = "Henderson";

            private function creationHandler():void
            {
                var params:String = "firstName=" + F_NAME +
                                    "&lastName=" + L_NAME;
                moduleLoader.url = "NameModule.swf?" + params;
            }
        ]]>
    </mx:Script>

    <mx:ModuleLoader id="moduleLoader" />

</mx:Application>
```

После создания экземпляра `<mx:ModuleLoader>` **и формирования начальной раскладки родительского приложения строка запроса строится из «свойство-значение». В данном примере свойства** `firstName` **и** `lastName` **передаются загруженному модулю в виде констант, но с таким же успехом это могут быть значения, полученные в результате вызова службы.**

Загруженный модуль разбирает запрос при помощи свойства `loaderInfo`**:**

```
<mx:Module
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="100%" height="100%"
    creationComplete="creationHandler();">

    <mx:Script>
        <![CDATA[
            import mx.utils.ObjectProxy;

            [Bindable]
            private var _proxy:ObjectProxy;

            private function creationHandler():void
            {
                _proxy = new ObjectProxy();

                var pattern:RegExp = /.*\?/;
                var query:String = loaderInfo.url.toString();
```

```
                query = query.replace( pattern, "" );

                var params:Array = query.split( "&" );

                for( var i:int = 0; i < params.length; i++ )
                {
                    var keyVal:Array = ( params[i] ).
                        toString().split("=");
                    _proxy[keyVal[0]] = keyVal[1];
                }
            }

        ]]>
    </mx:Script>

    <mx:Text text="{'Hello, ' + _proxy.firstName + ' '
        + _proxy.lastName}" />
</mx:Module>
```

Загруженный модуль разбирает строку запроса, выделяет параметры и добавляет их в виде свойств класса `ObjectProxy`. Благодаря встроенной поддержки привязки данных класса `ObjectProxy` значения параметров `firstName` и `lastName` отображаются в компоненте `Text` при обновлении этих свойств.

Передача данных в строке запроса – удобный способ получения и обработки данных загруженным модулем после инициализации. Строки запроса не создают жесткой логической привязки между родительским приложением и модулем, так как модуль обрабатывает данные по своему усмотрению и следит за тем, чтобы при этом не происходило ошибок времени выполнения. С другой стороны, такой механизм передачи данных модулям подвержен ошибкам «человеческого фактора»; опечатка в строке запроса приведет к нарушению обмена данными между родительским приложением и модулем.

См. также

Рецепт 18.10.

18.12. Оптимизация модулей с использованием отчетов компоновки

Задача

Требуется сократить размер файла и ускорить загрузку приложения.

Решение

Воспользуйтесь параметром командной строки `link-report` программы mxmlc при компиляции приложения, чтобы создать файл с отчетом

компоновки. Передайте файл отчета в качестве входного значения параметра командной строки `load-externs` при компиляции модуля. В результате будут откомпилированы только те классы, которые необходимы для работы модуля.

Обсуждение

При компиляции модуля в сгенерированный файл SWF включается весь пользовательский код и код Framework, от которого зависит модуль. Однако часть этого кода (особенно кода Framework) может быть общей для модуля и родительского приложения. Чтобы устранить избыточность и сократить размер файла модуля, передайте при компиляции модуля файл отчета компоновки.

Этот файл содержит список классов, задействованных в работе приложения. Он создается параметром командной строки `link-report` при компиляции приложения.

Следующая команда создает отчет компоновки с именем report.xml в текущем каталоге:

```
>mxmlc -link-report=report.xml MyApplication.mxml
```

Файл отчета используется при компиляции модуля для устранения избыточности и сокращения размера файла модуля. Включите в командную строку компиляции параметр `link-externs` и укажите в качестве значения имя созданного файла:

```
>mxmlc -link-externs=report.xml MyModule.mxml
```

В результате из сгенерированного файла SWF модуля исключается код, задействованный как в приложении, так и в модуле. Это замечательное средство оптимизации: представьте код Framework, который может быть скомпилирован как в приложении, так и в модуле. С другой стороны, компиляция модулей с отчетом компоновки формирует зависимость между приложением и модулем. Если в приложение будут внесены изменения, возможно, вам придется заново сгенерировать отчет компоновки и перекомпилировать модуль, чтобы обеспечить доступность этого кода.

Если приложение загружает более одного модуля, этот механизм оптимизации также может использоваться при компиляции кода, который не обязательно присутствует в родительском приложении, но является общим для нескольких модулей.

В общем случае все классы менеджеров (например, `mx.managers.DragManager` или `mx.managers.PopUpManager`), от которых могут зависеть модули, рекомендуется компилировать в составе родительского приложения. Это объясняется тем, что модули не могут обращаться к кодовым ресурсам других модулей; например, если один модуль попытается обратиться к объекту `DragManager` из другого модуля, произойдет исключение.

Чтобы модули могли обращаться к тому же объекту-менеджеру в составе приложения, необходимо импортировать и объявить локальную переменную класса в файле основного приложения:

```
import mx.managers.PopUpManager;
var popUpManager:PopUpManager;
```

Далее следует построить отчет компоновки для компиляции модулей, проследить за тем, чтобы модули использовали одну ссылку на объект-менеджер, и оптимизировать их по размеру файла модуля.

См. также

Рецепты 18.5 и 18.6.

19

AIR API

В Flex SDK входят классы для разработки приложений для настольных систем на базе *AIR* (Adobe Integrated Runtime). Adobe AIR – кроссплатформенная среда времени выполнения, которая позволяет разработчику адаптировать существующие веб-технологии RIA на настольных компьютерах. Среда AIR нейтральна по отношению к операционным системам; разработчик пишет приложение для платформы Adobe AIR и ему не приходится строить и развертывать несколько отдельных приложений для разных операционных систем. Хотя инфраструктура AIR дает возможность создавать настольные приложения на базе HTML и Ajax, в примерах этой главы основное внимание уделяется использованию Flex Framework.

Чтобы запустить приложение, написанное для платформы AIR, необходимо предварительно установить эту среду с сайта Adobe по адресу *http://labs.adobe.com/technologies/air/*. Приложения, выполняемые под управлением среды Adobe AIR, устанавливаются и запускаются точно так же, как все остальные «родные» настольные приложения.

Разработка приложений AIR с использованием Flex Framework напоминает создание приложений Flex для броузера. В Flex 3 SDK включены классы для выполнения операций с файловой системой, буфером обмена операционной системы и локальными базами данных, причем это далеко не все возможности, доступные в AIR API. Для упаковки приложения AIR в установочный файл вам понадобится файл SWF приложения и файл-дескриптор приложения; кроме того, приложение должно быть подписано с использованием шифрованного сертификата. Подпись гарантирует конечному пользователю, что он устанавливает исходную, неповрежденную версию вашего приложения. Для создания подписи могут использоваться как самостоятельно сгенерированные сертификаты, так и сертификаты от доверенного источника (такого как VeriSign

или Thawte). Установочные файлы AIR имеют расширение .air, а после установки выполняются под управлением среды Adobe AIR.

Описание всех возможностей AIR API могло бы стать темой для отдельной книги. В этой главе рассмотрены некоторые важнейшие возможности, которые помогут вам освоить разработку настольных приложений для среды Adobe AIR.

19.1. Создание приложения AIR с использованием Flex Framework

Задача

Требуется создать настольное приложение с использованием Flex и Adobe AIR.

Решение

Создайте главный файл приложения в MXML, с корневым тегом `<mx:WindowedApplication>` и файлом-дескриптором приложения, который задает свойства установки и запуска приложения AIR.

Обсуждение

Разработка приложений для платформы AIR с использованием Flex Framework в целом напоминает разработку Flex-приложений для Web, однако разработчик получает возможность взаимодействовать с операционной системой. При разработке Flex-приложений для Web основной файл приложения содержит корневой тег `<mx:Application>`. Для приложений AIR, выполняемых на настольном компьютере в окнах операционной системы, основное приложение содержит корневой тег `<mx:WindowsApplication>`.

Класс `mx.core.WindowedApplication` представляет собой расширение `mx.core.Application` со специфическими свойствами и методами, предназначенными для работы с окнами операционной системы. `WindowedApplication` представляет главное окно приложения AIR. Это окно может создавать другие окна, обращаться к файловой системе и взаимодействовать с операционной системой – словом, выполнять большинство операций, характерных для настольного приложения.

Пример файла простейшего приложения AIR:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    title="Hello World">

    <mx:Label text="Welcome to AIR" />

</mx:WindowedApplication>
```

При запуске приложения в окне приложения и на системной панели задач отображается текст *Hello World*. В компоненте выводится текст *Welcome to AIR*.

Компиляция приложений AIR осуществляется программой командной строки amxmlc из каталога /bin установки Flex 3 SDK. Вероятно, вы уже знакомы с программой mxmlc, которая тоже находится в каталоге /bin; эта программа предназначена для компиляции приложений на базе Flex и ActionScript. При запуске mxmlc использует стандартный конфигурационный файл flex-config.xml из подкаталога /frameworks установочного каталога SDK. Программа amxmlc использует конфигурационный файл air-config.xml из каталога /frameworks, а при ее компиляции используется архив airglobal.swc из каталога /libs/air.

Чтобы откомпилировать приложение AIR, убедитесь в том, что путь к каталогу /bin задан в системной переменной PATH, откройте приглашение командной строки и введите следующую команду:

```
>amxmlc HelloWorld.mxml
```

Команда создает файл HwlloWorld.swf в текущем каталоге командной строки. Запустив файл SWF щелчком, вы не увидите ничего, кроме стандартного черного фона. Это объясняется тем, что приложение запускается в Flash Player. Чтобы увидеть приложение AIR, необходимо запустить его в среде Adobe AIR. Сгенерированный файл SWF необходим для развертывания и упаковки приложения AIR, как и другой файл, называемый дескриптором приложения.

Дескриптор приложения представляет собой файл XML, содержащий описание свойств установки, запуска и идентификации приложения AIR для операционной системы. Этот файл необходим для развертывания приложения AIR в ходе разработки и упаковки приложения в установочный файл для дальнейшего распространения.

Наряду с такими параметрами, как каталог установки приложения в каталоге Program Files операционной системы и файл графического значка, дескриптор приложения позволяет задать свойства главного окна приложения; после запуска эти свойства становятся доступными только для чтения.

Пример файла дескриптора для приложения AIR, созданного в предыдущем примере:

```
<application xmlns="http://ns.adobe.com/air/application/1.0">
    <id>com.oreilly.flexcookbook.HelloWorld</id>
    <filename>HelloWorld</filename>
    <name>Hello World</name>
    <version>0.1</version>
    <description>A Hello World Application</description>

    <initialWindow>
        <content>HelloWorld.swf</content>
        <systemChrome>standard</systemChrome>
```

```
        <transparent>false</transparent>
        <visible>true</visible>
        <width>400</width>
        <height>400</height>
    </initialWindow>

</application>
```

Параметры, заданные в этом примере, составляют лишь небольшое подмножество доступных свойств, но по всей вероятности, такая конфигурация будет характерна для большинства создаваемых вами приложений AIR. Корневой тег файла дескриптора `<application>` содержит атрибут, определяющий URI пространства имен AIR. Последний сегмент пространства имен (`1.0`) определяет версию среды AIR, для которой предназначено приложение.

Обязательными параметрами дескриптора приложения являются `<id>`, `<filename>` и `<version>`, родительский узел `<initialWindow>` и дочерние узлы `<content>`. Элемент `id` содержит уникальный идентификатор приложения. В качестве его значения рекомендуется использовать строку DNS, переставленную в обратном порядке и разделенную точками. Идентификатор используется для установки приложения и обращения к каталогу, в котором хранятся приложения. Элемент `filename` используется для обращения к приложению из операционной системы; его значение включает имя исполняемого файла приложения и путь установки. Элемент `version` содержит назначенный разработчиком номер версии приложения; какие-либо требования к стандартному формату отсутствуют, но он необходим для распространения обновлений приложения. Элемент `content` узла `<initialWindow>` определяет основной файл SWF приложения, загружаемый в AIR при запуске приложения. Файл SWF создается программой командной строки amxmlc, включенной в Flex 3 SDK, которой в качестве входного файла передается основной файл приложения AIR.

Помимо обязательных примеров дескриптора приложения, в этом примере также были добавлены параметры `<name>`, `<description>` и параметры окна. Значения элементов `name` и `descrtiption` отображаются в окне установки.

Свойства `systemChrome`, `transparent` и `visible` узла `<initialWindow>` относятся к визуальному оформлению главного окна приложения. Эти свойства могут задаваться либо в файле дескриптора, либо в корневом теге `<mx:WindowedApplication>` главного файла, но после запуска приложения они доступны только для чтения.

Свойства `width` и `height` определяют размеры окна приложения при запуске. Эти параметры не являются обязательными (как и другие параметры `initialWindow`, относящиеся к ограничениям размеров и позиции окна), однако они были включены в пример, поскольку эти значения часто используются на практике.

19.2. Инструментарий командной строки AIR

Задача

Требуется использовать программы командной строки Flex 3 SDK, предназначенные для развертывания, отладки и упаковки приложений AIR.

Решение

Используйте программы amxmlc, adl и adt из подкаталога /bin установочного каталога Flex 3 SDK.

Обсуждение

Flex 3 SDK включает подборку программ командной строки для компиляции, запуска и упаковки приложений AIR в установочные файлы. Компиляция приложений AIR осуществляется программой mxmlc с указанием файла основного приложения. Этим файлом может быть файл HTML, файл ActionScript или файл MXML. Примеры этой главы ориентируются на разработку приложений AIR с использованием Flex Framework, а файл основного приложения имеет расширение .mxml. Программа adl используется для запуска приложения AIR в среде Adobe AIR во время разработки, до упаковки приложения для установки. Программа adt создает и назначает сертификаты, а также упаковывает приложения AIR в установочные файлы.

Программа amxmlc, используемая для создания основного файла SWF, имеет много общего с компилятором командной строки mxmlc. Более того, команда amxmlc запускает компилятор MXML и приказывает ему использовать конфигурационный файл air-config.xml из подкаталога /frameworks установочного каталога SDK. При запуске amxmlc доступны все параметры командной строки, используемые при запуске mxmlc, но в примерах этой главы для построения файлов SWF будет использоваться базовый набор параметров.

В качестве исходного файла amxmlc передается основной файл приложения с корневым тегом `<mx:WindowedApplication>`, использующий Flex Framework. Пример простого файла основного приложения AIR:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    title="Hello World">
    <mx:Label text="Welcome to AIR" />
</mx:WindowedApplication>
```

Класс `mx.core.WindowedApplication` является контейнером для приложений AIR на базе Flex. Класс `WindowedApplication` расширяет `mx.core.Application` и соответствует главному окну операционной системы для приложения AIR.

Компиляция этого приложения для работы в среде AIR осуществляется командой amxmlc, по аналогии с компиляцией веб-приложений Flex командой mxmlc:

```
>amxmlc -output HelloWorld.swf HelloWorld.mxml
```

Команда создает файл SWF с именем HelloWorld.swf в текущем каталоге командной строки. Параметр `output` не является обязательным; при его отсутствии результат будет тем же. Если параметр `output` не указан, компилятор присваивает сгенерированному файлу SWF имя, совпадающее с именем входного файла. Если вы предпочитаете задать файлу SWF другое имя, включите параметр `output`, но помните, что правильное имя должно присутствовать в дескрипторе приложения, который используется для развертывания и упаковки приложения AIR.

Как упоминалось ранее в этой главе, *дескриптор приложения* представляет собой файл **XML с описанием свойств установки, запуска и идентификации** приложения AIR для операционной системы. Запустив файл SWF, созданный компилятором amxmlc, в Flash Player, вы увидите только черный фон, потому что это приложение работает только в среде Adobe AIR. Для запуска приложения, откомпилированного командой amxmlc, потребуется файл-дескриптор приложения для файла SWF.

Пример простого дескриптора, сохраненного под именем HelloWorld-app.xml, в котором ранее созданный файл SWF указывается в качестве файла приложения:

```
<application xmlns="http://ns.adobe.com/air/application/1.0">

    <id>com.oreilly.flexcookbook.HelloWorld</id>
    <filename>HelloWorld</filename>
    <name>Hello World</name>
    <version>0.1</version>
    <description>A Hello World Application</description>

    <initialWindow>
        <content>HelloWorld.swf</content>
        <systemChrome>standard</systemChrome>
        <transparent>false</transparent>
        <visible>true</visible>
        <width>400</width>
        <height>400</height>
    </initialWindow>

</application>
```

В файле дескриптора приведены значения свойств исходного окна приложения (экземпляр `WindowedApplication` основного файла) и параметры для установки и идентификации приложения в операционной системе. Обратите внимание: в элементе `content` указывается местонахождение SWF-файла приложения, созданного программой amxmlc.

В процессе разработки приложения AIR программа adl (Adobe Debug Launcher) используется для запуска приложения и его предварительного просмотра перед упаковкой и установкой. Программа adl также обеспечивает вывод трассировочных данных на консоль во время работы FDB (Flash Debugger). Чтобы запустить приложение AIR программой adl до того, как оно будет упаковано, введите следующую команду с указанием имени файла дескриптора приложения:

```
>adl HelloWorld-app.xml
```

Если команда будет успешно выполнена, приложение запускается в окне, показанном на рис. 19.1.

Рис. 19.1. *Приложение AIR Hello World*

Если приложение работает так, как предполагалось, его можно упаковать в установочный файл программой AIR Developer Tool (adt). Установочные файлы AIR должны быть подписаны с использованием цифрового сертификата, подтверждающего подлинность приложения. Для построения установочного файла понадобится сертификат от доверенного источника (например, VeriSign или Thawte); впрочем, вы также можете создать сертификат самостоятельно. Первый вариант подтверждает личность стороны, выдавшей сертификат, и гарантирует конечному пользователю, что приложение не подвергалось злонамеренным модификациям. Хотя сертификаты, самостоятельно сгенерированные программой adt, также подтверждают, что приложение не изменялось с момента подписи, они не могут использоваться для проверки «личности» стороны, выдавшей сертификат.

Приложения **AIR рекомендуется подписывать сертификатами, выданными доверенными источником**, но в примерах этой главы для построения установочных файлов будут использоваться самостоятельно сгенерированные сертификаты.

Чтобы создать сертификат программой adt, введите следующую команду:

```
>adt -certificate -cn HelloWorld 1024-RSA certificate.pfx password
```

Команда создает файл с именем certificate.pfx в текущем каталоге командной строки. Наряду с командой построения сертификата и соответствующего закрытого ключа указывается общее имя сертификата (HelloWorld) и тип ключа (1024-RSA). Допустимые типы ключей – 1024-RSA и 2048-RSA – описывают тип шифрования ключа. Файлы сертификатов могут иметь расширения .pfx и .p12, соответствующие типам файлов Personal Information Exchange.

Сертификат и закрытый ключ, созданные программой adt, сохраняются в файле типа PKCS12. В дальнейшем сертификат и ключ используются для сертификации и упаковки приложения AIR командой `package` программы adt. Чтобы упаковать приложение в установочный файл, выполните следующую команду:

```
>adt -package -storetype pkcs12 -keystore certificate.pfx HelloWorld.air
HelloWorld-app.xml HelloWorld.swf
```

Параметр `storetype` определяет версию хранилища ключей PKCS (Public-Key Cryptography Standards), используемую для хранения закрытого ключа и сертификата. В параметре `keystore` передается сертификат, созданный ранее командой `certificate` программы adt. Последние три входных параметра определяют соответственно имя установочного файла, имя файла с дескриптором приложения и имя файла SWF приложения. Вы можете включить дополнительные имена файлов, необходимых для работы приложения, в конец команды; эти файлы будут упакованы и установлены вместе с приложением. Местонахождение файлов задается по отношению к текущему каталогу, но параметр `-C` позволяет сменять каталоги с включаемыми файлами.

При выполнении команды `package` программы adt вам будет предложено ввести пароль, использованный при построении сертификата; в нашем примере используется строка *password*. Если выполнение команды завершилось успешно, а пароль был принят, в текущем каталоге командной строки создается файл с именем HelloWorld.air – установочный файл приложения Hello World. Чтобы установить приложение, перейдите в каталог, который был текущим на момент выполнения команды, и сделайте двойной щелчок на файле HelloWorld.air. На экране появляется окно, показанное на рис. 19.2.

Это окно увидит пользователь, который будет устанавливать ваше приложение **AIR. В качестве имени приложения выводится значение, заданное** свойству `<name>` в файле дескриптора, а издатель **(Publisher) приложения** считается неизвестным, поскольку при упаковке приложения AIR использовался самостоятельно созданный сертификат. Если щелкнуть на кнопке Install, появится следующее окно, показанное на рис. 19.3.

Кнопка Continue во втором окне на рис. 19.3 устанавливает приложение и создает ярлык (shortcut) на рабочем столе. Вы можете указать каталог для установки приложения; если оставить в поле Installation Location имя каталога по умолчанию вашей операционной системы, приложение будет установлено в каталог C:\Program Files\HelloWorld

Рис. 19.2. *Начальное окно установки приложения Hello World*

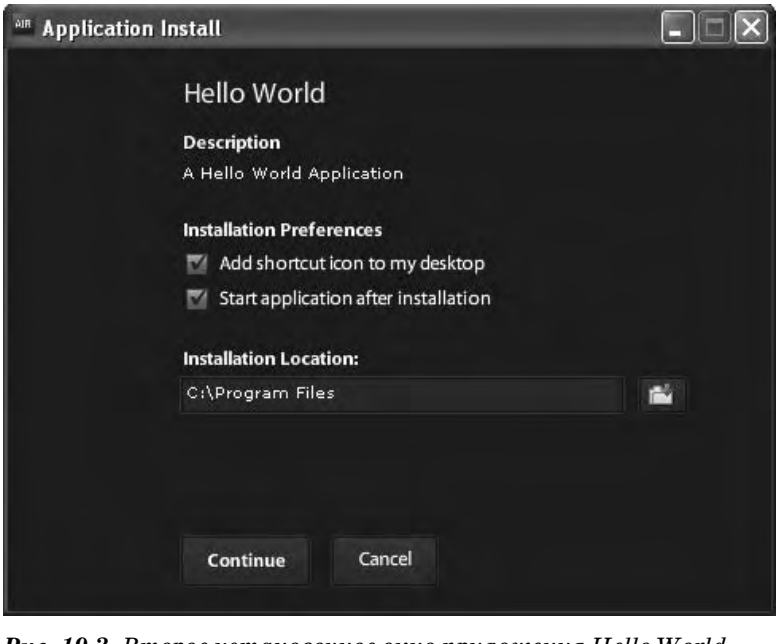

Рис. 19.3. *Второе установочное окно приложения Hello World*

в Windows или HD/Applications/HelloWorld в Mac OS X. Каталог, в который устанавливается приложение, называется *каталогом приложения*. Также в процессе установки создается каталог для хранения данных; в системе Windows для этой цели может использоваться каталог C:\Documents and Settings\<*пользователь*>\Application Data\com.oreilly.flexcookbook.HelloWorld, а в Mac OS X – каталог /Users/<*пользователь*>/Library/Preferences/com.oreilly.flexcookbook.HelloWorld. Имя каталога представляет собой записанное в обратном порядке имя DNS, указанное в качестве идентификатора приложения в файле дескриптора. Оба каталога, созданные во время установки – каталог приложения и каталог хранения данных, – доступны через API файловой системы AIR. Файлы в каталоге хранения данных можно как читать, так и записывать, но каталог приложения доступен только для чтения.

Приложения AIR удаляются так же, как и любые другие настольные приложения: программой Установка и удаление программ в системе Windows или перетаскиванием каталога приложения в Корзину в Mac OS X. При удалении приложения уничтожаются все файлы, созданные при установке, но остаются дополнительные файлы, созданные после установки, включая каталог хранения данных.

См. также

Рецепт 19.1.

19.3. Управление окнами

Задача

Требуется открывать «родные» окна операционной системы из приложения AIR.

Решение

Используйте классы `flash.display.NativeWindow` и `mx.core.Window` из оконного AIR API.

Обсуждение

Приложения AIR могут создавать окна, соответствующие «родным» окнам операционной системы не только по функциональности, но и по внешнему виду и поведению. Наряду с простым созданием окон, обладающих тем же оформлением и прямоугольной формой, что и окна других настольных приложений, вы можете создавать и применять пользовательские скины при помощи стилевых свойств и нестандартной графики. Кроме настройки оформления, содержащего элементы управления окном, вы также можете прослушивать события, передаваемые «родными» окнами для этих элементов.

В качестве корневого тега приложения MXML для среды AIR используется тег <mx:WindowedApplication>. Он представляет исходное окно приложения, отображаемое при его запуске. Пользовательские настройки задаются в файле дескриптора приложения и объявляются в теге <mx:WindowedApplication>. Окно WindowedApplication выполняет функции контейнера для экземпляра класса flash.display.NativeWindow и позволяет включать компоненты Flex непосредственно в список отображения из разметки MXML. Обращение к экземпляру NativeWindow из класса WindowedApplication осуществляется через свойство nativeWindow.

Класс NativeWindow предоставляет интерфейс для управления «родными» окнами настольной системы. Чтобы создать новое окно в приложении AIR с использованием Flex Framework, следует создать новый экземпляр класса mx.core.Window. Как и класс WindowedApplication, класс Window выполняет функции контейнера для базового экземпляра NativeWindow, доступного через свойство nativeWindow. Инициализация свойств окна определяется в корневом теге <mx:Window> пользовательского компонента окна.

Пример создания пользовательского окна посредством расширения класса Window:

```
<mx:Window
    xmlns:mx="http://www.adobe.com/2006/mxml"
    title="Hello"
    width="200" height="200">

    <mx:Label text="I am a Window!" />

</mx:Window>
```

Когда приложение отдаст команду на открытие окна, оно будет размещено на рабочем столе с размерами 200×200, а в строке заголовка будет содержаться текст *Hello*. В корневом теге <mx:Window> могут определяться и другие свойства оформления окна и прозрачности, а также обработчики событий, передаваемых экземпляром NativeWindow. Следующий пример удаляет стандартное системное оформление, отключает кнопки изменения размеров и добавляет слушателей событий для элементов управления окном:

```
<mx:Window
    xmlns:mx="http://www.adobe.com/2006/mxml"
    title="Hello"
    systemChrome="none" transparent="true"
    maximizable="false" minimizable="false"
    width="200" height="200"
    windowComplete="completeHandler();"
    closing="closingHandler(event);">

    <mx:Script>
        <![CDATA[
```

```
            private function completeHandler():void
            {
                nativeWindow.addEventListener( NativeWindowBoundsEvent.
                    RESIZING, resizeHandler );
            }
            private function resizeHandler(
                evt:NativeWindowBoundsEvent ):void
            {
                trace( evt.beforeBounds + " : " + evt.afterBounds );
            }
            private function closingHandler( evt:Event ):void
            {
                trace( "goodbye!" );
            }
        ]]>
    </mx:Script>

    <mx:Label text="I am a Window!" />

</mx:Window>
```

При открытии окно оформляется с применением стандартного оформления Flex, и у него отключаются кнопки изменения размеров. Тег назначает обработчики событий, передаваемых экземпляром `NativeWindow` при создании, изменении размеров и закрытии окна.

Оконный API предоставляет возможность выбора типа окна. Свойство `type` задает значение из перечисления в классе `NativeWindowTypes`; в результате к окну применяется определенная комбинация пользовательского оформления и элементов. По умолчанию свойство `type` равно `normal`. Окна с типами `utility` и `lightweight` не отображаются на панели задач операционной системы Windows и в оконном меню Mac OS X. К первому типу относятся упрощенные окна с сокращенным визуальным оформлением, имеющие только кнопку закрытия окна. У окон второго типа свойство `systemChrome` должно быть равно `false`; такие окна используются в основном для передачи информации пользователю (как стандартные окна оповещений в других настольных приложениях).

Внешний вид строки заголовка, строки состояния и манипулятора изменения размеров также могут настраиваться при помощи свойств `showTitleBar`, `showStatusBar` и `showGripper` компонента `<mx:Window>`. Чтобы удалить стандартное оформление Flex, задайте свойству `showFlexChrome` тега `<mx:Window>` значение `false`; в полученном окне отображается только текущее содержимое сцены без стандартных системных элементов оформления. В приведенном примере это приведет к тому, что текст *I am a Window!* выводится на экране без фона или оконного оформления.

Чтобы отобразить компонент `<mx:Window>`, следует сначала создать экземпляр окна, а затем вызвать для него метод `open`:

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

```
    layout="vertical"
    windowComplete="initHandler();"
    closing="closeHandler();">

    <mx:Script>
        <![CDATA[
            import com.oreilly.flexcookbook.CustomWindow;

            private var window:CustomWindow;
            private function initHanlder():void
            {
                window = new CustomWindow();
                window.alwaysInFront = true;
                window.open();
            }

            private function closeHandler():void
            {
                if( !window.closed ) window.close();
            }
        ]]>
    </mx:Script>

</mx:WindowedApplication>
```

Когда приложение завершит формирование начальной раскладки и откроет экземпляр `NativeWindow`, передается событие `windowComplete` и вызывается метод `initHandler`. В методе `initHandler` создается и открывается экземпляр класса `CustomWindow`. Чтобы окно всегда оставалось видимым, т. е. находилось в начале z-порядка окон на экране, свойству `alwaysInFront` задается значение `true`. Окна также можно вручную позиционировать на основании z-порядка других окон (включая экземпляр `WindowedApplication`) при помощи методов `orderInFrontOf`, `orderInBackOf`, `orderToFront` и `orderToBack` класса `Window`.

Обратите внимание на важное обстоятельство: закрытие главного окна приложения до закрытия всех открытых им окон не приведет к завершению приложения. По этой причине слушатель события `closing` проверяет, что все остальные окна были закрыты должным образом.

См. также

Рецепты 19.1 и 19.2.

19.4. Создание меню

Задача

Требуется предоставить пользователю «родное» меню операционной системы для выполнения команд.

Решение

Используйте функции API для создания меню в приложениях и окнах.

Обсуждение

Классы API работы с меню позволяют взаимодействовать с поддержкой меню в операционной системе. Вы можете добавлять команды меню и прослушивать события, передаваемые при выборе. Существует много разновидностей меню; способ создания и взаимодействия с ними зависит от операционной системы, в которой выполняется приложение AIR. А это означает, что при включении «родных» меню в приложение необходимо обеспечить поддержку всех целевых операционных систем.

Операционная система Mac OS X поддерживает *меню приложения* – глобальное меню, вызываемое с панели инструментов приложения, которое не зависит от того, какое окно приложения обладает фокусом в настоящий момент.

Меню приложения автоматически создается операционной системой Mac OS X; вы можете добавлять к стандартным меню команды и подменю, а также обработчики событий.

Меню операционной системы Windows относятся к категории *оконных меню*. Такие меню отображаются в окнах под строкой заголовка и присутствуют только в «родных» окнах с системным оформлением. Как следствие, оконные меню не могут добавляться в экземпляры `<mx:Window>` со свойством `systemChrome=none`.

Чтобы определить, в какой операционной системе выполняется приложение AIR, используйте свойства `NativeWindow.supportsMenu` и `NativeApplication.supportsMenu` во время выполнения:

```
if( NativeWindow.supportsMenu )
{
    // Windows
}
else if( NativeApplication.supportsMenu )
{
    // Mac OS X
}
```

Если свойство `NativeWindow.supportsMenu` равно `true`, значит, оконные меню поддерживаются операционной системой, и приложение работает в среде Windows. Класс `flash.desktop.NativeApplication`, предоставляющий информацию и функции уровня приложения, также содержит статическое свойство `supportsMenu`. Если объект `NativeApplication` поддерживает меню приложения, значит, приложение выполняется в среде Mac OS X.

Чтобы добавить объект «родного» меню, которое должно стать корневым, создайте новый экземпляр `NativeMenu` и задайте его свойству `menu` экземпляра `NativeWindow` или `NativeApplication`:

```
if( NativeWindow.supportsMenu )
{
    stage.nativeWindow.menu = new NativeMenu();
}
else if( NativeApplication.supportsMenu )
{
    NativeApplication.nativeApplication.menu = new NativeMenu();
}
```

После назначения корневого меню в объект `NativeMenu` **добавляются новые меню и подменю. В следующем примере для объекта** `<mx:WindowsApplication>` **создается меню приложения или оконное меню (в зависимости от того, в какой операционной системе работает приложение):**

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="initHandler();">

    <mx:Script>
        <![CDATA[

            private function initHandler():void
            {
                var fItem:NativeMenuItem = new
                    NativeMenuItem( "File" );
                var fileMenu:NativeMenuItem;
                if( NativeWindow.supportsMenu )
                {
                    stage.nativeWindow.menu = new
                    NativeMenu();
                    fileMenu =
                    stage.nativeWindow.menu.addItem( fItem );
                }
                else if( NativeApplication.supportsMenu )
                {
                    NativeApplication.nativeApplication.menu =
                    new NativeMenu();
                    fileMenu =
                    NativeApplication.nativeApplication.menu.addItem(fItem);
                }
                fileMenu.submenu = createFileMenu();
            }

            private function createFileMenu():NativeMenu
            {
                var menu:NativeMenu = new NativeMenu();
                var openItem:NativeMenuItem = new
                    NativeMenuItem( "Open" );
                var openCmd:NativeMenuItem = menu.addItem(openItem );
                openCmd.addEventListener(
```

```
                    Event.SELECT, openHandler );
                var saveItem:NativeMenuItem =
                    new NativeMenuItem( "Save" );
                var saveCmd:NativeMenuItem =
                    menu.addItem( saveItem );
                saveCmd.addEventListener(
                    Event.SELECT, saveHandler );
                return menu;
            }
            private function openHandler( evt:Event ):void
            {
                printOut.text += "You selected open.\n";
            }
            private function saveHandler( evt:Event ):void
            {
                printOut.text += "You selected save.\n";
            }

        ]]>
    </mx:Script>
    <mx:TextArea id="printOut"
        width="100%" height="100%"
        />
</mx:WindowedApplication>
```

После создания приложения и формирования начальной раскладки вызывается метод `initHandler`; новое меню File включается в экземпляр `menu` экземпляра `WindowedApplication` или объект `NativeApplication` в зависимости от операционной системы (Windows или Mac OS X соответственно). Команды подменю включаются в меню File методом `createFileMenu`, и назначаются обработчики событий, реагирующие на событие `select` каждой команды.

При запуске приложения в операционной системе Windows главное меню приложения располагается под строкой заголовка и содержит единственную команду File. При выборе этой команды открывается подменю с несколькими командами. Щелчок на каждой команде приводит к вызову зарегистрированного обработчика события и выводу текста в объекте `<mx:TextArea>`.

Класс `NativeMenuItem` также поддерживает разделительные линии. По умолчанию аргумент конструктора `isSeparator` равен `false`, но если задать ему значение `true`, в меню включается горизонтальный разделитель:

```
var rule:NativeMenuItem = new NativeMenuItem( "Line", true );
```

Свойство `enabled` класса `NativeMenuItem` позволяет устанавливать и снимать блокировку команд меню во время выполнения:

```
var saveItem:NativeMenuItem = new NativeMenuItem( "Save" );
saveItem.enabled = false;
```

Устанавливая и снимая блокировку команд, можно управлять доступностью команд меню в зависимости от текущей ситуации и требований приложения.

Концепция «родного» меню относится не только к меню приложения и оконным меню. *Контекстные меню* (открываемые щелчком правой кнопки мыши или щелчком с нажатой клавишей Command в окнах, системной панели, значках и т. д.) тоже относятся к этой категории. Более того, классы `flash.ui.ContextMenu` и `flash.ui.ContextMenuItem` представляют собой расширения классов, входящих в AIR API для работы с меню.

См. также

Рецепт 19.3.

19.5. Чтение и запись в файл

Задача

Требуется обеспечить создание, чтение и запись файлов в файловой системе.

Решение

Используйте классы `File`, `FileStream` и `FileMode` **AIR API файловой системы.**

Обсуждение

Объект `File` представляет файл или каталог. Для выполнения операций чтения/записи с файлами на жестком диске используется объект `FileStream`. Файлы открываются для чтения и записи синхронными и асинхронными методами класса `FileStream`. При использовании синхронного метода `FileStream.open` файл интерпретируется как объект `ByteArray`, а все остальные операции приостанавливаются до завершения чтения или записи. При асинхронном методе `FileStream.openAsync` файл интерпретируется скорее как объект `URLStream`, а данные помещаются в буфер по мере поступления. Выбор между синхронным и асинхронным методом зависит от особенностей приложения, однако следует помнить, что после завершения операции поток необходимо закрыть.

Действие, выполняемое с файлом, задается строковыми константами класса `FileMode` при вызове методов `FileStream.open` и `FileStream.openAsync`. Константа `FileMode.WRITE` означает, что при открытии файлового потока файл будет создан (если он не существует) или перезаписан (если файл существует). С константой `FileMode.APPEND` данные, находящиеся в буфере, записываются в конец файла, а константа `FileMode.UPDATE` предоставляет доступ к файлу для чтения/записи. Все директивы записи создают новый файл, если он не существует. При использовании константы `FileMode.`

READ данные из файла читаются в буфер, причем файл обязательно должен существовать.

Чтобы выполнить чтение или запись в файл, необходимо связать объект File с файлом или каталогом на компьютере пользователя. Класс File содержит ряд статических свойств и методов для связывания со стандартными каталогами файловой системы, каталогом приложения и каталогом хранения данных.

В следующем примере асинхронный метод FileStream.open используется для записи данных в файл на рабочем столе:

```
var file:File = File.desktopDirectory.resolvePath( "test.txt" );
var stream:FileStream = new FileStream();
stream.open( file, FileMode.WRITE );
stream.writeUTFBytes( "Hello World" );
stream.close();
```

Файл с именем test.txt создается или открывается и помещается в буфер объекта FileStream. Текст *Hello World* записывается в файл методом FileStream.writeUTFBytes. Чтобы открыть файл и прочитать из него данные, используйте метод FileStream.readUTFBytes:

```
var file:File = File.desktopDirectory.resolvePath( "test.txt" );
var stream:FileStream = new FileStream();
stream.open( file, FileMode.READ );
trace( stream.readUTFBytes( stream.bytesAvailable ) );
stream.close();
```

При открытии файла для чтения в режиме FileMode.READ объект FileStream немедленно приступает к чтению данных в буфер. Для обращения к данным в буфере используется свойство FileStream.bytesAvailable. При использовании синхронного метода FileStream.open все операции приостанавливаются до того момента, когда данные будут помещены в буфер, поскольку эта операция выполняется в программном потоке основного приложения. Также можно воспользоваться асинхронным методом FileStream.openAsync, открыть файл в асинхронном режиме и прослушивать события передачи/завершения для чтения данных, постоянно поступающих в буфер чтения.

Если файл повторно помещается в буфер записи посредством указания аргумента FileMode.WRITE при открытии потока, это приведет к перезаписи всех содержащихся в нем данных. Чтобы узнать, доступен ли файл с заданным именем, можно воспользоваться свойством exists класса File:

```
var file:File = File.desktopDirectory.resolvePath( "test.txt" );
if( file.exists )
{
    trace( "ile created: " + file.creationDate );
    file = File.desktopDirectory.resolvePath( "test2.txt" );
}
var stream:FileStream = new FileStream();
```

```
stream.open( file, FileMode.WRITE );
stream.writeUTFBytes( "Hello World" );
stream.close();
```

Методы `FileStream.writeUTFBytes` и `FileStream.readUTFByte` в этом примере составляют небольшое подмножество методов класса `FileStream` для чтения и записи данных в буфер файлового потока.

Говоря об операциях чтения/записи файлов, важно учитывать разрешения доступа чтения/записи для каталогов, созданных в файловой системе при установке приложения AIR. В процессе установки для приложения создаются два основных каталога: каталог приложения и каталог хранения данных. Пути к этим каталогам хранятся в свойствах `File.applicationDirectory` и `File.applicationStorageDirectory` соответственно. Каталог приложения доступен только для чтения; это означает, что файлы из этого каталога могут помещаться только в буфер чтения. Каталог хранения данных доступен как для чтения, так и для записи, и после установки в него можно записывать любые файлы, необходимые для работы приложения.

19.6. Сериализация объектов

Задача

Требуется сериализовать объекты пользовательских классов вашего приложения в файлы на жестком диске.

Решение

Зарегистрируйте псевдонимы классов и используйте метод `FileStream.writeObject` для сохранения объектов в файле в формате AMF (ActionScript Message Format).

Обсуждение

API файловой системы для приложений AUR позволяет записать объекты в буфер файлового потока в кодировке AMF. Для большинства объектных типов языка ActionScript (например, `String` и `Boolean`) сериализация поддерживается автоматически. Такие типы можно преобразовать в двоичный формат с использованием AMF; они сохранят свои значения при десериализации. Однако для пользовательских типов автоматическая сериализация не поддерживается. Чтобы пользовательские объекты можно было сериализовать, для класса необходимо зарегистрировать *псевдоним* (alias) – либо методом `registerClassAlias`, либо объявлением тега метаданных `[RemoteClass]` перед определением класса.

Допустим, приложение выводит информацию о пользователе в виде объекта, который может быть загружен позднее в этом же приложении или в другом приложении, которое умеет работать с такими объектами. Класс с информацией о пользователе выглядит примерно так:

```
package com.oreilly.flexcookbook
{
    [RemoteClass]
    [Bindable]
    public class UserData
    {
        public var firstName:String;
        public var lastName:String;
        public var age:Number;
        public var sessionLength:Number;
        public function UserData( firstName:String = "",
                                  lastName:String = "" )
        {
            this.firstName = firstName;
            this.lastName = lastName;
        }
    }
}
```

При каждом запуске приложения создается новый объект `UserData`, заполняемый пользователем. При желании можно сохранить эту информацию в текстовом файле в виде отформатированной строки, однако сериализация объекта в файле позволит легко загрузить его в приложении без разбора строки и задания свойств нового экземпляра `UserData`. Чтобы сериализовать объект и автоматически восстановить его свойства при десериализации, зарегистрируйте класс при помощи тега метаданных `[RemoteClass]`.

Добавление тега `[RemoteClass]` перед определением класса означает, что класс может сохранять информацию типа и значения свойств при сериализации/десериализации в формате AMF. Тег метаданных `[Transient]` перед определением переменной исключает соответствующее свойство из процесса сериализации. Так, свойство `sessionLength` в предыдущем примере обновляется во время выполнения приложения, и его сохранение между сеансами не имеет смысла, поэтому свойство помечается тегом `[Transient]`:

```
package com.oreilly.flexcookbook
{
    [RemoteClass]
    public class UserData
    {
        public var firstName:String;
        public var lastName:String;
        public var age:Number;
        [Transient]
        public var sessionLength:Number;

        public function UserData( firstName:String = "",
                                  lastName:String = "" )
        {
```

```
            this.firstName = firstName;
            this.lastName = lastName;
        }

    }
}
```

Сериализация пользовательских объектов в файле осуществляется методом `FileStream.writeObject`. По умолчанию при чтении и записи двоичных данных методами `writeObject` и `readObject` используется спецификация AMF 3. Свойство `FileStream.objectEncoding` также позволяет использовать формат AMF 0. В следующем примере приложение сохраняет информацию о пользователе в файле с расширением `.user`:

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

    <mx:Script>
        <![CDATA[
            import com.oreilly.flexcookbook.UserData;

            private var userData:UserData = new UserData();
            private static const EXT:String = ".user";
            private function submitHandler():void
            {
                userData.firstName = firstField.text;
                userData.lastName = lastField.text;
                userData.age = ageField.value;
                saveUserData();
            }
            private function saveUserData():void
            {
                var fnm:String = userData.firstName + "_" +
                                 userData.lastName + EXT;
                var file:File =
                    File.desktopDirectory.resolvePath( fnm );
                var stream:FileStream = new FileStream();
                stream.open( file, FileMode.WRITE );
                stream.writeObject( userData );
                stream.close();
            }

        ]]>
    </mx:Script>

    <mx:Form>
        <mx:FormItem label="First Name:">
            <mx:TextInput id="firstField"
                change="{submitBtn.enabled =
                    firstField.text != ''}"
                />
```

```
        </mx:FormItem>
        <mx:FormItem label="Last Name:">
            <mx:TextInput id="lastField"
                change="{submitBtn.enabled =
                    lastField.text != ''}"
                />
        </mx:FormItem>
        <mx:FormItem label="Age:">
            <mx:NumericStepper id="ageField"
                minimum="18" maximum="110"
                />
        </mx:FormItem>
        <mx:Button id="submitBtn" label="submit"
            enabled="false"
            click="submitHandler();"
            />
    </mx:Form>

</mx:WindowedApplication>
```

После того как пользователь ввел и отправил данные, вызывается обработчик события `submitHandler`, который обновляет объект `UserData` и вызывает метод `saveUserData`. Метод `saveUserData` создает новый объект `File` на основании свойств `firstName` и `lastName` экземпляра `UserData`, и файл записывается на рабочий стол с расширением .user. При открытии этого файла и его записи в буфер чтения значения свойств автоматически восстанавливаются в ходе десериализации. В следующем примере приложение дополняется возможностью открытия файлов .user и заполнением полей формы:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Script>
        <![CDATA[
            import com.oreilly.flexcookbook.UserData;

            [Bindable]
            private var userData:UserData = new UserData();
            private var file:File = File.desktopDirectory;
            private var filter:FileFilter = new
                 FileFilter("User File", "*.user");
            private static const EXT:String = ".user";

            private function submitHandler():void
            {
                userData.firstName = firstField.text;
                userData.lastName = lastField.text;
                userData.age = ageField.value;
                saveUserData();
            }
```

```
        private function saveUserData():void
        {
            var fnm:String = userData.firstName + "_" +
                             userData.lastName + EXT;
            var file:File = File.desktopDirectory.resolvePath( fnm );
            var stream:FileStream = new FileStream();
            stream.open( file, FileMode.WRITE );
            stream.writeObject( userData );
            stream.close();
        }

        private function openHandler():void
        {
            file.browseForOpen( "Open User", [filter] );
            file.addEventListener( Event.SELECT, selectHandler );
        }
        private function selectHandler( evt:Event ):void
        {
            var stream:FileStream = new FileStream();
            stream.open( file, FileMode.READ );
            userData = stream.readObject();
        }
    ]]>
</mx:Script>

<mx:Form>
    <mx:FormItem label="First Name:">
        <mx:TextInput id="firstField"
            text="{userData.firstName}"
            change="{submitBtn.enabled =
            firstField.text != ''}"
            />
    </mx:FormItem>
    <mx:FormItem label="Last Name:">
        <mx:TextInput id="lastField"
            text="{userData.lastName}"
            change="{submitBtn.enabled =
            lastField.text != ''}"
            />
    </mx:FormItem>
    <mx:FormItem label="Age:">
        <mx:NumericStepper id="ageField"
            value="{userData.age}"
            minimum="18" maximum="110"
            />
    </mx:FormItem>
    <mx:Button id="submitBtn" label="submit"
        enabled="false"
        click="submitHandler();"
        />
</mx:Form>
```

```
        <mx:HRule width="100%" />
        <mx:Button label="open" click="openHandler();" />

    </mx:WindowedApplication>
```

Когда пользователь открывает свой профиль, приложение вызывает метод `File.browseForOpen` для создания диалогового окна, в котором отображаются только файлы с расширением .user. При выборе открываемого файла вызывается метод `selectHandler`, и файл помещается в буфер чтения объекта `FileStream` методом `FileStream.readObject`. Метод `readObject` десериализует объект `UserData` и использует привязку данных для заполнения полей формы сохраненными значениями свойств.

Методы `FileStream.writeObject` и `FileStream.readObject` заметно упрощают сериализацию пользовательских объектов данных в файловой системе. Регистрируя класс при помощи тега метаданных `[RemoteClass]`, вы фактически объявляете о том, что объекты класса должны кодироваться в двоичный формат в кодировке AMF. Сериализация пользовательских объектов экономит время и усилия, которые пришлось бы потратить на загрузку и разбор текстовых файлов с парами «ключ/значение».

См. также

Рецепт 19.5.

19.7. Шифрование при локальном хранении данных

Задача

Требуется сохранить данные на жестком диске пользователя так, чтобы другие приложения не могли выполнять с ними операции чтения/записи.

Решение

Используйте шифрованное локальное хранилище данных приложения AIR для безопасного хранения информации.

Обсуждение

Устанавливаемые в системе приложения AIR получают доступ к шифрованному локальному хранилищу для безопасного хранения информации. Использование DPAPI (Data Protection API) для приложений AIR в системе Windows или Keychain для Mac OS X обеспечивает шифрование данных и их доступность только в пределах той же изолированной среды безопасности («песочницы»). Максимальный объем пространства, выделяемого под шифрованное локальное хранилище, составляет 10 Мбайт.

Данные хранятся в виде хеш-таблицы; чтение и запись данных осуществляется по строковому ключу. Данные сохраняются в виде сериализованных объектов `ByteArray`, что позволяет сохранять большинство встроенных объектных типов и пользовательские объекты с зарегистрированными псевдонимами классов. Для обращения к шифрованному локальному хранилищу используются статические методы класса `flash.data.EncryptedLocalStore`. Методам `getItem` и `setItem` при вызове передается строковый ключ для чтения и записи данных. Также поддерживается операция удаления данных по строковому ключу и очистки всего хранилища методами класса `EncryptedLocalStore`.

Пример использования `EncryptedLocalStore` для хранения пользовательских данных:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="completeHandler();">

    <mx:Script>
        <![CDATA[
            import com.oreilly.flexcookbook.UserData;

            [Bindable]
            public var userData:UserData;

            private function submitHandler():void
            {
                userData = new UserData(
                    firstField.text, lastField.text );

                var bytes:ByteArray = new ByteArray();
                bytes.writeObject( userData );
                EncryptedLocalStore.setItem( "user", bytes );

                views.selectedChild = userCanvas;
            }

            private function completeHandler():void
            {
                var user:ByteArray =
                    EncryptedLocalStore.getItem( "user" );
                if( user != null )
                {
                    userData = user.readObject() as UserData;
                    views.selectedChild = userCanvas;
                }
            }

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

```
        <mx:ViewStack id="views"
            width="300" height="300"
            backgroundColor="0xEEEEEE">
            <!-- Entry Form -->

            <mx:Form id="inputForm">
                <mx:FormItem label="First Name:">
                    <mx:TextInput id="firstField" />
                </mx:FormItem>
                <mx:FormItem label="Last Name:">
                    <mx:TextInput id="lastField" />
                </mx:FormItem>
                <mx:Button label="submit"
                    click="submitHandler();" />

            </mx:Form>
            <!-- Welcom Display -->
            <mx:VBox id="userCanvas">
                <mx:Label text="Hello," />
                <mx:HBox>
                    <mx:Label text="{userData.firstName}" />
                    <mx:Label text="{userData.lastName}" />
                </mx:HBox>
            </mx:VBox>
        </mx:ViewStack>

    </mx:WindowedApplication>
```

После того как приложение запустится и завершит формирование начальной раскладки, вызывается метод `completeHandler`. Он проверяет, присутствуют ли в шифрованном локальном хранилище данные с ключом `user`. Если такие данные имеются, приложение читает объект `ByteArray`, десериализует его и преобразует в объект `UserData`. Если же данные недоступны, пользователь может ввести информацию и передать ее для сохранения. В методе `submitHandler` данные сериализуются в объект `ByteArray` методом `ByteArray.writeObject` и сохраняются с ключом `user`. Любые данные, сериализуемые методами `ByteArray` – строки в кодировке UTF, логические значения, числа, – могут сохраняться в шифрованном локальном хранилище данных.

См. также

Рецепт 19.6.

19.8. Открытие и сохранение файлов

Задача

Требуется вызвать диалоговые окна для открытия или сохранения файлов.

Решение

Используйте методы класса `flash.filesystem.File`.

Обсуждение

Класс `File` дает возможность открыть диалоговое окно для выбора одного или нескольких открываемых файлов. При использовании метода `File.browseForOpen` выбор файла в диалоговом окне приводит к передаче события `select`. Когда пользователь выбирает файлы в окне, открытом методом `File.browseForOpenMultiple`, передается событие `selectMultiple`.

В следующем примере открывается диалоговое окно для выбора одного файла с заданным расширением:

```
private var file:File = new File();
private var filter:FileFilter = new FileFilter(
    "Text", "*.txt; *.xml; *.html" );

private function initHandler():void
{
    file.browseForOpen( "Open File", [filter] );
    file.addEventListener( Event.SELECT, selectHandler );
    file.addEventListener( Event.CANCEL, cancelHandler );
}

private function selectHandler( evt:Event ):void
{
    var stream:FileStream = new FileStream();
    stream.open( file, FileMode.READ );
    trace( stream.readUTFBytes( stream.bytesAvailable ) );
    stream.close();
}

private function cancelHandler( evt:Event ):void
{
    trace( "Browse cancelled." );
}
```

Приложение создает новый объект `File` и вызывает метод `File.browserForOpen` для вызова диалогового окна открытия файла. Класс `FileFilter` позволяет выбрать типы файлов, отображаемых в диалоговом окне, и ограничить пользователя выбором файлов с определенным расширением. Вы можете прослушивать событие `select`, передаваемое объектом `File` при выборе файла в диалоговом окне. В обработчике события `selectHandler` открывается новый объект `FileStream`, и в его буфер чтения помещается экземпляр `File`.

Вызов метода `File.browseForOpenMultiple` открывает диалоговое окно с возможностью выбора нескольких файлов. После выбора файлов передается событие `selectMultiple`, а выбранные объекты `File` содержатся в свойстве `files` объекта `FileListEvent`:

```
private var file:File;
private var filter:FileFilter = new FileFilter(
    "Text", "*.txt; *.xml; *.html" );

private function initHandler():void
{
    file = File.desktopDirectory;
    file.browseForOpenMultiple( "Open File", [filter] );
    file.addEventListener( FileListEvent.SELECT_MULTIPLE,
        selectHandler );
}

private function selectHandler( evt:FileListEvent ):void
{
    trace( "Selected files from:  " + file.url + "\n" );
    var files:Array = evt.files;
    for( var i:int = 0; i < files.length; i++ )
    {
        trace( ( files[i] as File ).name + "\n" );
    }
}
```

В этом примере открывается диалоговое окно с содержимым рабочего стола. После того как пользователь выберет файлы и подтвердит их открытие, вызывается метод `selectHandler`, который выводит свойство `name` каждого выбранного объекта `File` на отладочную консоль.

Класс `File` также обладает поддержкой диалогового окна сохранения файлов. В этом диалоговом окне пользователь выбирает каталог для сохранения и вводит в текстовом поле имя файла.

Событие `select` прослушивается так же, как при использовании метода `File.browseForOpen`. В следующем примере открывается диалоговое окно сохранения файла, а в заданный файл записывается строка *Hello World*:

```
private var file:File;

private function initHandler():void
{
    file = File.desktopDirectory;
    file.browseForSave( "Save As" );
    file.addEventListener( Event.SELECT,
        selectHandler );
}

private function selectHandler( evt:Event ):void
{
    var stream:FileStream = new FileStream();
    stream.open( evt.target as File, FileMode.WRITE );
    stream.writeUTFBytes( "Hello World." );
    stream.close();
}
```

Класс File предоставляет удобный способ организации интерфейса для выбора файлов, соответствующего диалоговым окнам операционной системы. Однако возможности графического представления файлов и каталогов не ограничиваются диалоговыми окнами, открытыми методами класса File. В SDK также входят другие классы, которые могут использоваться в приложениях AIR.

См. также

Рецепты 19.5 и 19.6.

19.9. Навигация по файловой системе в AIR

Задача

Требуется добавить компоненты для навигации по файловой системе и отображения содержимого ее каталогов.

Решение

Используйте компоненты файловой системы из SDK.

Обсуждение

AIR API из Flex 3 SDK включает компоненты для просмотра каталогов файловой системы компьютера. Компоненты, доступные для приложений AIR, сочетают функциональность списковых элементов Framework (таких, как Tree, List и DataGrid) со специфическими особенностями файловой системы. Хотя компоненты файловой системы выглядят и работают так же, как их аналоги из общих компонентов Flex, данные файловой системы передаются компонентам в свойстве directory (а не в свойстве dataProvider).

В следующем примере компоненты FileSystemComboBox и FileSystemList используются для организации перемещения по файловой системе и вывода содержимого каталогов:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="initHandler();">
    <mx:Script>
        <![CDATA[
            import mx.events.FileEvent;
            private function initHandler():void
            {
                fileCB.directory = File.documentsDirectory;
            }
            private function changeHandler( evt:FileEvent ):void
            {
                trace( evt.file.nativePath );
```

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

    <mx:FileSystemComboBox id="fileCB"
        directory="{fileList.directory}"
        directoryChange="changeHandler(event);"
        />
    <mx:FileSystemList id="fileList"
        directory="{fileCB.directory}"
        />

</mx:WindowedApplication>
```

Компонент `mx.controls.FileSystemComboBox` отображает иерархию каталогов в виде раскрывающегося списка. После загрузки и формирования начальной раскладки исходным каталогом компонента назначается каталог документов пользователя. Открыв список компонента, вы увидите в нем иерархическое дерево, ведущее к этому каталогу.

В данном примере при выборе строки в компоненте `FileSystemComboBox` происходит обновление каталога, отображаемого в `FileSystemList`, и вызывается метод `changeHandler`. Объект события, передаваемый с событием `directoryChange`, относится к типу `FileEvent`. Текущий выбранный каталог берется из свойства `file` объекта `FileEvent`, а путь к нему выводится на отладочную консоль. Благодаря двусторонней привязке свойств `directory` компонентов `FileSystemComboBox` и `FileSystemList` оба компонента обновляются при выборе пользователем нового каталога.

Наряду с отображением скрытых файлов или файлов с определенным расширением класс `FileSystemList` также позволяет работать с историей просмотра каталогов. Для решения этой задачи используются экземпляры класса `FileSystemHistoryButton`.

В следующем примере компоненты `FileSystemHistoryButton` упрощают возврат к каталогам, ранее просматривавшимся в компоненте `FileSystemList`:

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

    <mx:HBox width="100%">
        <mx:FileSystemHistoryButton label="Back"
            dataProvider="{fileList.backHistory}"
            enabled="{fileList.canNavigateBack}"
            click="fileList.navigateBack();"
            itemClick="fileList.navigateBack(event.index)"
            />
        <mx:FileSystemHistoryButton label="Forward"
            dataProvider="{fileList.forwardHistory}"
            enabled="{fileList.canNavigateForward}"
```

```
            click="fileList.navigateForward();"
            itemClick=
                "fileList.navigateForward(event.index)"
            />
    </mx:HBox>

    <mx:FileSystemList id="fileList"
        width="100%" height="250"
        directory="{File.documentsDirectory}"
        />

</mx:WindowedApplication>
```

Провайдер данных для кнопки истории просмотра представляет собой массив объектов `File`, выбиравшихся ранее в каталогах, отображаемых в `FileSystemList`. Внутренняя реализация компонента `FileSystemList` обновляет свойства `backHistory`, `forwardHistory`, `canNavigateBack` и `canNavigateForward`. Для доступа к конкретным данным каталогов, хранящимся в компоненте `FileSystemList`, используется раскрывающееся контекстное меню компонентов `FileSystemHistoryButton`. С событием `itemClick` экземпляра `FileSystemHistoryButton` передается объект `MenuEvent`. Свойство `index` объекта события дает возможность выбрать определенную точку в истории просмотра и обновить содержимое каталога, отображаемое компонентом `FileSystemList`.

Компонент `FileSystemList` отображает текущее содержимое (файлы и подкаталоги) заданного каталога, однако он не содержит информации об иерархической структуре каталогов в файловой системе. Для отображения дерева каталогов следует использовать компонент `FileSystemTree`. Настройка класса `FileSystemTree` позволяет включить режим показа скрытых файлов, файлов с определенным расширением, только каталогов, а также задать другие параметры фильтрации. Наряду с настройкой параметров отображения данных и навигации компонент `FileSystemTree` передает события `directoryClosing`, `directoryOpening` и `directoryChange`. Вы можете назначить обработчики для этих событий, передающих объект `FileEvent`.

В следующем примере экземпляр класса `FileSystemTree` заполняется содержимым корневого каталога файловой системы при создании экземпляра:

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

    <mx:FileSystemTree id="fileTree"
        width="100%" height="100%"
        directory="{FileSystemTree.COMPUTER}"
        />

</mx:WindowedApplication>
```

Компонент FileSystemTree является удобным средством просмотра иерархии выбранного каталога. Элементы содержимого компонента FileSystemTree помечаются именами файлов или папок. Для вывода дополнительной информации об объектах файлов и папок используется класс FileSystemDataGrid, который автоматически отображает экземпляры DataGridColumn для имени файла, типа, размера, даты создания и последнего изменения. Двойной щелчок на элементах данных компонента FileSystemDataGrid используется для перехода между каталогами. В следующем примере компонент FileSystemDataGrid используется для перехода к каталогу рабочего стола файловой системы:

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

    <mx:FileSystemDataGrid id="fileGrid"
        width="100%" height="100%"
        directory="{File.desktopDirectory}"
        />

</mx:WindowedApplication>
```

При двойном щелчке в строке изображение таблицы данных обновляется, и в ней отображаются файлы и папки выбранного каталога. Эта операция приводит к открытию каталога и отображению его содержимого, однако компонент не содержит компонентов для возврата по иерархии файловой системы. Как и FileSystemList, компонент FileSystemDataGrid ведет внутреннюю историю просмотра, а для взаимодействия с ней могут использоваться экземпляры FileSystemHistoryButton.

Компоненты файловой системы, включенные в AIR API, предоставляют удобный механизм перемещения по файловой системе компьютера. Следует помнить, что AIR не реагирует на оповещения файловой системы, и если во время использования компонентов файловой системы какой-либо файл или каталог будет удален, изображение не обновляется. Чтобы удостовериться в том, что операция не выполняется с несуществующими файлами, вызовите метод refresh, поддерживаемый компонентами FileSystemList, FileSystemTree и FileSystemDataGrid.

См. также

Рецепты 19.5 и 19.8.

19.10. Внешний API перетаскивания мышью

Задача

Требуется обеспечить перетаскивание данных мышью в окно приложения и за его пределы.

Решение

Добавьте данные в объект Clipboard и обеспечьте управление операциями перетаскивания с использованием класса NativeDragManager.

Обсуждение

Внешний API перетаскивания мышью позволяет реализовать пересылку данных между файловой системой и приложениями AIR. При выполнении жеста перетаскивания данные в некотором формате включаются в объект Clipboard и передаются NativeDragManager методом doDrag. Вы можете зарегистрировать слушателей событий, передаваемых NativeDragManager во время и при завершении операции перетаскивания. События представляются экземплярами класса NativeDragEvent; для обращения к данным Clipboard в обработчике события используется свойство NativeDragEvent.clipboard.

Жест перетаскивания инициируется при выделении элемента приложения мышью. Если пользователь удерживает нажатой кнопку мыши, операция входит в *фазу перетаскивания*. Операция перетаскивания может быть принята любым зарегистрированным компонентом, расширяющим класс flash.display.InteractiveObject. Жест перетаскивания завершается в момент отпускания кнопки мыши. Компонент, инициировавший жест перетаскивания, называется *источником* (инициатором) перетаскивания, а экземпляр InteractiveObject, принимающий операцию, называется *приемником перетаскивания*.

Flex Framework предоставляет поддержку перетаскивания в приложениях, однако внешний API перетаскивания позволяет перетаскивать данные между приложением AIR и файловой системой (или другими приложениями). При перетаскивании данных внутри приложения AIR рекомендуется использовать API перетаскивания Flex Framework. Если вы уже умеете пользоваться классом mx.managers.DragManager для передачи данных в приложениях Flex, вероятно, класс flash.desktop.NativeDragManager покажется вам знакомым. Одно из важных различий между этими классами заключается в том, что объекты DragSource добавляются в экземпляр DragManager, тогда как объекты Clipboard добавляются в экземпляр NativeDragManager.

В следующем примере класс NativeDragManager используется для включения в объект Clipboard графического файла, который будет перемещаться в каталог файловой системы посредством перетаскивания:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="initHandler();"
    closing="closeHandler();">

    <mx:Script>
        <![CDATA[
```

```
            import mx.graphics.codec.PNGEncoder;

            private var tempDir:File = File.createTempDirectory();
            private var imageData:BitmapData;

            private function initHandler():void
            {
                imageData = new BitmapData( image.width, image.height );
                imageData.draw( image );
            }

            private function closeHandler():void
            {
                tempDir.deleteDirectory();
            }

            private function clickHandler():void
            {
                var transfer:Clipboard = new Clipboard();
                transfer.setData(ClipboardFormats.FILE_LIST_FORMAT,
                    [getImageFile()], false );
                NativeDragManager.dropAction = NativeDragActions.COPY;
                NativeDragManager.doDrag( this, transfer, imageData );
            }

            private function getImageFile():File
            {
                var tempFile:File = tempDir.resolvePath("img.png" );
                var png:ByteArray = new PNGEncoder().encode(imageData );
                var stream:FileStream = new FileStream();
                stream.open( tempFile, FileMode.WRITE );
                stream.writeBytes( png );
                stream.close();
                return tempFile
            }
        ]]>
    </mx:Script>
    <mx:Image id="image"
        source="@Embed(source='assets/bigshakey.png')"
        buttonMode="true" useHandCursor="true"
        mouseDown="clickHandler();"
        />

</mx:WindowedApplication>
```

При передаче компонентом `Image` **события** `mouseDown` **вызывается метод** `clickHandler` **и создается новый объект** `Clipboard`**. Представление передаваемых данных создается методом** `Clipboard.setData`**. Формат данных объекта** `Clipboard` **представляет собой массив, состоящий из единственного элемента: графического файла, находящегося во временном каталоге файловой системы. Когда пользователь перетаскивает изображение из**

приложения в каталог файловой системы и отпускает кнопку мыши, графический файл перемещается в указанное место.

Данные также могут перетаскиваться в приложение из файловой системы или другого приложения AIR. Чтобы приложение принимало внешние операции перетаскивания, прослушивайте события перетаскивания, передаваемые классом `NativeDragManager`. Приложение из следующего примера принимает графические файлы от операций перетаскивания:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    windowComplete="initHandler();">

    <mx:Script>
        <![CDATA[
            import mx.controls.Image;

            private var loader:Loader;
            private var xposition:Number;
            private var yposition:Number;

            private function initHandler():void
            {
                addEventListener(
                    NativeDragEvent.NATIVE_DRAG_ENTER,
                    dragEnterHandler );
                addEventListener(
                    NativeDragEvent.NATIVE_DRAG_DROP,
                    dragDropHandler );
            }

            private function dragEnterHandler( evt:NativeDragEvent ):void
            {
                if( evt.clipboard.hasFormat(
                    ClipboardFormats.FILE_LIST_FORMAT ) )
                    NativeDragManager.acceptDragDrop( this );
            }

            private function dragDropHandler( evt:NativeDragEvent ):void
            {
                var pt:Point = globalToLocal(new Point(
                    evt.localX, evt.localY ));
                xposition = pt.x;
                yposition = pt.y;
                var files:Array = evt.clipboard.getData(
                    ClipboardFormats.FILE_LIST_FORMAT )
                    as Array;
                loader = new Loader();
                loader.contentLoaderInfo.addEventListener(
                    Event.COMPLETE,
                    completeHandler );
```

```
                loader.load( new URLRequest( files[0].url ) );
            }

            private function completeHandler(
                    evt:Event ):void
            {
                var bmp:Bitmap = loader.content as Bitmap;
                var image:Image = new Image();
                image.source = bmp;
                image.x = xposition;
                image.y = yposition;
                addChild( image );
            }

        ]]>
    </mx:Script>

</mx:WindowedApplication>
```

После того как приложение загрузится и сформирует начальную раскладку, назначаются обработчики событий `nativeDragEnterEvent` и `nativeDragDropEvent`. При перетаскивании файла в приложение последнему разрешается прослушивать события завершения перетаскивания, для чего метод `NativeDragManager.acceptDragDrop` вызывается с передачей объекта самого приложения. При выполнении действия `dragDropEvent` передаваемые данные интерпретируются как массив объектов `File`; первый элемент массива загружается в компонент `Image`.

Хотя примеры этого рецепта демонстрируют взаимодействие приложений AIR с файловой системой при передаче данных с использованием `NativeDragManager`, ваши возможности не ограничиваются работой с объектами `File`. Действия перетаскивания также могут использоваться для передачи растровой графики, строк с кодом HTML, URL-адресов или простого текста.

См. также

Рецепт 19.5.

19.11. Взаимодействие с буфером обмена операционной системы

Задача

Требуется поместить и прочитать данные из буфера обмена операционной системы.

Решение

Используйте статическое свойство `generalClipboard` класса `Clipboard`.

Обсуждение

Передача объектов и данных может осуществляться не только операциями перетаскивания, но и взаимодействием с буфером обмена операционной системы через статическое свойство `generalClipboard` класса `flash.desktop.Clipboard`. Поддерживается копирование данных в приложения AIR и из них в стандартных форматах растровой графики, списков файлов и текста (простого, HTML и в форме URL-адресов). Такие данные преобразуются в объекты `BitmapData`, массив объектов `File` и `String` соответственно. Также можно добавлять пользовательские форматы данных, передаваемые в объектах `Clipboard`, однако эти данные будут доступны только другим приложениям AIR, которые умеют работать с этим форматом.

Следующий пример позволяет добавлять, читать и удалять данные из буфера обмена операционной системы:

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

    <mx:Script>
        <![CDATA[

            private function addHandler():void
            {
                Clipboard.generalClipboard.setData(
                    ClipboardFormats.TEXT_FORMAT, textField.text );
            }

            private function removeHandler():void
            {
                Clipboard.generalClipboard.clear();
            }

            private function pasteHandler():void
            {
                if( Clipboard.generalClipboard.hasFormat(
                    ClipboardFormats.TEXT_FORMAT ) )
                {
                    textField.text =
                        Clipboard.generalClipboard.getData(
                            ClipboardFormats.TEXT_FORMAT ) as String;
                }
            }

        ]]>
    </mx:Script>
    <mx:TextArea id="textField"
        width="100%" height="100%"
        />
```

```
        <mx:Button label="add to clipboard"
            click="addHandler();"
            />
        <mx:Button label="remove from clipboard"
            click="removeHandler();"
            />
        <mx:Button label="past from clipboard"
            click="pasteHandler();"
            />
    </mx:WindowedApplication>
```

Обработчик события `addHandler` помещает текстовую строку, отображаемую в компоненте `<mx:TextArea>`, в буфер обмена операционной системы. Скопированные данные могут быть вставлены в любое другое приложение независимо от того, работает оно в среде Adobe AIR или нет. Также приложение может вставить в компонент `<mx:TextArea>` строковые данные, скопированные в буфер другим приложением. В методе `pasteHandler` формат данных, находящихся в буфере обмена, проверяется методом `hasFormat`. Если это строковые данные, они преобразуются в объект ActionScript `String` и вставляются в компонент. Очистка буфера выполняется методом `clear`, как это сделано в методе `removeHandler`.

Типы формата данных не ограничиваются константами из перечисления в классе `ClipboardFormats` – можно использовать любое строковое значение, поддерживаемое несколькими приложениями AIR. Допустим, информация о пользователе обрабатывается двумя и более приложениями. Чтобы обеспечить передачу информации между приложениями через буфер обмена операционной системы, поместите данные в буфер так, как это сделано в следующем фрагменте:

```
Clipboard.generalClipboard.setData( "userObject",
new UserObject( 'Ted', 'Henderson' ) );
```

Любое приложение AIR, которое умеет обрабатывать данные в формате `userObject`, сможет обращаться к экземплярам `UserObject` в буфере обмена операционной системы.

См. также

Рецепт 19.10.

19.12. Отображение HTML

Задача

Требуется вывести в приложении контент в формате HTML.

Решение

Используйте компонент `<mx:HTML>` для загрузки и отображения контента HTML в контейнере с прокруткой.

Обсуждение

Среда Adobe AIR поддерживает отображение данных в формате HTML в приложениях на базе SWF. Механизм отображения создан на основе технологии WebKit и поддерживает контент, который может воспроизводиться в любом броузере на базе WebKit (например, Safari). Центральное место в нем занимает класс `flash.html.HTMLLoader` из AIR HTML API. Объект `HTMLLoader`, являющийся расширением класса `flash.display.Sprite`, может включаться в приложения на базе ActionScript и Flex для среды Adobe AIR.

Также в API входит компонент `<mx:HTML>` для удобного добавления контента HTML в контейнеры с прокруткой в приложениях, использующих Flex Framework. Класс `mx.controls.HTML` во внутренней реализации взаимодействует с экземпляром класса `HTMLLoader` и предоставляет доступ к объектам HTML DOM (Document Object Model). Это дает возможность взаимодействовать с объектами JavaScript, доступными в DOM, и стилями CSS, примененными к странице HTML. Ваши возможности не ограничиваются загрузкой страниц HTML с сервера; страницы также могут загружаться из локальной «песочницы» и даже из отформатированных строк.

Чтобы загрузить страницу HTML в компонент `<mx:HTML>`, задайте значение свойства `location`. Строка `location` передается внутреннему экземпляру `HTMLLoader`, который загружает и выводит запрашиваемую страницу. В следующем примере веб-страница загружается в компонент `<mx:HTML>`, а другие компоненты обеспечивают возможность перехода на другие страницы:

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

    <mx:Script>
        <![CDATA[
            [Bindable]
            public var urlLocation:String =
                "http://www.adobe.com";

        ]]>
    </mx:Script>

    <mx:Form width="100%">
        <mx:FormItem width="100%">
            <mx:HBox width="100%">
                <mx:TextInput id="urlField"
                    width="100%"
                    text="{html.location}"/>
                <mx:Button label="go"
                    click="{urlLocation = urlField.text}"
                    />
```

```
            </mx:HBox>
        </mx:FormItem>
    </mx:Form>
    <mx:HTML id="html"
        width="100%" height="100%"
        location="{urlLocation}"
        />

</mx:WindowedApplication>
```

Приложение при запуске загружает веб-страницу по адресу *http://www.adobe.com*, а его компоненты позволяют выбрать и загрузить другие страницы HTML посредством изменения свойства `location` компонента `<mx:HTML>`. После загрузки страницы пользователь может взаимодействовать с контентом HTML точно так же, как в броузере. Экземпляр `HTMLLoader` также ведет внутреннюю историю просмотра и предоставляет удобные средства для возврата к ранее просматривавшимся страницам. В следующем примере добавляются компоненты для перехода в прямом и обратном направлении по истории объекта `HTMLLoader` с использованием методов компонента `<mx:HTML>`:

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

    <mx:Script>
        <![CDATA[

            [Bindable]
            public var urlLocation:String =
                "http://www.adobe.com";

        ]]>
    </mx:Script>
    <mx:Form width="100%">
        <mx:FormItem width="100%">
            <mx:HBox width="100%">
                <mx:Button label="back"
                    click="html.historyBack();"
                    />
                <mx:Button label="forward"
                    click="html.historyForward();"
                    />
                <mx:TextInput id="urlField"
                    width="100%"
                    text="{html.location}"/>
                <mx:Button label="go"
                    click="{urlLocation = urlField.text}"
                    />
            </mx:HBox>
        </mx:FormItem>
```

```
        </mx:Form>
        <mx:HTML id="html"
            width="100%" height="100%"
            location="{urlLocation}"
            />

    </mx:WindowedApplication>
```

Методы `HTML.historyBack` и `HTML.historyForward` перебирают историю просмотра, переданную объекту `HTMLLoader`. Если вы знакомы с построением приложений Ajax, управление историей в классе `HTMLLoader` организовано аналогично объекту JavaScript `window.history`.

Кроме методов перемещения в прямом и обратном направлении по истории просмотра, также предусмотрены методы для получения длины истории, определения текущей позиции и загрузки заданной страницы (`historyGo`). Этот пример наглядно показывает, как быстро реализуется простейший броузер на базе SWF.

Экземпляр `HTMLLoader` передает несколько событий в фазах загрузки и воспроизведения, при обновлении свойства `location`. Вы можете прослушивать эти события и назначать обработчики в объявлениях компонента `<mx:HTML>`. Обратите внимание на порядок следования этих событий и на то, в какой момент появляется возможность взаимодействия с загруженной моделью HTML DOM:

```
<mx:HTML id="html"
    width="100%" height="100%"
    location="http://www.adobe.com"
    htmlDOMInitialize="initHandler();"
    locationChange="changeHandler();"
    complete="completeHandler();"
    htmlRender="renderHandler();"
    />
```

Событие `locationChange` передается при обновлении свойства `location` внутреннего экземпляра `HTMLLoader`. Событие `htmlDOMInitialized` передается после создания документа, но до того, как вы сможете взаимодействовать с объектами HTML DOM. Взаимодействие с HTML DOM становится возможным после передачи события `complete`. Событие `htmlRender` передается после формирования начального изображения и всех последующих перерисовок контента HTML.

19.13. Взаимодействие между ActionScript и JavaScript

Задача

Требуется обращаться к узлам элементов HTML, переменным и функциям JavaScript и изменять стили CSS загруженной страницы HTML.

Решение

Прослушивайте событие `complete`. Для обращения к HTML DOM используется свойство `domWindow` компонента `<mx:HTML>`.

Обсуждение

Класс `HTMLLoader` предоставляет доступ к объектам DOM загруженного документа HTML. Вы можете обращаться к узловым элементам окна HTML и взаимодействовать с кодом JavaScript, содержащимся в странице, обращаться к переменным и методам, а также вызывать методы ActionScript из JavaScript.

Обращение к глобальному объекту JavaScript документа HTML, загруженного в компоненте `<mx:HTML>`, осуществляется через свойство `domWindow`. Свойство `domWindow` имеет обобщенный тип `Object`, а для обращения к свойствам HTML DOM используется такая же точечная запись, как и для других объектов ActionScript. Используя свойство `domWindow`, вы можете обращаться к узловым элементам HTML, переменным и функциям объектов JavaScript и стилевых таблиц CSS в загруженном документе. Предположим, в компоненте HTML приложения загружается следующий документ HTML:

```
<html>
    <body>
        <p id="helloField">Hello World</p>
    </body>
</html>
```

Эта простая страница выводит текст *Hello World* при загрузке в броузере или компоненте `<mx:HTML>` приложения AIR. Обращение к элементу `helloField` этой страницы после загрузки контента осуществляется методом `getElementId`, так же как в JavaScript:

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

    <mx:Script>
        <![CDATA[
            private function completeHandler():void
            {
                var p1:String =
                   html.domWindow.document.getElementById(
                   'helloField').innerHTML;
                trace( p1 );
            }

        ]]>
    </mx:Script>

    <mx:HTML id="html"
```

```
        width="100%" height="100%"
        location="test.html"
        complete="completeHandler();"
        />

</mx:WindowedApplication>
```

Событие `complete` передается внутренним экземпляром `HTMLLoader` компонента `<mx:HTML>` при завершении загрузки контента и доступности DOM. В методе `completeHandler` приложение обращается к значению узла `helloField` загруженного документа `test.html` и выводит строку на отладочной консоли. Свойство `domWindow` компонента `<mx:HTML>` содержит ActionScript-представление окна HTML DOM. Доступ к данным, содержащимся в загруженном документе HTML, не ограничивается только чтением; вы также можете изменить текущее содержимое элемента `helloField`:

```
html.domWindow.document.getElementById('helloField').innerHTML = "Hola!";
```

Для обращения к стилевым таблицам CSS используется свойство `styleSheets`. Его значение представляет собой массив, элементы которого соответствуют порядку объявления каскадных стилевых таблиц в документе. В следующем фрагменте документ HTML дополняется стилем для элемента `helloField`:

```
<html>
    <style>
        #helloField {
            font-size: 24px;
            color: #FF0000;
        }
    </style>
    <body>
        <p id="helloField">Hello World</p>
    </body>
</html>
```

Чтобы изменить стиль, примененный к элементу `helloField`, вы обращаетесь к стилевой таблице свойства `styleSheets`:

```
var styleSheets:Object = html.domWindow.document.styleSheets;
trace( styleSheets[0].cssRules[0].style.fontSize );
styleSheets[0].cssRules[0].style.color = "#FFCCFF";
```

Каждое объявление стиля в таблице CSS помещается в массив, а для его выборки используется свойство `cssRules`. Обновление стилевых свойств в объявлениях CSS осуществляется через объект `style`.

Изменения стилей не сохраняются в истории экземпляра `HTMLLoader`. Если свойство `location` компонента `<mx:HTML>` было обновлено, то любые изменения в оформлении документа HTML теряются при вызове метода `HTML.historyBack`.

Свойство domWindow также позволяет обращаться к переменным и функциям JavaScript из документа HTML. В документ HTML из нашего примера добавляется свойство age и метод для его изменения:

```
<html>
    <script>
        var age = 18;
        function setAge( num )
        {
            age = num;
            handleAgeChange( age );
        }
        function sayHello()
        {
            return "Hello";
        }
    </script>
    <body>
        <p id="helloField">Hello World</p>
    </body>
</html>
```

Вероятно, вы заметили вызов несуществующей функции handleAgeChange в методе setAge. Хотя методы ActionScript могут назначаться делегатами для вызовов функций JavaScript из HTML DOM, если при вызове метода setAge не найдется ни функции JavaScript, ни делегированного обработчика ActionScript, произойдет исключение ReferenceError. Аналогично, при обращении к несуществующему объекту JavaScript через свойство domWindow произойдет ошибка. Это создает зависимость между сценариями ActionScript и JavaScript, но поскольку свойства относятся к обобщенному типу Object, формируемая логическая привязка не является жесткой и приложение не зависит от конкретного документа HTML или наоборот.

В следующем примере приложение загружает документ test.html, обновляет свойство age и вызывает метод sayHello для объекта JavaScript:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">
    <mx:Script>
        <![CDATA[

            private function completeHandler():void
            {
                trace( html.domWindow.sayHello() );

                html.domWindow.handleAgeChange = ageHandler;
                trace( "Current Age: " + html.domWindow.age );
                html.domWindow.setAge( 30 );
            }
            private function ageHandler( value:Object ):void
```

```
            {
                trace( "Age changed = " + value );
            }

        ]]>
    </mx:Script>

    <mx:HTML id="html"
        width="100%" height="100%"
        location="test.html"
        complete="completeHandler();"
        />

</mx:WindowedApplication>
```

При вызове метода JavaScript `setAge` приложение не выдает ошибки, потому что функция `handleAgeChange` передает управление методу `ageHandler` в коде ActionScript. Между JavaScript и ActionScript можно передавать данные любого типа, но данные, не относящиеся к простым типам (например, объекты `Date`), необходимо преобразовать к соответствующему типу.

См. также

Рецепт 19.12.

19.14. Работа с локальными базами данных SQL

Задача

Требуется сохранять и загружать данные, используемые вашим приложением.

Решение

Создайте файл базы данных на жестком диске пользователя. Выполните команды в стандартном синтаксисе SQL (Structured Query Language).

Обсуждение

Среда Adobe AIR включает ядро баз данных SQL, позволяющее создавать локальные базы данных для хранения информации. База данных хранится в файле, а доступ к ней не ограничивается отдельным каталогом файловой системы, благодаря чему любое приложение может работать с базой данных. Ядро SQL в AIR позволяет создавать реляционные базы данных, использующие стандартные команды SQL для хранения и выборки сложных данных.

В API баз данных SQL включены классы для создания и открытия файлов баз данных, определения команд, прослушивания событий и получе-

ния информации о схеме базы данных. Файл базы данных создается при помощи класса flash.silesystem.File и может иметь любое разрешение. Подключение к базе данных открывается вызовом методов open или openAsync класса flash.dataSQLConnection с передачей пути к файлу. Если методам open и openAsync не передается ссылка на файл, методы создают базу данных в памяти и открывают ее для выполнения команд. Команды выполняются в синхронном или асинхронном режиме. Во время выполнения команды можно прослушивать события успеха и неудачи, но в асинхронном режиме команды выполняются вне основного программного потока, что позволяет выполнять другой код во время обработки команды.

В следующем примере приложение создает новый файл базы данных, если он не был создан ранее, и открывает подключение:

```
var db:File = File.applicationStorageDirectory.
    resolvePath( "Authors.db" );

var sqlConn:SQLConnection = new SQLConnection();
sqlConn.addEventListener( SQLEvent.OPEN, openHandler );
sqlConn.addEventListener( SQLErrorEvent.ERROR, errorHandler );
sqlConn.openAsync( db );

private function openHandler( evt:SQLEvent ):void
{
    trace( "Database created." );
}
private function errorHandler( evt:SQLErrorEvent ):void
{
    trace( "Error " + evt.error.message + " :: " + evt.error.details );
}
```

Экземпляр SQLConnection создает файл с именем Authors.db в каталоге хранения данных, после чего файл открывается методом openAsync. Если операция была выполнена успешно, передается событие SQLEvent и вызывается метод openHandler. Если во время выполнения произойдет ошибка, передается событие SQLErrorEvent. Свойство error объекта SQLErrorEvent содержит объект SQLError со свойствами, унаследованными от класса flash.errors.Error (например, message), и свойствами с описанием неудачной операции.

Чтобы выполнить команду SQL, необходимо задать значение свойства text объекта SQLStatement и вызвать метод execute. В следующем примере команда CREATE TABLE языка SQL создает новую таблицу в базе данных:

```
var db:File = File.applicationStorageDirectory.
    resolvePath( "Authors.db" );

var sqlConn:SQLConnection = new SQLConnection();
sqlConn.addEventListener( SQLEvent.OPEN, openHandler );
sqlConn.addEventListener( SQLErrorEvent.ERROR, errorHandler );
sqlConn.openAsync( db );
```

```
private function openHandler( evt:SQLEvent ):void
{
    var sql:String = "CREATE TABLE IF NOT EXISTS authors (" +
                        "authorId     INTEGER     PRIMARY KEY," +
                        "firstName    TEXT        NOT NULL," +
                        "lastName     TEXT        NOT NULL" +
                        ");";

    var statement:SQLStatement = new SQLStatement();
    statement.sqlConnection = sqlConn;
    statement.text = sql;
    statement.addEventListener( SQLEvent.RESULT, resultHandler );
    statement.addEventListener( SQLErrorEvent.ERROR, errorHandler );

    statement.execute();
}
private function resultHandler( evt:SQLEvent ):void
{
    trace( "Table created." );
}

private function errorHandler( evt:SQLErrorEvent ):void
{
    trace( "Error " + evt.error.message +
           ":: " + evt.error.details );
}
```

При открытии подключения к базе данных выполняется команда SQL, которая создает новую таблицу `authors` с именами столбцов (т. е. свойств) `authorId`, `firstName` и `lastName`. Чтобы выполнить команду SQL, задайте ее текст свойству `text` объекта `SQLStatement`. Экземпляр `SQLConnection` передается объекту `SQLStatement` в свойстве `sqlConnection`; кроме того, перед выполнением создаются слушатели событий. Так как подключение было открыто асинхронно методом `SQLConnection.openAsync`, команда создания таблицы тоже будет выполнена асинхронно, и во время операции сможет выполняться другой код.

Запросы формулируются на декларативном языке SQL. Помимо команды `CREATE TABLE`, при работе с базами данных часто используются команды `INSERT`, `SELECT`, `UPDATE` и `DELETE`. Пример выполнения запроса `INSERT` с включением новых данных в таблицу:

```
private var insertQuery:SQLStatement = new SQLStatement();

private function addAuthor( fName:String, lName:String ):void
{

    var sql:String = "INSERT INTO authors VALUES (" +
                            "null," +
                            "'" + fName + "'," +
                            "'" + lName + "'" +
                            ");";
```

```
        insertQuery.sqlConnection = sqlConn;
        insertQuery.text = sql;
        insertQuery.addEventListener( SQLEvent.RESULT, insertHandler );
        insertQuery.addEventListener( SQLErrorEvent.ERROR, errorHandler );

        insertQuery.execute();
    }
    private function insertHandler( evt:SQLEvent ):void
    {
        var result:SQLResult = insertQuery.getResult();
        trace( "Row ID : " + result.lastInsertRowID + " / " +
                "# Rows Affected : " + result.rowsAffected );
    }
```

Новые записи в таблице `authors` добавляются командой SQL `INSERT INTO`. Если операция была выполнена успешно, вызывается метод `insertHandler`, и метод `getResult` экземпляра `SQLStatement`, использованного в запросе, возвращает объект `SQLResult`.

Для удобства и повышения эффективности выполнения запросов в класс `SQLStatement` включены свойства `parameters` и `itemClass`. В подобных ситуациях, когда одна команда может вызываться многократно для разных входных данных, в классе `SQLStatement` объявляется общая команда, а свойство `parameters` используется для изменения параметров при выполнении. Свойство `parameters` представляет собой ассоциативный массив, в котором хранятся пары «ключ/значение» (для именованных и неименованных параметров).

Вот как выглядит предыдущий пример с использованием именованных параметров:

```
var insertSql:String = "INSERT INTO authors VALUES (" +
                    "null,:firstName,:lastName);";

insertQuery.sqlConnection = sqlConn;
insertQuery.text = insertSql;
insertQuery.parameters[":firstName"] = fName;
insertQuery.parameters[":lastName"] = lName;
```

Значения заменяются при выполнении команды; перед именем свойства ставится префикс `:` или `@`. Также возможен другой вариант: значения сохраняются с числовыми индексами, а их позиция в команде SQL представляется символами `?`:

```
var insertSql:String = "INSERT INTO authors VALUES (" + "null,?,?);";

insertQuery.sqlConnection = sqlConn;
insertQuery.text = insertSql;
insertQuery.parameters[0] = fName;
insertQuery.parameters[1] = lName;
```

В ходе выполнение команды каждый знак `?` в команде заменяется значением элемента с соответствующим индексом. Параметризация команд

SQL не только повышает эффективность выполнения, но и обеспечивает дополнительную защиту от атак внедрения SQL.

Если приложение содержит объекты данных, связанные с содержимым таблицы локальной базы данных, свойство `itemClass` класса `SQLStatement` предоставляет удобные средства использования результатов запроса `SELECT`. В следующем фрагменте свойство `itemClass` связывает информацию, полученную из базы данных, с классом `com.oreilly.flexcookbook.Author`:

```
import com.oreilly.flexcookbook.Author;

private var selectQuery:SQLStatement = new SQLStatement();

private function getAuthors():void
{
    var sql:String = "SELECT authorId, firstName, lastName FROM authors";
    selectQuery.sqlConnection = sqlConn;
    selectQuery.text = sql;
    selectQuery.itemClass = Author;
    selectQuery.addEventListener( SQLEvent.RESULT, selectHandler );
    selectQuery.addEventListener( SQLErrorEvent.ERROR, errorHandler );

    selectQuery.execute();
}

private function selectHandler( evt:SQLEvent ):void
{
    var authors:Array = selectQuery.getResult().data;
    for( var i:int = 0; i < authors.length; i++ )
    {
        var author:Author = authors[i] as Author;
        trace( author.firstName + " " + author.lastName );
    }
}
```

После успешного выполнения запроса вызывается метод `selectHandler`, и метод `getResult` экземпляра `SQLStatement`, использованного в запросе, возвращает объект `SQLResult`. Свойство `data` объекта `SQLResult` содержит массив записей, прочитанных из базы данных, в данном примере массив с описанием авторов. Каждому элементу массива ставится в соответствие объект класса `Author`, указанного в свойстве `itemClass` перед выполнением команды.

К сожалению, привести полное описание языка SQL и его команд, поддерживаемых в Adobe AIR, в книге невозможно. Тем не менее этот рецепт дает представление о некоторых важных приемах выполнения команд к локальным базам данных.

См. также

Рецепты 19.5–19.7.

19.15. Обнаружение и отслеживание сетевых подключений

Задача

Требуется обнаружить подключение к Интернету и следить за его доступностью.

Решение

Используйте классы `ServiceMonitor`, `SocketMonitor` и `URLMonitor` из AIR API мониторинга.

Обсуждение

Adobe AIR включает классы для проверки доступности подключения к сетевому ресурсу. Наряду с оповещениями пользователей об изменениях состояния, эти классы также позволяют разработчику создавать приложения с поддержкой *периодического подключения*, т. е. организации беспроблемной рабочей среды, в которой сетевой ресурс используется тогда, когда он доступен, а в остальное время приложение работает с сохраненной локальной копией. Локальные данные могут включать сериализованные объекты, хранящиеся в отдельных файлах или в зашифрованном хранилище данных, и информацию, введенную в локальную базу данных.

Для обнаружения изменений в сетевом подключении прослушивается событие `networkChange`, передаваемое экземпляром `NativeApplication`. Это событие передается тогда, когда подключение становится или перестает быть доступным. Событие не содержит информации о доступности подключения, поэтому для проверки возможности работы с нужным ресурсом следует использовать запросы с обработчиками событий.

В следующем примере создается слушатель для события `networkChange`:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    networkChange="networkChangeHandler();">

    <mx:Script>
        <![CDATA[
            private function networkChangeHandler(
                evt:Event ):void
            {
                // Проверка подключения
            }
        ]]>
    </mx:Script>

</mx:WindowedApplication>
```

Обработчик события networkChange в этом примере объявляется в корневом теге <mx:WindowedApplication> и вызывается каждый раз, когда NativeApplication обнаружит изменение в сетевом подключении. При запуске приложения сетевой ресурс не проверяется; если его доступность необходима для работы приложения, выполните соответствующие действия после завершения загрузки и проверьте наличие связи.

Событие networkChange оповещает приложение только об изменении в сети, а не о доступности или недоступности ресурса. В зависимости от специфики приложения для проверки доступности службы можно использовать классы SocketMonitor и URLMonitor. Класс ServiceMonitor является базовым для всех классов мониторинга; он предоставляет удобный механизм постоянного опроса доступности. Для обнаружения изменений в состоянии подключений HTTP используются объекты URLMonitor, производного от ServiceMonitor. Следующий фрагмент проверяет доступность сайта посредством запроса заголовка HTTP:

```
private var monitor:URLMonitor;

private function startMonitor():void
{
    var req:URLRequest = new URLRequest( "http://www.adobe.com" );
    req.method = URLRequestMethod.HEAD;
    monitor = new URLMonitor( req );
    monitor.pollInterval = 30000;
    monitor.addEventListener( StatusEvent.STATUS, statusHandler );
    monitor.start();
}

private function statusHandler( evt:StatusEvent ):void
{
    trace( "Available: " + monitor.available );
    trace( "Event code: " + evt.code );
}
```

Значение свойства pollInterval объекта URLMonitor задается в миллисекундах. В этом примере заголовок HTTP запрашивается с интервалом в 30 секунд. Служба периодически проверяет подключение, но передает событие StatusEvent только при исходном запросе и при последующих изменениях в доступности подключения. В данном примере метод statusHandler будет в первый раз вызван через 30 секунд после вызова метода staerMonitor. В дальнейшем метод statusHandler будет вызываться только при изменении свойства available экземпляра URLMonitor, например при изменении состояния сетевого подключения.

Использование экземпляра SocketMonitor для проверки подключения к сокету напоминает проверку состояния HTTP с экземпляром URLMonitor; при создании экземпляра задаются аргументы host и port:

```
socketMonitor = new SocketMonitor( "www.adobe.com", 1025 );
socketMonitor.addEventListener( StatusEvent.STATUS, statusHandler );
socketMonitor.start();
```

19.16. Проверка бездействия пользователя

Задача

Требуется узнать, когда пользователь перестал взаимодействовать с приложением.

Решение

Задайте свойство `idleThreshold` экземпляра `NativeApplication` и прослушивайте события `userIdle` и `userPresent`.

Обсуждение

Присутствие пользователя обнаруживается по действиям с мышью и клавиатурой. Бездействие определяется отсутствием таких действий в течение некоторого времени. Экземпляр `NativeApplication`, созданный при запуске приложения, содержит свойства и методы, специфические для данного приложения и доступные в любой его точке. Помимо событий изменения состояния сетевых подключений и активизации приложения также передаются события активности пользователя.

В следующем примере экземпляр `NativeApplication` используется для обнаружения активности пользователя:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="completeHandler();">

    <mx:Script>
        <![CDATA[
            private function completeHandler():void
            {
                NativeApplication.nativeApplication.idleThreshold = 10;
                NativeApplication.nativeApplication.addEventListener(
                    Event.USER_IDLE, idleHandler );
                NativeApplication.nativeApplication.addEventListener(
                    Event.USER_PRESENT, presenceHandler );
            }

            private function idleHandler( evt:Event ):void
            {
                trace( "Hello?!?!" );
            }
            private function presenceHandler( evt:Event ):void
            {
                trace( "Welcome Back!" );
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

После запуска приложения и формирования начальной раскладки вызывается метод `completeHandler`, который задает продолжительность периода бездействия и создает слушателей для событий активности пользователя, передаваемых экземпляром `NativeApplication`. Свойство `idleThreshold` задается в секундах, через 10 секунд бездействия пользователя вызывается метод `idleHandler`. Когда после выявленного отсутствия активности операционная система обнаружит событие от клавиатуры или операцию с мышью, вызывается метод `presenceHandler`, который выводит приветствие для вернувшегося пользователя. События активности пользователя особенно полезны в приложениях, которые работают только при непосредственном присутствии пользователя за компьютером.

19.17. Создание фоновых приложений

Задача

Приложение должно работать в фоновом режиме и может не обладать основным интерфейсом.

Решение

Задайте атрибут `visible=false` в корневом теге `<mx:WindowedApplication>` и файле дескриптора приложения. Используйте классы `DockIcon` и `SystemTrayIcon` для создания значка приложения.

Обсуждение

Среда AIR позволяет создавать приложения, не обладающие основным пользовательским интерфейсом, выполняемые в фоновом режиме. Такие приложения представляются значками в системной панели (system tray) или доке (dock). Значки приложений поддерживаются как в Mac OS X, так и в Windows, но в этих операционных системах используются разные конвенции пользовательского интерфейса. Соответственно AIR предоставляет классы для представления значков приложений конкретной операционной системы: класс `DockIcon` для приложений, выполняемых в Mac OS X, и класс `SystemTrayIcon` для приложений Windows. Чтобы узнать, какой тип значков используется операционной системой, в которой работает приложение, используйте свойства `supportsDockIcon` и `supportsSystemTrayIcon` класса `NativeApplication`.

Классы `DockIcon` и `SystemTrayIcon` расширяют абстрактный базовый класс `flash.desktop.InteractiveIcon`. Свойство `icon` экземпляра `NativeApplication` содержит ссылку на значок приложения, поддерживаемый операционной системой. Графическое изображение значка задается свойством `bitmaps`. Массив `bitmaps` содержит объекты `BitmapData` со стандартными размерами значков приложений операционной системы. Если свойство `bitmaps` оставлено пустым, в док-панели Mac OS X выводится значок по умолчанию, а в системной панели Windows значок не отображается.

Помимо графики значка, для значка приложения также можно определить контекстные меню, вызываемые щелчком на значке. Для контекстных меню можно прослушивать события выбора и выполнения команд и реагировать на них так, как требует приложение. В следующем примере представлено приложение, которое выполняется в системной панели, а его меню предоставляет команду выхода из приложения:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    visible="false"
    windowComplete="completeHandler();">

    <mx:Script>
        <![CDATA[
            [Embed(source='assets/AIRApp_16.png')]
            private var icon16:Class;
            [Embed(source='assets/AIRApp_32.png')]
            private var icon32:Class;
            [Embed(source='assets/AIRApp_48.png')]
            private var icon48:Class;
            [Embed(source='assets/AIRApp_128.png')]
            private var icon128:Class;

            private function completeHandler():void
            {
                var shellMenu:NativeMenu = createShellMenu();

                var icon:InteractiveIcon =
                   NativeApplication.nativeApplication.icon;
                if( NativeApplication.supportsDockIcon )
                {
                    ( icon as DockIcon ).menu = shellMenu;
                }
                else
                {
                   ( icon as SystemTrayIcon ).menu = shellMenu;
                   ( icon as SystemTrayIcon ).tooltip = "My App";
                }

                var bitmaps:Array =
                    [new icon16(), new icon32(),
                    new icon48(), new icon128()];
                icon.bitmaps = bitmaps;
            }

            private function createShellMenu():NativeMenu
            {
                var menu:NativeMenu = new NativeMenu();
                var quitCmd:NativeMenuItem = new NativeMenuItem( "Quit" );
                quitCmd.addEventListener( Event.SELECT, quitHandler );
```

```
                menu.addItem( quitCmd );
                return menu;
            }

            private function quitHandler( evt:Event ):void
            {
                NativeApplication.nativeApplication.exit();
            }
        ]]>
    </mx:Script>

</mx:WindowedApplication>
```

После запуска и формирования начальной раскладки приложение определяет текущую операционную систему при помощи свойства `NativeApplication.supportsDockIcon`. Если значение равно `true`, значит, приложение работает в Mac OS X, а свойство `icon` экземпляра `NativeApplication` содержит объект `DockIcon`. Метод `createShellMenu` создает и возвращает контекстное меню операционной системы, которое включается в свойство `menu` значка приложения. В меню добавляется единственная команда `quit`, при выборе которой пользователем вызывается метод `quitHandler`.

Среди важных свойств, задаваемых в корневом теге `<mx:WindowedApplication>`, следует обратить внимание на свойство `visible`. Если это свойство равно `false`, главное окно приложения скрывается, а весь доступ к приложению ограничивается системной панелью или док-панелью. Свойству `visible` также необходимо задать значение `false` и в файле дескриптора приложения, чтобы интерфейс скрывался после фазы инициализации приложения. Простой файл дескриптора для приложения, выполняемого в системной панели/док-панели:

```
<application xmlns="http://ns.adobe.com/air/application/1.0">

    <id>SystTrayApp</id>
    <name>SystTrayApp</name>
    <filename>SystTrayApp</filename>
    <version>0.1</version>
    <initialWindow>
        <content>SystTrayApp.swf</content>
        <systemChrome>none</systemChrome>
        <transparent>true</transparent>
        <visible>true</visible>
    </initialWindow>

</application>
```

20

FlexUnit и модульное тестирование

По мере роста размеров и сложности приложений **Flex практика и концепция** модульного тестирования постепенно приобретают популярность в сообществе Flex. *Модульное тестирование* (*unit testing*) – процесс проверки того, что добавления или изменения в проекте не приводят к появлению ошибок или отходу от ожидаемого поведения – позволяет организовать параллельную работу в больших группах и убедиться в том, что малые самостоятельные блоки программы, вплоть до конкретных методов, возвращают ожидаемые результаты. Это заметно ускоряет поиск ошибок, потому что правильно написанный модульный тест проверяет поведение одного метода или очень малого функционального блока.

Центральное место в модульном тестировании занимает *тестовый сценарий* (*test case*) – процедура, которая передает значение методу приложения и сообщает о прохождении теста при получении правильного возвращаемого значения. Тестовые сценарии значительно различаются по сложности, от простой проверки того, что метод возвращает правильное целое значение, до проверки правильности выполнения логики вывода или получения правильного типа объекта от веб-службы. Объединение нескольких тестовых сценариев называется *тестовым пакетом* (*test suite*). Тестовые пакеты предназначены для тестирования всего приложения или отдельного аспекта очень большого приложения. В выходных данных тестового пакета содержится информация обо всех прохождениях тестов, как успешных, так и ошибочных. По мере добавления в приложение нового кода для него пишутся новые тестовые сценарии, после чего выполняется весь тестовый пакет. Это гарантирует, что новый код не помешает выполнению прежнего работоспособного кода и интегрируется в приложение в соответствии с его требованиями.

Инфраструктура FlexUnit позволяет создавать тестовые сценарии и асинхронные тесты, а также выполнять тестовые пакеты в управляющей программе, обеспечивающей визуальное отображение информации

обо всех тестах пакета. Рецепты этой главы (все они были написаны и разработаны Дэниелом Райнхартом (Daniel Rinehart), показывают, как создать осмысленные тестовые сценарии и интегрировать их в тестовые пакеты. Также в этой главе продемонстрирована работа с более совершенными инструментами (такими как библиотека Antennae), упрощающими автоматическое построение тестов в ходе разработки приложения.

20.1. Создание приложения с использованием FlexUnit Framework

Задача

Требуется создать приложение, использующее классы FlexUnit Framework для создания и запуска тестов.

Решение

Загрузите и распакуйте FlexUnit Framework. Включите файл flexunit.swc в путь компиляции приложения.

Обсуждение

FlexUnit Framework включает графическую программу запуска тестов и базовые классы, используемые для создания пользовательских тестов. Пакет загружается с сайта Google Code по адресу *http://code.google.com/p/as3flexunitlib/*. Проследите за тем, чтобы загружаемая версия была самой новой.

Распакуйте ZIP-файл в выбранную вами папку. Чтобы использовать FlexUnit Framework, просто включите flexunit.swc в путь компиляции приложения. Если вы используете Flex Builder, выполните команду Project→Properties→Flex Build Path→Library Path→Add SWC, а затем перейдите к файлу flexunit/bin/flexunit.swc. Если вы предпочитаете командную строку, включите в аргументы mxmlc команду `-library-path+=flexunit/bin/flexunit.swc` (задайте нужный путь по своему усмотрению).

20.2. Создание приложения для запуска тестов FlexUnit

Задача

Требуется создать приложение, которое запускает тесты FlexUnit и выводит результаты в графическом виде.

Решение

Используйте экземпляр `TestSuite` и компонент `TestRunnerBase`.

Обсуждение

TestRunnerBase – стандартная оболочка для выполнения графических тестов, включенная в поставку FlexUnit Framework. Чтобы создать приложение с включением TestRunnerBase, приведите основной файл приложения MXML к следующему виду:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:flexui="flexunit.flexui.*">
    <flexui:TestRunnerBase id="testRunner" width="100%" height="100%"/>
</mx:Application>
```

На рис. 20.1 показано, как выглядит тестовое приложение FlexUnit после компиляции и запуска.

***Рис. 20.1.** Исходный вид тестового приложения FlexUnit*

Создайте экземпляр TestSuite для хранения коллекции выполняемых тестов. Для этого в тот же файл MXML включается блок <mx:Script>:

```
<mx:Script>
    <![CDATA[
        import flexunit.framework.TestSuite;

        private function createTestSuite():TestSuite
        {
            var testSuite:TestSuite = new TestSuite();
            return testSuite;
        }
    ]]>
</mx:Script>
```

В этом примере в экземпляр TestSuite никакие тесты не добавляются; мы просто создаем объект, в который можно добавлять экземпляры TestCase.

Далее экземпляр TestRunnerBase связывается с экземпляром TestSuite; это делается в функции handleCreationComplete:

```
        private function handleCreationComplete():void
        {
```

```
            testRunner.test = createTestSuite();
            testRunner.startTest();
        }
```

Функция автоматически запускает тесты после загрузки приложения. Чтобы вызвать ее, добавьте в приложение обработчик события `creationComplete`:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:flexui="flexunit.flexui.*" creationComplete=
    "handleCreationComplete();">
```

После компиляции и запуска приложение по-прежнему будет выглядеть так, как показано на рис. 20.1, однако ситуация изменилась: теперь в экземпляр `TestSuite` можно добавить экземпляры `TestCase`, и они будут запущены при запуске приложения.

Итоговый файл MXML:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:flexui=
"flexunit.flexui.*" creationComplete="handleCreationComplete();">
    <mx:Script>
        <![CDATA[
            import flexunit.framework.TestSuite;

            private function createTestSuite():TestSuite
            {
                var testSuite:TestSuite = new TestSuite();
                return testSuite;
            }

            private function handleCreationComplete():void
            {
                testRunner.test = createTestSuite();
                testRunner.startTest();
            }
        ]]>
    </mx:Script>
    <flexui:TestRunnerBase id="testRunner" width="100%" height="100%"/>
</mx:Application>
```

20.3. Создание тестового сценария FlexUnit

Задача

Требуется создать класс FlexUnit `TestCase` для тестового кода.

Решение

Создайте класс, расширяющий `TestCase`. Включите в него один или несколько открытых методов с именами, начинающимися с префикса `test`.

Обсуждение

При создании класса ActionScript, расширяющего `TestCase`, используются стандартная схема формирования имени: имя класса совпадает с именем тестируемого класса, и к нему добавляется суффикс `Test`. Например, если тестируемый класс называется `RegExp`, класс `TestCase` будет называться `RegExpTest`. Традиционно класс `TestCase` помещается в один пакет с тестируемым классом. Для класса `mx.core.UITextFormat` создается класс `TestCase` `mx.core.UITextFormatTest`.

Например, класс ActionScript с именем `RegExpTest`, расширяющий `TestCase`, выглядит так:

```
package
{
    import flexunit.framework.TestCase;
    public class RegExpTest extends TestCase
    {
    }
}
```

Для поиска методов `TestCase`, которые должны быть выполнены, FlexUnit Framework использует механизм рефлексии. Имена функций, начинающиеся с префикса `test`, сообщают FlexUnit, что функция должна быть включена в состав выполняемых. В примере этого рецепта в `RegExpTest` включается метод `testRegExp`:

```
public function testRegExp():void
{
}
```

На следующей фазе процесса определяется одно или несколько проверочных *утверждений* (assertions). Утверждение представляет собой программный способ проверки некоторого условия. Самой распространенной формой является сравнение ожидаемого значения с фактическим, полученным при вызове операции. FlexUnit содержит несколько типов утверждений для проверки разных ситуаций. Наиболее распространенные утверждения и их функции:

`assertEquals`

Сравнение оператором `==`.

`assertTrue`

Проверка истинности условия.

`assertNull`

Проверка условия на равенство `null`.

`assertStrictlyEquals`

Сравнение оператором `===`.

FlexUnit также предоставляет различные вспомогательные утверждения для проверки обратных условий, например `assertFalse` и `assertNot-`

Null (полный список приведен в документации Assert API из поставки FlexUnit).

Каждая функция утверждения может получать необязательный строковый аргумент. Если проверяемое условие не выполняется, в выходных данных эта строка будет выведена перед стандартным сообщением об ошибке «expected X but was Y». Помните, что при нарушении утверждения оставшаяся часть тестового метода не выполняется.

TestCase расширяет класс Assert, в котором определяются все функции утверждений. Это позволяет субклассам TestCase напрямую вызывать функции утверждений. Следующий пример демонстрирует различные методы утверждений (этот фрагмент включается в функцию testRegExp):

```
var regExp:RegExp = new RegExp("a", "i");
assertFalse(regExp.test("b"));
assertFalse("regExp doesn't match", regExp.test("b"));
assertNull(regExp.exec("b"));
assertNull("regExp doesn't match", regExp.exec("b"));
assertNotNull(regExp.exec("Apple"));
assertNotNull("regExp matches", regExp.exec("Apple"));
assertTrue(regExp.exec("Apple") is Array);
assertTrue("regExp exec returned an Array",
    regExp.exec("Apple") is Array);
assertEquals("A", regExp.exec("Apple")[0]);
assertEquals("regExp matched A in Apple", "A",
    regExp.exec("Apple")[0]);
assertStrictlyEquals(regExp, regExp);
assertStrictlyEquals("regExp object identity", regExp, regExp);
```

Вы можете дополнить TestCase новыми тестовыми методами для проверки других групп операций. Предполагается, что каждый тестовый метод специализируется на проверке определенной операции или задачи. При тестировании операций создания, выборки, обновления и удаления следует предусмотреть для каждого случая отдельный метод test, например testCreate, testRetrieve и т. д. В этом случае при нарушении утверждения вы получите более точную информацию, которая поможет выявить проблему.

Помните, что тестовые методы в TestCase выполняются в случайном порядке. Каждый тест должен создавать собственные данные без каких-либо предположений относительно предшествующего выполнения других тестов. Полный код ActionScript для этого примера:

```
package
{
    import flexunit.framework.TestCase;

    public class RegExpTest extends TestCase
    {
        public function testRegExp():void
        {
```

```
            var regExp:RegExp = new RegExp("a", "i");
            assertFalse(regExp.test("b"));
            assertFalse("regExp doesn't match", regExp.test("b"));

            assertNull(regExp.exec("b"));
            assertNull("regExp doesn't match", regExp.exec("b"));

            assertNotNull(regExp.exec("Apple"));
            assertNotNull("regExp matches", regExp.exec("Apple"));

            assertTrue(regExp.exec("Apple") is Array);
            assertTrue("regExp exec returned an Array",
                regExp.exec("Apple") is Array);

            assertEquals("A", regExp.exec("Apple")[0]);
            assertEquals("regExp matched A in Apple", "A",
                regExp.exec("Apple")[0]);

            assertStrictlyEquals(regExp, regExp);
            assertStrictlyEquals("regExp object identity", regExp, regExp);
        }
    }
}
```

Завершающим шагом должно стать включение созданного экземпляра `TestCase` в `TestSuite`; об этом рассказано в следующем рецепте.

См. также

Рецепт 20.2 и 20.4.

20.4. Включение тестового сценария в тестовый пакет

Задача

Требуется добавить тестовый сценарий в существующий тестовый пакет.

Решение

Используйте метод `addTestSuite` класса `TestSuite`.

Обсуждение

Чтобы включить тестовый сценарий в тестовый пакет, воспользуйтесь методом `addTestSuite`, которому при вызове передается ссылка на класс `TestCase`. Во внутренней реализации FlexUnit использует механизм рефлексии для поиска всех методов, имена которых начинаются с префикса `test`, и включает их в список выполнения.

В следующем примере приведена обновленная версия метода `createTestSuite` из рецепта 20.2, в которой `RegExpTest` включается в состав выполняемых тестов:

```
private function createTestSuite():TestSuite
{
    var testSuite:TestSuite = new TestSuite();
    testSuite.addTestSuite(RegExpTest);
    return testSuite;
}
```

Если `TestCase` не входит в пакет по умолчанию, не забудьте добавить директиву импортирования класса. Пример:

```
import mx.core.UITextFormatTest;

private function createTestSuite():TestSuite
{
    var testSuite:TestSuite = new TestSuite();
    testSuite.addTestSuite(RegExpTest);
    testSuite.addTestSuite(UITextFormatTest);
    return testSuite;
}
```

Тестовые сценарии, включаемые в тестовый пакет, выполняются в порядке добавления.

См. также

Рецепты 20.2 и 20.3.

20.5. Выполнение кода перед и после каждого теста

Задача

Требуется выполнять некоторый фрагмент кода до и после каждого теста в тестовом сценарии.

Решение

Переопределите метод `setUp` и `tearDown` в `TestCase`.

Обсуждение

По умолчанию каждый метод `test`, определяемый в `TestCase`, выполняется в отдельном экземпляре `TestCase`. Если нескольким тестовым методам необходимы одни и те же системные данные или информация состояния, используйте метод `setUp` для централизованной настройки конфигурации теста без необходимости явного вызова метода в начале каждого теста. Аналогично, если после каждого теста необходимо

уничтожать некоторые объекты или выполнять иные завершающие действия, метод tearDown гарантированно выполняется независимо от любых невыполненных условий и ошибок.

Помните, что выполнение метода test прекращается при первом же нарушенном утверждении или выданной ошибке. Это обстоятельство является важным аргументом в пользу метода tearDown, если тесты используют внешние ресурсы или объекты, которые должны безусловно освобождаться.

Метод setUp чаще всего используется для приведения системы в заведомо корректное состояние. Для сложных тестов это может потребовать подключения нескольких объектов или подключения к внешним ресурсам. Чтобы создать фрагмент кода, выполняемый перед каждым тестом в TestCase, переопределите метод setUp следующим образом:

```
override public function setUp():void
{
}
```

В методе setUp можно разместить любой код, включая тестовые утверждения. Возможность использования утверждений в setUp позволяет быстро отменять тесты в случае недоступности ресурса или объекта. Если в методе setUp нарушается утверждение или происходит ошибка, не вызывается ни предполагаемый метод test, ни метод tearDown. Это единственная ситуация, в которой метод tearDown не будет вызван.

У setUp имеется парный метод tearDown, выполняемый после каждого теста в TestCase независимо от нарушенных утверждений и ошибок. Считайте его чем-то вроде секции finally в блоке try…catch…finally. Впрочем, метод tearDown обычно не является необходимым. Напомню, что по умолчанию каждый метод test выполняется в отдельном экземпляре TestCase. Это означает, что переменные уровня класса инициализируются заново без учета изменений, внесенных предыдущим тестовым методом. В частности, метод tearDown удобен для выполнения общих утверждений после каждого метода test или освобождения ресурсов, внешних по отношению к системе (например, отключения сокета). Чтобы создать код, выполняемый после каждого метода test в TestCase, переопределите метод tearDown:

```
override public function tearDown():void
{
}
```

Следующий пример демонстрирует вызов методов setUp, кода test и методов tearDown:

```
package
{
    import flexunit.framework.TestCase;
    public class SetUpTearDownTest extends TestCase
    {
```

```
        private var _phase:String = "instance";
        override public function setUp():void
        {
            updatePhase("setUp()");
        }

        override public function tearDown():void
        {
            updatePhase("tearDown()");
        }

        public function testOne():void
        {
            updatePhase("testOne()");
        }

        public function testFail():void
        {
            updatePhase("testFail()");
            fail("testFail() always fails");
        }

        public function testError():void
        {
            updatePhase("testError()");
            this["badPropertyName"] = "newValue";
        }

        private function updatePhase(_phase:String):void
        {
            trace("Running test", methodName, "old phase", _phase,
                "new phase", phase);
            _phase = phase;
        }
    }
}
```

Примерный результат выполнения `SetUpTearDownTest`**:**

```
Running test testFail old phase instance new phase setUp()
Running test testFail old phase setUp() new phase testFail()
Running test testFail old phase testFail() new phase tearDown()
Running test testError old phase instance new phase setUp()
Running test testError old phase setUp() new phase testError()
Running test testError old phase testError() new phase tearDown()
Running test testOne old phase instance new phase setUp()
Running test testOne old phase setUp() new phase testOne()
Running test testOne old phase testOne() new phase tearDown()
```

Обратите внимание: в начале каждого теста значение `_phase` **равно** `instance`**, а методы** `setUp` **и** `tearDown` **выполняются независимо от исхода проверки утверждений или возникновения ошибок.**

См. также

Рецепт 20.2.

20.6. Передача данных между тестовыми сценариями

Задача

Требуется обеспечить передачу простых и сложных тестовых данных между тестовыми сценариями.

Решение

Создайте фабрику, генерирующую необходимые экземпляры тестовых данных.

Обсуждение

В процессе тестирования довольно часто возникает ситуация, когда несколько тестовых сценариев должны использовать одни и те же (или похожие) тестовые данные. Данные могут быть как простыми (например, объект, представляющий адрес), так и сложными (заказ с несколькими взаимозависимыми позициями). Копирование/вставка фрагментов кода или попытки загрузки данных с внешних ресурсов для создания и инициализации таких объектов для каждого тестового сценария – не оптимальное решение; лучше использовать централизованное создание объектов в фабрике. Паттерны централизации данных такого рода часто обозначаются термином «ObjectMother».

В простейшей форме `ObjectMother` представляет собой вспомогательный класс со статическим методом для создания всех типов необходимых объектов. Как правило, методы создания создаются в двух формах. В первой форме передаются значения каждого задаваемого свойства, а метод просто «собирает» объект по значениям. Вторая форма вызывается с минимальным набором аргументов (или вообще без аргументов), а каждому свойству задается реалистичное, разумное значение по умолчанию. Низкоуровневые методы создания могут использоваться при конструировании более сложных объектов.

Другое преимущество класса `ObjectMother` – возможность определения тестовых констант или других общих значений, доступных для всех тестов.

Пример простейшей реализации `ObjectMother`:

```
package
{
    public class ObjectMother
    {
        public static const SHIPPING_ZIP_CODE:String = "0123";
```

```
        public static function createAddress(line:String,
            city:String, state:String, zip:String):Address
        {

            var address:Address = new Address();
            address.line = line;
            address.city = city;
            address.state = state;
            address.zip = zip;
            return address;
        }

        public static function createAddressShipping():Address
        {
            return createAddress("123 A Street", "Boston", "MA",
                SHIPPING_ZIP_CODE);
        }

        public static function createAddressBilling():Address
        {
            return createAddress("321 B Street", "Cambridge", "MA",
                "02138");
        }

        public static function createOrder(lineItems:Array = null):Order
        {
            var order:Order = new Order();
            order.shippingAddress = createAddressShipping();
            order.billingAddress = createAddressBilling();
            for each (var lineItem:LineItem in lineItems)
            {
                addLineItemToOrder(order, lineItem);
            }
            return order;
        }

        public static function addLineItemToOrder(order:Order,
            lineItem:LineItem):void
        {
            order.addLineItem(lineItem);
        }
    }
}
```

Начиная с простого объекта `Address`, определяется стандартизированный метод создания параметров `createAddress`. Две вспомогательные функции, `createAddressShipping` и `createAddressBilling`, предоставляют методам `TestCase` быстрый доступ к конкретизированным экземплярам `Address`. Вспомогательные функции строятся на базе обобщенной функции `createAddress` и всей логики создания экземпляров, уже запрограммированной в этом методе. Многоуровневая схема создания особенно удобна для

сложных объектов, как в примере с `createOrder`, или в том случае, если один объект создается за несколько шагов.

Так как при каждом вызове метода создается новый экземпляр объекта, изменения, вносимые одним экземпляром `TestCase`, не будут иметь нежелательных побочных эффектов для другого экземпляра `TestCase`. В то же время из-за централизации объектов тестовых данных изменение данных в `ObjectMother` для поддержки нового теста может нарушить работоспособность существующих тестов. Впрочем, обычно эта проблема отходит на второй план в сравнении с удобствами доступа к тестовым данным.

20.7. Обработка событий в тестовых сценариях

Задача

Требуется обеспечить ожидание события в `TestCase`.

Решение

Используйте метод FlexUnit `addAsync`.

Обсуждение

Тестирование поведения в `TestCase` часто сопряжено с ожиданием асинхронных событий. Если методы `TestCase` используют только синхронные события (например, события изменения, инициируемые при задании свойств), специальная обработка не нужна. Но при использовании асинхронных событий необходимо принять специальные меры при тестировании. Стандартная ситуация, требующая прослушивания асинхронных событий при тестировании, – ожидание завершения загрузки `URLLoad` или завершения создания `UIComponent`. В этом рецепте описан синтаксис и различные нюансы обработки событий в `TestCase` на примере класса `URLLoad` и вымышленного объекта `Configuration`.

Обработка событий в `TestCase` должна быть организована особым образом, так как если FlexUnit не будет знать об ожидании события, сразу же при выходе из метода `test` FlexUnit запустит следующий метод `test`. Это может привести к выдаче ложной информации об успешном прохождении теста; FlexUnit выдаст зеленую полоску, тогда как где-то «на заднем плане» тест завершится неудачей, или еще хуже – выдаст сообщение об ошибке.

Чтобы сообщить FlexUnit о необходимости дождаться события, прежде чем помечать тест как успешный или сбойный, следует заменить слушателя, передаваемого `addEventListener`, вызовом `addAsync`. Первые два аргумента `addAsync` обязательны, а два других – нет. Первый обязательный аргумент определяет слушателя, который должен вызываться при передаче события. В нем указывается метод, который бы передавался в качестве слушателя до введения `addAsync`. Второй обязательный ар-

гумент определяет продолжительность интервала для срабатывания события (тайм-аут). Если событие не будет передано до истечения тайм-аута, FlexUnit пометит тест как неудачный и продолжит выполнение других тестовых методов.

Типичный пример использования `addAsync`:

```
package
{
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    import flexunit.framework.TestCase;

    public class ConfigurationTest extends TestCase
    {
        public function testParse():void
        {
            var urlLoader:URLLoader = new URLLoader();
            urlLoader.addEventListener(Event.COMPLETE,
                addAsync(verifyParse, 1000));
            urlLoader.load(new URLRequest("sample.xml"));
        }

        private function verifyParse(event:Event):void
        {
            var configuration:Configuration = new Configuration();
            assertFalse(configuration.complete);
            configuration.parse(new XML(event.target.data));
            assertTrue(configuration.complete);
        }
    }
}
```

Тестирование метода `parse` для объекта `Configuration` разбито на два метода. Первый метод конструирует объекты и инициирует операцию, требующую ожидания события. Затем метод `verify` использует результат события для выполнения своих действий и утверждений. При нормальном выполнении метод `verifyParse` указывался бы непосредственно как слушатель `addEventListener`, но в данном случае он «завернут» в вызов `addAsync` с 1000-миллисекундным тайм-аутом. Обратите внимание: имя функции-слушателя не начинается с префикса `test`; в противном случае `FlexUnit` попытается выполнить этот метод как дополнительный тест, а это явно нежелательно.

FlexUnit учитывает тип события, что позволяет слушателю привести свой аргумент к соответствующему типу. Если бы в предыдущем примере вместо обобщенного события `Event` использовалось событие `FlexEvent` или другого субкласса, производного от `Event`, слушатель мог бы свободно преобразовать свой параметр к этому типу. В случае несоот-

ветствия типов событий во время выполнения произошла бы ошибка «Type Coercion».

На этом этапе необходимо сделать два важных предупреждения по поводу addAsync. Во-первых, никогда не используйте ожидание более чем по одному вызову addAsync в любой момент времени. FlexUnit Framework некорректно обнаруживает и обрабатывает сбои тестов при нескольких addAsync. В крайнем случае можно использовать цепочечные вызовы addAsync, когда в слушателе, вызываемом по одному вызову addAsync, создается новый вызов addAsync, как показано в следующем фрагменте кода. Во-вторых, не регистрируйте addAsync для событий, многократно инициируемых в ходе тестирования. Так как механизм addAsync используется FlexUnit Framework в качестве «контрольной точки» для принятия решения об успехе или неудаче при выполнении теста, многократный вызов addAsync приведет к ложным срабатываниям и всевозможным странностям.

Если вы используете addAsync, вместо замыкания (closure) или создания переменной экземпляра для передачи информации слушателю также можно воспользоваться необязательным третьим аргументом addAsync. Передавать можно любые данные, что обеспечивает необходимую гибкость. Например, определяемый ранее тест можно записать так, чтобы перед инициированием загрузки XML он создавал и проверял флаг завершения объекта Configuration. Такой подход соответствует принципу «скорейшего сбоя», принятому в модульном тестировании для сведения к минимуму времени выполнения всего тестового пакета. Измененная версия тестового метода с передачей данных выглядит так:

```
public function testComplete():void
{
    var configuration:Configuration = new Configuration();
    assertFalse(configuration.complete);
    var urlLoader:URLLoader = new URLLoader();
    urlLoader.addEventListener(Event.COMPLETE,
        addAsync(verifyComplete, 1000, configuration));
    urlLoader.load(new URLRequest("sample.xml"));
}
private function verifyComplete(event:Event,
    configuration:Configuration):void
{
    configuration.parse(new XML(event.target.data));
    assertTrue(configuration.complete);
}
```

Объект, созданный в методе test, передается verifyComplete во втором аргументе. Этот механизм также позволяет передавать обобщенные объекты или примитивные типы (например, int).

Если событие не произойдет в течение заданного интервала, по умолчанию FlexUnit выдает ошибку. Если же при отсутствии события должны выполняться некоторые пользовательские действия, укажите в четвер-

том необязательном аргументе addAsync вызываемую функцию. Определение пользовательского обработчика событий особенно удобно для выяснения причин отсутствия события или специфических операций удаления объектов, задействованных в тесте. Пользовательский обработчик ошибки всегда получает копию передаваемых данных, даже если это null. Чтобы убедиться в том, что объект Configuration никогда не выдает события complete, попробуйте использовать следующую схему:

```
public function testCompleteEvent():void
{
    var configuration:Configuration = new Configuration();
    assertFalse(configuration.complete);
    configuration.addEventListener(Event.COMPLETE,
        addAsync(verifyEvent, 250, configuration,
        verifyNoEvent));
}
private function verifyEvent(event:Event,
    configuration:Configuration):void
{
    fail("Unexpected Event.COMPLETE
        from Configuration instance");
}
private function verifyNoEvent(configuration:
    Configuration):void
{
    assertFalse(configuration.complete);
}
```

Для события все равно необходимо определить слушателя события, но как показывает этот пример, в случае возникновения событие представляет условие ошибки. Пользовательский обработчик ошибки убеждается в том, что Configuration находится в корректном состоянии (для отсутствия события).

Если настройка или тестирование объекта требует последовательного срабатывания нескольких асинхронных событий, очень важно, чтобы в любой момент времени был активен только один вызов addAsync (см. ранее). Чтобы обойти это ограничение, следует организовать цепочечный вызов addAsync. Дополним предыдущий пример: если разбор конфигурационных данных требует загрузки дополнительных файлов, изменение статуса завершения может потребовать определенного времени. Пример цепочечного объединения этих двух событий:

```
public function testComplexComplete():void
{
    var configuration:Configuration = new Configuration();
    assertFalse(configuration.complete);
    var urlLoader:URLLoader = new URLLoader();
    urlLoader.addEventListener(Event.COMPLETE,
        addAsync(verifyComplexParse, 1000, configuration));
    urlLoader.load(new URLRequest("complex.xml"));
}
```

```
private function verifyComplexParse(event:Event,
    configuration:Configuration):void
{
    configuration.addEventListener(Event.COMPLETE,
        addAsync(verifyComplexComplete, 1000, configuration));
    configuration.parse(new XML(event.target.data));
    assertFalse(configuration.complete);
}

private function verifyComplexComplete(event:Event,
    configuration:Configuration):void
{
    assertTrue(configuration.complete);
}
```

В функцию `verifyComplexParse`, указанную при первом вызове `addAsync`, включается второй вызов `addAsync` для следующего события в цепочке. Длина цепочки может быть сколь угодно большой.

См. также

Рецепт 20.8.

20.8. Тестирование визуальных компонентов в FlexUnit

Задача

Требуется протестировать визуальный компонент.

Решение

Временно включите компонент в иерархию отображения и протестируйте его.

Обсуждение

Иногда приходится слышать, что тестирование визуальных компонентов отклоняется от основной цели модульного тестирования, потому что оно затрудняет изоляцию тестируемого класса при проверке тестовых условий. Тестирование компонентов затрудняется широтой возможностей Flex Framework, прежде всего вызовом таких методов, как `measure`. Стили и родительские контейнеры тоже могут повлиять на поведение компонента. Соответственно тестирование визуальных компонентов лучше рассматривать как автоматизацию функционального тестирования.

Чтобы вы могли протестировать поведение визуального компонента, последний должен пройти через несколько фаз своего жизненного цикла. Flex Framework автоматически выполняет необходимые действия при

включении компонента в иерархию отображения. Однако экземпляры TestCase не являются визуальными компонентами, а это означает, что компонент должен быть связан с объектом, внешним по отношению к TestCase. Наличие внешней связи означает, что вам придется дополнительно позаботиться о выполнении «зачистки» после как успешных, так и сбойных тестов; в противном случае «бесхозные» компоненты могут случайно повлиять на выполнение других тестов.

Схема тестирования компонентов

Простейший способ получения ссылки на визуальный объект, к которому можно добавить тестируемый компонент, основан на использовании Application.application. Поскольку TestCase выполняется в приложении Flex, этот синглетный экземпляр доступен всегда. Создание и активизация визуального компонента не являются синхронными операциями; перед тестированием экземпляр TestCase должен дождаться, пока компонент перейдет в корректное состояние. Чтобы обнаружить переход в это состояние для вновь созданного компонента, проще всего организовать ожидание события FlexEvent.CREATION_COMPLETE при помощи addAsync. Чтобы один метод TestCase не мешал выполнению другого метода TestCase, необходимо уничтожить созданный компонент и все внешние ссылки на него. Для решения этих двух задач лучше всего использовать метод tearDown и переменную с экземпляром класса. В следующем фрагменте продемонстрированы создание, присоединение, активизация и зачистка для экземпляра Tile:

```
package mx.containers
{
    import flexunit.framework.TestCase;
    import mx.core.Application;
    import mx.events.FlexEvent;
    public class TileTest extends TestCase
    {
        // Переменная класса обеспечивает доступ
        // к экземпляру из tearDown()
        private var _tile:Tile;
        override public function tearDown():void
        {
            try
            {
                Application.application.removeChild(_tile);
            }
            catch (argumentError:ArgumentError)
            {
                // Означает, что компонент
                // никогда не добавлялся
            }
            _tile = null;
        }

        public function testTile():void
```

```
        {
            _tile = new Tile();
            _tile.addEventListener(
                FlexEvent.CREATION_COMPLETE,
                addAsync(verifyTile, 1000));
            Application.application.addChild(_tile);
        }
        private function verifyTile(flexEvent:FlexEvent):void
        {
            // Компонент готов к тестированию
            assertTrue(_tile.initialized);
        }
    }
}
```

Обратите внимание на определение переменной класса, которая позволяет методу `tearDown` получить доступ к экземпляру, созданному и добавленному в `Application.application`. Добавление компонента в `Application.application` может завершиться неудачей, поэтому метод `tearDown` «заворачивает» вызов `removeChild` в блок `try…catch` для предотвращения выдачи ошибок. Метод `test` использует `addAsync` для ожидания перехода компонента в корректное состояние, прежде чем приступать к выполнению тестов.

Тестирование создания компонента

Теоретически можно вручную вызвать `measure` и другие методы Flex Framework для экземпляра компонента, чтобы он стал частью иерархии отображения, `test` лучше имитирует среду, в которой должен работать компонент. В отличие от модульного тестирования, внешняя по отношению к компоненту среда не находится под жестким контролем; это означает, что вам придется принять дополнительные меры, чтобы объектом тестирования был сам компонент, а не его окружение. Для примера протестируем логику раскладки контейнера `Tile` при добавлении дочерних компонентов:

```
public function testTileLayout():void
        {
            _tile = new Tile();
            var canvas:Canvas = new Canvas();
            canvas.width = 100;
            canvas.height = 100;
            _tile.addChild(canvas);
            canvas = new Canvas();
            canvas.width = 50;
            canvas.height = 50;
            _tile.addChild(canvas);
            canvas = new Canvas();
            canvas.width = 150;
            canvas.height = 50;
            _tile.addChild(canvas);
```

```
        _tile.addEventListener(
            FlexEvent.CREATION_COMPLETE,
            addAsync(verifyTileLayout, 1000));
        Application.application.addChild(_tile);
    }

    private function verifyTileLayout(
        flexEvent:FlexEvent):void
    {
        var horizontalGap:int =
            int(_tile.getStyle("horizontalGap"));
        var verticalGap:int =
            int(_tile.getStyle("verticalGap"));
        assertEquals(300 + horizontalGap, _tile.width);
        assertEquals(200 + verticalGap, _tile.height);
        assertEquals(3, _tile.numChildren);
        assertEquals(0, _tile.getChildAt(0).x);
        assertEquals(0, _tile.getChildAt(0).y);
        assertEquals(150 + horizontalGap, _tile.getChildAt(1).x);
        assertEquals(0, _tile.getChildAt(1).y);
        assertEquals(0, _tile.getChildAt(2).x);
        assertEquals(100 + verticalGap, _tile.getChildAt(2).y);
    }
```

В этом примере в контейнер `Tile` добавляются три дочерних компонента разных размеров. Согласно логике формирования раскладки, компонент `Tile` должен создать сетку 2×2 и назначить каждому дочернему компоненту максимальные значения ширины и высоты среди всех дочерних компонентов. Метод `verify` убеждается в том, что стандартная логика привела к желаемому результату. Важно заметить, что тест сосредоточен исключительно на логике, используемой компонентом. Он не проверяет, хорошо смотрится раскладка или нет, а лишь то, что его поведение соответствует документации. Также важно учитывать то, как стили могут отразиться на внешнем виде компонента. Динамическое определение `horizontalGap` и `verticalGap` в методе `verify` – один из способов повышения устойчивости теста на случай изменения значений по умолчанию. Вместо этого метод `test` также мог бы задать стилевые значения при создании экземпляра, чтобы гарантировать использование нужных данных.

Тестирование после создания

После того как компонент будет создан, тестирование дополнительных изменений в нем усложняется. Возникает соблазн воспользоваться обобщенным событием `FlexEvent.UPDATE_COMPLETE`, но оно может сработать многократно при одном изменении компонента. Теоретически возможно создать логику, которая корректно обрабатывает эти многократные события, но в конечном итоге `TestCase` начнет тестировать логику события Flex Framework и обновления пользовательского интерфейса (вместо логики, содержащейся в компоненте). Таким образом, проектирование

тестов, ограничивающихся логикой компонента, сродни изящному искусству. Это еще одна причина, по которой многие специалисты склонны рассматривать тестирование компонентов на этом уровне как функциональное, а не как модульное тестирование.

Предположим, в созданный ранее контейнер `Tile` добавляется еще один дочерний компонент. Следующий пример обнаруживает это изменение:

```
// Переменная класса для отслеживания
// последнего экземпляра функции addAsync()
private var _async:Function;

public function
    testTileLayoutChangeAfterCreate():void
{
    _tile = new Tile();
    var canvas:Canvas = new Canvas();
    canvas.width = 100;
    canvas.height = 100;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 50;
    canvas.height = 50;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 150;
    canvas.height = 50;
    _tile.addChild(canvas);
    _tile.addEventListener(
        FlexEvent.CREATION_COMPLETE,
        addAsync(verifyTileLayoutAfterCreate,
        1000));
    Application.application.addChild(_tile);
}

private function verifyTileLayoutAfterCreate(
    flexEvent:FlexEvent):void
{
    var horizontalGap:int =
        int(_tile.getStyle("horizontalGap"));
    var verticalGap:int =
        int(_tile.getStyle("verticalGap"));
    assertEquals(300 + horizontalGap, _tile.width);
    assertEquals(200 + verticalGap, _tile.height);
    assertEquals(3, _tile.numChildren);
    assertEquals(0, _tile.getChildAt(0).x);
    assertEquals(0, _tile.getChildAt(0).y);
    assertEquals(150 + horizontalGap,
        _tile.getChildAt(1).x);
    assertEquals(0, _tile.getChildAt(1).y);
    assertEquals(0, _tile.getChildAt(2).x);
```

```
        assertEquals(100 + verticalGap,
            _tile.getChildAt(2).y);
        var canvas:Canvas = new Canvas();
        canvas.width = 200;
        canvas.height = 100;
        _tile.addChild(canvas);
        _async = addAsync(verifyTileLayoutChanging, 1000);
        _tile.addEventListener(
            FlexEvent.UPDATE_COMPLETE, _async);
    }

    private function verifyTileLayoutChanging(
            flexEvent:FlexEvent):void
    {
        _tile.removeEventListener(
            FlexEvent.UPDATE_COMPLETE, _async);
        _tile.addEventListener(
            FlexEvent.UPDATE_COMPLETE, addAsync(
            verifyTileLayoutChangeAfterCreate, 1000));
    }
    private function verifyTileLayoutChangeAfterCreate(
        flexEvent:FlexEvent):void
    {
        var horizontalGap:int =
            int(_tile.getStyle("horizontalGap"));
        var verticalGap:int =
            int(_tile.getStyle("verticalGap"));
        assertEquals(400 + horizontalGap, _tile.width);
        assertEquals(200 + verticalGap, _tile.height);
        assertEquals(4, _tile.numChildren);
        assertEquals(0, _tile.getChildAt(0).x);
        assertEquals(0, _tile.getChildAt(0).y);
        assertEquals(200 + horizontalGap,
            _tile.getChildAt(1).x);
        assertEquals(0, _tile.getChildAt(1).y);
        assertEquals(0, _tile.getChildAt(2).x);
        assertEquals(100 + verticalGap,
            _tile.getChildAt(2).y);
        assertEquals(200 + horizontalGap,
            _tile.getChildAt(3).x);
        assertEquals(100 + verticalGap,
            _tile.getChildAt(3).y);
    }
```

Логика обработки события использует переменную класса для отслеживания последней асинхронной функции, созданной вызовом `addAsync`, для ее последующего удаления и добавления другого слушателя при повторной выдаче того же события. Если произойдет дополнительное изменение, которое потребует передачи еще одного события `FlexEvent.UPDATE_COMPLETE`, метод `verifyTileLayoutChanging` также должен сохранить свою функцию `addAsync`, чтобы сделать возможным ее удаление. Цепо-

чечная обработка событий в определенном смысле ненадежна: если изменится логика выдачи событий в Flex Framework, программа может завершиться неудачей. Для нашего теста неважно, чтобы два события `FlexEvent.UPDATE_COMPLETE` были переданы последовательно; впрочем, это лишь побочный эффект воспроизведения логики компонента на этом уровне.

Хотя компонент может передавать дополнительные события (например, `Event.RESIZE`), состояние компонента в точке передачи события обычно является нестабильным. Так, если контейнер `Tile` передает событие `Event.RESIZE`, ширина компонента уже изменилась, а позиция дочерних компонентов – нет. Также могут присутствовать действия, поставленные в очередь посредством `callLater`, и исключение компонента из иерархии отображения приведет к ошибкам при попытке выполнения этих действий. Некоторых проблем такого рода удается избежать при тестировании компонентов с синхронной логикой обновления, снимающей необходимость в обработке событий. Возможен и другой вариант: тестируемый компонент передает событие, которое четко определяет момент полной реализации изменения. Какой бы способ вы ни выбрали, следует учесть, что эти схемы достаточно хрупки и приводят к побочному тестированию поведения за пределами компонента.

Тестирование по таймеру

Если в компонент одновременно вносится множество сложных изменений, количество и порядок передаваемых событий становится слишком громоздким. Вместо того чтобы дожидаться конкретного события, можно пойти по другому пути и ожидать в течение заданного времени. Такой подход упрощает обработку обновлений нескольких объектов или компонента, использующего экземпляры `Effect` с заранее известной продолжительностью. Недостаток данного решения заключается в том, что тестирование по таймеру может привести к ложным положительным срабатываниям при изменении скорости или доступности ресурсов в тестовой среде. Ожидание в течение фиксированного времени также означает, что время выполнения всего тестового пакета растет быстрее, чем при простом добавлении синхронных или управляемых событиями тестов.

Переработанная версия предыдущего примера с `Tile` выглядит следующим образом:

```
private function waitToTest(
        listener:Function, waitTime:int):void
{
    var timer:Timer = new Timer(waitTime, 1);
    timer.addEventListener(
        TimerEvent.TIMER_COMPLETE,
        addAsync(listener, waitTime + 250));
    timer.start();
}
```

```
public function testTileLayoutWithTimer():void
{
    _tile = new Tile();
    var canvas:Canvas = new Canvas();
    canvas.width = 100;
    canvas.height = 100;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 50;
    canvas.height = 50;
    _tile.addChild(canvas);
    canvas = new Canvas();
    canvas.width = 150;
    canvas.height = 50;
    _tile.addChild(canvas);
    Application.application.addChild(_tile);
    waitToTest(verifyTileLayoutCreateWithTimer,
        500);
}

private function verifyTileLayoutCreateWithTimer(
    timerEvent:TimerEvent):void
{
    var horizontalGap:int =
        int(_tile.getStyle("horizontalGap"));
    var verticalGap:int =
        int(_tile.getStyle("verticalGap"));
    assertEquals(300 + horizontalGap, _tile.width);
    assertEquals(200 + verticalGap, _tile.height);
    assertEquals(3, _tile.numChildren);
    assertEquals(0, _tile.getChildAt(0).x);
    assertEquals(0, _tile.getChildAt(0).y);
    assertEquals(150 + horizontalGap,
        _tile.getChildAt(1).x);
    assertEquals(0, _tile.getChildAt(1).y);
    assertEquals(0, _tile.getChildAt(2).x);
    assertEquals(100 + verticalGap,
        _tile.getChildAt(2).y);
    var _canvas:Canvas = new Canvas();
    _canvas.width = 200;
    _canvas.height = 100;
    _tile.addChild(canvas);
    waitToTest(verifyTileLayoutChangeWithTimer,
        500);
}

private function verifyTileLayoutChangeWithTimer(
    timerEvent:TimerEvent):void
{
    var horizontalGap:int =
        int(_tile.getStyle("horizontalGap"));
```

```
            var verticalGap:int =
                int(_tile.getStyle("verticalGap"));
            assertEquals(400 + horizontalGap, _tile.width);
            assertEquals(200 + verticalGap, _tile.height);
            assertEquals(4, _tile.numChildren);
            assertEquals(0, _tile.getChildAt(0).x);
            assertEquals(0, _tile.getChildAt(0).y);
            assertEquals(200 + horizontalGap,
                _tile.getChildAt(1).x);
            assertEquals(0, _tile.getChildAt(1).y);
            assertEquals(0, _tile.getChildAt(2).x);
            assertEquals(100 + verticalGap,
                _tile.getChildAt(2).y);
            assertEquals(200 + horizontalGap,
                _tile.getChildAt(3).x);
            assertEquals(100 + verticalGap,
                _tile.getChildAt(3).y);
        }
```

В отличие от предыдущих тестовых примеров, которые могли завершиться сразу же после выдачи событий, эта версия теста будет выполняться как минимум одну секунду. Дополнительное время, прибавляемое к задержке таймера при вызове `addAsync`, компенсирует небольшие расхождения в периодичности срабатывания таймера. Промежуточный метод переключения слушателей `FlexEvent.UPDATE_COMPLETE` исчез из кода примера, но в остальном тестовый код остался неизменным.

Программное тестирование визуальных компонентов

Возможность получения низкоуровневых растровых данных позволяет легко проверить некоторые визуальные аспекты компонента на программном уровне. Например, вы можете убедиться в том, что изменение цвета фона и стиля обрамления компонента действительно отражается на его внешнем виде. После создания экземпляра компонента сохраните и проанализируйте его растровые данные. В следующем примере тест проверяет, что добавление обрамления к контейнеру `Canvas` приводит к желаемому результату:

```
package mx.containers
{
    import flash.display.BitmapData;
    import flexunit.framework.TestCase;

    import mx.core.Application;
    import mx.events.FlexEvent;

    public class CanvasTest extends TestCase
    {
        // Переменная класса обеспечивает доступ
        // к экземпляру из tearDown()
        private var _canvas:Canvas;
```

```
override public function tearDown():void
{
    try
    {
        Application.application.removeChild(_canvas);
    }
    catch (argumentError:ArgumentError)
    {
        // Ошибку можно спокойно проигнорировать;
        // она означает лишь то, что компонент
        // не был добавлен.
    }
    _canvas = null;
}

private function captureBitmapData():BitmapData
{
    var bitmapData:BitmapData =
        new BitmapData(canvas.width, canvas.height);
    bitmapData.draw(_canvas);
    return bitmapData;
}

public function testBackgroundColor():void
{
    _canvas = new Canvas();
    _canvas.width = 10;
    _canvas.height = 10;
    _canvas.setStyle("backgroundColor", 0xFF0000);
    _canvas.addEventListener(FlexEvent.CREATION_COMPLETE, addAsync
        (verifyBackgroundColor, 1000));
    Application.application.addChild(_canvas);
}

private function verifyBackgroundColor(flexEvent:FlexEvent):void
{
    var bitmapData:BitmapData = captureBitmapData();
    for (var x:int = 0; x < bitmapData.width; x++)
    {
        for (var y:int = 0;
            y < bitmapData.height; y++)
        {
            assertEquals("Pixel (" + x + ",
            " + y + ")", 0xFF0000, bitmapData.
            getPixel(x, y));
        }
    }
}

public function testBorder():void
{
```

```
            _canvas = new Canvas();
            _canvas.width = 10;
            _canvas.height = 10;
            _canvas.setStyle("backgroundColor", 0xFF0000);
            _canvas.setStyle("borderColor", 0x00FF00);
            _canvas.setStyle("borderStyle", "solid");
            _canvas.setStyle("borderThickness", 1);
            _canvas.addEventListener(FlexEvent.CREATION_COMPLETE,
                addAsync(verifyBorder, 1000));
            Application.application.addChild(_canvas);
        }
        private function
            verifyBorder(flexEvent:FlexEvent):void
        {
            var bitmapData:BitmapData = captureBitmapData();
            for (var x:int = 0; x < bitmapData.width; x++)
            {
                for (var y:int = 0;
                    y < bitmapData.height; y++)
                {
                    if ((x == 0) || (y == 0) ||
                        (x == bitmapData.width - 1) ||
                        (y == bitmapData.height - 1))
                    {
                        assertEquals("Pixel (" + x + ",
                            " + y + ")", 0x00FF00,
                            bitmapData.getPixel(x, y));
                    } else {
                        assertEquals("Pixel (" + x + ",
                            " + y + ")", 0xFF0000,
                            bitmapData.getPixel(x, y));
                    }
                }
            }
        }
    }
}
```

Метод `testBackgroundColor` убеждается в том, что всем пикселам `Canvas` был правильно назначен цвет фона. Метод `testBorder` проверяет, что при добавлении обрамления к `Canvas` внешние пикселы окрашиваются в цвет рамки, а остальные пикселы сохраняют цвет фона. Графические данные сохраняются в методе `captureBitmapData`; при этом используется возможность прорисовки любого компонента Flex на экземпляре `BitmapData`. Этот полезный прием может использоваться для проверки программных скинов и других визуальных компонентов; без него модульное тестирование заметно затрудняется.

Пакет Visual FlexUnit (*http://code.google.com/p/visualflexunit/*) предоставляет альтернативный механизм тестирования визуального оформления компонентов.

Скрытие тестируемого компонента

Включение тестируемого компонента в `Application.application` приводит к некоторым побочным эффектам, одним из которых является прорисовка компонента. В результате добавление и удаление компонентов из иерархии отображения может сопровождаться изменением размеров и текущей позиции тестовой программы FlexUnit. Чтобы избежать этого нежелательного поведения, скройте тестируемые компоненты, задав их свойствам `visible` и `includeInLayout` значение `false` перед включением в иерархию отображения. Например, чтобы контейнер `Canvas` оставался скрытым в процессе его тестирования в приведенном ранее коде, запишите код его включения в иерархию отображения следующим образом:

```
_canvas.visible = false;
_canvas.includeInLayout = false;
Application.application.addChild(_canvas);
```

См. также

Рецепты 20.3 и 20.7.

20.9. Установка и настройка Antennae

Задача

Требуется автоматизировать сборку и тестирование приложений Flex.

Решение

Загрузите и распакуйте шаблоны Antennae, распространяемые с открытым кодом. Настройте их для вашей системы.

Обсуждение

Antennae – проект с открытым кодом, предназначенный для автоматизации сборки и тестирования приложений Flex. Он использует Ant и Java для реализации межплатформенных механизмов компиляции библиотек и приложений Flex, построения тестовых пакетов FlexUnit и автоматизированного запуска тестов FlexUnit. Antennae также определяет инфраструктуру построения крупных проектов с множественными зависимостями и интеллектуальной перекомпиляции. Antennae можно загрузить по адресу *http://code.google.com/p/antennae/*. Обязательно найдите самую свежую версию Antennae-*.zip. При распаковке все содержимое архива распаковывается в подкаталог с именем Antennae.

Antennae состоит из нескольких частей:

lib

Откомпилированная поддержка создания `TestSuite` для Java и Flex, приложения командной строки для запуска FlexUnit и получения

информации, библиотека FlexUnit и шаблон приложения FlexUnit.

src

Исходный код средств Java и Flex из каталога lib, кроме FlexUnit (этот исходный код доступен в другом месте).

templates

Шаблоны Antennae для построения библиотек Flex, приложений Flex, приложений FlexUnit и проектов Flex со сложными зависимостями.

tools

Обобщенные определения приемника, задачи и свойств Ant для автоматизации построения и тестирования.

tutorial

Подробные обучающие примеры, разъясняющие использование основных возможностей Ant и шаблонов Antennae.

Также базовый каталог содержит документацию и примеры файлов, используемых для настройки конфигурации Antennae на разных платформах.

Чтобы поэкспериментировать со всеми шаблонами и обучающими примерами, сначала необходимо настроить конфигурацию Antennae. Для обеспечения простой межплатформенной поддержки, ориентированной на коллективную работу, часто изменяемые свойства Antennae собраны в один файл, который можно изменять по мере надобности. В базовом каталоге находятся файлы build-user.properties.mac и build-user.properties.win, которые станут отправной точкой для настройки Antennae. Пользователи Linux используют в качестве примера файл build-user.properties.mac, но учтите, что некоторые возможности могут оказаться недоступными из-за ограничений Flex Framework.

Скопируйте нужный файл и присвойте ему имя build-user.properties (имя остается неизменным для всех платформ). На следующем этапе файл настраивается для конкретной системы; откройте его для редактирования в любом текстовом редакторе.

Важнейшее свойство, которое необходимо задать, — `flex2.dir`; задайте ему путь к каталогу Flex 2 или Flex 3 (базовый каталог с подкаталогами bin, lib и player). Используйте либо SDK из поставки Flex Builder, либо автономный Flex SDK.

Если вы собираетесь использовать средства автоматизации для командной строки FlexUnit, задайте свойство `flex2.standalone.player`. Рекомендуется использовать отладочную версию автономного проигрывателя; она позволяет получать данные трассировки стека, по которым проще определить причину возникновения ошибки.

Свойство `tomcat.webapps.dir` задается только при работе с примером tutorial/multi/app.

Наконец, различные свойства с префиксом `air` относятся к компиляции, запуску и упаковке приложений AIR на базе Flex. За дополнительной информацией о настройке этих свойств обращайтесь к вики-документации Antennae по адресу *http://code.google.com/p/antennae/w/list*.

Чтобы увидеть, как Antennae используется при сборке и тестировании проектов, просмотрите содержимое каталога tutorial. При запуске с целью `build` или `test` Ant перебирает все проекты в каталоге tutorial и передает в каждом случае нужную цель. В частности, цель `test` демонстрирует как успешный, так и ошибочный запуск FlexUnit.

В базовом каталоге находится файл README.txt с общим описанием процесса настройки Antennae, философии построения и тестирования проектов. В каждом из каталогов tutorial и template также присутствует файл README.txt, в котором описано предназначение и правила использования каждого проекта в каталоге.

20.10. Построение автоматизированных тестовых пакетов

Задача

Требуется автоматически построить тестовый пакет из всех готовых сценариев.

Решение

Используйте средства построения `TestSuite` в Antennae.

Обсуждение

Чтобы запустить тестовый сценарий `TestCase`, необходимо включить его в `TestSuite`. Процесс создания нового экземпляра `TestCase` и его включения в `TestSuite` быстро входит в привычку, но при работе нескольких разработчиков над общей кодовой базой вероятность того, что какой-либо тест будет забыт, повышается. Вместо того чтобы вручную включать каждый сценарий `TestCase` в `TestSuite`, можно сгенерировать `TestSuite` автоматически. Проект с открытым кодом Antennae содержит вспомогательное приложение автоматического построения `TestSuite`; приложение просматривает содержимое каталога исходного кода и включает все содержащиеся в нем тесты в тестовый пакет.

В подкаталоге lib базового каталога Antennae присутствует файл JAR с именем arc-flexunit2.jar, содержащий класс `com.allurent.flexunit2.framework.AllTestsFileGenerator`. При запуске `AllTestsFileGenerator` находит в каталоге с исходным кодом все классы с именами Test*.as или *Test.as и создает `TestSuite` с этими классами. Приложение направляет сгенерированный класс `TestSuite` в стандартный вывод с возможностью последующего перенаправления. Сгенерированный файл `TestSuite` явля-

ется частью корневого пакета и называется FlexUnitAllTests. Если файлы Antennae были распакованы в каталог ~/Antennae или C:\Antennae, запуск приложения выглядит примерно так:

```
java -cp ~/Antennae/lib/arc-flexunit2.jar
com.allurent.flexunit2.framework.AllTestsFileGenerator
~/FlexCookbook/src/ > ~/FlexCookbook/src/FlexUnitAllTests.as

java -cp C:\Antennae\lib\arc-flexunit2.jar
com.allurent.flexunit2.framework.AllTestsFileGenerator
C:\FlexCookbook\src\ > C:\FlexCookbook\src\FlexUnitAllTests.as
```

В первом примере ~/Antennae/lib/arc-flexunit2.jar – местонахождение файла JAR; com.allurent.flexunit2.framework.AllTestsFileGenerator – имя выполняемого класса; ~/FlexCookbook/src/ – корневой каталог дерева исходных кодов, в котором ищутся тесты; ~/FlexCookbook/src/FlexUnitAllTests.as – полное имя генерируемого файла.

Сгенерированный файл TestSuite выглядит примерно так:

```
package
{
    import flexunit.framework.*;
    import mx.containers.CanvasTest;
    import mx.containers.TileTest;
    public class FlexUnitAllTests
    {
        public static function suite() : TestSuite
        {
            var testSuite:TestSuite = new TestSuite();
            testSuite.addTestSuite(
                mx.containers.CanvasTest);
            testSuite.addTestSuite(mx.containers.TileTest);
            return testSuite;
        }
    }
}
```

Задачу построения файла FlexUnitAllTests можно автоматизировать, чтобы файл всегда строился перед компиляцией приложения FlexUnit (за дополнительной информацией об использовании AllTestsFileGenerator в Flex Builder обращайтесь к документации Antennae).

Вместо того чтобы вручную конструировать TestSuite в главном приложении, укажите класс FlexUnitAllTests в качестве выполняемого TestSuite. Каждый раз, когда потребуется заново сгенерировать класс FlexUnitAllTests, все включаемые тесты будут откомпилированы и запущены заново. Обновленное приложение FlexUnit с использованием FlexUnitAllTests выглядит так:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:flexui=
"flexunit.flexui.*" creationComplete="handleCreationComplete();">
    <mx:Script>
```

```
        <![CDATA[
            import flexunit.framework.TestSuite;

            private function handleCreationComplete():void
            {
                testRunner.test = FlexUnitAllTests.suite();
                testRunner.startTest();
            }
        ]]>
    </mx:Script>
    <flexui:TestRunnerBase id="testRunner"
        width="100%" height="100%"/>
</mx:Application>
```

По умолчанию в сгенерированный класс `TestSuite` включаются все файлы с именами Test*.as и *Test.as. Чтобы переопределить это поведение, создайте файл фильтров, в котором включаемые файлы определяются при помощи регулярных выражений. Каждая строка в файле фильтров задает одно регулярное выражение, по которому проверяется каждое имя файла. Если хотя бы одно регулярное выражение совпадет, файл включается в генерируемый пакет. Регулярное выражение применяется к полному имени файла, что позволяет организовать отбор `TestCase` по именам пакетов и по другим критериям. Простой файл фильтров может выглядеть так:

```
/mx/containers/.*Test.as
RegExpTest.as
```

Первая строка включает все тесты в любом подкаталоге каталога `/mx/containers`. Вторая строка включает конкретный тест RegExpTest.as, где бы он ни находился. Важно заметить, что AllTestsFileGenerator работает только на уровне файловой системы, поэтому в любом из правил должен присутствовать шаблон Test.as, который предотвращает выбор файлов, не являющихся файлами `TestCase`.

Если предыдущие правила хранятся в файле с именем filters.txt, вызов генератора выглядит так:

```
java -cp ~/Antennae/lib/arc-flexunit2.jar
com.allurent.flexunit2.framework.AllTestsFileGenerator
~/FlexCookbook/src/ filters.txt >
~/FlexCookbook/src/FlexUnitAllTests.as

java -cp C:\Antennae\lib\arc-flexunit2.jar
com.allurent.flexunit2.framework.AllTestsFileGenerator
C:\FlexCookbook\src\ filters.txt >
C:\FlexCookbook\src\FlexUnitAllTests.as
```

При использовании файла фильтров в `TestSuite` автоматически включается тест, который всегда завершается неудачей. Это напоминает разработчику о том, что некоторые тестовые сценарии могут отсутствовать в пакете и `TestSuite` не представляет полного набора тестов.

21

Компиляция и отладка

Приложения Flex обычно компилируются в Flex Builder или в командной строке компилятором MXML (mxmlc), но существуют и другие способы компиляции, перемещения файлов и запуска приложений. Например, такие утилиты, как make, Ant или Rake, позволяют упростить весь процесс компиляции и развертывания до одной команды.

Отладка приложений Flex осуществляется в отладочной версии Flash Player, позволяющей просматривать результаты трассировки. В Flex Builder 3 вы можете выполнить программу в пошаговом режиме, строка за строкой, и просматривать свойства переменных. Flex Building 3 также поддерживает новый профильный режим, в котором выводится информация об использовании памяти, создании и удалении объектов. Впрочем, кроме Flex Builder, существует немало других инструментов с открытым кодом. Например, вместо Flex Builder IDE для просмотра данных объектов можно использовать Xray и Console.as for Firebug, а для просмотра вывода команд трассировки – Flash Tracer или Output Panel. В рецептах этой главы рассматривается как отладка с использованием инструментария Flex Builder, так и трассировка и просмотр объектов в Xray и FlashTracer.

21.1. Трассировка без использования Flex Builder

Задача

Требуется включить в приложение команды трассировки, упрощающие его отладку (без использования Flex Builder 3).

Решение

Загрузите одну из многих существующих программ трассировки с открытым исходным кодом.

Обсуждение

С тех пор как фирма Adobe предоставила компилятор и библиотеку Flex 3 для свободного доступа, у разработчиков появились новые возможности для получения трассировочного вывода Flash Player. Теперь они не ограничиваются использованием Flash IDE или Flex Builder IDE, а могут выбрать наиболее подходящий для себя инструмент. Например, Xray Джона Грдена (John Grden) создает средства для просмотра результатов `trace` в Flash. Xray не только предоставляет доступ к данным трассировки во время выполнения, но и обеспечивает основные возможности просмотра объектов (рис. 21.1).

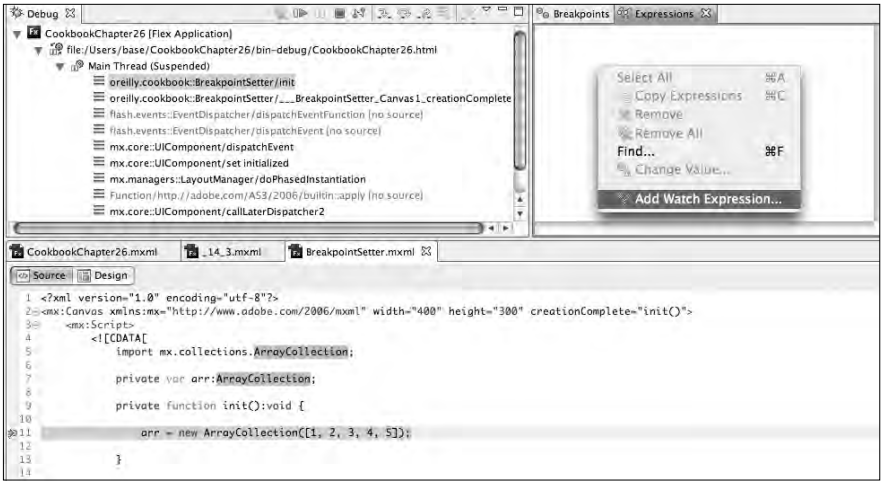

Рис. 21.1. *Просмотр вывода в Xray*

Еще один вариант: утилита FlashTrace, написанная Алессандро Круньола (Alessandro Crugnola). После установки этого дополнения к Firefox вы сможете получать результаты всех команд `trace`, выполняемых в приложении. При желании результаты трассировки также можно направить в файл.

Xray можно загрузить по адресу *http://osflash.org/xray#downloads*, а FlashTrace – по адресу *http://www.sephiroth.it/firefox*.

21.2. Компилятор компонентов

Задача

Требуется откомпилировать компонент Flex в файл SWC, который может использоваться в качестве общей библиотеки времени выполнения (RSL).

Решение

Используйте компилятор компонентов (compc). Информация передается компилятору либо в виде набора аргументов в командной строке, либо в конфигурационном файле XML (аргумент `load-config`).

Обсуждение

Командная строка запуска компилятора компонентов compc выглядит примерно так:

```
compc -source-path . -include-classes oreilly.cookbook.foo -output example.swc
```

Важнейшие параметры командной строки:

`-benchmark`

Компилятор должен измерить время, затраченное на компиляцию.

`-compiler.debug`

В сгенерированный файл SWC включается отладочная информация и отладочные функции.

`-compiler.external-library-path [path-element] [...]`

Файлы SWC и каталоги, используемые при компиляции.

`-compiler.include-libraries [library] [...]`

Библиотеки (SWC), полностью включаемые в SWF.

`-compiler.library-path [path-element] [...]`

Файлы SWC или каталоги с файлами SWC, которые должны использоваться при компиляции.

`-compiler.locale [locale-element] [...]`

Локальный контекст для интернализации.

`-compiler.optimize`

Оптимизация SWF после компоновки.

`-compiler.services <файл>`

Путь к конфигурационному файлу Flex Data Services.

`-compiler.theme [файл] [...]`

Перечень всех файлов CSS или SWC, применяемых в качестве тем оформления в приложениях.

`-compiler.use-resource-bundle-metadata`

Признак включения групп ресурсов в приложение.

`-include-classes [класс] [...]`

Перечень всех классов, включаемых в RSL; допускаются повторения и пути с метасимволами.

`-include-file <имя><путь>`

Перечень всех файлов, включаемых в RSL; допускаются повторения и пути с метасимволами.

`-include-resource-bundles [группа] [...]`

Признак включения группы ресурсов локализации.

`-load-config <файл>`

Загрузка файла с параметрами конфигурации.

`-output <файл>`

Имя и местонахождение файла, создаваемого компилятором compc.

`-runtime-shared-libraries [url] [...]`

Все внешние RSL, которые должны быть упакованы в RSL-библиотеку, создаваемую compc при компиляции.

`-runtime-shared-library-path [элемент] [rsl-url] [url-файла-политики] [rsl-url] [url-файла-политики]`

Местонахождение и другая информация о RSL-библиотеках, используемых приложением.

`-use-network`

Разрешение и запрет доступа к сетевым ресурсам из SWC.

Команда компиляции нескольких классов в RSL может оказаться очень длинной. Для ее упрощения можно использовать конфигурационные файлы.

По аналогии с компилятором MXML (mxmlc), конфигурационные файлы включатся в командную строку compc с параметром `load-config`. Как и с mxmlc, compc автоматически загружает конфигурационный файл по умолчанию flex-config.xml. Чтобы не повторять все содержимое flex-config.xml (большая часть которого является обязательной), добавьте свои данные к конфигурационному файлу по умолчанию с оператором `+=`:

```
compc -load-config+=configuration.xml
```

Все флаги, передаваемые компилятору, можно описать в XML и передать compc в параметре `-load-config`:

```
<include-sources>src/.</include-sources>
```

21.3. Установка задач Flex Ant

Задача

Требуется использовать задачи Flex Ant, включенные в поставку Flex 3 SDK.

Решение

Скопируйте файл flex_ant/lib/flexTasks.jar в каталог Ant lib ({ANT_root}/lib).

Обсуждение

Чтобы все задачи из библиотеки задач Flex Ant из Flex 3 SDK всегда были доступны для Ant, скопируйте их в каталог lib установки Ant. Если вы не скопировали файл в каталог lib, укажите его в параметре `-lib` командной строки при создании файла XML проекта.

21.4. Использование задач compc и mxmlc в задачах Flex Ant

Задача

Требуется использовать задачи mxmlc и compc, входящие в число задач Flex Ant, для упрощения компиляции приложений Flex и работы с Ant.

Решение

Установите задачи Flex Ant в библиотечные каталоги Ant, а затем используйте теги `<mxmlc>` или `<compc>` с передачей параметров компиляции в аргументах XML.

Обсуждение

Готовые задачи Flex Ant значительно упрощают работу с Ant при компиляции приложений Flex. Все параметры, передаваемые в командной строке mxmlc или compc, могут передаваться задачам Flex Ant. Например, после объявления задачи mxmlc объявление параметров вывода может выглядеть так:

```
<mxmlc file="C:/Flex/projects/app/App.mxml" output="C:/Flex/projects/bin/
App.swf">
```

Вместо того чтобы задавать местонахождение mxmlc и передавать параметры конфигурации в аргументах исполняемого файла, можно воспользоваться Ant-задачей mxmlc; это ускорит настройку, а файлы построения станут более удобочитаемыми. Дополнительные параметры задаются следующим образом:

```
    <!-- Получение параметров компилятора по умолчанию. -->
    <load-config filename="${FLEX_HOME}/frameworks/flex-config.xml"/>
    <!-- Список элементов, образующих корни иерархии классов ActionScript.
-->
    <source-path path-element="${FLEX_HOME}/frameworks"/>
    <!-- Список файлов и каталогов, содержащих файлы SWC. -->
```

```
        <compiler.library-path dir="${FLEX_HOME}/frameworks" append="true">
            <include name="libs" />
            <include name="../bundles/{locale}" />
        </compiler.library-path>
    </mxmlc>
```

Задача `<compc>` работает примерно так же; все параметры compc передаются через задачу `<compc>`:

```
<compc output="${output}/mylib.swc" locale="en_US">
```

21.5. Компиляция и развертывание приложений Flex, использующих RSL-библиотеки

Задача

Требуется развернуть приложение Flex, использующее одну или несколько общих библиотек времени выполнения (RSL).

Решение

Передайте в параметре компилятора `external-library-path` местонахождение RSL-библиотек после компиляции приложения.

Обсуждение

При инициализации приложение Flex должно знать, где находятся все необходимые RSL-библиотеки. Необходимую информацию оно берет из параметра командной строки `external-library-path`; благодаря наличию этой информации у компилятора Flash Player начинает загружать байты RSL немедленно, без необходимости загрузки отдельного файла SWF при создании экземпляров компонентов или классов.

Чтобы использовать файл с RSL-библиотекой, необходимо сначала создать ее. RSL-библиотеки хранятся в файлах SWC, к которым обращается приложение во время выполнения. Файл SWC компилируется compc, а SWF-файл приложения компилируется компилятором mxmlc. Чтобы приложение могло использовать RSL-библиотеку, необходимо передать mxmlc ее местонахождение в параметре `runtime-shared-libraries`. В следующем примере Ant используется для компиляции как файла SWC, так и приложения, в котором он используется; это означает, что использоваться должны как compc, так и mxmlc. В файле build.xml, используемом Ant, оба компилятора должны быть объявлены как переменные:

```
<property name="mxmlc" value="C:\FlexSDK\bin\mxmlc.exe"/>
<property name="compc" value="C:\FlexSDK\bin\compc.exe"/>
```

Далее compc задействуется для компиляции RSL-библиотеки, используемой в приложении, а задача `move` размещает ее в каталоге application/rsl:

```
<target name="compileRSL">
    <exec executable="${compc}">
      <arg line="-load-config+=rsl/configuration.xml" />
    </exec>
    <mkdir dir="application/rsl" />
    <move file="example.swc" todir="application/rsl" />
    <unzip src="application/rsl/example.swc" dest="application/rsl/" />
  </target>
```

Затем файл SWF приложения компилируется программой mxmlc. Обратите внимание на передачу компилятору файла XML с именем configuration.xml в параметре `-load-config`. Файл содержит всю информацию о том, как должно компилироваться приложение, включая местонахождение RSL:

```
<target name="compileApplication">
    <exec executable="${mxmlc}">
      <arg line="-load-config+=application/configuration.xml" />
    </exec>
  </target>

  <target name="compileAll" depends="compileRSL,compileApplication">
  </target>
```

В командных строках обоих компиляторов используется файл configuration.xml с информацией о местонахождении RSL, передаваемый mxmlc:

```
<flex-config>
  <compiler>
    <external-library-path>
      <path-element>example.swc</path-element>
    </external-library-path>
  </compiler>
  <file-specs>
    <path-element>RSLClientTest.mxml</path-element>
  </file-specs>
  <runtime-shared-libraries>
    <url>example.swf</url>
  </runtime-shared-libraries>
</flex-config>
```

Вместо включения флага `external-library-path` в командную строку mxmlc:

```
mxmlc -external-library-path=example.swc
```

файл configuration.xml передается в параметре load-config, а все параметры читаются из файла XML.

Аналогичный файл может передаваться compc:

```
<flex-config>
  <compiler>
```

```
    <source-path>
      <path-element>.</path-element>
    </source-path>
  </compiler>
  <output>example.swc</output>
  <include-classes>
    <class>oreilly.cookbook.shared.*</class>
  </include-classes>
</flex-config>
```

Полный файл Ant для этого рецепта:

```
<?xml version="1.0"?>
<project name="useRSL" basedir="./">

  <property name="mxmlc" value="C:\FlexSDK\bin\mxmlc.exe"/>
  <property name="compc" value="C:\FlexSDK\bin\compc.exe"/>

  <target name="compileRSL">
    <exec executable="${compc}">
      <arg line="-load-config+=rsl/configuration.xml" />
    </exec>
    <mkdir dir="application/rsl" />
    <move file="example.swc" todir="application/rsl" />
    <unzip src="application/rsl/example.swc"
        dest="application/rsl/" />
  </target>
  <target name="compileApplication">
    <exec executable="${mxmlc}">
      <arg line="-load-config+=
        application/configuration.xml" />
    </exec>
  </target>
  <target name="compileAll" depends="compileRSL,compileApplication">
  </target>
</project>
```

21.6. Создание и отслеживание выражений в отладчике Flex Builder

Задача

Требуется отслеживать изменения переменных в приложениях Flex в процессе их выполнения.

Решение

Используйте Flex Builder Debugger для запуска приложения; установите точку прерывания там, где нужно узнать значение переменной. Создайте новое выражение в окне Expressions отладчика Flex Builder.

Обсуждение

Выражения – мощный отладочный инструмент, который позволяет узнать значение любой переменной в области видимости. Создав выражение, вы сможете проанализировать содержимое любого объекта (рис. 21.2).

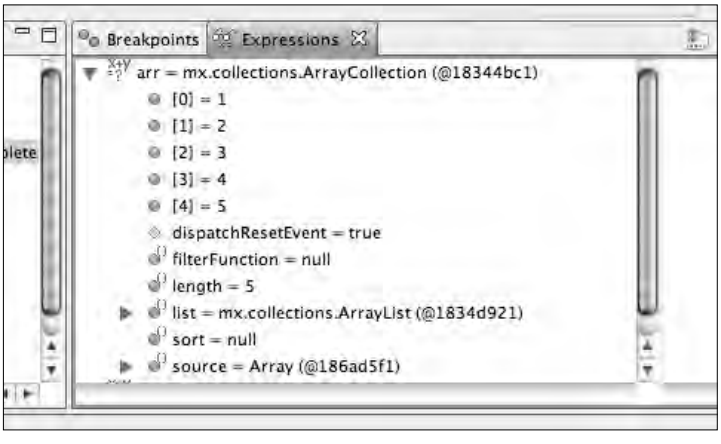

***Рис. 21.2.** Создание выражения*

Например, если установить точку прерывания в строке, в которой создается экземпляр массива (в следующем фрагменте она помечена комментарием *точка прерывания*):

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
    height="300" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            private var arr:ArrayCollection;

            private function init():void {
                arr = new ArrayCollection([1, 2, 3, 4, 5]);
                //точка прерывания
            }
            private function newFunc():void {
                var newArr:ArrayCollection =
                    new ArrayCollection([3, 4, 5, 6]);
            }

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

значение выражения `arr` будет равно `null`. Но если выполнить приложение в пошаговом режиме нажатием клавиши F6, значением выражения становится объект `ArrayCollection`, инкапсулирующий массив целых чисел (рис. 21.3).

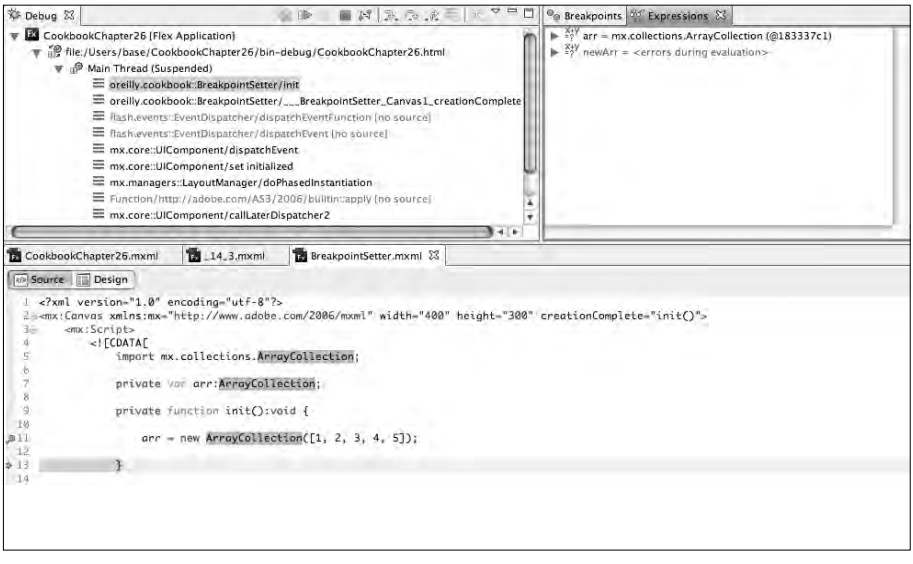

Рис. 21.3. *Выражение с текущим значением переменной*

Однако выражение `newArr` будет равно `null`, потому что переменная `newArr` находится вне области видимости (рис. 21.4).

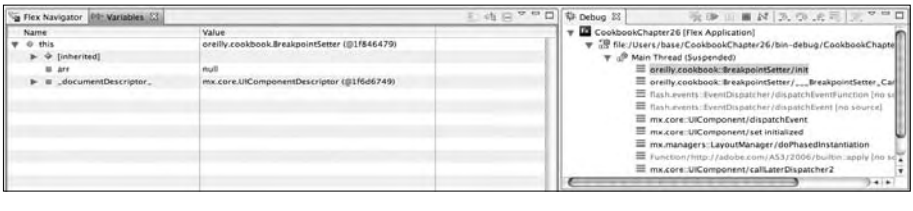

Рис. 21.4. *Проверяться могут только переменные в области видимости*

21.7. Установка Ant View в автономной версии Flex Builder

Материал предоставлен Райаном Тейлором (Ryan Tailor).

Задача

Вы не можете найти представление **Ant в автономной (stand-alone) версии** Flex Builder.

Решение

Установите Eclipse Java Development Tools.

Обсуждение

Чтобы получить доступ к Ant в автономной версии Flex Builder, необходимо установить Eclipse Java Development Tools. Это делается так:

1. Выберите в меню Flex Builder команду Help Software Updates Find and Install.
2. Выберите вариант Search for New Features и щелкните на кнопке Next.
3. Выберите в диалоговом окне вариант The Eclipse Project Updates и щелкните на кнопке Finish.
4. Открывается меню с предложением выбрать сетевой ресурс для загрузки файлов. Выберите любое место (желательно географически близкое для ускорения загрузки) и щелкните на кнопке OK.
5. Просмотрите различные версии SDK в дереве Eclipse Project Update; найдите среди них Eclipse Java Development Tools. Установите флажок рядом с этим узлом и щелкните на кнопке Next.
6. После того как Update Manager завершит загрузку нужных файлов, появляется диалоговое окно с запросом на подтверждение. Щелкните на кнопке Install All.
7. Дождитесь завершения установки и перезапустите Flex Builder.

Чтобы открыть представление Ant в Flex Builder, выполните команду Window Other Views Ant.

21.8. Создание файла сборки Ant для автоматизации стандартных задач

Материал предоставлен Райаном Тейлором (Ryan Taylor).

Задача

Требуется использовать возможности Ant для автоматизации стандартных задач (например, компиляции и построения документации).

Решение

Создайте файл сборки Ant и добавьте в него задачи для автоматизации своих процессов.

Обсуждение

Создать файл сборки Ant несложно, но это первый шаг на пути к автоматизации стандартных задач. Создайте документ XML с именем build.

xml, сохраните его в каталоге build в корневом каталоге проекта. Вообще говоря, сохранять файл именно в этом каталоге не обязательно, но это общепринятое соглашение.

Корневой узел файла сборки должен выглядеть примерно так:

```
<project name="MyAntTasks" basedir="..">
</project>
```

Задайте атрибуту `name` значение, уникальное для вашего проекта. Оно будет отображаться в представлении Ant в Eclipse. В атрибуте `basedir` должен быть задан корневой каталог проекта. Свойство `basedir` часто используется для определения других свойств, ссылающихся на файлы и подкаталоги в каталоге проекта.

Затем создаются дополнительные свойства для различных задач, которые будут добавлены позднее. Например, свойство со ссылкой на каталог исходного кода проекта определяется примерно так:

```
<project name="MyAntTasks" basedir="..">
    <property name="src" value="${basedir}/src" />
</project>
```

Пример также показывает, как использовать свойство после его объявления (синтаксис `${свойство}`).

Если вы определяете большое количество свойств, но при этом не хотите оставить в файле сборки минимум кода, объявите свойства в отдельном файле. Создайте новый тестовый файл с именем build.properties и сохраните его в одном каталоге с файлом build.xml. Файл содержит простые объявления свойств следующего вида:

```
src="${basedir}/src"
```

Например, свойства могут использоваться для определения путей к каталогу с исходным кодом, к каталогу bin и каталогу Flex 3 SDK. Вы довольно быстро поймете, какие свойства будут удобны в вашей конкретной ситуации, и сможете самостоятельно включить их в файл сборки.

См. также

Рецепт 21.3.

21.9. Компиляция приложения Flex с использованием mxmlc и Ant

Материал предоставлен Райаном Тейлором (Ryan Taylor).

Задача

Требуется включить в файл сборки Ant задачи для компиляции приложения.

Решение

Включите в файл сборки Ant задачи, использующие компилятор MXML для компиляции файлов.

Обсуждение

Самую распространенную (и самую полезную) категорию целей, включаемых в файлы сборки Ant, составляют цели компиляции. Приложения Flex компилируются программой mxmlc – бесплатным компилятором командной строки, включенным в **Flex 3 SDK. Включение целей компиляции** в файл сборки позволяет автоматизировать процесс построения приложения: Ant компилирует все файлы, а вам даже не приходится открывать окно командной строки или терминала.

Компилятор MXML (mxmlc) включается в SDK в нескольких форматах. Для исполняемой версии создается цель следующего вида:

```
<!-- COMPILE MAIN -->
<target name="compileMain"

    description="Compiles the main application files.">
    <echo>Compiling '${bin.dir}/main.swf'...</echo>
    <exec executable="${FLEX_HOME}/bin/mxmlc.exe" spawn="false">
        <arg line="-source-path '${src.dir}'" />
        <arg line="-library-path '${FLEX_HOME}/frameworks'" />
        <arg line="'${src.dir}/main.mxml'" />
        <arg line="-output '${bin.dir}/main.swf'" />
    </exec>

</target>
```

Также можно воспользоваться Java-версией; в этом случае задача выглядит немного иначе:

```
<!-- COMPILE MAIN -->
<target name="compileMain"

    description="Compiles the main application files.">
    <echo>Compiling '${bin.dir}/main.swf'...</echo>
    <java jar="${FLEX_HOME}/lib/mxmlc.jar" fork="true"
        failonerror="true">
        <arg value="+flexlib=${FLEX_HOME}/frameworks" />
        <arg value="-file-specs='${src.dir}/main.mxml'" />
        <arg value="-output='${bin.dir}/main.swf'" />
    </java>

</target>
```

Последний (и вероятно, лучший) вариант основан на использовании дополнительных задач mxmlc, включенных в поставку Flex 3 SDK. Установка этих задач описана в рецепте 21.3. Для обращения к ним из файла сборки необходимо сначала добавить определение задачи:

```
<!-- TASK DEFINITIONS -->
<taskdef resource="flexTasks.tasks"
    classpath="${FLEX_HOME}/ant/lib/flexTasks.jar" />
```

Импортирование дополнительных задач Flex позволяет компилировать их в более интуитивно понятном синтаксисе, а также использовать средства обнаружения ошибок в инструментах типа Eclipse при написании задачи. Пример:

```
<!-- COMPILE MAIN -->
<target name="compileMain"
    description="Compiles the main application files.">
    <echo>Compiling '${bin.dir}/main.swf'...</echo>
    <mxmlc file="${src.dir}/main.mxml"
        output="${bin.dir}/main.swf">
            <source-path path-element="${src.dir}" />
    </mxmlc>
</target>
```

Во всех этих примерах действуют одни и те же основные правила: вы должны определить свойства со ссылками на каталоги bin и src своего проекта, а также каталог Flex 3 SDK. Всем свойствам в этих примерах назначены рекомендуемые имена, кроме `FLEX_HOME` (это имя является обязательным). Свойство `FLEX_HOME` *должно* ссылаться на корневой каталог Flex 3 SDK. При использовании EXE- или JAR-версии mxmlc допускается использование других имен свойств, отличных от `FLEX_HOME`.

Однако истинная мощь компиляции проектов с применением Ant проявляется в цепочечном объединении целей. Например, вы можете создать цель `compileAll`, которая последовательно вызывает все отдельные цели компиляции:

```
<!-- COMPILE ALL -->
<target name="compileAll" description="Compiles all application files."
    depends="compileMain, compileNavigation, compileGallery,
    compileLibrary">
    <echo>Finishing compile process...</echo>
</target>
```

На первый взгляд примеры Ant выглядят устрашающе; но после непродолжительного использования Ant и конфигурационных файлов вы поймете, что они значительно повышают эффективность вашей работы. В частности, автоматизируя процесс компиляции при помощи сторонних инструментов (таких, как Ant), вы избавляетесь от привязки к одной конкретной среде разработки и можете легко задействовать Ant в любой среде разработки по своему усмотрению, например Flex Builder, FDT, TextMate или FlashDevelop.

См. также

Рецепт 21.3.

21.10. Построение документации в ASDoc и Ant

Материал предоставлен Райаном Тейлором (Ryan Taylor).

Задача

Требуется построить документацию для вашего приложения.

Решение

Включите в файл сборки Ant исполняемую задачу, которая использует ASDoc (входит в Flex 3 SDK) для построения документации.

Обсуждение

Бесплатная утилита построения документации ASDoc входит в поставку Flex 3 SDK. Стиль создаваемой в ASDoc документации знаком каждому, кто когда-либо использовал Adobe LiveDocs. Хотя открыть окно командной строки или терминала несложно, лучше включить цель в файл сборки Ant для дальнейшей автоматизации процесса.

Прежде чем создавать цель для построения документации, желательно создать дополнительную цель для очистки каталога docs. Определите свойство `docs.dir` и направьте его на каталог docs проекта:

```
<!-- CLEAN DOCS -->
<target name="cleanDocs"

    description="Cleans out the documentation directory.">
    <echo>Cleaning '${docs.dir}'...</echo>
    <delete includeemptydirs="true">
        <fileset dir="${docs.dir}" includes="**/*" />

    </delete>
</target>
```

Подготовив цель для очистки каталога документации, можно переходить к созданию цели, которая генерирует документацию. Обратите внимание: в следующем примере атрибут `depends` требует, чтобы цель `cleanDocs` была выполнена до инструкций, генерирующих документацию:

```
<!-- GENERATE DOCUMENTATION -->
<target name="generateDocs" description="Generates application
    documentation using ASDoc." depends="cleanDocs">

    <echo>Generating documentation...</echo>
    <exec executable="${FLEX_HOME}/bin/asdoc.exe" failOnError="true">

        <arg line="-source-path ${src.dir}" />
        <arg line="-doc-sources ${src.dir}" />
        <arg line="-main-title ${docs.title}" />
        <arg line="-window-title ${docs.title}" />
```

```
        <arg line="-footer ${docs.footer}" />
        <arg line="-output ${docs.dir}" />
    </exec>
</target>
```

Свойство `FLEX_HOME` должно содержать ссылку на корневой каталог Flex 3 SDK на вашем компьютере. Свойства `src.dir` и `docs.dir` представляют соответственно каталоги src и docs вашего проекта. Наконец, свойства `docs.title` и `docs.footer` задают тексты заголовка и завершения, отображаемые в документации. В завершителе часто размещается информация об авторских правах и URL-адреса.

ASDoc успешно генерирует документацию на основе программного кода, даже если в нем нет ни единого комментария. Конечно, на практике рекомендуется тщательно документировать код в формате Javadoc. Это не только сделает документацию более содержательной, но и поможет разобраться в коде посторонним программистам.

21.11. Компиляция приложений Flex с использованием Rake

Задача

Требуется откомпилировать приложение Flex с использованием Rake (утилиты построения программ для Ruby).

Решение

Загрузите и установите Ruby 1.9, если это не было сделано ранее, после чего загрузите и установите Rake.

Обсуждение

Хотя утилита Rake написана полностью на Ruby, она очень похожа на классическую утилиту make, используемую программистами C++ и C. После загрузки и установки Ruby и Rake создается простой файл Rake, который выглядит примерно так:

```
task :default do
    DEV_ROOT = "/Users/base/flex_development"
    PUBLIC = "#{DEV_ROOT}/bin"
    FLEX_ROOT = "#{DEV_ROOT}/src"
    system "/Developer/SDKs/Flex/bin/mxmlc
        --show-actionscript-warnings=true
        --strict=true -file-specs #{FLEX_ROOT}/App.mxml"
    system "cp #{FLEX_ROOT}/App.swf #{PUBLIC}/App.swf"
end
```

Задачи Rake являются аналогами целей Ant; иначе говоря, они определяют выполняемые действия. Действие по умолчанию (`default`) выполняется всегда, а прочие действия вызываются в других задачах.

Сами задачи могут содержать объявления переменных и использовать системные аргументы:

```
system "/Developer/SDKs/Flex/bin/mxmlc
    --show-actionscript-warnings=true
    --strict=true -file-specs #{FLEX_ROOT}/App.mxml"
```

В этом фрагменте содержится непосредственный вызов компилятора MXML, который создает файл SWF. Поскольку элемент в задаче Rake не выполняется до возврата управления предыдущей задачей, следующая строка может считать, что файл SWF был успешно построен и теперь его можно скопировать в другой каталог:

```
system "cp #{FLEX_ROOT}/App.swf #{PUBLIC}/App.swf"
```

В оставшейся части файла Rake объявляются переменные, которые будут использоваться для размещения файлов в нужных папках. Файл можно сохранить под любым именем и запустить в командной строке командой `rake`. Например, если файл был сохранен под именем `Rakefile`, команда его запуска будет выглядеть так:

```
rake Rakefile
```

21.12. Использование ExpressInstall в приложениях

Задача

Если у пользователя установлена слишком старая версия Flash Player, не подходящая для просмотра приложения Flex, необходимо автоматически установить новую версию.

Решение

Используйте при компиляции параметр `ExpressInstall`, чтобы при необходимости перенаправить пользователя на сайт Adobe для установки обновленной версии Flash Player.

Обсуждение

Чтобы использовать режим Express Install, необходимо установить параметр Use Express Install в конфигурации приложения в Flex Builder (рис. 21.5).

Если вы не используете Flex Builder для разработки, установите переменную `pluginspage` в теге `Object` тега `embed` страницы HTML, в которую встроен файл SWF:

```
pluginspage="http://www.adobe.com/go/getflashplayer"
```

Пример директивы `<embed>` для броузера на базе Netscape (например, Firefox):

```
<embed src="CookbookChapter26.swf" id="CookbookChapter26"
    quality="high" bgcolor="#869ca7" name="CookbookChapter26"
    allowscriptaccess="sameDomain" pluginspage="http://www.adobe.com/go/
    getflashplayer" type="application/x-shockwave-flash" align="middle"
    height="100%" width="100%">
```

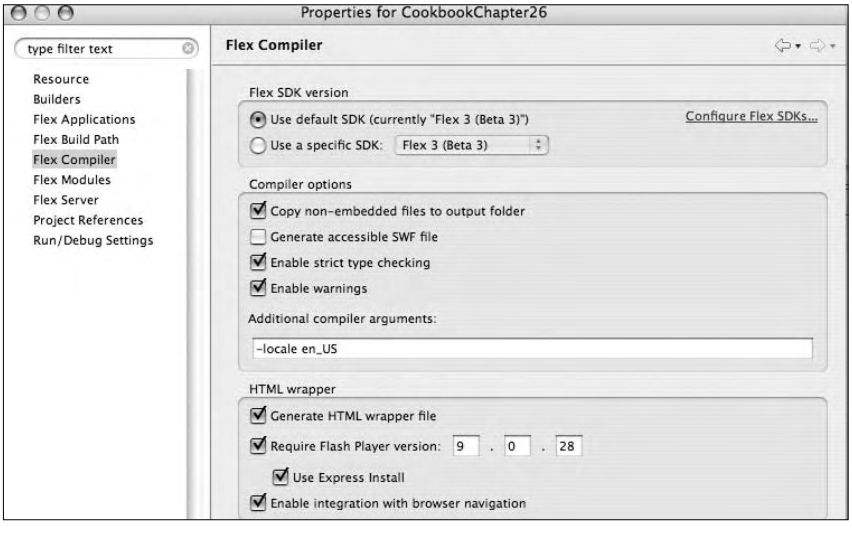

Рис. 21.5. *Включение режима Express Install*

21.13. Профилирование памяти в Flex Builder 3

Задача

Требуется просмотреть все объекты, созданные в памяти Flash Player, во время выполнения.

Решение

Используйте представление Memory Profiler в Flex Builder 3 для запуска приложения и просмотра информации о создаваемых и уничтожаемых объектах.

Обсуждение

Flex Profiler принадлежит к числу новшеств Flex Builder 3. Это мощный инструмент для получения информации о выделении и освобождении памяти. Flex Profiler подключается к приложению через локальный сокет. Возможно, для его использования придется отключить антивирусную программу (если она запрещает передачу данных через сокеты).

В ходе выполнения Profiler каждые несколько миллисекунд получает «снимок» данных и сохраняет текущее состояние Flash Player. Обраба-

тывая полученные данные, Profiler выдает информацию о ходе операций приложения. Он регистрирует время выполнения операции, а также общую структуру памяти объектов Flash Player на момент сохранения.

При запуске приложения в Profiler на экране появляется диалоговое окно Connection Established (рис. 21.6). В нем включается профилирование для выявления проблем с выделением памяти в приложении, а также профилирование быстродействия для ускорения работы приложения.

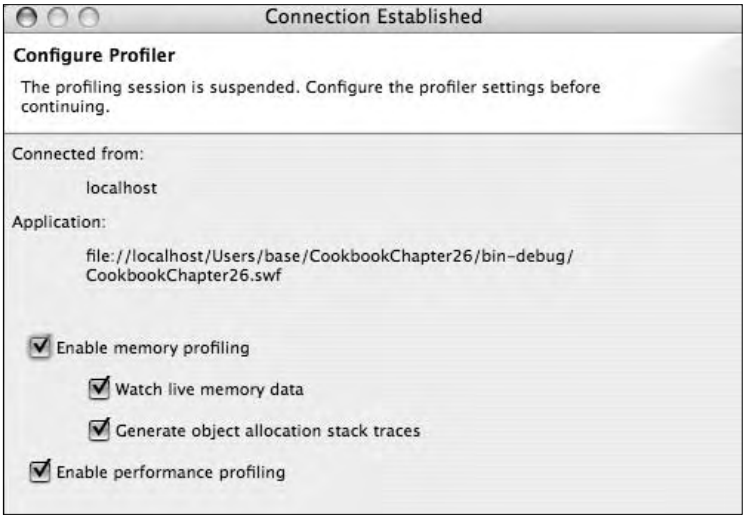

Рис. 21.6. *Выбор типа профилирования*

При установке флажка **Watch Live Memory Data в представлении profiling** отображаются активные графики жизненного цикла объектов, создаваемых в Flash Player (рис. 21.7).

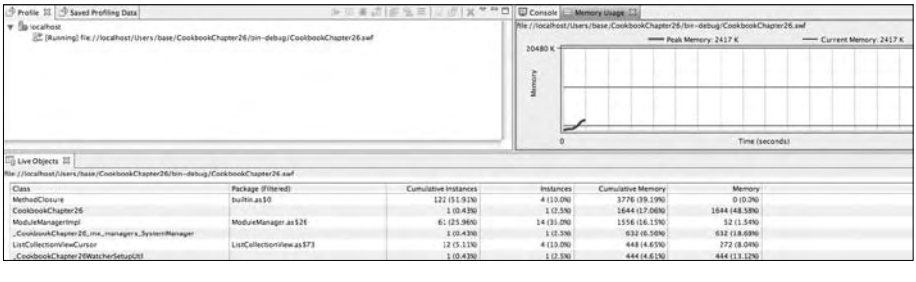

Рис. 21.7. *Активные графики и данные о выделении памяти в представлении Profiling*

Profiler дает возможность в любой момент получить «моментальный снимок» состояния памяти, получить данные о количестве экземпляров любого объекта и памяти, необходимой для их хранения (рис. 21.8).

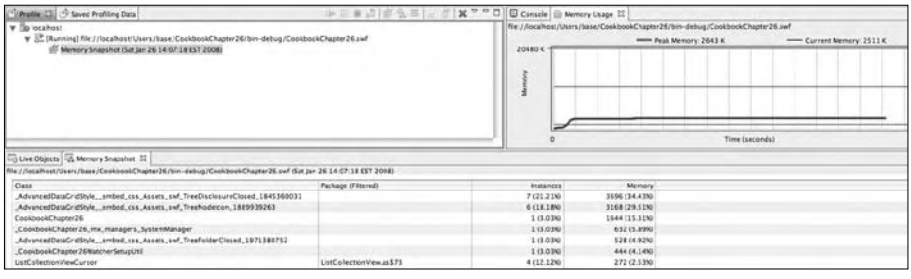

Рис. 21.8. *Информация о количестве экземпляров и затратах памяти*

Наконец, сравнение двух «снимков» состояния памяти позволит найти объекты, созданные после первого «снимка», но существующие во втором «снимке». Информация об имени класса, затратах памяти и количестве экземпляров хранится в представлении Loitering Objects (рис. 21.9).

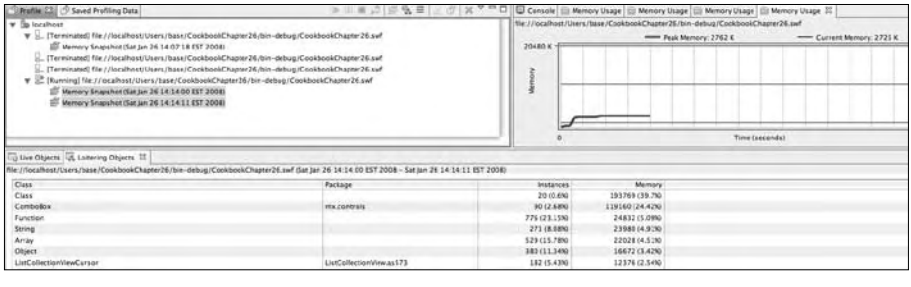

Рис. 21.9. *Просмотр недавно созданных объектов в представлении Loitering Objects*

22

Настройка, интернационализация и печать

Чтобы ваши приложения были доступны для самого широкого круга пользователей, Flex 3 представляет широкие возможности настройки доступности для пользователей с ограниченными возможностями, интернационализации и печати. Например, если ваш проект должен удовлетворять повышенным стандартам доступности, средства обнаружения «экранного диктора» и перебора с клавиатуры помогут пользователям со слабым зрением и тем пользователям, у которых возникают затруднения с использованием указательных устройств. Поддержка локализации и интернационализации в Flex 3 также была заметно усовершенствована. Среди новых средств локализации стоит отметить встроенный менеджер ресурсов интернационализации, возможность определения и переключения локального контекста во время выполнения и возможность запроса ресурсных модулей во время выполнения. Если вам понадобятся печатные материалы для раздачи, новейшая версия Flex поможет и в этом. Flex 3 дает возможность печатать компоненты Flex и содержит специальный компонент для печати повторяющихся многостраничных данных.

22.1. Международные символы в приложении

Задача

Требуется вывести в приложении текст в идеографическом письме (например, китайском или корейском).

Решение

Используйте встроенные шрифты, заведомо доступные для Flash Player.

Обсуждение

Приложения Flex могут выводить текст на языках с расширенным набором символов, включая текст в кодировке Юникод (например, знаки корейской или китайской письменности), при условии, что шрифт с такими символами доступен для Flash Player. Разработчик может включить нужный шрифт в приложение точно так же, как это делается с западными шрифтами. Однако следует помнить, что за удобство приходится расплачиваться: большое количество символов в большинстве идеографических языков приводит к разрастанию файла SWF. Принимая решение об использовании встроенных шрифтов, сравните достоинства (правильный вывод текста) с недостатками (увеличение SWF).

В следующем примере ChineseFonts.mxml представлены два подхода к выводу международных символов.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
  <mx:Style>
      @font-face {
        src: local("LiSong Pro");
        fontFamily: EmbeddedChinese;
        fontStyle: normal;
        fontWeight: normal;
       }
  </mx:Style>
  <mx:Form>
   <mx:FormItem label="System Font">
       <mx:Label text="快的棕色狐狸慢慢地跳過了懶惰灰色灰鼠" />
   </mx:FormItem>
   <mx:FormItem label="Embedded Font">
    <mx:Label fontFamily=
        "EmbeddedChinese" text="快的棕色狐狸慢慢地跳過了懶惰灰色灰鼠" />
   </mx:FormItem>
  </mx:Form>
</mx:Application>
```

При запуске приложения ChineseFonts.mxml вы увидите китайский текст рядом с надписью `System Font` только в том случае, если необходимые символы присутствуют в шрифте вашей системы. Строка `Embedded Font` нормально выводится во всех системах.

В коде MXML метода, использующего текст в кодировке Юникод, нет ничего особенного. Загружаемые им данные содержат текст на упрощенном китайском языке:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Style>
        @font-face {
            src: local("LiSong Pro");
            fontFamily: EmbeddedChinese;
```

```
            fontStyle: normal;
            fontWeight: normal;
        }
    </mx:Style>
    <mx:XML source="books.xml" id="booksData" />
    <mx:VBox fontFamily="EmbeddedChinese">
       <mx:Repeater id="iterator" dataProvider="{booksData.book}">
          <mx:VBox backgroundColor="0xffffff">
            <mx:Label text="{iterator.currentItem.@title}" />
            <mx:Text width="200" text="{iterator.currentItem.toString()}" />
            <mx:HRule width="200" />
          </mx:VBox>
       </mx:Repeater>
    </mx:VBox>
</mx:Application>
```

Пример загруженного документа:

```
<books>
    <book title="阿波罗为Adobe 导电线开发商口袋指南">
    现在您能建立和部署基于闪光的富有的互联网应用(RIAs) 对桌面使用Adobe 的导电线框架。由阿波罗产品队的成员写，这是正式指南对于 Adobe 阿波罗，新发怒平台桌面运行时间阿尔法发行从Adobe 实验室。众多的例子说明怎么阿波罗工作因此您可能立即开始大厦 RIAs 为桌面。
    </book>
    <book title="编程的导电线2">
    编程的导电线 2 谈论导电线框架在上下文。作者介绍特点以告诉读者不仅怎样，而且原因为什么使用一个特殊特点，何时使用它，和何时不是的实用和有用的例子。这本书被写为发展专家。当书不假设观众早先工作了以一刹那技术，读者最将受益于书如果他们早先建立了基于互联网，ntiered 应用。
    </book>
    <book title="ActionScript 3.0 设计样式">
    如果您是老练的闪光或屈曲开发商准备好应付老练编程技术与 ActionScript 3.0，这实践介绍逐步设计样式作为您通过过程。您得知各种各样的类型设计样式和修建小抽象例子在尝试您的手之前在大厦完全的运作的应用被概述在书。
    </book>
</books>
```

22.2. Применение групп ресурсов для локализации приложений

Задача

Требуется обеспечить поддержку небольшого количества альтернативных языков в приложении.

Решение

Оформите локализованные ресурсы в виде группы ресурсов (resource bundle).

Обсуждение

Группы ресурсов обеспечивают простейшую локализацию в приложениях Flex. Они представляют собой объекты ActionScript, предоставляющие интерфейс для работы с локализованным контентом из файлов свойств либо через привязку данных, либо из кода ActionScript. Каждая группа, используемая в приложении, представляет один файл свойств локализации. *Файл свойств* (*property file*) представляет собой текстовый файл со списком ключей свойств локализации и связанных с ними значений. Пары «ключ-значение» определяются в файле в формате `ключ=значение`, а файл сохраняется с расширением .properties.

Локализованные текстовые строки, встроенные ресурсы (например, графики), ссылки на определения классов ActionScript – все это может определяться в файлах свойств. При локализации приложения в файле свойств создается отдельная запись для каждого элемента приложения, изменение которого необходимо для полноценной локализации альтернативных языков. Пример файла свойств с определениями нескольких свойств для американской версии английского языка:

```
# Ресурсы локализации для американского английского языка
pageTitle=Internationalization Demo
language=American English
flag=Embed("assets/usa.png")
borderSkin=ClassReference("skins.en_US.LocalizedSkin")
```

В ходе локализации приложения необходимо создать отдельную копию файла свойств для каждого поддерживаемого языка. Скажем, если приложение должно поддерживать американский английский и французский языки, вы создаете второй файл свойств с переведенным текстом, ссылкой на изображение французского флага (вместо американского) и ссылкой на скин для франкоязычной версии приложения:

```
# Ресурсы локализации, En Francais
pageTitle=Demo d'internationalisation
language=Francais
flag=Embed("assets/france.png")
borderSkin=ClassReference("skins.fr_FR.LocalizedSkin")
```

При создании файла свойств необходимо учитывать ряд факторов, прежде всего размер и сложность приложения. Возможны разные решения, например создание отдельного файл свойств для каждого пользовательского компонента в приложении или для пакета взаимосвязанных компонентов с совместно используемыми ресурсами. Также можно определить файлы свойств, используемые в глобальном масштабе, например файл с сообщениями об ошибках или стандартными надписями (скажем, на кнопках).

Какой бы вариант разбиения свойств локализации вы ни выбрали, создайте структуру каталогов для упорядочения файлов. На рис. 22.1 показан оптимальный вариант: основной каталог с именем locale или localization, содержащий подкаталоги с именами идентификаторов локальных контекстов. В подкаталогах хранятся все файлы свойств для

локального контекста. Используя подобную структуру, вы без труда «объясните» компилятору, где следует искать файлы свойств.

При построении приложения компилятор создает для каждого файла свойств класс, производный от `ResourceBundle`. К элементам, определяемым в файле свойств, проще всего обращаться при помощи директивы `@Resource`; в этом случае вам вообще не придется писать код, работающий с экземплярами `ResourceBundle`; компилятор сделает все за вас. Директива `@Resource` получает два аргумента: идентификатор группы и ключ, используемый для поиска соответствующего значения в файле свойств. Например, для обращения к свойству `applicationTitle` в файле свойств используется следующая конструкция:

```
@Resource(key='applicationTitle', bundle='localizationProperties')
```

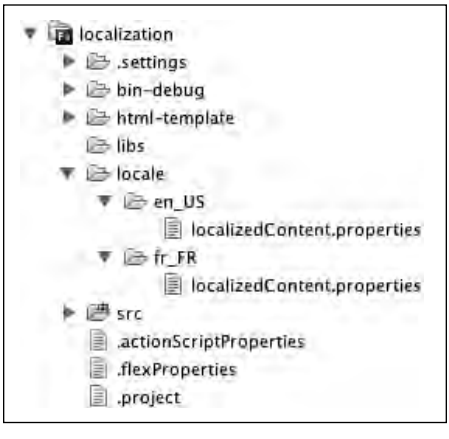

***Рис. 22.1.** Структура каталогов для файлов свойств локализации*

В более подробном примере LocalizationResource.mxml определяется небольшое приложение, использующее файлы свойств из двух предшествующих фрагментов:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Metadata>
        [ResourceBundle("localizedContent")]
    </mx:Metadata>
    <mx:VBox    horizontalCenter="0"
                verticalCenter="0"
                horizontalAlign="center"
                borderSkin="@Resource(key='borderSkin',
                    bundle='localizedContent')">
        <mx:Label fontSize="24"
            text="@Resource(key='pageTitle',
            bundle='localizedContent')" />
        <mx:Label fontSize="24" text="@Resource(key='language',
           bundle='localizedContent')" />
```

```
        <mx:Image source="@Resource(key='flag',bundle='localizedContent')" />
    </mx:VBox>
</mx:Application>
```

Метаданные `[ResourceBundle]` сообщают компилятору, что для данного компонента необходимы группы ресурсов. Это важно, поскольку группы ресурсов строятся на стадии компиляции, а все необходимые ресурсы для поддерживаемых языков должны быть скомпилированы в файл SWF приложения.

Компилятор также должен быть настроен для поддержки локализации в нужных локальных контекстах. В Flex Builder 3 эти параметры задаются в диалоговом окне свойств проекта Flex. На панели Flex Build Path определяется исходный путь, ведущий к файлам локализации. Если вы последовали проверенным рекомендациям и создали каталог locale, то путь будет иметь вид `locale/{locale}` (рис. 22.2). Также необходимо перечислить поддерживаемые локальные контексты в поле Additional compiler arguments на панели Flex Compiler. Например, для поддержки американского английского и французского локальных контекстов следует ввести строку `-locale en_US,fr_FR` (рис. 22.3). Во время построения приложения и поиска файлов свойств компилятор подставляет каждый идентификатор локального контекста в выражение пути, т. е. путь принимает вид `locale/en_US` для файлов свойств американского английского и `locale/fr_FR` для файлов свойств французского языка.

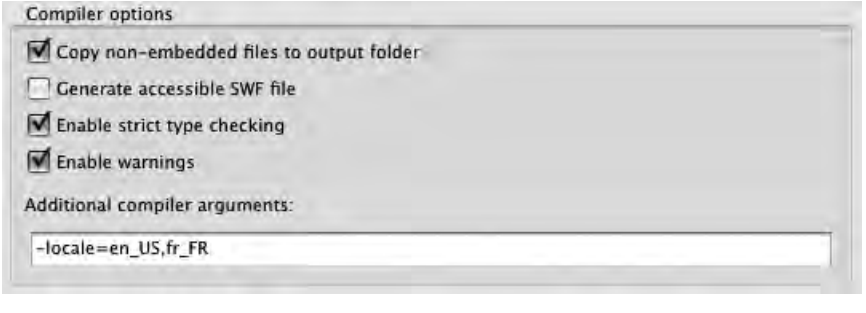

Рис. 22.2. *Аргументы компилятора, относящиеся к локализации*

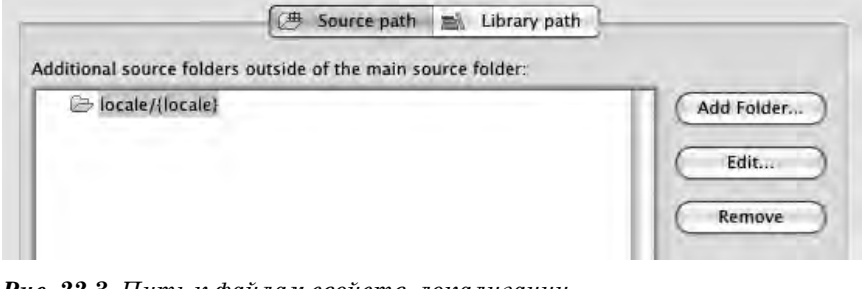

Рис. 22.3. *Путь к файлам свойств локализации*

Перед тем как приступать к построению локализованных приложений, необходимо выполнить еще одну операцию: локализовать соответствующий контент Flex Framework (например, сообщения об ошибках). Воспользуйтесь программой командной строки copylocale, предоставленной фирмой Adobe, для копирования этих файлов в новый локальный контекст. Эта операция должна быть выполнена один раз для каждого локального контекста, но созданная копия будет доступна для любого проекта, строящегося в данной установке Flex Framework. Учтите, что команда copylocale не создает локализованные копии файлов, а лишь позволяет откомпилировать приложение. Команда copylocale находится в подкаталоге bin установки Flex 3 SDK. При запуске ей передается идентификатор локального контекста по умолчанию и идентификатор того локального контекста, для которого создается копия:

```
Copylocale.exe en_US fr_FR
```

До выхода Flex 3 локальные контексты не могли переключаться во время выполнения. Это означало, что локализованные приложения можно было строить только одним способом – компиляцией отдельной копии приложения для каждого поддерживаемого локального контекста. В некоторых ситуациях этот подход приемлем, например, если объем локализации невелик, а файл SWF должен иметь минимальный размер. Чтобы использовать этот способ, передайте в аргументе компилятора `-locale` нужный локальный контекст и постройте свое приложение. Два основных преимущества этого способа – простота и малый размер файла SWF (так как компилятор включает только один комплект файлов свойств локализации). Если вам захочется увидеть этот способ в действии, попробуйте откомпилировать пример LocalizationResource.mxml для американского английского (`en_US` – рис. 22.4) или французского языка (`fr_FR` – рис. 22.5) с передачей локального контекста в аргументе компилятора.

Рис. 22.4. *Приложение LocalizationResource.mxml с аргументом компилятора "-locale=en_US"*

Рис. 22.5. *Приложение LocalizationResource.mxml с аргументом компилятора "-locale=fr_FR"*

22.3. Локализация с использованием ResourceManager

Задача

Требуется обеспечить поддержку небольшого количества локальных контекстов, которые должны определяться либо на программном уровне во время выполнения, либо выбираться пользователем.

Решение

Используйте класс `ResourceManager` для поддержки нескольких локальных контекстов. Обеспечьте возможность их переключения во время выполнения.

Обсуждение

Класс `ResourceManager` предоставляет программисту Flex основной интерфейс ActionScript для взаимодействия с группами ресурсов, создаваемыми компилятором. Он позволяет извлекать различные виды ресурсов из групп и предоставляет механизм для назначения локального контекста во время выполнения. Менеджер ресурсов представляет собой единственный (синглетный) экземпляр, обеспечивающий локализацию всего приложения. Каждый класс, производный от `UIComponent`, содержит защищенное свойство `resourceManager` со ссылкой на экземпляр менеджера ресурсов.

Директива `@Resource`, отлично подходящая для привязки локализованного контекста в тегах MXML, значительно менее удобна для «чистых» компонентов и методов ActionScript, зависящих от локализованных ресурсов. В таких ситуациях лучше использовать менеджер ресурсов, который предоставляет методы для обращения к локализованным данным и может использоваться в качестве приемника в выражениях привязки данных в коде ActionScript или MXML. В следующем фрагменте из приложения LocalizationManager.mxml директивы `@Resource` из приложения LocalizationResource.mxml (см. рецепт 22.2) заменяются методами `ResourceManager`:

```
<mx:VBox     horizontalCenter="0"
        verticalCenter="0"
        horizontalAlign="center"
        borderSkin="{resourceManager.getClass(
            'localizedContent', 'borderSkin')}">
    <mx:Label fontSize="24" text="{
        resourceManager.getString('localizedContent', 'pageTitle')}" />
    <mx:Label fontSize="24" text="{
        resourceManager.getString('localizedContent', 'language')}" />
    <mx:Image source="{resourceManager.getClass(
            'localizedContent', 'flag')}" />
</mx:VBox>
```

Имена методов зависят от типа ресурсов, аргументы в целом напоминают аргументы директивы `@Resource` (имя группы ресурсов и ключ свойства).

Привязка значений свойств с использованием менеджера ресурсов имеет дополнительное преимущество: в Flex 3 локальные контексты могут переключаться во время выполнения, и вам уже не придется строить отдельный локализованный файл SWF для каждого поддерживаемого контекста. Привязка свойств к методам ресурсов позволяет приложению оперативно перейти на нужный контекст. В примере LocalizationManager.mxml созданы кнопки для переключения между английским и французским языком:

```
<mx:HBox>
    <mx:Button label="In English"
        click="resourceManager.localeChain = ['en_US']" />
    <mx:Button label="En Francais"
        click="resourceManager.localeChain = ['fr_FR']" />
</mx:HBox>
```

Свойство `localeChain` изменяется в зависимости от того, какую кнопку выберет пользователь. Свойство `localeChain` содержит массив строк, представляющий упорядоченный список локальных контекстов. Например, оно пригодится при передаче приложению информации о языковых предпочтениях пользователя от броузера через заголовок HTTP Accept-Language или языковые предпочтения операционной системы хоста. Например, для пользователя из Великобритании предпочтительным является локальный контекст `en_GB`, но пользователь также может принять контекст `en_US`. При вызове метода менеджера ресурсов производится поиск группы ресурсов с заданным именем в одном из локальных контекстов цепочки в порядке их следования. Таким образом, приложение, локализованное для контекста `en_US`, будет успешно работать у пользователя из Великобритании при следующем значении `localeChain`:

```
resourceManager.localeChain = ["en_GB", "en_US"];
```

Проследите за тем, чтобы где-то в списке присутствовал локальный контекст `en_US`. Многие компоненты Flex Framework зависят от присутствия ресурсов локализации и выдают ошибку, если им не удается найти локализованный контент. Включение `en_US` в конец цепочки свойств защитит классы Framework от сбоев, обусловленных тем, что им не удалось найти локализованные ресурсы.

Если менеджер ресурсов используется приложением для обычной привязки, все делается практически так же, как при использовании директив `@Resource`. Но если приложение должно поддерживать несколько локальных контекстов, один из которых выбирается на стадии выполнения, ресурсы всех поддерживаемых контекстов должны быть откомпилированы в составе приложения. Передайте компилятору список поддерживаемых контекстов, разделенных запятыми, вместо одного контекста (см. рецепт 22.2).

22.4. Локализация с использованием ресурсных модулей

Задача

Требуется обеспечить поддержку большого количества локальных контекстов в приложении.

Решение

Используйте ресурсные модули для загрузки только той поддержки локальных контекстов, которая необходима приложению на стадии выполнения.

Обсуждение

Группы ресурсов при компиляции включаются в приложение, в результате чего файл SW увеличивается с каждым поддерживаемым локальным контекстом. Подавляющее большинство пользователей на практике ограничивается ресурсами одного локального контекста, в результате чего загружаемый файл приложения содержит большой объем «балласта». В Flex 3 была добавлена возможность компиляции групп ресурсов локального контекста в *ресурсные модули*, загружаемые приложением динамически на стадии выполнения. Вы можете на программном уровне определить предпочтительный контекст и загрузить только тот ресурсный модуль, который необходим для выбранного контекста.

Чтобы построить ресурсные модули для файлов свойств локализации, необходимо сначала определить, какие ресурсы необходимы для вашего приложения. Учитываются не только ресурсы, определяемые разработчиком, но и ресурсы, необходимые для Flex Framework. Компилятор mxmlc может проанализировать приложение и выдать список необходимых ресурсов. Это можно сделать в Flex Builder 3 при помощи поля Additional Compiler Arguments в диалоговом окне свойств проекта, но задача также легко решается из командной строки. Это избавит вас от необходимости возвращаться к окну свойств при каждом обновлении списка. Если при вызове компилятора локальный контекст не указан, компилятор записывает результаты анализа в файл:

```
mxmlc -locale= -resource-bundle-list=resources.txt ResourceModules.mxml
```

После завершения команды выходной файл `resources.txt` выглядит примерно так:

```
bundles = containers controls core effects localizedContent skins styles
```

По этим данным можно сообщить компилятору, какие группы ресурсов должны быть включены в ресурсный модуль. Компиляция приложения с использованием ресурсных модулей осуществляется только компилятором командной строки. Укажите путь к файлам свойств локализации,

список групп ресурсов из предыдущего шага, а также имя итогового файла SWF. Компилятор строит группы ресурсов из файлов свойств и упаковывает их в один файл SWF. Например, для построения групп ресурсов примера ResourceModule.mxml из рецепта 22.4 используется следующая команда:

```
mxmlc -locale=en_US -source-path=.,locale/{locale}
-include-resource-bundles=containers,controls,core,effects,
    localizedContent,skins,styles
-output en_US_resources.swf
```

Команда компиляции ресурсных модулей для французского языка:

```
mxmlc -locale=fr_FR -source-path=.,locale/{locale}
-include-resource-bundles=containers,controls,core,effects,
    localizedContent,skins,styles
-output fr_FR_resources.swf
```

В этой команде заслуживает внимания ряд обстоятельств. Во-первых, локальный контекст, используемый при посторении, указывается в аргументе `-locale`, так же как при компиляции групп ресурсов в составе приложения. Во-вторых, хотя аргумент `-source-path` выглядит знакомо, в данном случае важно включить в него признак текущего каталога – точку (.). Пример содержит ссылку на встроенный класс в файле свойств `localizedContent`, и без указания корневого каталога приложения у компилятора могут возникнуть проблемы с разрешением ссылки на класс. Обратите внимание: предполагается, что **mxmlc запускается из корневого** каталога иерархии исходного кода проекта. Аргумент `-include-resource-bundles` заполняется на основании списка, сгенерированного в предыдущем примере. Список разделяется запятыми, а разделители не могут отделяться от имен групп пробелами. Наконец, команда приказывает компилятору записать выходные данные в файл **en_US_resources.swf**. Выходному файлу SWF можно присвоить любое имя, но на практике рекомендуется включать в имя файла идентификатор локального контекста. Это позволит на программном уровне определить имя ресурсного модуля, который должен быть загружен приложением, по идентификатору локального контекста.

При компиляции ресурсных модулей программой mxmlc ссылки на встроенные ресурсы (например, графику) разрешаются относительно местонахождения файлов свойств локализации, содержащих ссылку. Если приложение использует группы ресурсов со встроенными ресурсами, определяемыми относительно корневого каталога исходного кода проекта, их придется обновить.

В приложении используется метод `loadResourceModule` менеджера ресурсов. При вызове методу передается URL-адрес, идентифицирующий файл **SWF ресурсного модуля, который вы хотите использовать. Ме**тод сходен с другими механизмами загрузки объектов ActionScript во время выполнения, такими как `SWFLoader` или традиционные модули. Приложение обращается к серверу с запросом на получение нужного

файла SWF, который принимается броузером. Для запроса ресурсов из других доменов потребуется файл междоменной политики. Прежде чем использовать ресурсные модули в приложении, необходимо дождаться завершения их загрузки. Когда ресурсный модуль будет готов к использованию, передается событие `ResourceEvent`. Чтобы прослушивать эти события, определите слушателя для событий `ResourceEvent.COMPLETE`. Метод `loadResourceModule` возвращает ссылку на объект, реализующий интерфейс `IEventDispatcher`, который используется для регистрации слушателей. Следующий фрагмент примера **ResourceModules.mxml** демонстрирует загрузку и использование ресурсных модулей:

```
import mx.events.ResourceEvent;
import mx.resources.ResourceManager;

private var selectedLocale:String;

private function setAppLocale(locale:String):void
{
    this.selectedLocale = locale;
    if (resourceManager.getLocales().indexOf(locale) == -1)
    {
        var dispatcher:IEventDispatcher =
            resourceManager.loadResourceModule(locale +
            "_resources.swf");
        dispatcher.addEventListener(ResourceEvent.COMPLETE,
            onResourceLoaded);
    }
    else
    {
        onResourceLoaded(null);
    }
}

private function onResourceLoaded(e:ResourceEvent):void
{
    resourceManager.localeChain = [this.selectedLocale];
    views.selectedIndex = 1;

    contentBackground.setStyle("borderSkin",
        resourceManager.getClass('localizedContent', 'borderSkin'));
    contentBackground.invalidateDisplayList();
    contentBackground.validateNow();
}
```

Пользователю предлагается выбрать между американским английским и французским языком. Когда пользователь выбират язык, функция `setAppLocale` вызывается для загрузки необходимого ресурсного модуля. Метод сначала проверяет, не были ли ресурсы запрашиваемого локального контекста загружены ранее; для этого он анализирует результат вызова метода `getLocales` менеджера ресурсов. Подобные проверки избавляют от лишних затрат на запрос и загрузку уже имеющихся ре-

сурсов. Если запрашиваемый локальный контекст не был загружен ранее, приложение вызывает метод `loadResourceModule` для его получения и регистрирует слушателя события `complete`, чтобы узнать о завершении загрузки и готовности модуля к использованию.

При обработке события `complete` приложение задает свойство `localeChain`, чтобы использовать загруженный ресурсный модуль. Обратите внимание на вызовы трех методов объекта `contentBackground`. В Flex настройки стилей не могут использоваться в привязке, поэтому объекты, ссылающиеся на стилевые свойства из ресурсных модулей, должны обновляться на программном уровне для отслеживания изменений в стилевых свойствах.

Кроме URL-адреса ресурсного модуля, метод `loadResourceModule` может получать несколько дополнительных параметров. Если приложение загружает несколько ресурсных модулей, вероятно, их следует загружать с параметром `update=false` у всех модулей, кроме последнего. Это избавит вас от лишних затрат на периодическое выполнение функций обновления менеджера ресурсов.

22.5. Поддержка устройств IME

Задача

Требуется распространять приложение на языке с многобайтовой кодировкой символов (например, японском, корейском или китайском).

Решение

Используйте класс `Capabilities` для обнаружения редактора IME (Input Method Editor) и класс `IME` для управления его взаимодействием с приложением Flex.

Обсуждение

В азиатских языках (например, в китайском) слова представляются идеограммами, а не комбинациями букв, как в языках латинской группы. В последних количество символов относительно невелико, что позволяет легко разместить их на клавиатуре с небольшим количеством клавиш. Для азиатских языков такое решение не подходит: клавиатура должна состоять из тысяч клавиш. В них применяются специальные программы, называемые редакторами IME; такие программы позволяют вводить символы нажатием нескольких клавиш. Редакторы IME работают на уровне операционной системы, т. е. являются внешними по отношению к Flash Player.

Класс `Capabilities` содержит свойство `hasIME`, которое может использоваться для проверки наличия IME. При помощи объекта `flash.system.IME` можно узнать, доступен ли объект IME и какой режим преобразования

в нем выбран. В следующем примере приложение проверяет наличие редактора IME, и если он обнаружен, запускает его с выбором нужного режима преобразования:

```
private function detectIME():void
{
    if (Capabilities.hasIME == true)
    {
        output.text = "Your system has an IME installed.\n";
        if (flash.system.IME.enabled == true)
        {
            output.text +=
                "Your IME is enabled. and set to " +
                flash.system.IME.conversionMode;
        }
        else
        {
            output.text += "Your IME is disabled\n";
            try
            {
                flash.system.IME.enabled = true;
                flash.system.IME.conversionMode =
                    IMEConversionMode.JAPANESE_HIRAGANA;
                output.text +=
                "Your IME has been enabled successfully";
            }
            catch (e:Error)
            {
                output.text +=
                    "Your IME could not be enabled.\n"
            }
        }
    }
    else
        output.text = "You do not have an IME installed.\n";
}
```

Всегда используйте блок `try…catch` при изменении конфигурации IME. Если используемый редактор IME не поддерживает заданные параметры, при вызове происходит ошибка.

В некоторых случаях IME рекомендуется отключать, например для текстовых полей, заполняемых числовыми данными. Вы можете вызвать функцию отключения IME при получении фокуса таким компонентом, а затем снова включить IME после потери фокуса:

```
<mx:Script>
    <[[
        private function enableIME(enable:Boolean):void
        {
            if (Capabilities.hasIME)
            {
```

```
                try
                {
                    flash.system.IME.enabled = enable;
                    trace("IME " +
                        (enable ? "enable" : "disable"));
                }
                catch (e:Error)
                {
                    Alert.show("Could not "
                        (enable ? "enable" : "disable") + " IME");
                }
            }
        }
    ]]>
</mx:Script>
<mx:VBox horizontalCenter="0" verticalCenter="0" >
    <mx:TextInput id="numericInput"
    focusIn="enableIME(false)" focusOut="enableIME(true)" />
    <mx:TextInput id="textInput" />
</mx:VBox>
```

Чтобы узнать о завершении построения знака пользователем, прослушивайте события объекта `System.ime`:

```
System.ime.addEventListener(IMEEvent.IME_COMPOSITION, onComposition);
```

22.6. Обнаружение экранного диктора

Задача

Требуется обеспечить поддержку пользователей с ослабленным зрением и адаптировать приложение для применения экранного диктора.

Решение

Используйте статическое свойство `active` класса `Accessibility` для обнаружения экранного диктора.

Обсуждение

Широкие мультимедийные возможности и визуальный характер взаимодействия – отличительные особенности приложений **RIA (Rich Internet Application)**. К сожалению, эти же особенности затрудняют работу с приложениями **Flash для пользователей с ослабленным зрением. Поддержка экранных дикторов важна для таких пользователей; иногда она остается единственным способом взаимодействия с приложением. Если поддержка таких пользователей входит в число требований к приложению, вероятно, в схему взаимодействия придется внести некоторые изменения. При помощи свойства `active` класса `Accessibility` можно узнать об использовании экранного диктора. В следующем фрагменте из

примера ScreenReader.mxml свойство `Accessibility.active` используется для принятия решения о воспроизведении анимации:

```
private function showNextPage():void
{
    if (Accessibility.active == false)
    {
        page2.visible = true;
        pageChangeAnimation.play();
    } else {
        page1.visible = false;
        page2.alpha = 1;
    }
}
```

22.7. Определение порядка перебора

Задача

Требуется обеспечить поддержку пользователей, у которых возникают проблемы с использованием указательных устройств (мышей, сенсорных панелей и т. д.).

Решение

Определите в приложении порядок перебора компонентов, чтобы пользователь мог работать с приложением без использования указательного устройства.

Обсуждение

Порядок перебора (tab order) является важным аспектом удобства пользования приложением. Фактически он позволяет работать с приложением без частых переключений между клавиатурой и указательным устройством. Для пользователей с физическими недостатками, для которых работа с указательным устройством затруднена и невозможна, поскольку без перебора с клавиатуры они не смогут работать с приложением. Чтобы определить порядок перебора, задайте свойство `tabIndex` для каждого компонента в приложении. В примере TabOrder.mxml этот порядок определяется так, чтобы пользователь мог легко перебирать поля адресной формы без использования мыши:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" creationComplete="firstName.setFocus()">
    <mx:Canvas width="228" height="215" x="50" y="50"
        backgroundColor="#FFFFFF">
        <mx:Label x="10" y="10" text="First Name" tabIndex="1" />
        <mx:TextInput x="10" y="36" width="100"
            id="firstName" tabIndex="2"/>
        <mx:Label x="118" y="10" text="Last Name" tabIndex="3" />
```

```
        <mx:TextInput x="118" y="36" width="100"
            id="lastName" tabIndex="4"/>
        <mx:Label x="10" y="69" text="Address"
            tabIndex="5" />
        <mx:TextInput x="10" y="95" width="208"
            id="address" tabIndex="6"/>
        <mx:Label x="10" y="125" text="City"
            tabIndex="7"/>
        <mx:TextInput x="10" y="151" width="100"
            id="city" tabIndex="8"/>
        <mx:Label x="118" y="125" text="State"
            tabIndex="9"/>
        <mx:TextInput x="118" y="151" width="34"
            id="state" tabIndex="10"/>
        <mx:Label x="160" y="125" text="Zip"
            tabIndex="11"/>
        <mx:TextInput x="160" y="151" width="58"
            id="zip" tabIndex="12"/>
        <mx:Button x="153" y="181" label="Submit"
            id="submit" tabIndex="13"/>
    </mx:Canvas>
</mx:Application>
```

Обратите внимание: свойство `tabIndex` задается не только для текстовых полей и кнопок, но и для статических надписей, которые не могут получать фокус ввода. Для пользователей, пользующихся экранными дикторами, порядок перебора также определяет порядок описания элементов страницы. Задание свойства `tabIndex` для всех доступных компонентов (а не только для тех, которые способны получать фокус) поможет пользователям экранных дикторов. Все элементы, у которых свойство `tabIndex` не задано, размещаются в конце порядка перебора, а следовательно, порядок, в котором зачитываются их описания, отличается от порядка их визуального размещения; это затруднит работу пользователей с экранными дикторами.

22.8. Печать отдельных элементов приложения

Задача

Требуется создать печатные материалы в приложении.

Решение

Используйте классы пакета `mx.printing` для определения, форматирования и непосредственного вывода данных на печать.

Обсуждение

Пакет `mx.printing` содержит реализации несколько классов, используемых при печати. Например, класс `FlexPrintJob` определяет задание

печати, формирует его содержание и отправляет задание на принтер. Приложение BasicPrintJob.mxml создает задание печати, включает в него две страницы и отправляет на печать:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400"
height="300">
    <mx:Script>
        <![CDATA[
            import mx.printing.FlexPrintJob;

            public function print():void
            {
                var printJob:FlexPrintJob = new
                    FlexPrintJob();
                if (printJob.start())
                {
                    printJob.addObject(pageContainer1);
                    printJob.addObject(pageContainer2);
                    printJob.send();
                }
            }

        ]]>
    </mx:Script>
    <mx:VBox width="380" height="260" verticalCenter="-20"
        horizontalCenter="0">
        <mx:VBox id="pageContainer1">
            <mx:Label text="Page 1" />
            <mx:TextArea id="page1" width="100%" height="100%" />
        </mx:VBox>
        <mx:VBox id="pageContainer2">
            <mx:Label text="page 2" />
            <mx:TextArea id="page2" width="100%" height="100%" />
        </mx:VBox>
    </mx:VBox>
    <mx:Button bottom="5" right="10" label="Print"
        click="print();" />
```

При вызове метода `start` операционная система выводит диалоговое окно печати. Дальнейшее выполнение приостанавливается до тех пор, пока пользователь не завершит настройку задания печати. Если пользователь решит отменить печать, метод `start` возвращает `false`. В противном случае функция вызывает метод `addObject` для включения текстовой области в задание печати и передает задание принтеру вызовом метода `send`.

Каждый вызов `addObject` размещает элемент вместе со всеми дочерними элементами на новой странице. В печатном выводе, созданном в этом примере, метки страниц и текстовые поля, содержащиеся в `pageContainer1` и `pageContainer2`, передаются принтеру на разных страницах.

Метод `addObject` также получает необязательный параметр, в котором содержится информация о масштабировании добавленного элемента.

Если элемент слишком велик, задание печати может вывести его на нескольких страницах. По умолчанию элемент масштабируется по ширине страницы, но существует несколько других режимов масштабирования, определяемых в виде статических констант класса `FlexPrintJobScaleType`. Например, столбцовую диаграмму можно масштабировать так, чтобы она умещалась по вертикали на странице. Это позволит прочитать значения всех столбцов на одной странице:

```
Public function print():void
{
    if (printJob.start())
    {
        printJob.addObject(columnChart,
            FlexPrintJobScaleType.MATCH_HEIGHT);
        printJob.send();
    }
}
```

Если диаграмма из-за своей ширины не помещается на одной странице, излишки переходят на другую страницу. Пример ScaleExample.mxml демонстрирует эффект от применения различных режимов масштабирования.

22.9. Форматирование контента приложения для печати

Задача

Приложение должно выдавать выходные данные, специально отформатированные для печати.

Решение

Создайте пользовательский рендерер, который будет форматировать данные специально для печати.

Обсуждение

Часто печатные данные отличаются от тех, которые должны выводиться на экран для пользователя. Например, может возникнуть необходимость в создании печатных версий объектов приложения или построении отчетов с данными, не выводимыми в приложении. Задача решается при помощи *рендерера печати* – компонента, создаваемого специально для формирования печатных данных.

Предположим, в примере BasicPrintJob.mxml из рецепта 22.8 вы решили отказаться от печати рамки текстового поля или надписи. Кроме того, вводимый текст должен печататься так, как он создается в редакторе, с заполнением страницы по ширине без масштабирования и переходом на следующую страницу при заполнении текущей. Форматирование

блока текста для печати может осуществляться таким компонентом, как BasicTextRenderer.mxml:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
backgroundColor="0xffffff">
    <mx:String id="textToPrint" />
    <mx:Text width="100%" text="{textToPrint}" />
</mx:Canvas>
```

Если рендерер печати участвует в форматировании печатных выходных данных, включите его в список отображения, чтобы инфраструктура Flex могла сформировать раскладку визуальных аспектов компонента. Будьте внимательны при выборе позиции для добавления компонента: в некоторых ситуациях возможны непредвиденные последствия вроде сдвига раскладки или появления нежелательных полос прокрутки. В следующем фрагменте, взятом из примера BasicPrintRenderer.mxml, рендерер включается в список отображения родительского приложения для предотвращения появления полос прокрутки:

```
public function print():void
{
    var printJob:FlexPrintJob = new FlexPrintJob();
    if (printJob.start())
    {
        var printRenderer:BasicTextRenderer = new BasicTextRenderer();
        printRenderer.width = printJob.pageWidth;
        printRenderer.textToPrint = page1.text;
        printRenderer.visible = false;
        Application.application.addChild(printRenderer);
        printJob.addObject(printRenderer);
        printJob.send();
        Application.application.removeChild(printRenderer);
    }
}
```

Обратите внимание на свойство `pageWidth` **объекта задания печати. Парные свойства** `pageWidth` **и** `pageHeight` **задаются при возврате управления методом** `start`**. Если вы пишете компонент рендерера печати, используйте эти свойства при изменении размеров компонентов. В частности, они обеспечат нормальную работу рендерера в книжной и альбомной ориентации печати, для разных размеров листа и типов принтеров.**

22.10. Управление многостраничной печатью контента неизвестной длины

Задача

Требуется управлять печатным выво Специальные символы дом на нескольких страницах. Ни объем печатаемых данных, ни размеры компонентов заранее не известны.

Решение

Воспользуйтесь компонентом PrintDataGrid для управления многостраничным выводом табличных данных.

Обсуждение

При работе с табличными данными (например, отчетами в формате электронной таблицы) для форматирования многостраничного результата следует использовать компонент PrintDataGrid. В следующем фрагменте из примера MultipageDataGrid.mxml компонент PrintDataGrid используется для печати отчета:

```
public function print():void
{
    var printJob:FlexPrintJob = new FlexPrintJob();
    if (printJob.start())
    {
        var printGrid:PrintDataGrid = new PrintDataGrid();
        printGrid.width = printJob.pageWidth;
        printGrid.height = printJob.pageHeight;
        printGrid.columns = populationGrid.columns;
        printGrid.dataProvider = populationData.state;
        printGrid.visible = false;
        Application.application.addChild(printGrid);
        printJob.addObject(printGrid);
        while (printGrid.validNextPage)
        {

            printGrid.nextPage();
            printJob.addObject(printGrid);

        }
        printJob.send();
        Application.application.removeChild(printGrid);
    }
}
```

Размер компонента PrintDataGrid задается в соответствии с размером страницы. Включение компонента в задание печати добавляет первую страницу. Существование дополнительных страниц данных можно проверить при помощи свойства validNextPage, а переход к следующей странице осуществляется методом nextPage.

При разумном использовании компонент PrintDataGrid упрощает вывод многих видов данных, он не ограничивается печатью одного лишь табличного текста. В сочетании с рендерером элементов компонент PrintDataGrid может использоваться для печати диаграмм, графики или сложных компонентов. В примере GridSquares.mxml компонент PrintDataGrid в сочетании с рендерером элементов используется для печати красных квадратов:

```
public function print(itemSize:int, itemCount:int):void
{
    var printData:Array = new Array();
    for (var i:int = 0;  i < itemCount;  i++)
    {
        printData.push(itemSize);
    }

    var column:DataGridColumn = new DataGridColumn();
    column.headerText = "";
    column.itemRenderer =
        new ClassFactory(SquareRenderer);

    var printGrid:PrintDataGrid = new PrintDataGrid();
    printGrid.showHeaders = false;
    printGrid.visible = false;
    printGrid.setStyle("horizontalGridLines", false);
    printGrid.setStyle("verticalGridLines", false);
    printGrid.setStyle("borderStyle", "none");
    printGrid.columns = [column];
    printGrid.dataProvider = printData;
    Application.application.addChild(printGrid);

    var printJob:FlexPrintJob = new FlexPrintJob();
    if (printJob.start())
    {
        printGrid.width = printJob.pageWidth;
        printGrid.height = printJob.pageHeight;
        printJob.addObject(printGrid);
        while (printGrid.validNextPage)
        {
            printGrid.nextPage();
            printJob.addObject(printGrid);
        }
        printJob.send();
    }

    Application.application.removeChild(printGrid);
}
```

22.11. Печать колонтитулов

Задача

Требуется вывести верхние и нижние колонтитулы в печатных результатах.

Решение

Создайте рендерер печати для управления раскладкой страницы.

Обсуждение

Объединение рендерера печати с PrintDataGrid позволяет гораздо точнее управлять раскладкой печати, чем при использовании одного лишь компонента PrintDataGrid. Одна из типичных задач управления печатью – вывод колонтитулов на печатных страницах. Для этого приложение должно проверить, включены ли колонтитулы в выходные данные, и проверить результат при помощи свойства validNextPage компонента PrintDataGrid. В приложении HeaderFooterPrintRenderer.mxml определяется рендерер печати, который создает многостраничный вывод с включением колонтитулов там, где это уместно:

```
<?xml version="1.0"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
        backgroundColor="#ffffff" horizontalAlign="center">

    <mx:Script>
        <![CDATA[

            public function startJob():void
            {
                // Пытаемся вывести на одной странице
                header.visible = true;
                header.includeInLayout = true;
                footer.visible = true;
                footer.includeInLayout = true;

                this.validateNow();

                if (printGrid.validNextPage)
                {
                    // Таблица не помещается
                    // на одной странице
                    footer.visible = false;
                    footer.visible = false;
                    this.validateNow();
                }
            }

            public function nextPage():Boolean
            {
                header.visible = false;
                header.includeInLayout = false;

                printGrid.nextPage();

                footer.visible = !printGrid.validNextPage;
                footer.includeInLayout = !printGrid.validNextPage;

                this.validateNow();
                return printGrid.validNextPage;
```

```
            }
        ]]>
    </mx:Script>
    <mx:DateFormatter id="formatter" formatString="M/D/YYYY" />
    <mx:Canvas id="header" height="80" width="100%">
        <mx:Label   text="Population by State"
                    fontSize="24"
                    color="0x666666"
                    horizontalCenter="0"
                    verticalCenter="0"
                    width="100%"
                    textAlign="center" />
    </mx:Canvas>
    <mx:VBox height="100%" width="80%">
        <mx:PrintDataGrid id="printGrid" width="100%"
            height="100%">
            <mx:columns>
                <mx:DataGridColumn     dataField="@name"
                    headerText="State" />
                <mx:DataGridColumn
                    dataField="@population"
                    headerText="Population"/>
            </mx:columns>
        </mx:PrintDataGrid>
    </mx:VBox>
    <mx:DateFormatter id="format" formatString="m/d/yyyy" />
    <mx:Canvas id="footer" height="80" width="100%">
        <mx:Label    text="{formatter.format(new Date())}"
                left="20" bottom="5" />
    </mx:Canvas>
</mx:VBox>
```

Компонент определяет верхний колонтитул с названием отчета и нижний колонтитул с датой печати. Метод `startJob` инициирует построение раскладки первой страницы. Сначала он пытается скомпоновать страницу так, словно все данные помещаются на одной странице. После вызова метода `validateNow` он проверяет, помещаются ли данные отчета на одной странице, при помощи свойства `validNextPage` компонента `PrintDataGrid`. Если свойство равно `false`, значит, отчет поместился на одной странице, а если нет – нижний колонтитул скрывается, и макет формируется заново. На этой стадии первая страница готова к включению в задание печати независимо от того, состоит ли отчет из одной или нескольких страниц. Если отчет выводится на нескольких страницах, метод `nextPage` готовит следующую страницу к выводу. Он скрывает верхний колонтитул (который выводится только на первой странице) и отображает нижний колонтитул там, где это уместно.

Выделение логики формирования страниц в рендерер значительно упрощает логику печати. Следующий фрагмент из примера HeaderFooter.mxml демонстрирует практическое применение рендерера печати в приложении:

```
public function print():void
{
    var printJob:FlexPrintJob = new FlexPrintJob();
    if (printJob.start())
    {
        var printRenderer:HeaderFooterPrintRenderer =
            new HeaderFooterPrintRenderer();
        printRenderer.visible = false;
        this.addChild(printRenderer);
        printRenderer.width = printJob.pageWidth;
        printRenderer.height = printJob.pageHeight;
        printRenderer. .dataProvider = populationData.state;
        printRenderer.startJob()

        do
        {
            printJob.addObject(printRenderer);
        }
        while (printRenderer.nextPage());

        // Отправка последней страницы
        printJob.addObject(printRenderer);
        printJob.send();

        this.removeChild(printRenderer);
    }
}
```

Метод `print` начинает задание печати и настраивает рендерер по аналогии с предыдущими примерами. Эта часть кода завершается вызовом метода `startJob` рендерера печати, который формирует первую страницу. В следующей секции метода блок `do…while` включает страницы в задание до тех пор, пока метод `nextPage` не вернет `false` (признак последней страницы). Но поскольку блок `do…while` завершается вызовом `nextPage`, при завершении цикла последняя страница еще не была включена в задание печати. Функция ставит в очередь последнюю страницу отдельной командой, отправляет задание печати и исключает рендерер из списка отображения.

Алфавитный указатель

Специальные символы

A

B

C

D

E

F

G

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

A

Б

В

Г

Д

З

И

К

P

C