Секреты программирования во Flash MX

2-е издание

Предисловие Гэри Гроссмана, создателя ActionScript

ActionScript для Flash MX

Подробное руководство

Символ®

O'REILLY®

Колин Мук

ActionScript for Flash MX

The Definitive Guide

Second Edition

Colin Moock

O'REILLY®

ActionScript для Flash MX

Подробное руководство

Второе издание

Колин Мук

Санкт-Петербург — Москва
2004

Колин Мук

ActionScript для Flash MX Подробное руководство, 2-е издание

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*М. Антипин*
Редактор	*В. Овчинников*
Корректор	*С. Беляева*
Верстка	*Н. Гриценко*

Мук К.
ActionScript для Flash MX. Подробное руководство. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 1120 с., ил.
ISBN 5-93286-068-5

Второе издание бестселлера «ActionScript для Flash MX. Подробное руководство» – исчерпывающий и незаменимый ресурс по ActionScript, настольная книга дизайнеров и Flash-программистов, пишущих серьезные приложения. Из-под пера Мука вышел полностью переработанный справочник, не только подробно описывающий бесчисленные нововведения Flash MX, но и раскрывающий в ясной и доступной манере глубинные изменения, произошедшие на методологическом уровне (особенно касающиеся компонентов и событий).

Издание уникально – это самая точная и хорошо организованная книга по программированию во Flash MX. Объем справочника увеличился почти вдвое и включает описание более 250 новых классов, объектов, методов и свойств. Мук добавил в книгу сотни новых примеров кода, иллюстрирующих практическое применение новых технологий Flash MX: отрисовку фигур на этапе исполнения, преобразование массивов в экранные таблицы, создание компонентов многократного использования, работу с XML и звуком. Помимо стандартных методик описываются лучшие приемы программирования и скрытые возможности, особое внимание уделяется объектно-ориентированному программированию и новой модели событий Flash MX. Книгу оценят как новички, так и специалисты.

ISBN 5-93286-068-5
ISBN 0-596-00396-X (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 25.06.2004. Формат 70×100 1/16 . Печать офсетная.
Объем 70 печ. л. Тираж 3000 экз. Заказ № 782
Отпечатано с готовых диапозитивов в ОАО «Техническая книга»
190005, Санкт-Петербург, Измайловский пр., 29.

Оглавление

Отзывы на 2-е издание книги «ActionScript для Flash MX. Подробное руководство»

Подобно своему предшественнику, 2-е издание книги «ActionScript for Flash MX: The Definitive Guide» (ASDG2) – настольная книга тех, кто пишет на ActionScript. Из-под пера Мука вышел полностью переработанный справочник, не только подробно описывающий бесчисленные нововведения Flash MX, но и раскрывающий в ясной и доступной манере глубинные изменения, произошедшие на методологическом уровне (особенно касающиеся компонентов и событий).

Роберт Пеннер, автор «Robert Penner's Programming Macromedia Flash MX»

Мне нравится эта книга. Это доступное и в то же время весьма глубокое исследование ActionScript. Если вы работаете с Flash MX как дизайнер или программист, вы должны иметь эту книгу.

Хилман Кертис, основатель hillmancurtis, Inc. (http://www.hillmancurtis.com), автор «Flash Web Design» и «MTIV: Process, Inspiration and Practice for the New Media Designer»

Об этой книге невозможно говорить, не прибегая к превосходным степеням. Второе издание «ActionScript for Flash MX: The Definitive Guide» Мука – одна из самых необходимых книг по программированию в среде Flash, какую можно купить. В этом исчерпывающем и основополагающем справочнике одинаковое внимание уделяется стандартным методикам, лучшим приемам и скрытым возможностям. Прочитав его, любой, кто пишет на ActioScript, узнает об этом языке что-то новое.

Найджел Пег, разработчик компонентов Flash UI, Macromedia

Как разработчик, преподаватель и коллега по перу я не могу нахвалиться на ASDG2. Эта книга просто незаменима для всех, кто работает с Macromedia Flash. Колин прекрасно владеет ActionScript и умеет изложить свои знания понятным, кратким и зачастую остроумным способом.

Брэнден Дж. Холл, основатель и администратор Flashcoders List, автор «Object-oriented Programming with ActionScript»

Колин еще раз написал незаменимый справочник Flash-разработчика. Всесторонний охват ActionScript делает этот справочник самым полным информационным ресурсом из всех доступных. Я опять выбрал эту книгу.

Дейв Янг, главный исполнительный директор (CEO) и разработчик, Quantumwave Interactive Inc.

Первое издание этой книги содержало информацию, которую просто нельзя было найти где-либо в другом месте. То же самое и в еще большей мере справедливо в отношении второго издания, посвященного более сложной системе Flash MX. В очередной раз Колин Мук и O'Reilly создали отлично написанный и организованный, тщательно выверенный источник информации для программистов Flash MX всех уровней. Кем бы вы ни были – проектировщиком, программистом или художником, эта книга – ваш бесценный помощник в разработке приложений и в художественном самовыражении средствами Flash MX.

Амит Питару, художник и технолог (http://www.pitaru.com, http://www.insert-silence.com), преподаватель Pratt Institute & NYU (http://itp.nyu.edu), соавтор «New Masters of Flash: The 2002 Annual»

Эта книга – исчерпывающий ресурс по ActionScript, созданный человеком, который, собственно, и стоял у его истоков.

Тодд Пургасон, креативный директор и соучредитель Juxt Interactive (http://www.juxtinteractive.com), автор «Flash Deconstruction»

ASDG2 – бесспорно, основополагающая книга по программированию на ActionScript во Flash MX. Она оказалась неоценимой даже для инженеров Macromedia Flash, которые рассматривают ее как дополнение к нашей собственной промышленной документации. Дотошное внимание, которое Мук уделяет деталям, всюду просматривается в этой прекрасной книге и в сочетании с легко воспринимаемым стилем обучения гарантирует, что книгу оценят как новички, так и специалисты.

Гэри Гроссман, создатель ActionScript, Macromedia

ASDG2 – великолепный по лаконичности и непосредственности источник информации. Он отличается полнотой и доскональностью, не обходя, по сути, молчанием ни одного аспекта Flash. Это единственная книга по Flash, с которой я работаю, и не будь ее у меня, я не раз оказался бы в затруднительном положении. В умении Колина подать информацию или идею есть что-то такое, благодаря чему даже сугубо технические области Flash для меня совершенно прозрачны.

Джеймс Паттерсон, художник (http://www.presstube.com), соавтор «New Masters of Flash»

ASDG2 – это единственный источник, к которому следует обращаться, чтобы узнать обо всех возможностях, предоставляемых Flash MX. Даже Macromedia не вникает в подробности так глубоко, как это делает Колин Мук. Если вам надо освоить ActionScript за неделю или найти решение какой-то задачи за час, то ASDG2 – единственный справочник, который позволит вам это сделать. Самое замечательное, что Мук – изумительный и увлеченный писатель. Я не встречал другой книги по программированию, которую было бы столь же приятно читать.

Джош Улм, основатель ioResearch Studios (http://www.ioresearch.com) и The Remedi Project (http://www.theremediproject.com)

Книга Мука – из тех, которые постоянно находятся на рабочем столе. В любых случаях – когда надо найти решение в условиях поджимающих сроков выполнения проекта или углубить свои навыки программирования, эта книга оказывается бесценным руководством, показывающим все входы и выходы ActionScript.

Гленн Томас, директор и основатель Smashing Ideas, Inc. (http://www.smashingideas.com), автор «Flash Studio Secrets», соавтор «Flash Enabled»

ASDG2 – это действительно самая точная и хорошо организованная книга по программированию во Flash из всех напечатанных. Очевидно, что Колин Мук увлечен предметом, и этим обусловлена удача книги.

Джаред Тарбелл, специалист по вычислительным технологиям (http://www.levitated.net), соавтор «Flash Math Creativity» и «Fresh Flash»

На рынке немало книг по Flash (в том числе и написанных мной самим!), но я не устаю рекомендовать только одну из них – ASDG2 Колина Мука. Используемая самостоятельно или в сочетании с любой другой книгой по Flash, она представляет собой поистине Святой Грааль новых идей для тех, кто профессионально работает с Flash, и помогает нам не совершать синтаксических ошибок.

Джошуа Дэйвис, художник и технолог (http://www.praystation.com), автор «Flash to the Core»

Предисловие

Прошло всего восемнадцать месяцев с того момента, когда я написал предисловие к первому изданию «ActionScript: подробное руководство». За это время первое издание зарекомендовало себя как необходимое руководство по программированию на ActionScript. Оно стало необходимо столь многим разработчикам, что кажется, будто оно существует гораздо дольше.

Flash MX, выпущенный в марте 2002 года, стал самым амбициозным к настоящему моменту выпуском Flash. Команда талантливых личностей, участвовавших в его создании, была более многочисленна, чем когда-либо прежде, и мы добавили в продукт свыше 100 существенных новых функций. ActionScript стал средоточием главных усилий, что потребовало изменений в способе разработки. До Flash MX разработка ActionScript осуществлялась небольшой группой лиц, в которую входил и я. В MX наши грандиозные планы в отношении ActionScript потребовали большого количества разработчиков. Располагая дополнительными ресурсами, мы смогли значительно усовершенствовать редактор сценариев и отладчик, оптимизировать производительность и добавить массу новых API, предоставивших новые возможности программирующим на ActionScript.

Сегодня Macromedia проявляет к Flash повышенный интерес. Принято считать Flash просто средством создания анимации, но в сообществе Flash-разработчиков растет убеждение, что Flash представляет собой нечто большее. Благодаря Flash MX у веб-разработчиков появилась возможность предоставлять через Сеть насыщенную интерактивную среду для пользователя – не только традиционные для Flash мультфильмы и динамическую графику, но и сложные веб-приложения.

Flash всегда был и, видимо, останется лучшим средством для придания вебсайту некой веселости, а разработчики серьезных веб-приложений стремятся преодолеть ограничения HTML. Они ищут новую платформу, которая предлагала бы пользователям более заманчивые, привлекательные и практичные ощущения – *обогащенного клиента*, – и обнаруживают, что Flash идеально подходит для доставки такого содержимого. Единообразная работа Flash на разных платформах и повсеместное распространение этого продукта предлагают технологию этапа исполнения, на которой разработчики могут создавать веб-приложения нового рода, более интересные и живые по сравнению с существовавшими до сих пор. Бьюсь об заклад, что мы увидим широкий спектр новых применений Flash, от многопользовательских игр до электронной торговли и визуализации данных. И в Macromedia стремятся обеспечить соответствие Flash новым требованиям, которые предъявляют к нему разработчики приложений. ActionScript играет важную роль в этом новом представлении о Flash MX. Поскольку полезность платформы Flash зависит

от мощи его языка сценариев, мы постарались сделать ActionScript достаточно мощным, чтобы удовлетворить самых требовательных веб-разработчиков.

Стремление сделать Flash подлинной платформой приложений заставило разработчиков Flash MX решать особые задачи. Flash можно считать продуктом, развитие которого стимулируется сразу в нескольких направлениях, поскольку он должен удовлетворять требованиям различных клиентов, от создателей анимированных персонажей до тех, кто занимается динамической графикой (motion graphics) и разработкой мощных приложений. Усовершенствование механизма сценариев рассматривалось как решающая задача, но мы сознавали, что столь же важно усилить средства Flash для творческого выражения, поскольку изобразительное искусство – душа и сердце Flash.

С целью гарантировать удовлетворение разнообразных потребностей наших клиентов, мы разделили всю команду разработчиков Flash на три группы, каждой из которых была назначена своя цель:

Доступность

Предоставить отличные начальные условия новичкам

Созидательность

Усилить средства Flash для творческого выражения

Мощь

Расширить мощь ActionScript как инструмента для разработки сложных приложений

К моему удовольствию, я возглавил последнюю группу, которая занялась усовершенствованием ActionScript для поддержки представления о Flash как о платформе. Мы пересмотрели и улучшили модели объектов и событий Flash; мы развили «умные» клипы Flash 5 в более надежную архитектуру компонентов и переписали наиболее употребительные объекты ActionScript с целью оптимизации эффективности их работы. Кроме того, мы добавили средства облегчения труда разработчиков, введя подсказчик кода (Code Hints) и переработав отладчик.

Однако над ActionScript трудились не мы одни. Объединение Macromedia с Allaire в 2001 году принесло в компанию солидный опыт работы с серверами. Сотрудники из нового офиса Macromedia в Ньютоне, штат Массачусетс, создали Macromedia Flash Remoting MX (Flash Remoting), новую серверную технологию, позволяющую осуществлять прямую и легкую связь с сервером. Бригада блестящих разработчиков Macromedia Flash Communication Server MX (Comm Server) показала, что можно сделать с ActionScript, введя новые API ActionScript (в том числе ServerSide ActionScript), которые обеспечивают великолепные возможности: живую двустороннюю связь и сотрудничество через Интернет!

Еще одна группа была выделена для создания компонентов. Эта группа, два человека из состава которой были научными редакторами данной книги, разработала компоненты UI, позволяющие быстро создавать HTML-подобные формы, и дополнительные элементы управления, которые выводят за

рамки того, что можно сделать в HTML, например полнофункциональный элемент дерева, календарь, сетку данных. В сочетании с Flash Remoting эти компоненты образуют мощные средства создания приложений, управляемых данными.

Компоненты Flash MX дают очень хорошее представление о том, что ожидает нас в будущем: абстракции высокого уровня, из которых можно быстро собирать интерактивный контент и приложения. Мы в Macromedia будем стремиться к тому, чтобы сделать создание и применение компонентов в будущих версиях Flash еще более простым и эффективным. Компоненты, поставляемые с Comm Server, дают отличный пример такой мощи. Даже без компонентов, используя Comm Server, довольно просто создать приложение для видеоконференций с помощью всего нескольких строк ActionScript. Компоненты Comm Server делают решение этой задачи еще более простым: с помощью обычного перетаскивания нескольких компонентов новичок может фактически создать сценарий, не пользуясь ActionScript. Мы заинтересованы в развитии этого направления, поскольку оно позволяет новичкам сразу же создавать полезные продукты. Будьте уверены, что по мере облегчения доступности ActionScript и Flash большие возможности станут открываться и перед опытными разработчиками. Взяв на себя черновую работу и разработку употребительных компонентов интерфейса пользователя, мы даем возможность опытным пользователям и программистам еще более повысить свою продуктивность. Усовершенствованная объектная модель и архитектура компонентов Flash MX позволяют квалифицированным разработчикам расширять существующие компоненты или разрабатывать собственные пользовательские библиотеки. Поэтому, хотя данная книга не описывает подробно имеющиеся компоненты, она предлагает продвинутым и честолюбивым разработчикам средства для создания собственных компонентов. Всегда радует, когда разработчики открывают новые направления применения ActionScript, получив инструменты и разобравшись с тем, как их использовать.

Таким образом, второе издание бесспорно является необходимой книгой для программирования на ActionScript во Flash MX. Она оказалась незаменимой даже для разработчиков команды Macromedia Flash, которые считают ее дополняющей нашу собственную документацию. Эта книга стала результатом безграничного таланта и энергии Колина Мука, которые заставили его глубоко вникнуть в ActionScript, раскрывая для читателя внутренние тайны этого языка. Дотошное внимание Колина к деталям, явно просматриваемое на протяжении всей этой замечательной книги, в сочетании с неторопливым педагогическим стилем, гарантируют, что книга будет оценена как новичками, так и специалистами. Получите удовольствие от книги и от ActionScript во Flash MX!

Гэри Гроссман,
создатель ActionScript,
Senior Engineering Manger,
Macromedia Flash Team,
октябрь 2002

Введение

Добро пожаловать во второе издание «ActionScript for Flash MX: The Definitive Guide»! По сравнению с первым оно сильно изменилось: появились сотни новых страниц, а старый материал значительно переработан, что привело его в соответствие с лучшей современной практикой работы во Flash MX. Надеюсь, вы с таким же удовольствием прочтете эту книгу, с каким я ее писал!

Как и первое издание, эта книга обучает читателя языку ActionScript с самых основ, рассказывая о фундаментальных понятиях и более сложных случаях применения языка, но особое внимание в ней уделяется технологии Macromedia Flash MX. В части I мы изучим основы ActionScript, начав с переменных и управления клипами, и дойдем до более сложных тем, таких как объекты, классы и связь с сервером. В части II «Справочник по языку» мы расскажем обо всех объектах, классах, свойствах, методах и обработчиках событий базового языка ActionScript. Сюда вы будете заглядывать регулярно, узнавая новое и вспоминая то, что часто забывается, поэтому книга должна лежать на вашем рабочем столе, а не на полке!

Во Flash MX язык ActionScript стал сложнее, но для чтения книги не обязательно быть программистом. В этом издании я по-прежнему старался проявлять заботу о новичках. Изложение достаточно насыщенно, но для освоения материала не нужна предварительная подготовка в программировании. Достаточно иметь опыт работы с Flash без использования ActionScript и желание учиться. Конечно, лучше иметь опыт программирования – тогда вы сразу сможете применить свои навыки к ActionScript. Чтобы облегчить переход на Flash опытным программистам, я намеренно приводил полезные аналогии с такими языками, как JavaScript, Java и C.

Кроме того, эта книга – действительно «подробное руководство» по ActionScript для Flash MX. Это результат почти четырех лет исследований, тысяч электронных писем сотрудникам Macromedia и откликов, полученных от пользователей всех уровней подготовки. Надеюсь, что книга явственно демонстрирует мой горячий интерес к предмету и приобретенный нелегким трудом практический опыт, которым вы можете непосредственно воспользоваться. Она представляет собой источник исчерпывающей и – благодаря рецензированию, выполненному Гэри Гроссманом (Gary Grossman), создателем ActionScript, – исключительно точной информации по ActionScript.

Краткий обзор отличий второго издания

Тем, кто знаком с первым изданием этой книги и кому не терпится узнать, что же содержится во втором, я могу указать на несколько отличительных

особенностей последнего. Однако не ограничивайтесь приводимым списком. Читайте книгу, и вы найдете много других важных изменений.

Следующие главы части I «Основы ActionScript» были существенно переработаны. В них рассказывается о некоторых наиболее примечательных нововведениях, таких как компоненты и важные изменения в способах обработки событий и действий с объектами.

- Глава 9 «Функции»
- Глава 10 «События и обработка событий»
- Глава 12 «Объекты и классы»
- Глава 14 «Подклассы класса MovieClip и компоненты»

Обратите также внимание на переработанные и новые приложения, в особенности следующие:

- Приложение C «Обратная совместимость и обновление версий проигрывателя»
- Приложение E «Поддержка HTML в текстовых полях»
- Приложение F «Поддержка GET и POST»
- Приложение G «Обзор компонентов Flash UI»
- Приложение H «Встраивание фильмов Flash в веб-страницы»

Нижеперечисленные статьи в части II «Справочник по языку» или переписаны заново, или весьма существенно переработаны по сравнению с первым изданием. Например, вам надо будет внимательно прочесть статью о *SharedObject* и изучить методы Drawing API, добавленные в класс *MovieClip*.

- *Accessibility*, объект
- *Button*, класс
- *Capabilities*, объект
- *Function*, класс
- `_global`, объект
- `#initclip` и `#endinitclip`, прагмы
- *LoadVars*, класс
- *LocalConnection*, класс
- *MovieClip*, класс (новые события и Drawing API)
- *Object*, класс
- *setInterval( )* и *clearInterval( )*, глобальные функции
- *SharedObject*, объект
- *Sound*, класс
- *Stage*, объект
- *System*, объект
- *TextField*, класс
- *TextFormat*, класс

- События приемника для *Key*, *Mouse*, *TextField* и *Stage* (см. табл. B.1)

Что нового в ActionScript для Flash MX

ActionScript чрезвычайно эволюционировал с переходом от Flash 5 к Flash MX (так называется новая среда разработки) и соответствующему Flash Player 6, и эта книга претерпела изменения вместе с ним. Подробные сведения о названиях версий Flash можно найти в табл. B.2.

Действие многих новых функций можно увидеть на сайте:
http://www.moock.org/webdesign/lectures/newInMX

В табл. B.1 представлен общий обзор основных нововведений в ActionScript. В ней указывается, где можно найти дополнительную информацию по каждой новой теме данной книги. Если не оговорено иное, перекрестные ссылки указывают на часть II «Справочник по языку».

Таблица B.1. Новые функции во Flash MX ActionScript

Функция	Подробное описание см. в
Drawing API: рисование фигур и заливок на этапе исполнения при помощи новых методов *MovieClip*	*MovieClip.beginFill( ), MovieClip.beginGradientFill( ), MovieClip.clear( ), MovieClip.curveTo( ), MovieClip.endFill( ), MovieClip.lineStyle( ), MovieClip.lineTo( ), MovieClip.moveTo( )*; «Рисование в клипе на этапе выполнения» в главе 13
Загрузка изображений в формате JPEG на этапе исполнения	*MovieClip.loadMovie( ), loadMovie( )*
Загрузка звука в формате MP3 на этапе исполнения	*Sound.loadSound( )*
Получение длительности звукового фрагмента и текущей продолжительности его исполнения	*Sound*.`position`, *Sound*.`duration`
Событие завершения исполнения звукового фрагмента	*Sound.onSoundComplete( )*
Создание, обработка и форматирование текстовых полей на этапе исполнения	Класс *TextField*, класс *TextFormat*, *MovieClip.createTextField( )*
Установка и снятие маски клипа на этапе исполнения	*MovieClip.setMask( )*
Создание клипов с нуля на этапе исполнения	*MovieClip.createEmptyMovieClip( )*
Определение глубины клипа на этапе исполнения	*MovieClip.getDepth( )*

Таблица В.1 (продолжение)

Функция	Подробное описание см. в
Периодическое выполнение функции или метода	*setInterval( )*, *clearInterval( )*
Ускорение обработки XML, строк и массивов данных благодаря повышению производительности	Класс *XML*, класс *String*, класс *Array*
Локальное хранение данных (аналогично cookies в JavaScript)	Объект *SharedObject*
Создание упакованных модулей кода с помощью подклассов MovieClip и компонентов	`#initclip`, `#endinitclip`, *Object.registerClass( )*, *attachMovie( )*; глава 14
Обмен данными между двумя Flash-проигрывателями на одном и том же компьютере	Класс *LocalConnection*
Объявление глобальных переменных	`_global`; «Переменные клипов и глобальные переменные» в главе 2
Использование международных символов в кодировке Unicode	«Строковый тип» в главе 4, приложение С
Определение обработчиков событий клипов с помощью функций обратного вызова	Глава 10
Приемники событий, генерируемых любыми объектами	Глава 10 и *Key.addListener( ), Mouse. AddListener( ), Stage.addListener( ), Selection. AddListener( ), TextField.addListener( )*
Задание режима кнопок для клипов	«Использование клипов в качестве кнопок» в главе 13
Объекты кнопок управления на этапе исполнения	Класс *Button*
Средства обеспечения доступности содержимого людям с нарушениями зрения	Объект *Accessibility*
Получение ширины и высоты фильма на этапе исполнения и перемещение элементов при изменении размеров фильма	*Stage.height*, *Stage.width*, *Stage.onResize( )*
Использование лексической или вложенной области видимости функции либо выполнение функции как метода произвольного объекта	*Function.call( ), Function.apply( )*; «Цепочка областей видимости» в главе 2; «Область видимости функций» в главе 9

Функция	Подробное описание см. в
Получение информации о проигрывателе и системе (о разрешении экрана, типе операционной системы и об установленном языке)	Объект *Capabilities*
Перехват событий от клавиатуры и мыши через централизованный API ввода	Объект *Key*, объект *Mouse*
Загрузка переменных с помощью класса LoadVars вместо функции *loadVariables( )*	Класс *LoadVars*
Контроль хода загрузки XML или переменных	*XML.getBytesLoaded( ), LoadVars.getBytesLoaded( )*
Управление порядком обхода кнопок, текстовых полей и клипов	*TextField*.tabIndex, *Button*.tabIndex, *MovieClip*.tabIndex
Отключение изменения курсора для кнопок	*Button*.useHandCursor, *MovieClip*.useHandCursor
Добавление свойств доступа к объекту и получение уведомления при изменении свойства	*Object.addProperty( ), Object.watch( )*

Что нового во втором издании

Второе издание «ActionScript for Flash MX: The Definitive Guide» – не просто дополнение к первому (называвшемуся «ActionScript: The Definitive Guide»). Текст полностью переработан, а изменение структуры руководства делает более заметными новые функции, появившиеся во Flash MX ActionScript. Практически каждый абзац подвергся редактированию, добавлено 400 страниц, посвященных новым возможностям ActionScript. Описания синтаксиса старого ActionScript для Flash 4 перемещены из основной части книги в приложение C и статьи в Интернете. Мы приняли такое решение, чтобы сделать книгу тоньше, но она все равно оказалась заметно более увесистой, чем первое издание. Когда эта книга дойдет до читателя, Flash Player 6 будет распространен почти повсеместно, поэтому больше нет смысла подробно рассматривать Flash 4, о котором здесь рассказано достаточно для того, чтобы помочь понять и модифицировать старый код, с которым вы можете столкнуться. Кроме того, мы тщательно рассмотрели различия между версиями Flash 5 и 6, что поможет вам разобраться в новых принципах и системах понятий и модифицировать имеющийся код. Примеры старого кода из первого издания по-прежнему доступны на *http://www.moock.org/asdg/codedepot*.

Обновленные примеры кода

Все примеры кода из первого издания были переработаны с учетом изменений синтаксиса ActionScript во Flash MX и лучших приемов работы. Например:

- В примерах с вопросником для обработчиков событий кнопок теперь используются функции обратного вызова вместо обработчиков *on( )* в стиле Flash 5.
- Текстовые поля, ранее создававшиеся в среде разработчика, теперь генерируются программно с помощью *createTextField( )*.
- Классы определяются в `_global` (новое свойство, в котором содержатся глобальные переменные).
- Вместо прежней глобальной функции *loadVariables( )* используется класс *LoadVars*.

Добавлены десятки новых примеров, специфических для Flash MX. Вот наиболее интересные из них:

- Полностью основанный на коде объектно-ориентированный вопросник; его можно загрузить из сетевого хранилища кода (Code Depot), о котором рассказывается ниже.
- Настраиваемая бегущая строка текста (text ticker) (см. *TextField*.`hscroll`).
- Преобразователь массивов в таблицы (см. *TextFormat*.`tabStops`).
- Загрузчик звука (см. *Sound.getBytesLoaded( )*).

Сотни усовершенствований

Содержимое первого издания улучшено массой небольших дополнений. Вот лишь некоторые из сотен улучшений:

- В описании свойства *MovieClip*.`_x` рассказано о *twips* (минимальном расстоянии, на которое можно переместить клип).
- В описание *MovieClip*.`_visible` добавлено предупреждение о том, что события кнопок не генерируются, когда `_visible` имеет значение `false`.
- В статье «Справочника про языку», посвященной методу *XML.parseXML( )*, подробно рассказано о секции CDATA и о предопределенных сущностях XML (`&`, `<`, `>`, `"` и `'`).
- В статье «Справочника по языку», посвященной методу *MovieClip.getBytesLoaded( )*, приведен список возможных значений, возвращаемых при асинхронном выполнении *loadMovie( )*.
- В главе 2 обсуждаются квалифицированные и неквалифицированные ссылки на переменные и венгерскую нотацию (*Hungarian notation*).
- В главе 4 подробно сравнивается `null` с `undefined` при описании операции *delete*.

Конечно, есть и более существенные изменения. Расскажем о них.

Главные изменения по сравнению с первым изданием

Ниже перечислены основные изменения в содержании и структуре второго издания. Обратите внимание, что некоторые из этих глав в первом издании располагались в части II «Применение ActionScript». Остальной материал его второй части теперь распределен по различным разделам второго издания, а часть сведений перемещена в файлы онлайн-справки. Несмотря на такие структурные изменения, не сомневайтесь, что во втором издании присутствуют десятки прикладных примеров, щедро разбросанных по всей книге. «Справочник по языку», бывший в первом издании частью III, стал теперь частью II.

Глава 1 «Легкое введение для непрограммистов»

- Добавлено введение в объектно-ориентированное программирование.
- Учебный пример вопросника переработан для Flash MX.
- Раздел по обработке событий переработан для Flash MX.

Глава 2 «Переменные»

- Добавлены рекомендации по суффиксам имен переменных.
- Добавлено описание глобальных переменных.
- Добавлен раздел по загрузке внешних переменных.
- Добавлено подробное обсуждение цепочки областей видимости.

Глава 3 «Данные и типы данных»

- Добавлен раздел «Копирование, сравнение и передача данных» (ранее находившийся в главе 15).

Глава 4 «Элементарные типы данных»

- Добавлено описание Unicode.

Глава 5 «Операции»

- Добавлено описание операторов строгого равенства и *instanceof*.

Глава 6 «Инструкции»

- Добавлено описание предложения *switch*.
- Пересмотрено описание *with*, в которое включены цепочки областей видимости.
- Удален старый оператор *call* (теперь о нем говорится только в «Справочнике по языку»).

Глава 8 «Инструкции цикла»

- Добавлен раздел о *setInterval( )*, позволяющей периодически выполнять код.
- Переработан раздел «Циклы временной диаграммы и событий клипа» с целью описания возможностей Flash MX (*MovieClip.createEmptyMovieClip( )* и обработчика *MovieClip.onEnterFrame( )*).

Глава 9 «Функции»

- Добавлен раздел о различиях между литералами функций и предложением *function*.
- Добавлено описание вложенных функций.
- Пересмотрен раздел «Область видимости функций», в который включено более подробное описание лексической области видимости.
- Учебный пример вопросника переработан для Flash MX.

Глава 10 «События и обработка событий»

- Добавлено полное описание свойств обработчиков событий.
- Добавлено описание приемников событий (event listeners), появившихся во Flash MX.
- Добавлено углубленное обсуждение областей видимости, включая табл. 10.1, сравнивающую старые и новые правила видимости.
- Добавлено описание ключевого слова `this` в различных обработчиках, включающее общую сводку в табл. 10.2.
- Все конкретные описания событий кнопок и клипов перемещены в «Справочник по языку» (см. также табл. 10.3).

Глава 11 «Массивы»

- Добавлено описание метода *Array.sortOn( )*.
- Учебный пример вопросника переработан для Flash MX.

Глава 12 «Объекты и классы»

- Глава полностью переработана и уделяет больше внимания процедуре создания класса с методами и свойствами.
- Добавлено описание ключевого слова Flash MX *super*, с помощью которого вызывается конструктор надкласса (superclass) и его методы.
- Добавлено формальное описание *цепочки прототипов*.
- Добавлено формальное описание вопросов, связанных с *организацией наследования классов*.
- Добавлен раздел по статическим методам и свойствам.
- Добавлено описание показа объекта на экране.
- Добавлен шаблон приложения объектно-ориентированного программирования (ООП).
- Добавлен раздел «Краткий справочник по ООП».
- Добавлено краткое описание UML и паттернов проектирования.

Глава 13 «Клипы»

- Добавлены сведения о создании чистого клипа с нуля с помощью *MovieClip.createEmptyMovieClip( )*.
- Добавлен раздел о рисовании в клипе на этапе исполнения с помощью нового Drawing API.

- Добавлен раздел о реализации в клипе поведения кнопки.
- Добавлен раздел о работе с фокусом ввода в клипах.
- Переработано (исправлено) содержащее ошибки описание *MovieClip.duplicateMovieClip( )*.
- Список методов и свойств *MovieClip* перемещен в «Справочник по языку».
- Описание устаревшего *tellTarget* перенесено в приложение C.
- Усовершенствован пример часов, в котором теперь использованы лучшие приемы работы с Flash MX.
- Пример вопросника заменен новой загружаемой ООП-версией (старую по-прежнему можно найти в Интернете).

Глава 14 «Подклассы класса MovieClip и компоненты» (полностью новая)

- Рассмотрено создание подклассов клипов (специальных типов клипов, связанных с классом).
- Рассказано, как создать элементарный компонент; сложным примером компонентов являются компоненты пользовательского интерфейса Flash.

Глава 15 «Лексическая структура» (глава 14 первого издания)

- Переработан список зарезервированных слов.
- Удалены или переработаны следующие темы главы 15:
  - «Копирование, сравнение и передача данных» перенесено в главу 3.
  - «Поразрядное программирование» перенесено в статьи на *http://www.moock.org/asdg/technotes*.
  - Удален раздел «Более сложные проблемы областей видимости функций» (проблема была решена во Flash MX).
  - Раздел «Тип данных MovieClip» перенесен в статьи на *http://www.moock. org/asdg/technotes*.

Глава 16 «Среда разработки ActionScript»

- Переработан раздел о старых «умных клипах» (smart clip), в котором теперь рассказывается о новой компонентной архитектуре Flash MX.

Глава 17 «Создание формы Flash»

- Переработан код примеров с целью использования класса *LoadVars* вместо *loadVariables( )*.

Расформирована глава 18 «Экранные текстовые поля» из первого издания:

- Содержимое всей главы перемещено в «Справочник по языку» (в описание класса *TextField*) и в приложение E (с существенными дополнениями к классу *TextField*).

Удалена прежняя глава 19 «Отладка»

- Вся глава полностью перемещена в статьи на *http://www.moock.org/asdg/technotes*.

Часть II «Справочник по языку» (бывшая часть III)

- Ранее в этом предисловии мы отметили основные изменения и дополнения, касающиеся «Справочника по языку». Полный список новых методов, свойств, классов, объектов, глобальных функций и директив, добавленных в «Справочник по языку», можно найти на *http://www.moock.org/webdesign/lectures/newInMX*. (Обратите внимание, что статьи *CustomActions* и *LivePreview* не включены в «Справочник по языку», о чем говорится далее.)

Чего в этой книге нет

Книга велика, но ActionScript больше. Сейчас невозможно рассказать в одной книге обо всем, что есть в ActionScript. Из этого издания было специально изъято формальное описание следующих тем (о которых все же попутно рассказывается, когда это уместно):

- Функции, применяемые исключительно для расширения среды разработки Flash MX (например, *CustomActions* и *LivePreview*). О них рассказывается в статье Macromedia «Creating Components in Flash MX», расположенной по адресу *http://www.macromedia.com/support/flash/applications/creating_comps*.
- Библиотека Macromedia Flash UI Components, расширяющая возможности среды разработки за рамки базового языка. В приложении G «Сводка по компонентам интерфейса пользователя Flash» рассмотрены свойства и методы компонентов. Об источниках информации, подробно освещающих Flash UI Components, рассказано в «Резюме» главы 14.
- Macromedia Flash Communication Server MX (Comm Server) API (например, *Remote SharedObject*, *Camera*, *Microphone*, *NetConnection* и *NetStream*). Comm Server применяется для создания многопользовательских вебприложений со звуком и видео. См. подробности в *http://www.macromedia.com/software/flashcom/*.
- Основы использования среды разработчика Flash MX. Однако если вы – программист и не знакомы с Flash, мы дадим ряд советов относительно ввода и выполнения примеров кода. Чтобы изучить анимацию и разработку графики в среде Flash MX, начните с электронной подсказки и учебника; затем посетите сайты, перечисленные по адресу *http://www.moock.org/moockmarks*.

К книге не прилагается CD, но все примеры могут быть загружены из хранилища кода в Интернете, о котором говорится ниже.

Недокументированные функции ActionScript

Сообщество разработчиков Flash наловчилось раскапывать так называемые *недокументированные функции* ActionScript – внутренние возможности языка, о которых Macromedia официально не сообщает и не санкционирует

их применение. Как правило, применять недокументированные функции не рекомендуется по следующим причинам:

- Они не тестировались для внешнего использования и потому могут содержать ошибки или работать неустойчиво.
- Они могут быть удалены из последующих версий языка без предупреждения.

В этой книге мы ставили задачу обеспечить возможно лучшую документацию по тем функциям, которые поддерживаются, но документированы недостаточно подробно или неправильно. Поэтому о функциях, документация или поддержка для которых отсутствует, не сообщается, за исключением следующих случаев:

- Источники в Macromedia передали или подтвердили имеющуюся информацию.
- Функция применяется так широко, что ее необходимо описать.

В любом случае описания недокументированных возможностей снабжены соответствующими предупреждениям на видном месте. Книга освещает следующие недокументированные темы:

- `__proto__` (применяется для установки наследования)
- *ASBroadcaster* (частично освещена в главе 12)
- *ASSetPropFlags( )* (частично освещена в главе 8)
- *LoadVars.decode( )*
- *LoadVars.onData( )*
- *Object.hasOwnProperty( )*
- *System.showSettings( )*
- *TextField*.`condenseWhite`
- Задание шрифта при помощи свойства *TextFormat*.`font`
- Класс *XMLNode*

Узнать (сохраняя при этом благоразумие), что обнаружили сыщики в ActionScript , можно на сайте:

http://chattyfig.figleaf.com/flashcoders-wiki/index.php?Undocumented%20Features

Соглашения по именованию версий Flash

С появлением семейства продуктов MX, включающего Flash MX, Macromedia отказалась от обычной системы номеров версий для среды разработки Flash. Однако числовая нумерация версий Flash Player сохранена. В табл. В.2 приводятся соглашения по именам версий Flash, принятые в этой книге.

Таблица В.2. Соглашения по именам версий Flash, принятые в книге

Название	Смысл
Flash MX	Среда разработки Flash MX (в отличие от проигрывателя Flash Player).
Flash Player 6	Flash Player версии 6. Flash Player – модуль (plugin), подключаемый к большинству веб-броузеров в Windows и на Макинтошах. Существуют версии в виде ActiveX и в стиле Netscape, к которым применяется общее название «Flash Player 6», за исключением особо отмеченных случаев, как в статье, посвященной объекту *Accessibility*, в «Справочнике по языку».
Flash Player *x*.0.*y*.0	Flash Player конкретной версии, заданной *x* и *y*, как во Flash Player 6.0.47.0. См. подробности в «Справочнике по языку», в статье о `capabilities.version`.
Flash 6	Сокращение для «Flash Player 6», употребляемое преимущественно в «Справочнике по языку» или там, где различие между Flash MX (средой разработки) и Flash Player 6 (подключаемым модулем броузера) несущественно.
Flash 5, среда разработки	Среда разработки Flash 5 (в противоположность Flash Player), предшествовавшая Flash MX.
Flash Player 5	Flash Player версии 5.
Flash 5	Сокращение от «Flash Player 5», употребляемое в основном в «Справочнике по языку» или там, где различие между Flash 5 (средой разработки) и Flash Player 5 (подключаемым модулем броузера) несущественно.
Flash 2, Flash 3 и Flash 4	Версии Flash Player, предшествовавшие 5-й; употребляются преимущественно в «Справочнике по языку» и при этом указывается, какие версии Flash поддерживают данную функцию.
автономный проигрыватель	Версия Flash Player, работающая на машине непосредственно, а не как подключаемый модуль веб-броузера или элемент ActiveX.
Проектор (Projector)	Самодостаточный выполняемый модуль, содержащий в себе как файл *.swf*, так и автономный проигрыватель. Проекторы можно создавать для Mac OS или Windows с помощью функции Flash File → Publish.

Что умеет ActionScript?

ActionScript используется для создания любых типов интерактивных приложений, обычно для использования в Сети. Вот лишь некоторые возможности: проигрыватель MP3, многопользовательский графический редактор, трехмерное перемещение в здании, электронный магазин, доска объявлений, редактор HTML и игра Pacman. Каждое из этих приложений задействует некоторую группу возможностей ActionScript, выборочно рассмотренных ниже. Прикиньте, с помощью каких из этих приемов вы сможете создавать свои приложения.

Управление временной диаграммой

Фильмы Flash состоят из кадров, располагающихся в линейной последовательности, называемой *временной диаграммой* (*timeline*). ActionScript позволяет управлять воспроизведением временной диаграммы фильма, проигрывать отрезки фильма, выводить конкретный кадр, останавливать воспроизведение фильма, зацикливать анимации и синхронизировать анимационное содержимое. *Клипы* (*movie clips*) внутри главного фильма располагают собственными временными диаграммами.

Диалоговый режим

Фильмы Flash могут воспринимать вводимые пользователем данные и реагировать на них. С помощью ActionScript можно создавать следующие интерактивные элементы:

- Кнопки, реагирующие на щелчок мыши (например, классические кнопки навигации).
- Элементы GUI, такие как списки, выпадающие меню, флажки.
- Содержимое, анимируемое движениями мыши (например, шлейф мыши).
- Объекты, перемещаемые с помощью мыши или клавиатуры (например, автомобиль в симуляторе вождения).
- Текстовые поля, показывающие данные пользователю или позволяющие ему передавать данные в фильм (например, заполняемый бланк).

Управление визуальным и звуковым содержимым

ActionScript можно использовать для исследования или модификации свойств визуального и звукового содержимого фильма. Можно, например, изменять цвет и местоположение объекта, уменьшать громкость звука или устанавливать тип шрифта для текстового блока. Можно также многократно изменять эти свойства с течением времени, чтобы создавать такие эффекты, как движение, соответствующее законам физики, или обнаружение столкновений.

Программное создание содержимого

ActionScript позволяет генерировать визуальное и звуковое содержимое непосредственно из библиотеки фильма или посредством копирования содержимого, уже имеющегося в рабочем пространстве, или на сцене (stage). Во Flash MX можно посредством методов *createEmptyMovieClip( )* и *createTextField( )* из Drawing API класса *MovieClip* создавать графику и текст с нуля на этапе исполнения. Содержимое, генерируемое программно, может быть строго статическим элементом, например случайным узором или интерактивным элементом, таким как кнопка в диалоговом окне, вражеский космический корабль в видеоигре или пункт выпадающего меню.

Связь с сервером

Очень часто функциональность Flash расширяют путем организации связи с каким-либо приложением на стороне сервера, например Macromedia ColdFusion MX или сценарием Perl. Хотя связь с ColdFusion находится в сфере действия Macromedia Flash Remoting MX (Flash Remoting), базовый язык ActionScript предоставляет широкий набор инструментов для обмена данными с любыми приложениями или сценариями, выполняемыми на сервере (например, Java, PHP, ASP и т. д.). Обмен данными с сервером присутствует в следующих приложениях:

Ссылка на веб-страницу

См. *getURL( )*.

Гостевая книга

См. классы *LoadVars* и *XML*, главу 17 и хранилище кода, описываемое в следующем разделе.

Приложение чата

См. класс *XMLSocket* и пример на *http://www.moock.org/chat*.

Сетевая игра с нескольким участниками

См. класс *XMLSocket* и пример на *http://www.moock.org/unity*.

Операции торговли через Интернет

См. классы *LoadVars* и *XML*.

Персонализированный сайт с регистрацией и авторизацией пользователей

См. классы *LoadVars* и *XML*.

Подробное рассмотрение реализации даже такого ограниченного набора возможных приложений ActionScript выходит за рамки этой книги. Здесь мы постараемся обучить вас базовым навыкам, которые позволят вам самостоятельно исследовать бесчисленные возможности. Это не книга готовых рецептов, а уроки – как «приготовить» программный код с самого начала. Что будет в меню, решайте сами.

Хранилище кода

В последующих главах встретятся десятки примеров кода. Чтобы получить соответствующие файлы с исходным кодом, а также многие другие обучающие файлы, не включенные в книгу, посетите сетевое хранилище кода (Code Depot), находящееся здесь:

http://www.moock.org/asdg/codedepot

Хранилище кода – это развивающийся ресурс, содержащий практические приложения ActionScript и базовые коды. Вот список некоторых из них:

- Вопросник с вариантами ответов
- Средство просмотра графики с изменением масштаба

- Инструменты для текстовых полей, такие как конвертор массивов в таблицы и настраиваемая бегущая строка
- Приложение для чата, использующее XML
- Приложение гостевой книги
- Настраиваемые курсор мыши и кнопка
- Базовый код игры с астероидами
- Программные эффекты движения
- Демонстрация текстовых полей HTML
- Предзагрузчики
- Обработка строк
- Элементы интерфейса, такие как ползунки и средства прокрутки текста
- След мыши и другие зрительные эффекты
- Управление громкостью и звучанием

Кроме того, на указанный URL будут помещаться новые сведения о книге, уточнения, технические замечания и информация о выявленных ошибках.

Витрина

Практически в каждом существующем сайте, использующем Flash, есть хоть капелька ActionScript, а на некоторых сайтах его достаточно много. В табл. B.3 представлен ряд адресов, на которых вы можете получить вдохновение для собственной работы. Посмотрите также сайты, перечисленные в приложении A, и закладки, собранные автором на *http://www.moock.org/moockmarks*.

Таблица B.3. ActionScript Showcase

Тема	URL
Эксперименты в дизайне, интерактивности и сценариях	*http://www.yugop.com* *http://www.praystation.com** *http://www.presstube.com* *http://www.pitaru.com* *http://www.flight404.com* *http://www.bzort-12.com* *http://www.benchun.net/mx3d/** *http://www.protocol7.com** *http://www.uncontrol.com** *http://flash.onego.ru** *http://www.figleaf.com/development/flash5** *http://nuthing.com* *http://www.deconcept.com* *http://www.natzke.com*

* Можно загрузить файлы *.fla*. В других случаях доступны только файлы *.swf*.

Таблица В.3 (продолжение)

Тема	URL
Игры	*http://www.orisinal.com* *http://www.gigablast.com* *http://www.sadisticboxing.com* *http://www.huihui.de* *http://www.sarbakan.com* *http://www.electrotank.com/games/multiuser* *http://www.titoonic.dk/products/games/spider* *http://content.uselab.com/acno* *http://www.neave.com/webgames*
Интерфейс, приложения и динамическое содержимое	*http://www.mnh.si.edu/africanvoices* *http://www.curiousmedia.com* *http://www.smallblueprinter.com* *http://davinci.figleaf.com/davinci* *http://host.oddcast.com* *http://www.enteryourinformation.com/broadmoor/onescreen.cfm*

Типографские обозначения

В этой книге приняты следующие обозначения, призванные выделить различные синтаксические составляющие ActionScript:

Пункты меню

Выбор пункта меню обозначается символом → , например File → Open.

`Моноширинный шрифт`

Используется в примерах кода, именах экземпляров клипов, метках кадров, именах свойств и именах переменных. Имена переменных часто оканчиваются суффиксами, перечисленными в табл. 2.1 (например, `_mc` для переменных, ссылающихся на экземпляры клипов). Добавление суффиксов считается хорошим правилом, но иногда мы избегали их, если оказывалось, что они существенно затрудняют чтение окружающего текста. Поэтому для краткости суффиксы могут опускаться.

Курсив

Применяется в названиях функций, методов, классов, слоев, в URL, именах файлов и суффиксах файлов, например *.swf*. Кроме того, выделению функций и методов служат круглые скобки, например *duplicateMovieClip( )*.

`Моноширинный полужирный шрифт`

Указывает на текст, который надо ввести дословно при пошаговом выполнении процедуры. Его также используют внутри примеров кода, например для того, чтобы выделить важную строку в большом примере.

Моноширинный курсив

Этим шрифтом форматируется программный код, который надо заменить соответствующим значением (например, *здесь должно быть ваше имя*). Кроме того, он используется для выделения имен переменных, свойств, методов и функций, упоминаемых в комментариях в примерах кода.

Поэкспериментировав с разными шрифтами, мы пришли к выводу, что для «Справочника по языку» лучше всего подойдут следующие соглашения, не противоречащие нашему общему подходу:

- Свойства классов выделяются моноширинным шрифтом для имени класса и свойства, поскольку и то и другое должны быть введены буквально (например, `Stage.width`, `Math.NaN`).
- Свойства экземпляров выделяются моноширинным курсивом для класса или экземпляра объекта, поскольку их должно заменить имя конкретного экземпляра. Само свойство выделяется моноширинным шрифтом и должно быть введено точно так, как показано (например, *`Button`*`.tabEnabled`, где *`Button`* следует заменить экземпляром кнопки).
- Имена методов и функций, а также класс или объект, к которому они относятся, всегда выделяются курсивом с добавлением круглых скобок, как в *MovieClip.duplicateMovieClip( )*. Обратитесь к «Справочнику по языку», сопровождающему материалу или близлежащим примерам кода, чтобы выяснить, следует ли указать имя класса буквально, как в *TextField.getFontList( )*, или заменить его именем экземпляра, как в *ball_mc.duplicateMovieClip( )*.
- В «Справочнике по языку» при обсуждении свойства или метода класса для краткости имя класса часто опускается. Например, если при обсуждении свойства `htmlText` класса *TextField* мы говорим «установить свойство `htmlText`», по контексту должно быть понятно, что имеется в виду «установить свойство *`someField_txt`*`.htmlText`, где *`someField_txt`* – идентификатор конкретного текстового поля».
- В некоторых случаях свойство объекта содержит ссылку на метод или обработчик обратного вызова. При этом не всегда ясно, следовало ли с помощью моноширинного шрифта указать, что это свойство (хотя и хранящее имя метода), или применить курсив и скобки, указывая, что это метод (хотя и хранящийся в свойстве). Если граница между свойством, указывающим на метод, и самим методом оказывается расплывчатой, приносим свои извинения. Постоянно подчеркивая формальную разницу, мы сделали бы текст менее удобным для чтения.
- В сводке свойств класса их имена для экономии места могут выделяться *курсивом*, а не `моноширинным` шрифтом. Это происходит только при перечислении их под заголовком «Свойства» без круглых скобок, поэтому ясно, что речь идет о свойствах, а не о методах.

Что-то из сказанного может показаться сейчас неясным, но к тому моменту, когда вы доберетесь до «Справочника по языку», прочтя предварительно об объектах, классах и клипах в главах 12, 13 и 14, неясностей у вас не останется.

Обратите особое внимание на замечания и предупреждения, отделенные от текста следующими значками:

Это совет. В нем содержится полезная информация по рассматриваемой теме, часто освещающая важные понятия или лучшие приемы.

Это предостережение, которое поможет вам решить или избежать неприятных проблем или предупредит о неизбежном событии. За последствия игнорирования предупреждения вы будете отвечать сами.

Сообщите нам свое мнение

Информация, приведенная в данной книге, была проверена со всей возможной тщательностью, но некоторые функции могли измениться (не исключены даже ошибки!). Ждем ваших писем с сообщениями о найденных ошибках и предложениями для будущих изданий по адресу:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в США и Канаде)
(707) 829-0515 (международные/местные)
(707) 829-0104 (факс)

У этой книги есть веб-страница, содержащая сведения об уже найденных ошибках, примеры и различную дополнительную информацию:

http://www.oreilly.com/catalog/actscript2

С комментариями и техническими вопросами по данной книге обращайтесь по адресу:

bookquestions@oreilly.com

Дополнительные сведения о наших книгах, конференциях, программном обеспечении, центрах ресурсов и сети O'Reilly Network можно найти на нашем веб-сайте:

http://www.oreilly.com

Благодарности

Как и первое издание, эта книга имела бы бледный вид без огромного вклада сотрудников Macromedia – групп разработки, контроля качества, сопровождения и управления продуктом Flash MX. В частности, огромная благодарность Гэри Гроссману (Gary Grossman) за критику, руководство и терпение, а также за написанное им предисловие. Вот имена сотрудников Macromedia, участвовавших в формировании этого текста: Джонатан Гэй (Jonathan Gay), Дже-

реми Кларк (Jeremy Clark), Эрик Виттман (Eric Wittman), Майкл Уильямс (Michael Williams), Пит Сантанджели (Pete Santangeli), Мэтт Вобенсмит (Matt Wobensmith), Бен Чан (Ben Chun), Трой Эванс (Troy Evans), Ли Томасон (Lee Thomason), Бентли Вулф (Bentley Wolfe), Джон Дауделл (John Dowdell), Ребекка Сан (Rebecca Sun), Дженис Пирс (Janice Pearce), Брайан Дистер (Brian Dister), Генриэтта Коэн (Henriette Cohn), Джефф Мотт (Jeff Mott), Майкл Моррис (Michael Morris), Денеб Мекета (Deneb Meketa), Тайник Уро (Tinic Uro), Роберт Тацуми (Robert Tatsumi), Колм Мак-Кеон (Colm McKeon) и Майк Чамберс (Mike Chambers).

Эту книгу редактировал Брюс Эпштейн (Bruce Epstein), у которого, как мне кажется, сверхчеловеческие способности. Он обладает исключительными познаниями в писательском деле и программировании, а его способность применять эти познания к тексту поразительна. Мне необычайно повезло, что у меня был такой выдающийся редактор (и самостоятельный автор).

Далее имею честь представить технических редакторов данного издания, которые входят в команду разработчиков Macromedia Flash MX: Гэри Гроссман, Крис Тилген (Chris Thilgen), Жиль Дрю (Gilles Drieu), Нигель Пегг (Nigel Pegg), Славик Лозбен (Slavik Lozben) и Майкл Ричардс (Michael Richards). Эрика Нортон (Erica Norton) редактировала первое издание. Спасибо, друзья, за потраченное время и самоотверженность.

Читателями предварительного текста этого издания были известные разработчики Flash, пользующиеся моим огромным уважением: Роберт Пеннер (Robert Penner) (*http://www.robertpenner.com*), Дейв Янг (Dave Yang) (*http://www.quantumwave.com*), Бренден Холл (Branden Hall) (*http://www.waxpraxis.org*), Амит Питару (Amit Pitaru) (*http:/ /www.pitaru.com*), Майкл Кей (Michael Kay) (*http://www.peep.org/wizard/*) и Вероника Броссье (Veronique Brossier) (*http://www.v-ro.com*). Точность этой книги обеспечена остротой их взгляда.

Спасибо Тиму О'Рейли (Tim O'Reilly), который установил высокий стандарт основательности, качества и точности во всем, что им публикуется. Моя благодарность Брайену Сойеру (Brian Sawyer), Клэр Клотье (Claire Cloutier), Глену Бисигнани (Glenn Bisignani), Майку Сьерре (Mike Sierra), Робу Романо (Rob Romano), Эди Фридман (Edie Freedman), Сенди Торр (Sandy Torre) и многим другим редакторам, составителям указателей, корректорам, сотрудникам служб продажи и маркетинга в O'Reilly, усилия которых способствовали появлению этой книги на полках.

Выражаю признательность своему доброму другу Дереку Клейтону (Derek Clayton), который систематически делился со мной своим опытом программирования. Дерек написал код Perl для главы 17, сервер Java *XMLSocket* для «Справочника по языку» и общую систему базы данных на плоских (flat) файлах, которые можно получить из сетевого хранилища данных. Он также является ведущим разработчиком Unity Socket Server, коммерческого приложения moock.org для создания во Flash многопользовательских приложений (*http://www.moock.org/unity*).

Сообществу Flash: спасибо за вдохновение и красоту, которые вы создаете. В частности, благодарность Джеймсу Паттерсону (James Patterson), Юго Накамуре (Yugo Nakamura), Наоки Мицусе (Naoki Mitsuse), Джошуа Девису (Joshua Davis), Джеймсу Бейкеру (James Baker), Марселю Марсу (Marcell Mars), Филиппу Торроне (Phillip Torrone), Роберту Рейнгардту (Robert Reinhardt), Марку Феннелу (Mark Fennell), Джошу Улму (Josh Ulm), Даррелу Планту (Darrel Plant), Тодду Пургасону (Todd Purgason), Джону Накку (John Nack), Джейсону Крогу (Jason Krogh), Хилману Куртису (Hillman Curtis), Глену Томасу (Glenn Thomas), Хосс Гиффорд (Hoss Gifford), Мануэлю Клементу (Manuel Clement), Андреасу Хейму (Andreas Heim), Роберту Ходжину (Robert Hodgin), Маргарет Карлсон (Margaret Carlson), Эрику Нацке (Erik Natzke), Андреасу Одендаалу (Andries Odendaal), Джеймсу Тиндаллу (James Tindall), Джону Вильямсу (Jon Williams), Ферри Халим (Ferry Halim), Джобу Макару (Jobe Makar), Джаред Тарбелл (Jared Tarbell), Джеффу Стирнсу (Geoff Stearns), Полю Сципула (Paul Szypula), Линде Вейнман (Lynda Weinman), перечисленным выше читателям бета-версии и всем тем, кого я по каким-то причинам пропустил.

Большая благодарность и признание в любви моей жене Венди Шаффер (Wendy Schaffer), родителям, семье и друзьям. Надеюсь, что это издание не было для них таким обременительным, как первое.

И наконец, хочу поблагодарить вас, читатель, за то, что вы нашли время для чтения этой книги. Надеюсь, она сможет передать вам мою страсть.

Колин Мук
Торонто, Канада
декабрь 2002

14

Подклассы класса MovieClip и компоненты

В главе 12 мы построили пользовательский класс *Ball* и с его помощью создавали отдельные экземпляры объектов *Ball*. Когда потребовалось отобразить объекты шариков, перед нами предстало несколько возможностей. Можно поместить шарики в отдельный класс *Room* и сделать его ответственным за рисование их на экране с помощью Drawing API. Можно ввести в объект *Ball* свойство `mc`, которое будет содержать ссылку на клип, представляющий шарик. В некоторых случаях наличие объекта, который представляет сущность (например, шарик), и отдельного клипа, отображающего его на экране, оказывается избыточным. Клипы сами по себе уже являются объектами, поэтому часто оказывается разумным использовать клип одновременно для описания объекта и его отображения на экране. Во Flash MX это можно сделать, создав подкласс на базе класса *MovieClip*, который и станет специализированным типом объекта, обладающим встроенной возможностью отображения на экране.

Во Flash 5 создание подкласса *MovieClip* требовало ряда трудных обходных маневров, недоступных большинству разработчиков. Flash MX вводит формальную архитектуру для создания подклассов *MovieClip*, в которой участвуют следующие новые функции и выражения: `#initclip`, *new MovieClip( )*, *Object.registerClass( )* и параметр *`initObj`* функции *attachMovie( )*.

Чтобы посмотреть, как создаются подклассы *MovieClip*, мы реализуем класс *Ball* из главы 12 в виде подкласса *MovieClip*. В нашем примере каждый экземпляр шарика будет клипом, ответственным за собственное перемещение по экрану. Просмотрите код примера 14.1; мы разберем его в последующих разделах. В целом в коде использована технология ООП, изученная нами в главе 12, но с некоторыми ухищрениями, необходимыми в архитектуре создания подклассов *MovieClip*. Все примеры данной главы можно загрузить из сетевого хранилища кода.

Пример 14.1. Класс Ball, пример подкласса MovieClip

```
// ========================================
// Создать пространство имен для хранения классов
// ========================================
```

```
if (_global.org == undefined) {
  _global.org = new Object();
}
if (_global.org.moock == undefined) {
  _global.org.moock = new Object();
}
// ======================================
// Создать класс Ball
// ======================================
/*
 * Класс Ball. Расширяет MovieClip.
 *   Версия: 1.0.0
 *   Описание: Подкласс класса клипа для показа шариков.
 *             Требует наличия в библиотеке символа с именем ballSymbol.
 *
 * Параметры конструктора:
 *   radius          - Размер шарика (половина его диаметра)
 *   ballColor       - Цвет шарика
 *   x               - Начальная x-координата шарика
 *   y               - Начальная y-координата шарика
 *   velocity        - Вектор перемещения шарика: {x:0, y:0}
 *
 * Методы:
 *   setSize()       - Установить размер клипа в соответствии с заданным радиусом
 *   setPosition()   - Поместить клип шарика в точку с координатами (x,y)
 *   setColor()      - Задать RGB-значение цвета для клипа шарика
 *   move()          - Применить вектор перемещения шарика к его текущей позиции
 *
 * Обработчики событий:
 *   onEnterFrame()  - Вызывает move() в каждом кадре.
 */
/*
 * Конструктор класса Ball (параметры передаются через initObj функции attachMovie())
 */
org.moock.Ball = function () {
  // Создать свойства экземпляра
  this.velocity = this.params.velocity;
  // Инициализировать экземпляр
  this.setPosition(this.params.x, this.params.y);
  this.setColor(this.params.ballColor);
  this.setSize(this.params.radius);
  this.onEnterFrame = this.move;
  // Удалить из экземпляра объект params конструктора
  delete this.params;
}
/*
 * Задать MovieClip в качестве надкласса Ball
 */
org.moock.Ball.prototype = new MovieClip();
/*
 * Связать библиотечный ballSymbol с классом Ball
 */
Object.registerClass("ballSymbol", org.moock.Ball);
```

```
/*
 * Методы экземпляра
 */
/*
 * Метод: Ball.setSize()
 *   Описание: Устанавливает размер клипа шарика
 *
 * Параметры:
 *   newRadius     - Новый радиус шарика
 */
org.moock.Ball.prototype.setSize = function (newRadius) {
  this._width = this._height = (newRadius * 2);
};
/*
 * Метод: Ball.setPosition()
 *   Описание: Помещает клип шарика в точку (x, y)
 *
 * Параметры:
 *   x             - Новое положение по горизонтали
 *   y             - Новое положение по вертикали
 */
org.moock.Ball.prototype.setPosition = function (x, y) {
  this._x = x;
  this._y = y;
};
/*
 * Метод: Ball.setColor()
 *   Описание: Изменяет цвет клипа шарика
 *
 * Параметры:
 *   newColor      - Численное RGB-значение цвета, устанавливаемого для клипа
 *
 */
org.moock.Ball.prototype.setColor = function (newColor) {
  var col = new Color(this);
  col.setRGB(newColor);
};
/*
 * Метод: Ball.move()
 *   Описание: Передвигает шарик в соответствии с его вектором перемещения
 *
 * Параметры: Нет
 *
 */
org.moock.Ball.prototype.move = function () {
  this._x += this.velocity.x;
  this._y += this.velocity.y;
}
// ========================================
// ЗАПУСК ПРИЛОЖЕНИЯ
// ========================================
main();
// Функция: main()
```

```
// Описание: Точка входа в приложение
function main () {
  // Создать объект, свойства которого будут использованы как параметры конструктора Ball
  var ballParams = new Object();
  ballParams.radius = 10;
  ballParams.ballColor = 0xFF0000;
  ballParams.x = 250;
  ballParams.y = 250;
  ballParams.velocity = {x:12, y:4};

  // Присвоить ссылку на объект ballParams единственному свойству объекта initObj,
  // который мы передадим в attachMovie(). Свойства initObj копируются в каждый
  // экземпляр ball во время прикрепления.
  var initObj = new Object();
  initObj.params = ballParams;

  // Создать шарик
  this.attachMovie("ballSymbol", "redBall", 1, initObj);
}
```

Создание символа в библиотеке

Прежде чем вернуться к коду примера 14.1, создадим элементы, которые будет использовать наш фильм. Каждый подкласс клипа связывается с символом, хранящимся в библиотеке. Чтобы создавать экземпляры подкласса программным способом, требуется экспортировать этот символ для использования с ActionScript. Ниже перечислены операции, которые надо выполнить, чтобы создать и экспортировать символ для нашего класса *Ball*:

1. Создать символ с именем *ballSymbol*, содержащий круг.
2. В Library выбрать *ballSymbol*.
3. Во всплывающем меню Options для Library выбрать Linkage. Появится диалоговое окно Linkage Properties.
4. Выставить флажок Export For ActionScript.
5. В поле Identifier ввести `ballSymbol`.
6. Щелкнуть по кнопке OK.

Теперь мы создадим функцию-конструктор *Ball*, которая будет автоматически вызываться при каждом создании экземпляра *ballSymbol*.

Создание и вызов конструктора подкласса

В данный момент конструктор *Ball* (из примера 14.1) определен на главной временной диаграмме файла *.fla*. Позже мы увидим, как переместить этот конструктор в библиотечный символ *ballSymbol*. Конструктор *Ball* очень похож на функции-конструкторы, которые мы видели в главе 12, с той дополнительной особенностью, что он определен в глобальном пространстве имен, соответствующем доменному имени `moock.org` (ваши собственные классы должны быть определены в пространстве имен вашего личного домена типа `com.yoursite.YourClass`):

```
/*
 * Конструктор класса Ball.
 */
org.moock.Ball = function () {
  // Создать свойства экземпляра.
  // ...
  // Инициализировать экземпляр.
  // ...
};
```

Однако наш конструктор класса не определяет никаких параметров! Помните, что мы создаем подкласс *MovieClip*, а экземпляры клипов не создаются стандартным способом, как *new Object( )*, что способствовало бы использованию параметров. Экземпляры клипов создаются с помощью функций *attachMovie( )* или *duplicateMovieClip( )*, порождающих экземпляр и вызывающих его конструктор класса (если таковой определен). Поэтому передавать аргументы функции-конструктору *Ball* надо через *attachMovie( )* или *duplicateMovieClip( )*. Но передать аргументы непосредственно, как в синтаксисе *new Object(arg1,...argn)*, нельзя. Поэтому мы прибегаем к так называемому инициализирующему объекту *initObject*, свойства которого будут скопированы в новый экземпляр непосредственно перед выполнением функции-конструктора.

Например, следующий код прикрепляет экземпляр символа, экспортированный как `"boxSymbol""`:

```
var initObj = new Object();
initObj.description = "A four-sided figure";
this.attachMovie("boxSymbol", "box_mc", 2, initObj);
```

Этот код дает новому экземпляру имя `"box_mc"`, помещает его на глубину 2 и передает ему initObject с единственным свойством `description`. После выполнения этого кода экземпляр `box_mc` будет обладать свойством `description` со значением `"A four-sided figure"`.

Вместо настоящих параметров конструкторы подкласса *MovieClip* должны использовать свойства initObject в *attachMovie( )* или *duplicateMovieClip( )*. Например, наш конструктор *Ball* (показанный ниже) использует свойство `params` объекта initObject, которое само является объектом, для передачи в конструктор параметров `x`, `y`, `velocity`, `radius` и `ballColor`:

```
org.moock.Ball = function () {
  // Создать свойства экземпляра
  this.velocity = this.params.velocity;

  // Инициализировать экземпляр
  this.setPosition(this.params.x, this.params.y);
  this.setColor(this.params.ballColor);
  this.setSize(this.params.radius);
  this.onEnterFrame = this.move;

  // Удалить из экземпляра объект params конструктора
  delete this.params;
}
```

Прикрепляя новый экземпляр, мы создаем initObject следующим образом:

```
// Создать объект, свойства которого будут использованы
// как параметры конструктора Ball
var ballParams = new Object();
ballParams.radius = 10;
ballParams.ballColor = 0xFF0000;
ballParams.x = 250;
ballParams.y = 250;
ballParams.velocity = {x:12, y:4};

// Присвоить ссылку на объект ballParams единственному свойству initObj.params
var initObj = new Object();
initObj.params = ballParams;

// Теперь передать initObj в экземпляр через attachMovie(). Свойство params
// копируется в новый экземпляр вместе со всеми параметрами конструктора,
// определенными в ballParams.
this.attachMovie("ballSymbol", "redBall", 1, initObj);
```

В примере 14.1 все инструкции для создания экземпляра находятся в функции *main( )*, поэтому переменные ballParams и initObj оказываются локальными и автоматически удаляются по завершении выполнения *main( )*. В противном случае мы должны были бы удалить ballParams и initObj явно или передать объект initObject в виде литерала, чтобы он не сохранялся после передачи в *attachMovie( )*.

Может возникнуть вопрос, зачем обременять себя свойством params, а не определить свойства radius, velocity, ballColor, _x и _y непосредственно в initObject и прямо не использовать их в нашем конструкторе *Ball*, например так:

```
// Код конструктора
org.moock.Ball = function () {
  // Инициализировать экземпляр.
  this.setColor(this.ballColor);
  this.setSize(this.radius);
  this.onEnterFrame = this.move;
};

// Создание экземпляра
var initObj = new Object();
initObj.radius = 10;
initObj.velocity = {x:44, y:2};
initObj.ballColor = 0xFF0000;
initObj._x = 30;
initObj._y = 12;
this.attachMovie("ballSymbol", "redBall", 1, initObj);
```

Установить свойства непосредственно для объекта initObj можно, но при этом возникают две проблемы:

- Страдает централизация кода: ответственность за создание свойств экземпляра возлагается на функцию *attachMovie( )*, тогда как заниматься этим должна функция-конструктор.

- Параметры конструктора (которые должны быть временными переменными) сохраняются как постоянные свойства нового экземпляра. Например, в свойстве `ballColor` после выполнения конструктора нет необходимости, однако оно остается определенным, пока существует экземпляр.

Чтобы централизовать код и не загружать экземпляр ненужными свойствами параметров конструктора, мы определяем все параметры в одном свойстве `params` объекта initObject. Завершая свою работу, конструктор удаляет свойство `params`:

```
// Удалить объект конструктора params из экземпляра.
delete this.params;
```

Назначение MovieClip надклассом

Мы создали конструктор класса *Ball* и научились создавать экземпляры объектов *Ball* с помощью *attachMovie( )*. Теперь мы должны сделать класс *Ball* подклассом класса *MovieClip*. В главе 12 мы видели, как объявляется надкласс для класса:

```
SubClass.prototype = new SuperClass();
```

Поэтому неудивительно, что объявление *MovieClip* в качестве надкласса *Ball* выглядит так:

```
/*
 * Задать MovieClip в качестве надкласса Ball.
 */
org.moock.Ball.prototype = new MovieClip();
```

Однако хотя мы и определили *Ball* как подкласс *MovieClip*, нам еще надо сообщить интерпретатору, чтобы он выполнил конструктор *Ball* при создании нового экземпляра *ballSymbol*, экспортированного из библиотеки. Чтобы связать экспортированный символ с нужным конструктором класса, воспользуемся *Object.registerClass( )*. Ниже мы связываем *ballSymbol* с классом *Ball*:

```
/*
 * Связать библиотечный ballSymbol с классом Ball.
 */
Object.registerClass("ballSymbol", org.moock.Ball);
```

На человеческий язык это указание можно перевести так: «При создании экземпляра *ballSymbol* вызвать конструктор *org.moock.Ball( )*, так чтобы ключевое слово `this` указывало на новый экземпляр *ballSymbol*».

Инструкция *new MovieClip( )* только устанавливает класс в качестве подкласса *MovieClip* (то есть только задает наследование). С ее помощью *нельзя* создать новый клип, используемый в фильме! Создание новых экземпляров клипов или подклассов клипов, таких как *Ball*, осуществляется с помощью *attachMovie( )*, *duplicateMovieClip( )* или *createEmptyMovieClip( )*.

Подробнее об *Object.registerClass( )* см. в «Справочнике по языку».

Объединение кода подкласса с библиотечным символом

Назначив надкласс, мы создали полностью функциональный подкласс *MovieClip* с именем *Ball*. Добавление в него новых свойств и методов осуществляется через `Ball.prototype` так же, как и для любого класса (как показано в примере 14.1). Допустим, однако, что этот класс *Ball* надо использовать совместно с другим разработчиком. Разумно поместить наш класс во внешний файл *.as* и сообщить другому разработчику, что он может включить файл с классом в свой код с помощью `#include`. Но при этом нужно также сообщить ему о необходимости создать для этого класса символ в библиотеке и экспортировать его под именем `"ballSymbol"`. Если разработчик не выполнит экспорт символа или укажет при экспорте неверный идентификатор связи, класс работать не будет.

Наличие файла класса и сопутствующего символа клипа неудобно и чревато возникновением ошибок. Чтобы сделать подкласс *MovieClip* более самодостаточным, можно поместить код внутрь самого символа между прагмами `#initclip` и `#endinitclip` (создав так называемый *блок* `#initclip`).

Например, чтобы расположить класс *Ball* внутри клипа *ballSymbol*, поместим в кадр 1 временной диаграммы *ballSymbol* следующий код. (Для краткости код тела функции и ряд комментариев опущены.)

```
#initclip

// ========================================
// Создать пространство имен для хранения классов
// ========================================
if (_global.org == undefined) {
  _global.org = new Object();
}
if (_global.org.moock == undefined) {
  _global.org.moock = new Object();
}

// ======================================
// Создать класс Ball
// ======================================

// Конструктор класса
org.moock.Ball = function () { ... };

// Назначить MovieClip надклассом Ball
org.moock.Ball.prototype = new MovieClip();

// Связать библиотечный ballSymbol с классом Ball
Object.registerClass("ballSymbol", org.moock.Ball);

// Метод: Ball.setSize()
org.moock.Ball.prototype.setSize = function (newRadius) { ... };

// Метод: Ball.setPosition()
org.moock.Ball.prototype.setPosition = function (x, y) { ... };

// Метод: Ball.setColor()
```

```
org.moock.Ball.prototype.setColor = function (newColor) { ... };

// Метод: Ball.move()
org.moock.Ball.prototype.move = function () { ... };

#endinitclip
```

Блок `#initclip` может находиться только в кадре 1 временной диаграммы клипа. Если код размещен на главной временной диаграмме или не в кадре 1 временной диаграммы клипа, возникнет ошибка компиляции. Код внутри блока выполняется один и только один раз для каждого символа, непосредственно перед созданием первого экземпляра символа. Определив наш класс *Ball* внутри блока `#initclip` в *ballSymbol*, мы гарантируем доступность конструктора класса *Ball* перед появлением первого экземпляра *ballSymbol*. Методы и свойства, определенные *Ball*, сразу становятся доступными каждому новому экземпляру *Ball*.

Когда класс *Ball* хранится в *ballSymbol* в блоке `#initclip`, становится легко перемещать весь пакет между фильмами, перетаскивая *ballSymbol* из библиотеки одного файла *.fla* в библиотеку другого. Без всяких дополнительных усилий файл *.fla* получает возможность прикреплять и использовать экземпляры *ballSymbol*.

Подробнее об `#initclip` см. в «Справочнике по языку».

Создание компонентов

До сих пор мы не обращали внимания на то, что экземпляр нашего *ballSymbol* можно вручную перетаскивать на сцену в среде разработки Flash. В этом случае конструктор *Ball( )* будет запускаться на этапе выполнения даже для тех экземпляров, которые были помещены на сцену в среде разработки (при условии, что *Ball* был определен как подкласс перед размещением экземпляра на временной диаграмме). Но при создании экземпляра вручную во время разработки нет возможности задать параметры через initObject, как при использовании *attachMovie( )* или *duplicateMovieClip( )*. Чтобы решить эту проблему, надо определить символ клипа в виде компонента.

Компонент – это символ, предоставляющий GUI для задания свойств экземпляров в среде разработки. Обычно компоненты являются также подклассами *MovieClip*, которые используют свойства, заданные на этапе разработки, в качестве параметров конструктора.

Компоненты Flash MX представляют собой новую и улучшенную реализацию Smart Clips из Flash 5.

Создание компонентов мы более подробно изучим в главе 16. А сейчас посмотрим, как преобразовать наш *ballSymbol* в компонент. Продолжим работу с *ballSymbol* из предыдущего раздела, в блоке `#initclip` которого определен конструктор класса *Ball*.

1. Выбрать в библиотеке *ballSymbol*.

2. Во всплывающем меню Options окна Library выбрать Component Definition. Появится диалоговое окно Component Definition.
3. Нажать кнопку «+» пять раз, чтобы вставить позиции для пяти новых параметров компонента.
4. В поле Name для пяти параметров ввести `radius`, `velocity`, `color`, `x` и `y`. Это «косметические» имена параметров этапа разработки, которые увидит конечный пользователь. В них можно узнать параметры нашего конструктора *Ball*.
5. В поле Variable для этих пяти параметров ввести `cp_radius`, `cp_velocity`, `cp_ballColor`, `cp_x` и `cp_y`. Это имена свойств, которым будут присваиваться значения в каждом экземпляре *ballSymbol*. Префикс `cp_` представляет собой введенное нами соглашение, которое означает «component parameter». С его помощью мы найдем и удалим параметры компонента после того, как их конструктор завершит их использование.
6. Для этих пяти параметров вызвать всплывающее меню Type и задать Number, Object, Color, Number и Number. Это значения типа данных каждого из параметров. Тип Color предоставляет интерфейс для выбора пользователем цвета.
7. В поле Value для пяти параметров ввести `0`, `{x:0, y:0}`, `#000000`, `0` и `0`. Обратите внимание, что для `velocity` и `color` используется специальный интерфейс, упрощающий ввод свойств объекта и значений цвета. Это значения, которые устанавливаются по умолчанию, но пользователь может изменить их при перетаскивании экземпляра на сцену.
8. Щелкнуть по кнопке OK.

Определение нашего компонента *ballSymbol* показано на рис. 14.1.

В результате, когда пользователь будет перетаскивать экземпляр *ballSymbol* на сцену, инспектор свойств предоставит GUI для задания «параметров компонента» этого экземпляра, как показано на рис. 14.2.

Значения параметров, заданные пользователем в инспекторе свойств, автоматически присваиваются свойствам экземпляра *ballSymbol* (при этом берутся те имена свойств, которые мы задали раньше на шаге 5 – `cp_radius`, `cp_velocity`, `cp_ballColor`, `cp_x` и `cp_y`). Хотя можно было бы просто использовать эти свойства экземпляра «как есть», мы модифицируем конструктор *Ball*, чтобы улучшить модульность кода.

Вспомните, что наша исходная функция конструктора *Ball* требует, чтобы параметры были переданы ей в едином пакете – объекте `params`. Поэтому конструктор обращается к каждому параметру как к `this.params.`*`paramName`*, что показано в примере 14.1. Объект `params` также аккуратно хранит все параметры конструктора, чтобы их удобно было удалить после создания экземпляра. Напротив, параметры нашего компонента (`cp_radius`, `cp_velocity` и т. д.) определены непосредственно как свойства новых экземпляров *ballSymbol*, а не в объекте `params`. Так же как и в initObject функции *attachMovie( )*, можно использовать эти непосредственно заданные свойства в имеющемся виде, не привлекая функцию-конструктор. Однако это было бы посягательством

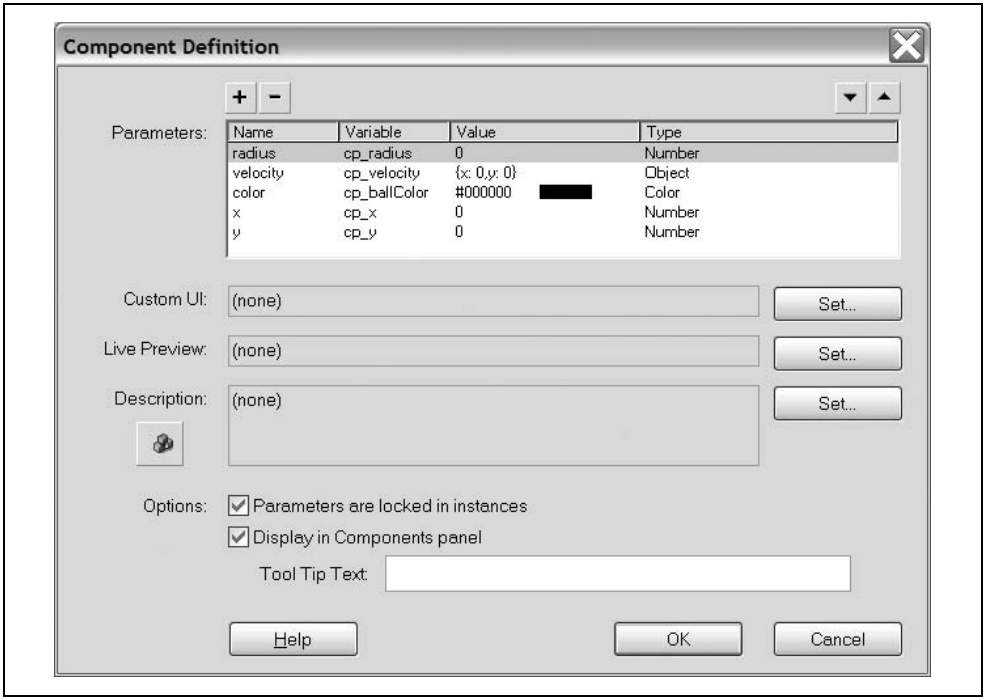

Рис. 14.1. *Определение компонента ballSymbol*

на обязанности конструктора. Правильная функция-конструктор отвечает за создание свойств экземпляра и осуществление инициализации экземпляра; она также использует параметры конструктора только для инициализации нового экземпляра.

Кроме того, обычные параметры конструктора являются локальными для функции конструктора и автоматически прекращают существование, когда конструктор завершает свою работу. Наш конструктор *Ball* заставляет параметры компонента (`cp_radius`, `cp_velocity` и т. д.) выполнить роль параметров конструктора, используя их только для инициализации нового экземпляра *Ball* и явным образом удаляя их в конце функции. В нашем конструкторе *Ball* параметры компонента являются временными и не приравнены к внутренним свойствам экземпляра. Например, можно было бы попросить поль-

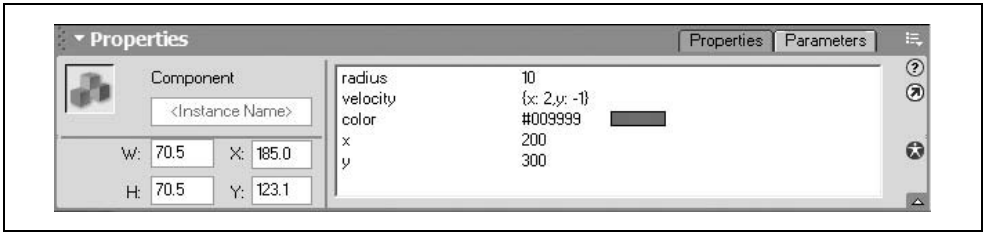

Рис. 14.2. *GUI для параметров компонента ballSymbol*

зователя задать диаметр шарика в качестве параметра компонента, но получить из него радиус и сохранить в виде свойства. Проведение различия между параметрами компонента и свойствами нашего класса *Ball* делает код централизованным: свойства создаются, как это и должно быть, конструктором, а не через GUI определения компонента.

Ниже приведен исправленный код для конструктора *Ball*, превращенного в компонент. Прочий код класса *Ball* остался прежним. Обратите внимание на важность, которую приобретают комментарии, – конструктор не определяет параметры, поэтому мы должны определить необходимые параметры, основываясь на комментариях. (Для краткости в последующем коде опущены комментарии, относящиеся к номеру версии, методам и обработчикам.) Конструктор распознает параметры компонента, которые надо удалить, по префиксу `cp_`, который мы задали ранее в диалоговом окне Component Definition.

```
/*
 * Класс Ball. Расширяет MovieClip.
 *
 * Параметры компоненты:
 *   cp_radius           - Размер шарика
 *   cp_ballColor        - Цвет шарика
 *   cp_x                - Начальная x-координата шарика
 *   cp_y                - Начальная y-координата шарика
 *   cp_velocity         - Вектор перемещения шарика: {x:0, y:0}
 *
 * Методы: ...
 *
 * Обработчики событий: ...
 */
/*
 * Конструктор класса. параметры передаются через initObj функции attachMovie().
 */
org.moock.Ball = function () {
  // Создать свойства экземпляра.
  this.velocity = this.cp_velocity;

  // Инициализировать экземпляр.
  this.setPosition(this.cp_x, this.cp_y);
  this.setColor(this.cp_ballColor);
  this.setSize(this.cp_radius);
  this.onEnterFrame = this.move;

  // Удалить из экземпляра параметры компонента (свойства, имена которых
  // начинаются с префикса cp_, являются параметрами компонента).
  for (var p in this) {
    if (p.indexOf("cp_") != -1) {
      delete this[p];
    }
  }
};
```

Конечно, теперь, после модификации конструктора *Ball*, мы должны модифицировать initObject, чтобы он использовал имена свойств компонента при

создании экземпляров *ballSymbol* с помощью *attachMovie( )*. Это позволит использовать одну и ту же функцию-конструктор независимо от того, создается ли компонент вручную на этапе разработки или программно на этапе выполнения.

Вот модифицированный вариант с применением *attachMovie( )*:

```
var initObj = new Object();
initObj.cp_radius = 10;
initObj.cp_ballColor = 0xFF0000;
initObj.cp_x = 250;
initObj.cp_y = 250;
initObj.cp_velocity = {x:12, y:4};

// Создать шарик.
this.attachMovie("ballSymbol", "redBall", 1, initObj);
```

Если все экземпляры подкласса клипа будут создаваться программно с помощью *attachMovie( )* или *duplicateMovieClip( )*, конструктор должен принимать параметры, которые прикреплены к единственному объекту `params` в объекте initObject. Однако при создании компонентов, экземпляры которых будут создаваться на этапе разработки, конструктор должен принимать отдельные параметры компонента, начинающиеся с префикса `cp_` (а при создании экземпляров компонента программным способом следует прикрепить одноименные параметры непосредственно к initObject).

Упаковка графики компонента

Многие визуальные компоненты динамически создают свое изображение на экране на этапе выполнения путем прикрепления экспортированных клипов к экземплярам компонентов. При перемещении компонента, зависящего от экспортированных элементов, из одного файла *.fla* в другой каждый такой элемент должен быть перемещен в библиотеку целевого файла. Во Flash MX нет формальных средств для упаковки компонента, состоящего из нескольких символов, в один объект. Однако традиционный подход состоит в размещении всех элементов в *направляющем слое* (*guide layer*) внутри компонента (см. описание направляющих слоев в электронной справке Flash). Элементы, расположенные в направляющем слое, автоматически перемещаются при перетаскивании компонента между библиотеками, но не видны в компоненте на этапе выполнения (если только компонент не прикрепит их явным образом).

Шаблон компонента

Ниже представлен код общего шаблона компонента, реализующего подкласс *MovieClip*. В шаблоне не используется пространство имен типа `org.moock`, как в нашем примере *Ball*. Кроме того, шаблон не хранит все параметры конструктора в едином объекте `params`. Эти дополнительные приемы вы можете реализовать по своему усмотрению. Однако в шаблоне предполагается, что параметры конструктора имеют префикс `cp_`, и после создания объекта все параметры компонента удаляются. Хотя удалять параметры конструктора не обязательно, такая практика написания кода полезна.

```
// КОД В КАДРЕ 1 БИБЛИОТЕЧНОГО СИМВОЛА someClassSymbol.

#initclip

/*
 * Класс SomeClass. Расширяет MovieClip
 *   Версия: 1.0.0
 *   Описание: Здесь находится ваше описание
 *
 * Параметры компонента:
 *   cp_param1       - Краткое описание
 *   cp_param2       - Краткое описание
 *
 * Методы:
 *   someMethod()    - Краткое описание
 */

/*
 * Конструктор класса
 * (Параметры передаются через initObj или устанавливаются как параметры компонента)
 */
_global.SomeClass = function () {
  // Создать свойства экземпляра
  this.prop1 = this.cp_param1;
  // Инициализировать экземпляр
  this.someMethod(this.cp_param2);
  // Удалить из экземпляра параметры (с префиксом cp_)
  for (var p in this) {
    if (p.indexOf("cp_") != -1) {
      delete this[p];
    }
  }
};

/*
 * Задать MovieClip в качестве надкласса SomeClass
 */
SomeClass.prototype = new MovieClip();

/*
 * Связать библиотечный someClassSymbol с классом SomeClass
 */
Object.registerClass("someClassSymbol", SomeClass);

/*
 * Методы экземпляра
 */

/*
 * Метод: SomeClass.someMethod()
 *   Описание: Краткое описание
 *
 * Параметры:
 *   param     - Краткое описание
 *
 */
```

```
SomeClass.prototype.someMethod = function (param) {
  trace("someMethod invoked with parameter of: " + param);
};

#endinitclip
```

В следующем коде показан пример применения шаблона *SomeClass*. Чтобы посмотреть на экземпляр *SomeClass*, созданный вручную, загрузите файл *componentTemplate.fla* из хранилища кода.

```
// Создать экземпляр
var initObj = new Object();
initObj.cp_param1 = 15;
initObj.cp_param2 = "Hello world";
this.attachMovie("someClassSymbol", "someClassInstance", 1, initObj);
```

Подклассы подклассов MovieClip

Как и обычный класс, подкласс *MovieClip* можно расширить посредством нового подкласса. Если родительский класс определен вне соответствующего ему символа (как в самом первом примере *Ball* в этой главе), определить его дочерний подкласс можно так:

- Присвоить прототипу дочернего класса новый экземпляр родительского класса
- Зарегистрировать дочерний класс в его символе с помощью *Object.registerClass( )*

Не забудьте также вызвать родительский конструктор из конструктора дочернего класса. Например, следующий код показывает, как создать подкласс *Baseball* из существующего класса *Ball*:

```
// Конструктор класса Baseball
org.moock.Baseball = function () {
  // ... код инициализации ...

  // Вызвать конструктор класса Ball
  super();
}

// Назначить Ball надклассом Baseball
org.moock.Baseball.prototype = new org.moock.Ball();

// Связать класс Baseball с собственным символом в библиотеке
Object.registerClass("baseballSymbol", org.moock.Baseball);
```

Однако когда внутри блока `#initclip` символа определены подкласс (*Ball*) и подкласс этого подкласса (*Baseball*), надо обеспечить инициализацию подкласса перед инициализацией любого из его собственных подклассов. Для этого применяется параметр `order` прагмы `#initclip`, который является неотрицательным целым числом, задающим порядок выполнения конкретного блока `#initclip` относительно всех остальных блоков `#initclip` в фильме. Блоки `#initclip` с меньшими значениями *order* выполняются раньше, чем блоки с большими значениями. Все блоки `#initclip`, для которых не задан параметр *order*, выполняются раньше блоков с заданным параметром *order*.

Например, чтобы создать класс *Baseball* внутри клипа *baseballSymbol*, мы должны заменить первую строку кода класса *Ball* следующей:

```
#initclip 0
```

После этого можно создать класс *Baseball* так:

```
// Использовать более высокий порядок #initclip, чем у Ball, чтобы гарантировать
// доступность Ball к моменту выполнения этого кода.
#initclip 1

// ================================================
// Создать пространство имен для хранения классов
// ================================================
if (_global.org == undefined) {
  _global.org = new Object();
}
if (_global.org.moock == undefined) {
  _global.org.moock = new Object();
}

// ================================================
// Создать класс Baseball
// ================================================

/*
 * Класс Baseball. Расширяет Ball.
 *   Версия: 1.0.0
 *   Описание: Подкласс клипа для отображения baseball
 *
 * Параметры конструктора:
 *   signature       - Сигнатура шарика
 *
 * Методы:
 *   setSignature()  - Установить сигнатуру для шарика
 */

/*
 * Конструктор класса. Параметры передаются через объект initObj метода attachMovie()
 */
org.moock.Baseball = function () {
  // Создать свойства экземпляра.
  this.sig = null;

  // Инициализировать экземпляр, прежде чем вызывать конструктор Ball.
  // (Иначе Ball удалит объект params прежде, чем мы успеем его использовать.)
  this.setSignature(this.params.signature);
  // Вызвать конструктор класса Ball.
  super();

  // Не требуется удалять объект params, поскольку это сделает конструктор Ball.
};
/*
 * Назначить Ball надклассом Baseball.
 */
```

```
org.moock.Baseball.prototype = new org.moock.Ball();

/*
 * Связать библиотечный baseballSymbol с классом Baseball
 */
Object.registerClass("baseballSymbol", org.moock.Baseball);
/*
 * Методы экземпляра
 */

/*
 *     Метод: Ball.setSignature()
 *  Описание: Устанавливает сигнатуру шарика
 *
 * Параметры:
 *   newSig     - Новая сигнатура шарика
 */
org.moock.Ball.prototype.setSignature = function (newSig) {
  this.sig = newSig;
};

#endinitclip
```

Обратите внимание, что у нескольких блоков `#initclip` может быть одинаковое значение *order*. Например, если мы создадим три подкласса *Ball*, то можем во всех их блоках `#initclip` сделать параметр `order` равным 1. Значение имеет только то, что в *Ball* порядок блоков `#initclip` ниже, чем во всех его подклассах.

Резюме

В этой главе мы изучили три взаимосвязанных, но различных типа подклассов *MovieClip*:

- Подкласс, определенный вне своего библиотечного символа.
- Подкласс, определенный в своем библиотечном символе (внутри директивы `#initclip`).
- Компонент, позволяющий задавать свои параметры с помощью GUI на этапе разработки.

Тип подкласса *MovieClip*, который вам следует использовать в своем приложении, целиком зависит от вашей ситуации. Если надо разрешить размещение экземпляров на этапе разработки, то вполне вероятно, что придется создать компонент. Аналогично, если вы собираетесь распространять свой подкласс, передавая его другим разработчикам, то компонент способствует созданию хорошего транспортного механизма. В противном случае шаг определения компонента можно пропустить. Но, как правило, следует определить подкласс непосредственно внутри связанного с ним библиотечного символа в рамках блока `#initclip`. Хотя это не требуется правилами, но хранение подкласса в связанном с ним символе облегчает обслуживание пакета в целом.

Компоненты и подклассы клипов – два краеугольных камня технологии разработки приложений Flash MX. Создавая библиотеки кода в виде компонентов, можно быстро собирать приложения из повторно используемых элементов. Macromedia включила во Flash MX ряд полезных компонентов интерфейса пользователя (UI Components). Обязательно ознакомьтесь с этими компонентами, их методами и параметрами (см. приложение G). Исходный код Flash UI Components также поставляется вместе с Flash MX, поэтому можно изучать эти компоненты и расширять их, создавая свои собственные.

Вот материалы для дополнительного изучения компонентов:

Введение в учебный материал по компонентам

В среде разработки Flash MX выберите Help → Tutorials → Introduction to Components.

Применение компонентов

В среде разработки Flash MX выберите Help → Using Flash → Using Components.

Создание адресной книги с помощью компонентов (Colin Moock)

http://www.macromedia.com/desdev/mx/flash/articles/addressbook.html

Создание форм и компонентов в Macromedia Flash MX

http://www.macromedia.com/support/flash/applications/creating_forms/

Создание компонентов в Macromedia Flash MX

http://www.macromedia.com/support/flash/applications/creating_comps/

Создание компонента Flash MX

http://www.flashcomponents.net/tutorials/triangle/triangle.html

Расширение компонентов в Macromedia Flash MX

http://www.macromedia.com/desdev/mx/flash/articles/fmx_components.html

Модификация компонентов ListBox и ComboBox

http://www.macromedia.com/desdev/mx/flash/articles/extending_components.html

Коллекция компонентов Macromedia Flash

http://www.macromedia.com/exchange/flash

Коллекция компонентов сторонних разработчиков

http://www.flashcomponents.net/

Вперед!

Компоненты и подклассы клипов – последняя большая тема в изучении основ языка ActionScript. Прежде чем перейти к «Справочнику по языку», мы поговорим о *лексической структуре* (о тонкостях синтаксиса ActionScript) и объединим все вместе в заключительном учебном приложении главы 17. Дополнительные сведения о процедуре определения компонентов см. в главе 16.