Flash-программирование

Предисловие Гари Гроссмана, создателя ActionScript

ActionScript

Подробное руководство

Символ

O'REILLY®

Колин Мук

ActionScript

The Definitive Guide

Colin Moock

O'REILLY®

ActionScript

Подробное руководство

Колин Мук

Санкт-Петербург
2002

Колин Мук

ActionScript. Подробное руководство

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Редактор	*Т. Маслова*
Корректура	*С. Журавина*
Верстка	*Н. Гриценко*

Мук К.
ActionScript. Подробное руководство. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 792 с., ил.
ISBN 5-93286-028-6

Издание полностью посвящено описанию ActionScript – объектно-ориентированного языка программирования Flash – и адресовано как разработчикам Flash, делающим первые шаги в программировании, так и тем, кто использует свое знание JavaScript при переходе на ActionScript (оба языка основаны на стандарте ECMAScript).

С помощью этой книги новички быстро осваивают программирование на ActionScript. Опытные программисты могут при изучении сложных вопросов, специфичных для Flash, поднять уровень своих знаний по JavaScript. Помимо теории в книге есть масса практических советов и примеров, в том числе касающихся текстовых полей с прокруткой, кнопок меню, вопросников с вариантами ответов, сайтов, управляемых XML, видеоигр с использованием законов физики, многопользовательских сред реального времени и многого другого. Книга содержит описание многих недокументированных или недостаточно документированных тем. Исчерпывающий и точный «Справочник по языку» послужит незаменимым попутчиком в ежедневной работе.

ISBN 5-93286-028-6
ISBN 1-56592-852-0 (англ)

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 31.10.2001. Формат 70x100$^{1}/_{16}$. Бумага офсетная.
Печать офсетная. Объем 49,5 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в ФГУП «Печатный Двор» им. А. М. Горького
Министерства РФ по делам печати, телерадиовещания
и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Оглавление

Предисловие 11

Введение 15

Часть I. Основы ActionScript 23

1. Легкое введение для непрограммистов 25

Некоторые основные фразы 27
Другие понятия ActionScript 36
Создание вопросника с вариантами ответов 45
Вперед! 59

2. Переменные 60

Создание переменных (объявление) 61
Присваивание значений переменным 63
Изменение и извлечение значений переменных 64
Типы значений 66
Область видимости переменной 68
Некоторые практические примеры 80
Вперед! 81

3. Данные и типы данных 82

Данные и информация 82
Сохранение смысла данных с помощью типов данных 83
Создание данных и задание их категорий 84
Преобразование типов данных 86
Элементарные и сложные типы данных 96
Вперед! 97

4. Элементарные типы данных 98

Тип Number 98
Целые числа и числа с плавающей точкой 98
Числовые литералы 99
Действия с числами 103

Строковый тип данных 104
Работа со строками 109
Булев тип 129
Тип undefined 131
Тип null 132
Вперед! 132

5. Операторы 133

Общие характеристики операторов 133
Оператор присваивания 138
Арифметические операторы 139
Операторы равенства и неравенства 144
Операторы сравнения 149
Строковые операторы 152
Логические операторы 153
Оператор группирования 159
Оператор запятая 159
Оператор void 160
Прочие операторы 160
Вперед! 164

6. Предложения 165

Типы предложений 166
Синтаксис предложений 166
Предложения ActionScript 168
Сравнение предложений с действиями 176
Вперед! 176

7. Условные предложения 177

Предложение if 178
Предложение else 180
Предложение else if 182
Эмуляция предложения switch 183
Компактный синтаксис условных предложений 185
Вперед! 185

8. Предложения цикла 186

Цикл while 186
Терминология циклов 190
Цикл do-while 191
Цикл for 192
Цикл for-in 193

Досрочное завершение цикла 195
Циклы временной диаграммы и событий клипа 198
Вперед! 205

9. Функции 206

Создание функций 207
Запуск функций 207
Передача функции информации 208
Выход из функций и возвращение значений 211
Литералы функций 214
Доступность и срок жизни функций 215
Область видимости функции 216
Еще раз о параметрах функции 221
Рекурсивные функции 225
Внутренние функции 227
Функции как объекты 228
Централизация кода 230
Еще раз вопросник с вариантами ответов 230
Вперед! 234

10. События и обработчики событий 235

Синхронное выполнение кода 235
Асинхронное выполнение кода, основанное на событиях 236
Типы событий 236
Обработчики событий 237
Синтаксис обработчика событий 238
Создание обработчиков событий 238
Область видимости обработчиков событий 242
События кнопок 246
Обзор событий клипов 251
События клипов, связанные с воспроизведением фильмов 252
События клипа, связанные с пользователем 258
Порядок выполнения кода 264
Копирование обработчиков событий клипа 267
Обновление экрана с помощьюupdateAfterEvent 267
Повторное использование кода 269
Динамические обработчики событий клипов 269
Применение обработчиков событий 270
Вперед! 272

11. Массивы 273

Что такое массив? 273
Анатомия массива 274

Создание массивов 276
Обращение к элементам массива 278
Определение размера массива 280
Именованные элементы массивов 282
Добавление элементов в массив 283
Удаление элементов из массива 289
Общие средства обработки массивов 292
Многомерные массивы 298
Вопросник с вариантами выбора: подход № 3 299
Вперед! 300

12. Объекты и классы 301

Анатомия объекта 304
Создание экземпляров объектов 305
Свойства объектов 305
Методы 307
Классы и объектно-ориентированное программирование 311
Встроенные классы и объекты ActionScript 327
Вперед! 329

13. Клипы 330

«Объектность» клипов 331
Типы клипов 332
Создание клипов фильмов 335
Порядок расположения фильмов и экземпляров в стеке 343
Ссылки на экземпляры и главные фильмы 348
Удаление экземпляров клипов и главных фильмов 362
Встроенные свойства клипов 365
Методы клипов 367
Практические примеры клипов 372
Последний вопросник 376
Вперед! 379

14. Лексическая структура 380

Пробельные символы 380
Символы, завершающие предложение (точка с запятой) 382
Комментарии 383
Зарезервированные слова 385
Идентификаторы 387
Чувствительность к регистру 387
Вперед! 388

15. Более сложные темы 389

Копирование, сравнение и передача данных 389
Поразрядное программирование 392
Более сложные вопросы областей видимости функций 404
Тип данных movieclip 407
Вперед! 408

Часть II. Применение ActionScript 409

16. Среда разработки ActionScript 411

Панель Actions 411
Помещение сценариев в кадры 414
Добавление кода к кнопкам 415
Добавление сценариев в клипы 417
Куда девался код? 417
Производительность 418
Сохранение кода ActionScript во внешних файлах 419
Создание интеллектуальных клипов 421
Вперед! 428

17. Формы Flash 429

Цикл данных форм Flash 429
Создание заполняемой формы Flash 432
Вперед! 438

18. Экранные текстовые поля 439

Динамические текстовые поля 439
Текстовые поля для ввода данныхпользователем 441
Параметры текстовых полей 442
Свойства текстовых полей 447
Поддержка HTML 449
Работа с выделением текста в текстовых полях 458
Пустые текстовые поля и предложение for-in 458
Вперед! 459

19. Отладка 460

Средства отладки 461
Методика отладки 467
Вперед! 472

Часть III. Справочник по языку 475

Справочник по языку ActionScript 477

Глобальные функции 477
Глобальные свойства 478
Встроенные классы и объекты 479
Заголовки статей 479
Алфавитный справочник по языку 480

Часть IV. Приложения 729

A. Ресурсы 731

B. Набор символов Latin 1 и коды клавиш 736

C. Обратная совместимость 743

D. Отличия от ECMA-262 и JavaScript 748

Алфавитный указатель 751

Предисловие

Когда летом 1998 года я прибыл в Macromedia, чтобы присоединиться к команде разработчиков Flash, небольшая и динамичная группа уже создала изумительный продукт. Flash 3 почти повсеместно был принят как стандарт для векторной анимации в приложениях Сети. Его преданные и энергичные пользователи из числа талантливых художников создавали впечатляющие произведения, с каждым днем появляющиеся на все большем количестве сайтов.

Корни ActionScript можно проследить до того пункта в списке планируемых характеристик Flash 4, который назывался «Повышение интерактивности». Flash 3 предлагал только базовый набор действий для управления клипами и кнопками Flash и обеспечения интерактивности. Однако я помню впечатление, которое произвели на меня сложность и временные затраты реализации с помощью действий Flash 3 игры в крестики-нолики – простой задачи для большинства языков программирования.

Это было до появления ActionScript. Сегодня никого не удивляют динамические веб-сайты, созданные исключительно с помощью Flash 4. А сейчас появляются сайты, использующие еще более изощренные возможности, предоставляемые ActionScript во Flash 5.

Главной задачей при создании ActionScript было обеспечить его доступность: важно, чтобы ActionScript было легко использовать не-программистам. Вместо чистого окна для редактирования сценариев мы создали во Flash 4 визуальный, легко понятный интерфейс для добавления к фильмам интерактивности. Простота Flash 4 ActionScript сделала легким его изучение и, что важно, позволила сохранить маленький размер Flash Player.

Flash Player создан так, чтобы быстро загружаться даже при низкой пропускной способности соединения. Команда Flash повторяет мантру «а насколько увеличится код проигрывателя?», прежде чем добавить в него новую функцию. ActionScript не был исключением из этого правила. Задача для ActionScript, как и для любой новой функции проигрывателя, в том, чтобы получить максимум шума (богатства функций) за минимум денег (увеличение размера проигрывателя).

Мы знали, что пользователи найдут для ActionScript самые неожиданные применения, и тем более приятной неожиданностью было увидеть, чего они смогли достичь с его помощью. Через месяц после выпуска Flash 4 в Сети стали появляться изумительные сайты, использующие ActionScript: сайты электронной коммерции, комнаты для чатов, доски объявлений, аркадные игры, настольные игры и даже Flash-сайты для создания Flash-сайтов. Шлюзы были открыты и впустили новую волну анимированного, интерактивного, насыщенного графикой содержимого Сети.

Когда настало время разрабатывать Flash 5, мне в первую очередь хотелось, чтобы ActionScript развился в полноценный язык сценариев с возможностями, которыми программисты привыкли пользоваться в таких языках, как JavaScript – функциями, объектами, сложными управляющими операторами и многочисленными типами данных. Такой инструментарий позволил программистам повысить свою производительность в других языках, и я хотел, чтобы ActionScript тоже поддерживал его. Вместо проектирования языка с чистого листа я решил сделать ActionScript очень похожим на JavaScript, который фактически является стандартом для сценариев, выполняющихся в Интернете на стороне клиента. Точнее, моделью для ActionScript послужил стандарт ECMAScript (ECMA-262). Поэтому для программистов JavaScript, переходящих на Flash, ActionScript довольно близок. Кроме того, программисты ActionScript смогут применить свое знание ActionScript при программировании на JavaScript и легко использовать уже существующий код в обоих языках.

Требования к доступности и минимизации размера проигрывателя оставались в силе. JavaScript является тонким и сложным языком, и мы стремились в полной мере предоставить его мощь продвинутым пользователям, в то же время сохранив простоту использования Flash 4 ActionScript. С этой целью в новой панели Flash 5 Actions существует два режима: обычный режим (Normal Mode), улучшенная версия редактора Flash 4 ActionScript, и экспертный режим (Expert Mode), текстовый редактор для опытных пользователей. Для того чтобы минимизировать размер проигрывателя, пришлось отчасти пожертвовать совместимостью ActionScript с ECMAScript. Например, ActionScript не поддерживает компиляцию кода на этапе выполнения с помощью функции *eval( )*: для поддержки такой функции пришлось бы целиком включить в проигрыватель компилятор ActionScript, что повлекло бы недопустимое увеличение его размера. По той же причине не поддерживается поиск с помощью регулярных выражений. Обе эти характеристики очень полезны и демонстрируют, какие сложные решения пришлось принимать команде Flash, чтобы обеспечить баланс между размером проигрывателя и его функциональными возможностями.

К этим двум требованиям мы добавили третье: совместимость. Flash 5 ActionScript был спроектирован так, чтобы можно было достаточно

спокойно обновить сценарии Flash 4 до синтаксиса Flash 5. Кроме того, Flash 5 поддерживает Flash 4 ActionScript в качестве подмножества, поэтому с помощью Flash 5 можно отлично создавать фильмы Flash 4. Колин Мук осветил вопросы обратной совместимости, а также основные различия между ActionScript и JavaScript (часто обусловленные соображениями совместимости) в приложении C «Обратная совместимость» и приложении D «Отличия от ECMA-262 и JavaScript».

В течение всего процесса разработки команда Flash получала бесценные данные от сообщества пользователей Flash – крепко спаянной группы, обладающей собственным голосом, огромными талантами и увлеченностью. Сообщество Flash сыграло большую руководящую роль в формировании тех функций, которые вошли в продукт. Задачей Macromedia является создание программного обеспечения, которое удовлетворяло бы потребностям ее клиентов; для этого она прислушивается к клиентам и изучает методы их работы.

В конце концов, история Flash не закончена. Это живая работа, которая постоянно ведется с целью удовлетворения ваших потребностей. Flash-разработчики – это художники века информации, и задача команды Flash состоит в том, чтобы изготовить для них самые лучшие кисти и резцы. Эта книга является первым исчерпывающим учебником и справочником, посвященным исключительно языку ActionScript. В этом качестве она знаменует поворотную точку в эволюции ActionScript: теперь ActionScript стал достаточно сложен, чтобы заслужить такую превосходную книгу, насыщенную новейшим материалом и не обошедшую вниманием ни одну возможность языка.

Наслаждайтесь этой книгой и наслаждайтесь Flash 5 ActionScript. Мы все мечтаем увидеть ваши достижения!

— Гари Гроссман
Ведущий разработчик Macromedia Flash
Март 2001

Введение

В этой книге рассказывается как об основах языка ActionScript, так и о его продвинутом применении. Около 800 содержательных страниц посвящены подробнейшему исследованию языка ActionScript – от базовых понятий переменных и управления клипами до таких глубоких тем, как объекты и классы, связь с сервером и XML. В конечном итоге будет охвачено все, что относится к Flash-программированию.

Книга написана не только для программистов. Изложение ведется достаточно быстро, но для чтения не требуется предварительного знания программирования. Все, что требуется, – это опыт работы с той частью Flash, которая не связана с использованием ActionScript, и желание учиться. Конечно, если вы программист, это еще лучше; вы сразу сможете применить в ActionScript свое мастерство написания программ.

Данное издание содержит подробно документированный материал, который не изложен или недостаточно полно изложен в документации Macromedia или в книгах других авторов. Известно, что во Flash есть функции и технологии, о которых знают лишь посвященные, и сведения о них передаются изустно. Каким образом слои, клипы и загружаемые фильмы помещаются в стек Flash Player? (см. главу 13 «Клипы»). Как определяется порядок выполнения кода для каждого конкретного кадра? (см. главу 13). Обладают ли обработчики событий локальной областью видимости? (см. главу 10 «События и обработчики событий»). Почему число 90 иногда выводится как 89.9999999997? (см. главу 4 «Элементарные типы данных»). Я счел своей особой задачей нанести на карту эти неизведанные области. Конечно, я рассказываю и об основных приемах программирования, которые требуются в любом языке, например, о том, как обеспечить многократное выполнение участка кода (см. главу 8 «Предложения цикла»).

Эта книга задумана так, чтобы стать настольной, а не пылиться на полке. Часть III «Справочник по языку» исчерпывающим образом описывает все объекты, классы, свойства, методы и обработчики событий, имеющиеся в ActionScript. Этим справочником нужно регулярно пользоваться, чтобы узнавать новое и вспоминать вещи, которые постоянно забываются.

Прежде всего эта книга является подробным руководством. Это результат многолетних исследований, обработки тысяч электронных писем в Macromedia и откликов, полученных от пользователей всех уровней. Очевидно, книга носит отпечаток как моего страстного отношения к предмету, так и мучительного практического опыта, которым вы можете непосредственно воспользоваться. Она исчерпывающе авторитетно описывает ActionScript, а точность ее беспрецедентна благодаря технической проверке Гари Гроссмана (Gary Grossman), создателя ActionScript.

Что может ActionScript?

Честно говоря, полноценные языки типа Flash 5 ActionScript не имеют практических границ применения. Взглянем на некоторые специфические возможности ActionScript, что позволит почувствовать, какого рода темы мы будем освещать на протяжении всей книги. Постарайтесь представить себе, как можно сочетать эти приемы для достижения ваших конкретных целей.

Управление временной диаграммой

Фильмы Flash состоят из кадров, располагающихся в линейной последовательности, называемой *временной диаграммой* (*timeline*). С помощью ActionScript можно управлять воспроизведением временной диаграммы фильма, проигрывать отрезки фильма, выводить конкретный кадр, останавливать воспроизведение фильма, зацикливать анимации и синхронизировать анимационное содержание.

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 мышью (т. е. классические кнопки навигации)
- Содержимое, анимируемое движениями мыши (например, след от мыши)
- Объекты, которые можно перемещать с помощью мыши или клавиатуры (например, автомобиль в игре вождения)
- Текстовые поля, разрешающие пользователю 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 моделировать такие необычные эффекты, как движение по законам физики или обнаружение столкновений.

Программная генерация содержимого

С помощью ActionScript можно генерировать визуальное и звуковое содержимое непосредственно из библиотеки фильма или путем копирования содержимого, уже имеющегося на рабочем столе. Содержимое, генерируемое программным образом, может служить строго статичным элементом, например, случайным узором, или интерактивным элементом, таким как вражеский космический корабль в видеоигре или опция в выпадающем меню.

Обмен данными с сервером

ActionScript предоставляет большое разнообразие средств для отправки информации на сервер и получения ее с сервера. Во всех перечисленных ниже приложениях используется обмен данными с сервером:

- Ссылка на сайт
- Гостевая книга
- Приложение чата
- Сетевая игра с несколькими участниками
- Операция электронной торговли
- Персональный сайт с регистрацией и авторизацией пользователей

Конечно, эти примеры дают лишь ограниченное представление о возможных приложениях ActionScript. Задача книги в том, чтобы дать базовые навыки, которые позволят вам самостоятельно исследовать великое множество других возможностей. Это не сборник рецептов, это урок по приготовлению кода, а что будет в меню, решайте сами.

Хранилище кода

В последующих главах встретятся десятки примеров кода. Чтобы получить соответствующие исходные файлы, а также многие другие обучающие файлы, не включенные в книгу, следует посетить сетевое хранилище кода (Code Depot), находящееся по адресу:

http://www.moock.org/asdg

Хранилище кода – это развивающийся ресурс, содержащий практические приложения ActionScript и базовые коды. Вот список некоторых примеров, которые можно в нем найти (они скачиваются как по отдельности, так и единым zip-файлом):

- Вопросник с вариантами ответов
- Приложение для чата, использующее XML
- Приложение гостевой книги
- 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, как и на сайт данной книги, помещаются новые сведения о книге, уточнения, технические замечания и выявленные ошибки.

Витрина

Практически в каждом имеющемся сайте, использующем Flash, есть хоть капелька ActionScript, а на некоторых сайтах даже довольно много. В следующей таблице представлен ряд адресов, на которых можно получить вдохновение для собственной работы. Посмотрите также сайты, перечисленные в приложении А «Ресурсы».

Таблица. Витрина ActionScript

Тема	URL
Экспериментирование 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://kaluzhny.nm.ru/3D.html** *http://www.protocol7.com** *http://www.uncontrol.com**

(продолжение)

Тема	URL
	*http://www.digitalnotions.com/dev/flash5**
	*http://flash.onego.ru**
	*http://www.figleaf.com/development/flash5**
Игры	*http://www.gigablast.com*
	http://www.sadisticboxing.com
	*http://www.flashkit.com/arcade**
	http://www.huihui.de
Интерфейс и динамическое содержимое	*http://www.mnh.si.edu/africanvoices*
	http://www.curiousmedia.com

* Имеются загружаемые файлы *.fla*. В остальных случаях доступны только файлы *.swf*.

Типографские обозначения

Для выделения различных синтаксических составляющих ActionScript в книге использованы следующие обозначения:

- Моноширинный шрифт используется в примерах кода, именах экземпляров клипов, метках кадров, именах свойств и именах переменных.
- *Курсив* используется в названиях функций, методов, классов, слоев, именах и расширениях файлов, например *.swf*.
- **Моноширинный полужирный** шрифт используется для кода, который нужно вводить с клавиатуры при поэтапном выполнении процедуры.
- *Моноширинный курсив* используется для кода, который нужно заменить соответствующим значением (например, *здесь должно быть ваше имя*), или для имен переменных и свойств, упоминаемых в комментарии кода.
- За именами функций и методов следуют круглые скобки.

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

Это совет. В нем содержится полезная дополнительная информация по рассматриваемой теме.

Это предостережение, которое поможет вам решить или избежать неприятностей.

Сообщите нам свое мнение

Информация, приведенная в данной книге, была протестирована и проверена со всей тщательностью, но вы можете обнаружить, что некоторые функции изменились (или даже найти ошибку!). Пожалуйста, пишите о найденных ошибках, а также ваши предложения для будущих изданий по адресу:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (in the U.S. or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

У этой книги есть сайт, на котором сообщается об ошибках, приводятся примеры и содержится различная дополнительная информация. Он находится по адресу:

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

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 сайте:

http://www.oreilly.com

Благодарности

Я глубоко обязан следующим людям, с которыми имел честь быть знакомым и совместно работать:

- Чрезвычайно талантливой команде Flash в Macromedia, находящейся в постоянном поиске нового и уважающей людей, сидящих за терминалами, которые и образуют Сеть. Сочетание профессионализма, исследовательского духа и личной увлеченности, свойственное Macromedia, редко встречается в структурах корпораций.
- Непревзойденным профессионалам в O'Reilly: Тиму O'Рейли (Tim O'Reilly), Трою Мотту (Troy Mott), Майку Сьерра (Mike Sierra), Робу Романо (Rob Romano), Эди Фридману (Edie Freedman) и многим редакторам, составителям указателей, корректорам, а также сотрудникам служб продаж и маркетинга, благодаря которым эта книга появилась на полках.
- Дереку Клейтону (Derek Clayton), моему личному наставнику в программировании и другу. Помимо почти каждодневного консультирования по коду Дерек написал код Perl для главы 17 «Фор-

мы Flash» и очень часто тренировал меня в Quake. Он также написал на Java сервер *XMLSocket* и на Perl – систему базы данных с плоскими файлами; то и другое можно загрузить из Хранилища кода.

- Уэнди Шаффер (Wendy Schaffer), которая, помимо корректирования чернового варианта рукописи, смогла поддерживать меня энергией и любовью во время всепоглощающего труда по написанию этой книги.
- Брюсу Эпштейну (Bruce Epstein), который выступал как развивающий редактор, помогая улучшить почти каждое предложение в рукописи и часто предлагая варианты завершения темы. Его проницательная редакторская работа и руководство неоценимы.
- Гари Гросману (Gary Grossman), создателю ActionScript в Macromedia, каким-то образом всегда находившему время для ответов на вопросы, разъяснения тонкостей и даже поддержки дискуссий во время разработки ActionScript. Гари выступал ведущим техническим редактором книги, помогая прояснить важные понятия и детали. Точность этого текста во многом является прямым результатом его участия.
- Славику Лозбену (Slavik Lozben), специалисту Macromedia Flash, которого я не могу в достаточной мере отблагодарить за создание событий в клипах и *swapDepths*. Без помощи интеллекта Славика и его готовности участвовать в обсуждении я бы до сих пор писал главу о событиях и их обработчиках. Славик внес также большой вклад в качестве технического редактора.
- Эрике Нортон (Erica Norton), специалисту по качеству Macromedia ActionScript, которая с готовностью отвечала на вопросы и исследовала их один за другим. Помимо ведения регулярного обсуждения Эрика выкраивала в своем напряженном графике работы время для того, чтобы осуществлять техническое редактирование.
- Джереми Кларку (Jeremy Clark), менеджеру продукта Macromedia Flash, который оказал активную поддержку книге своими мыслями, советами, дружбой и ответами на мои бесконечные вопросы. Эрику Витману (Eric Wittman), главному менеджеру продукта Macromedia Flash, чье прозорливое руководство на годы определило развитие Flash. Дженис Пирс (Janice Pearce) из группы контроля качества Macromedia Flash, которая разъяснила различные вопросы выпуска Flash и любезно предоставила первые версии Flash 5. Мэтту Вобенсмиту (Matt Wobensmith), менеджеру сообщества Macromedia Flash и постоянному ценному источнику информации. Трою Эвансу (Troy Evans), менеджеру продукта Macromedia Flash Player, который руководит Flash Player и борется за него. Бентли Вольфу (Bentley Wolfe) из группы технической поддержки Macromedia, который, кажется, никогда не расстается с клавиатурой.
- Ричарду Коману (Richard Koman) из O'Reilly & Associates, обеспечившему руководство редакторской работой на этапах заявки и первых черновых версий.

- Дэвиду Фугейту (David Fugate), моему литературному агенту из Waterside Productions, чье усердие и доверие сделали деловые вопросы практически незаметными.
- Д. Джо Дуонгу (D. Joe Duong), который знает слишком много для своего возраста, несмотря на то, что много времени посвящает обучению других. Мне повезло быть одним из них. Майку Линковичу (Mike Linkovich), философу программирования, который вдохновляет в такой же мере, как поучает. Джеймсу Портеру (James Porter) и Эндрю Мерфи (Andrew Murphy) за чтение и тестирование, снова чтение и тестирование и еще раз чтение и тестирование. И, конечно, Грэму Бартону (Graham Barton).
- Дугу Кили (Doug Keeley), Терри Маквайру (Terry Maguire) и Джону Николзу (Jon Nicholls), создавшим ICE – компанию, в которой позволено сосуществовать профессии и страсти.
- Профессору Полю Биму (Paul Beam), увидевшему связь между литературой, коммуникациями и компьютерами в невероятных рамках студенческой программы по английскому языку в Университете Ватерлоо.
- Профессору Джеку Грею (Jack Gray) за мудрость и дружбу.
- Эндрю Харрису (Andrew Harris), Дэвиду Люкстону (David Luxton), Майклу Каванагу (Michael Kavanagh), Стивену Бурку (Stephen Burke), Шерил Гула (Cheryl Gula), Кристине Нишино (Christine Nishino), Стивену Мумбу (Stephen Mumby), Карин Трговак (Karin Trgovac) и Юдит Зисман (Judith Zissman), которых я ценю за их искусство, идеи и дружбу.
- Сообществу Flash, из которого я черпал вдохновение и понимание, включая Джеймса Патерсона (James Patterson), Юго Накамура (Yugo Nakamura), Наоки Мицузе (Naoki Mitsuse), Джошуа Дэвису (Joshua Davis), Джеймсу Бейкеру (James Baker), Марселю Марсу (Marcell Mars), Филлипу Торроне (Phillip Torrone), Роберту Рейнхардту (Robert Reinhardt), Марку Феннелу (Mark Fennell), Брандену Холлу (Branden Hall), Джошу Ульму (Josh Ulm), Даррелу Планту (Darrel Plant), Тодду Пургасону (Todd Purgason), Джону Наку (John Nack), Джейсону Крогу (Jason Krogh), Хилману Куртису (Hillman Curtis), Гленну Томасу (Glenn Thomas) и всем остальным, кого я неизбежно пропустил.
- Семейству Мук (Moocks) – Маргарет, Майклу, Джейн и Биз, которые научили меня думать, мечтать, исследовать и любить.
- Семейству Шаффер (Schaffers), за годы семейной жизни и дружбы.

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

Колин Мук
Торонто, Канада
Апрель 2001

I

Основы ActionScript

В этой части рассказывается о базовом синтаксисе и грамматике языка ActionScript: переменных, данных, предложениях, функциях, обработчиках событий, массивах, объектах и клипах. По окончании изучения части I вы будете знать все необходимое для написания программ ActionScript.

- Глава 1 «Легкое введение для непрограммистов»
- Глава 2 «Переменные»
- Глава 3 «Данные и типы данных»
- Глава 4 «Элементарные типы данных»
- Глава 5 «Операторы»
- Глава 6 «Предложения»
- Глава 7 «Условные предложения»
- Глава 8 «Предложения цикла»
- Глава 9 «Функции»
- Глава 10 «События и обработчики событий»
- Глава 11 «Массивы»
- Глава 12 «Объекты и классы»
- Глава 13 «Клипы»
- Глава 14 «Лексическая структура»
- Глава 15 «Более сложные темы»

1

Легкое введение для непрограммистов

Я буду учить вас разговаривать с Flash.

Не просто программировать на Flash, но говорить ему что-то и ждать, что он скажет в ответ. Это не метафора или просто риторический прием. Это философский подход к программированию.

Языки программирования используются для отправки информации компьютерам и получения информации от них. Язык программирования, так же как и естественный язык, представляет собой совокупность словаря и грамматики, используемую для общения. Пользуясь языком программирования, мы указываем компьютеру, какие действия он должен совершить, или запрашиваем у него информацию. Компьютер слушает, пытается выполнить запрашиваемые действия и выдает ответы. Поэтому, хотя вы могли подумать, что читаете эту книгу для того, чтобы «научиться программировать», на самом деле вы учитесь общаться с Flash. Конечно, Flash не разговаривает ни по-английски, ни по-французски, ни по-немецки, ни по-китайски. Родным языком для Flash является ActionScript, и вы будете учиться разговаривать на нем.

Обучение компьютерному языку иногда рассматривается как синоним обучения программированию. Но программирование – это больше, чем простое изучение синтаксиса языка. Что бы было, если бы Flash мог разговаривать по-английски, т. е. для общения с ним не требовалось изучать ActionScript?

Что произошло бы, если бы мы сказали: «Flash, сделай так, чтобы шарик прыгал по экрану»?

Flash не смог бы выполнить наш запрос, поскольку он не понимает слово «шарик». Ладно, это просто семантическая проблема. Flash ждет, чтобы ему описывали объекты того мира, который ему известен: клипы, кнопки, кадры и т. д. Поэтому перефразируем наш запрос в терминах, понятных Flash, и посмотрим, что произойдет: «Flash, сделай так, чтобы клип с именем `ball_one` прыгал по экрану».

Flash по-прежнему не сможет выполнить нашу просьбу без дополнительной информации. Какого размера должен быть шарик? Куда нужно его поместить? В каком направлении он должен начать перемещение? С какой скоростью он должен двигаться? В какой части экрана он должен прыгать? Как долго? В двух измерениях или в трех? Да... Такого количества вопросов мы не ожидали. В действительности, Flash не задает нам эти вопросы. Если Flash не может нас понять, то просто не делает того, чего мы от него хотим, или выдает сообщение об ошибке. Но сейчас представим себе, что Flash запросил у нас более явные инструкции, и переформулируем свой запрос в виде последовательности шагов:

1. Шарик – это символ круглого клипа с именем `ball`.
2. Квадрат – это символ клипа с четырьмя сторонами и именем `square`.
3. Создать новый зеленый шарик диаметром 50 пикселов.
4. Дать новому шарику имя `ball_one`.
5. Создать новый черный квадрат шириной 300 пикселов и поместить его в центр стола.
6. Поместить `ball_one` где-нибудь наверху квадрата.
7. Перемещать `ball_one` в случайном направлении со скоростью 75 пикселов в секунду.
8. Если `ball_one` ударится об одну из сторон квадрата, заставить его отскочить (в обратном направлении).
9. Продолжать, пока не поступит команда остановиться.

Хотя мы давали свои команды на естественном языке, нам все же пришлось пройти всю логику управления нашим прыгающим мячиком, чтобы Flash смог нас понять. Очевидно, программирование не ограничивается лишь знанием синтаксиса языков программирования. Аналогично этому, знание большого количества слов английского языка не обязательно означает, что вы хороший собеседник.

Наш гипотетический пример Flash, разговаривающего на английском языке, выявляет четыре важных аспекта программирования:

- Каким бы ни был язык, искусство программирования заключается в формулировке логических шагов.
- Прежде чем сказать что-либо на языке компьютера, обычно полезно сказать это на человеческом языке.

- Разговор, ведущийся на одном языке, будучи переведен на другой язык, по-прежнему состоит из тех же основных утверждений.
- Компьютеры не очень догадливы. Кроме того, их словарь весьма ограничен.

По большей части программирование никак не связано с написанием кода. Прежде чем написать хоть одну строчку ActionScript, точно продумайте, что вы хотите сделать, и запишите функции своей системы в виде блок-схемы или проекта. Когда ваша программа станет достаточно хорошо описана на уровне понятий, можете перевести ее на ActionScript.

В программировании, так же как в любви, политике и бизнесе, эффективное общение является ключом к успеху. Чтобы Flash мог понять ваш ActionScript, синтаксис должен быть абсолютно правильным, вплоть до последней кавычки, знака равенства и точки с запятой. И чтобы гарантировать, что Flash поймет, о чем вы говорите, нужно ссылаться только на тот мир, который ему известен, и использовать понятные ему термины. То, что может быть очевидным для вас, не всегда очевидно компьютеру. Программирование на компьютере можно уподобить разговору с ребенком: ничто не следует считать само собой разумеющимся, нужно быть точным в каждой детали и перечислять все шаги, необходимые для выполнения задачи. Но помните, что, в отличие от детей, Flash точно сделает то, что вы ему скажете, и ничего сверх того.

Некоторые основные фразы

В первый же день занятий любым иностранным языком вы, скорее всего, выучите несколько основных фраз («Добрый день», «Как вы поживаете?» и т. д.). Даже если вы просто выучили фразу и не знаете значения каждого слова, можно понять действие фразы и повторить ее, чтобы воспроизвести это действие. Когда вы изучите правила грамматики, расширите свой словарь и станете использовать слова из заученных фраз в различном контексте, то сможете лучше понять свои первые фразы. Оставшаяся часть этой главы будет во многом напоминать первый день на курсах иностранного языка: вы увидите различные фрагменты кода и познакомитесь с некоторыми основами грамматики программирования. Остальная часть книги будет построена на этом фундаменте. Возможно, закончив чтение книги, вы захотите вернуться к этой главе, чтобы посмотреть, как далеко вы продвинулись.

Создание кода

В качестве первого упражнения мы попробуем добавить к фильму Flash четыре простые строки кода. Почти все программирование на ActionScript происходит в панели Actions (Действия). Все инструк-

ции, которые мы добавляем в панель Actions, выполняются Flash при воспроизведении фильма. Откройте теперь панель Actions с помощью следующих шагов:

1. Запустите Flash с новым пустым документом.
2. На главной временной диаграмме выберите кадр 1 слоя 1.
3. Выберите меню Window → Actions.

Панель Actions разделена на две части: окно Script (справа) и окно Toolbox (слева). В окне Script (Сценарии) размещается весь наш код. Окно Toolbox (Инструменты) предоставляет быстрый доступ к процедурам, операторам, функциям, свойствам и объектам (Actions, Operators, Functions, Properties и Objects) ActionScript. Вероятно, вы узнаете показанные на рис. 1.1 основные действия по прежним версиям Flash.

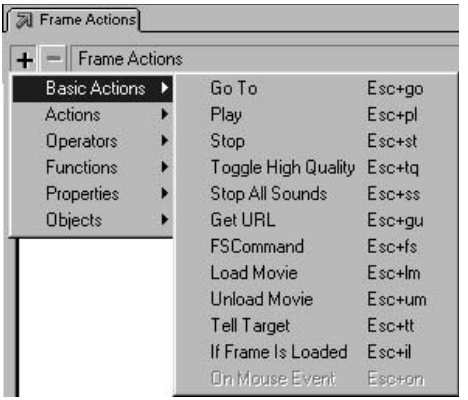

Рис. 1.1. *Основные действия Flash 5*

Но в окне Toolbox можно обнаружить и многое другое. На рис. 1.2 отражены все имеющиеся действия, в том числе есть несколько старых друзей из Flash 2, 3 и 4. Продолжив изучение окна Toolbox, вы даже найдете такие вещи, как Sound (звук), Array (массив) и XML. В данной книге мы расскажем обо всем этом.

С помощью меню окна Toolbox можно создавать код ActionScript. Однако, чтобы изучить синтаксис, принципы и структурный состав ActionScript, мы будем вводить весь код вручную.

Так называемые *Actions (Действия)* являются не просто действиями: они содержат различные базовые средства языка программирования – переменные, условные операторы, циклы, комментарии, вызовы функций и т. д. Они смешаны в кучу в одном меню, и общее название *Actions (Действия)* затемняет значение программных структур.

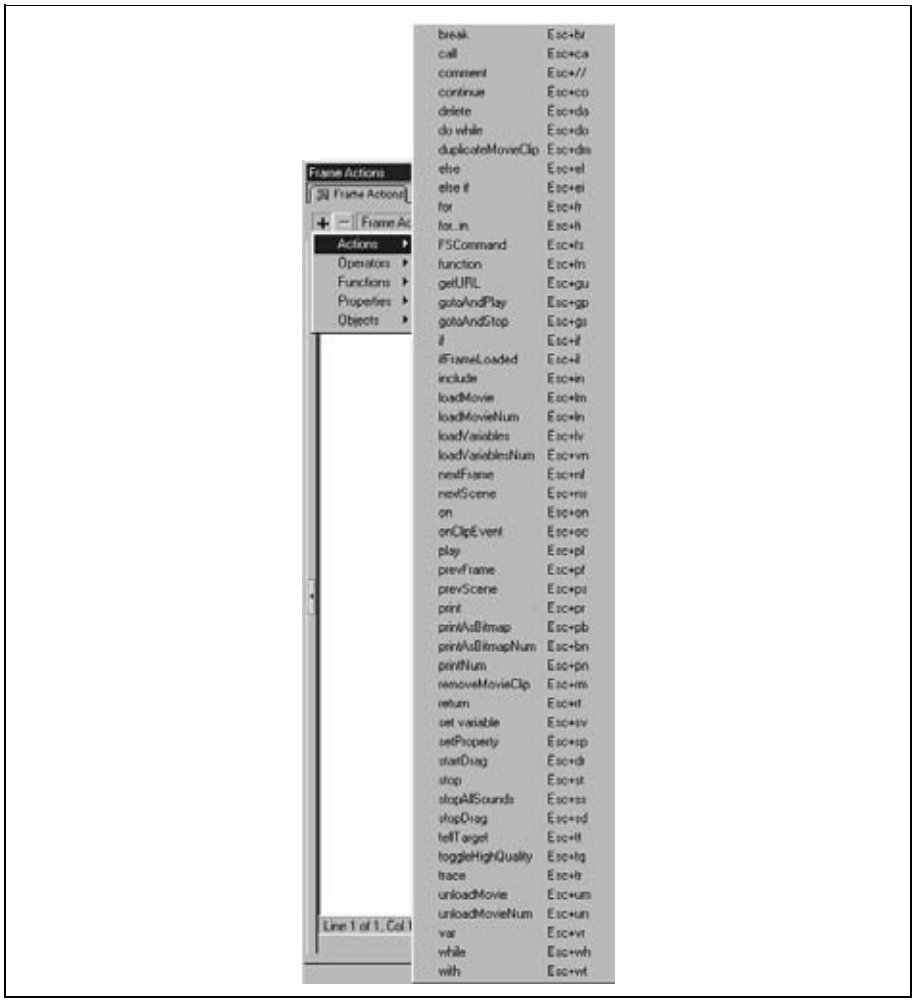

Рис. 1.2. *Расширенный список действий*

Мы будем раскрывать смысл Actions, чтобы взглянуть на них с точки зрения программиста. По всей книге для описания каждого рассматриваемого действия я использую подходящий термин программирования. Например, вместо «добавить действие *while*» я буду писать «создать цикл *while*». Вместо слов «добавить действие *if*» я буду писать «создать новый условный оператор». Вместо «добавить действие *play*» я буду писать «вызвать функцию (или метод) *play( )*». Эти различия составляют важную часть обучения языку ActionScript.

Готовы поработать руками? Тогда поздороваемся с Flash.

Поздороваемся с Flash

Прежде чем вводить код в панели Actions, нужно отключить автопилот ActionScript следующим образом:

1. Выбрать меню Edit → Preferences.
2. На вкладке General выбрать Actions Panel → Mode → Expert Mode.
3. Expert Mode (Экспертный режим) можно выбрать и через всплывающее меню, доступное по стрелке на правом краю панели Actions, хотя при этом режим устанавливается только для текущего кадра. См. главу 16 «Среда разработки ActionScript».

Как вам это нравится? Вы уже эксперт. При входе в режим Expert Mode окно Parameters внизу на панели Actions исчезает. Пусть вас это не тревожит: мы не программируем с помощью меню, и поэтому оно нам не понадобится.

Теперь выберите кадр 1 в слое 1. Ваш ActionScript (называемый также *кодом*) всегда должен быть прикреплен к кадру, клипу или кнопке; выбор кадра 1 влечет прикрепление к нему создаваемого вслед за этим кода. В режиме Expert Mode можно вводить код с клавиатуры прямо в окно Script в правой части панели Actions, где мы и будем осуществлять все программирование.

А теперь волнующий момент – ваша первая строка кода. Настало время познакомиться с Flash! Введите в окно Script следующее:

```
var message = "Hi there, Flash!";
```

Эта строка кода образует законченную команду, называемую *предложением (statement)*. Под этой строкой введите вторую и третью строки кода, показанные после этого абзаца. Замените *здесь ваше имя* на свое имя (всюду в этой книге, где вы увидите *код, набранный курсивом*, нужно заменить эту часть кода собственным содержимым):

```
var firstName = "здесь ваше имя";
trace (message);
```

Хм. Пока ничего не произошло. Это связано с тем, что наш код не может ничего выполнить, пока мы не экспортируем файл *.swf* и не воспроизведем наш фильм. Прежде чем сделать это, давайте попросим Flash ответить на наше приветствие. Введите под уже имеющимися строками четвертую строку кода (ну, у нас тут полоса удачи ...):

```
trace ("Hi there, " + firstName + ", nice to meet you.");
```

Теперь Flash готов вас встретить. Выберите меню Control → Test Movie и посмотрите, что произойдет. В окне Output должен появиться некий текст, как показано на рис. 1.3.

Ловко, да? Давайте разберемся, как все это произошло.

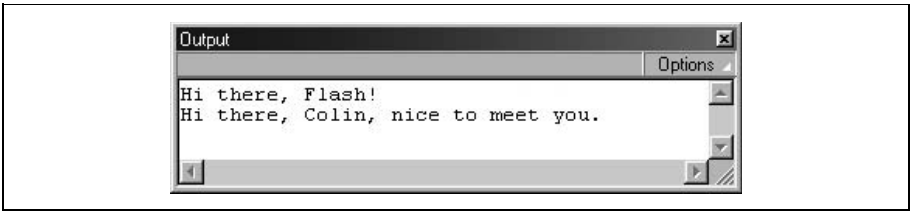

Рис. 1.3. *Flash становится дружелюбным*

Отслеживание событий (переменные)

Помните, что я назвал программирование просто общением с компьютером? Так оно и есть, но, пожалуй, оно менее личное, чем я это до сих пор представлял. В первой строке кода

```
var message = "Hi there, Flash!";
```

на самом деле вы не поздоровались с Flash. Скорее, вы сказали что-то вроде следующего:

> Flash, запомни, пожалуйста, кое-какую информацию для меня, а именно: фразу «Hi there, Flash!». Впоследствии мне может потребоваться эта информация, поэтому дай ей метку `message`. Если когда-нибудь я попрошу у тебя `message`, возврати мне текст «Hi there, Flash!»

Возможно, это не так дружественно, как сказать «Привет!», но при этом иллюстрируется одна из подлинных основ программирования: Flash может запомнить для вас нечто при условии, что оно будет помечено так, чтобы потом его можно было найти. Например, во второй строке кода мы хотели, чтобы Flash запомнил ваше имя, и пометили ссылку на него словом `firstName`. Flash запомнил ваше имя и вывел его в окне Output, когда вы тестировали свой фильм.

Способность Flash кое-что запоминать является для нас решающим обстоятельством в программировании на нем. Flash может запоминать данные любого типа, в том числе текст (например, ваше имя), числа (например, 3.14159) и более сложные типы данных, о которых мы расскажем позже.

Официальная номенклатура переменных

Пора дать несколько формальных терминов для описания того, как Flash осуществляет запоминание. Вы уже знаете, что Flash запоминает данные. Отдельный фрагмент данных называется *элементом данных* (*datum*). Элемент данных (например, «Hi there, Flash!») и идентифицирующая его метка (например, `message`) вместе называются *переменной* (*variable*). Метка переменной называется ее *именем* (*name*), а элемент данных переменной называется ее *значением* (*value*). Мы говорим, что переменная *хранит* (*stores*) или *содержит* (*contains*) свое

значение. Обратите внимание, что «Hi there, Flash!» заключено в кавычки для обозначения того, что это *строка* текста, а не число или другой *тип данных*.

В первой строке кода вы задали значение переменной `message`. Действие, связанное с приданием переменной значения, называется *присваиванием переменной значения* или просто *присваиванием*. Но прежде чем присвоить переменной значение, вы должны создать ее. Формально существование переменной начинается с ее *объявления* с помощью особого ключевого слова *var*, которое вы использовали ранее.

Поэтому, для того чтобы проинструктировать вас, как создать первую строку кода, можно было воспользоваться более формальным языком: объявить новую переменную с именем `message` и присвоить ей начальное значение «Hi there, Flash!». Тогда вы должны были бы написать:

```
var message = "Hi there, Flash!";
```

Волшебник за шторой (интерпретатор)

Вспомните первые две строки кода:

```
var message = "Hi there, Flash!";
var firstName = "здесь ваше имя";
```

В каждом из этих предложений вы создавали переменную и присваивали ей значение. Однако третья и четвертая строки несколько отличаются от них:

```
trace (message);
trace ("Hi there, " + firstName + ", nice to meet you.");
```

Эти предложения используют команду *trace( )*. Вы уже видели действие этой команды – она заставила Flash вывести ваш текст в окне Output. В третьей строке Flash вывел значение переменной `message`. В последней строке Flash также преобразовал переменную `firstName` в ее значение (то, которое вы ввели) и вставил его в предложение после слов «Hi there.». После этого команда *trace( )* заставила указанные данные появиться в окне Output (что удобно для слежения за происходящим во время выполнения программы).

Возникает вопрос: что заставило команду *trace( )* поместить ваш текст в окно Output? Когда вы создаете переменную или подаете команду, то фактически обращаетесь к *интерпретатору ActionScript*, который запускает ваши программы, управляет кодом, ждет инструкций, выполняет все команды ActionScript и выполняет предложения, запоминает ваши данные, посылает вам информацию, вычисляет значения и даже запускает основную среду программирования, когда фильм загружается в проигрыватель Flash.

Интерпретатор транслирует ваш ActionScript в тот язык, который понятен компьютеру и используется им для выполнения вашего кода. Во время воспроизведения фильма интерпретатор всегда активен и готов попытаться понять команды, которые вы ему подаете. Если интерпретатору понятны ваши команды, он посылает их на выполнение процессору компьютера. Если команда генерирует результат, интерпретатор передает вам соответствующий отклик. Когда интерпретатору не понятна команда, он посылает сообщение об ошибке. Интерпретатор, таким образом, действует как коммутатор для ActionScript: это аудитория, к которой вы обращаетесь в своем коде, и посредник, который возвращает вам сообщения из Flash.

Рассмотрим подробнее работу интерпретатора на примере обработки простого действия *trace( )*.

Рассмотрим следующую команду, как это сделал бы интерпретатор:

```
trace ("Nice night to learn ActionScript.");
```

Интерпретатор сразу распознает ключевое слово *trace* по специальному списку допустимых имен команд. Интерпретатору известно также, что *trace( )* используется для отображения текста в окне Output, поэтому он ждет указаний о том, какой текст выводить. Дальше он находит текст «Nice night to learn ActionScript.» между круглыми скобками, следующими за словом *trace*, и думает: «Ага! Это как раз то, что мне нужно. Я должен немедленно послать это в окно Output!».

Заметьте, что команда заканчивается точкой с запятой (;). Точка с запятой действует как точка в конце предложения. За редким исключением все предложения ActionScript должны заканчиваться точкой с запятой. Если предложение понято и есть вся необходимая информация, интерпретатор транслирует команду, чтобы ее мог выполнить процессор, в результате чего наш текст появляется в окне Output.

Это сильно упрощенное описание того, как работают процессор компьютера и интерпретатор, но оно иллюстрирует следующие пункты:

- Интерпретатор всегда ждет ваши инструкции.
- Интерпретатор должен прочесть ваш код буква за буквой и постараться понять его. Это похоже на то, как человек старается прочесть и понять предложение в книге.
- Интерпретатор читает ваш ActionScript, применяя строгие правила: если бы, к примеру, отсутствовали круглые скобки в предложении *trace( )*, то интерпретатор не смог бы разобраться в том, что происходит, и команда не была бы выполнена.

Вы еще только познакомились с интерпретатором, но вскоре вы станете с ним близки, как любовники: начнутся ссоры и крики «почему ты меня не слушаешь?!», и чудесные мгновения, когда вы понимаете друг друга в совершенстве. Мой отец всегда говорил мне, что лучший способ выучить новый язык – это найти возлюбленную, которая на нем

разговаривает. Позвольте мне стать первым, кто пожелает вам счастья в новых отношениях между вами и интерпретатором ActionScript. С этого момента, описывая выполнение инструкций ActionScript, я буду обычно говорить «интерпретатор», а не «Flash».

Дополнительные нужные данные (аргументы)

Вы уже сталкивались с передачей интерпретатору текста, который нужно отобразить, при подаче команды *trace( )*. Это обычный подход: мы будем часто подавать команду и снабжать интерпретатор дополнительными данными, используемыми при выполнении этой команды. Для данных, передаваемых команде, есть специальное название: *аргумент*, или, что то же самое, *параметр*. Чтобы послать команде аргумент, его нужно заключить в круглые скобки:

```
command(argument);
```

Чтобы снабдить команду несколькими аргументами, их следует разделить запятыми:

```
command(argument1, argument2, argument3);
```

Посылка команде аргументов называется *передачей* аргументов. Например, в коде `gotoAndPlay(5)` именем команды является *gotoAndPlay*, а 5 является передаваемым аргументом (в данном случае, номером кадра). Некоторые команды, например *stop( )*, требуют после себя скобок, но не принимают аргументы. Почему так происходит, мы узнаем в главе 9 «Функции».

Связующие элементы ActionScript (операторы)

Взглянем еще раз на четвертую строку кода, содержащую предложение *trace( )*:

```
trace ("Hi there, " + firstName + ", nice to meet you.");
```

Видите знаки плюс (+)? Они используются для соединения (*конкатенации*) нашего текста и являются одним из многих имеющихся *операторов*. Операторы в языке программирования сродни союзам («и», «или», «но» и т. д.) в естественных языках. Это средства, используемые для соединения фраз кода и манипуляций над ними. В примере *trace( )* оператор «плюс» присоединяет заключенный в кавычки текст «Hi there, » к тексту, содержащемуся в переменной `firstName`.

Все операторы связывают фразы кода вместе, при этом обрабатывая их. Фразы могут быть текстом, числами или иметь другой тип данных, и оператор, как правило, производит некоторое преобразование. Очень часто операторы соединяют вместе два элемента, как это делает оператор «плюс». А другие операторы сравнивают значения, присваи-

вают значения, участвуют в принятии логических решений, определяют типы данных, создают новые объекты и предоставляют различные другие полезные услуги.

При использовании с числовыми операндами знаки плюс (+) и минус (–) осуществляют элементарные арифметические действия. Следующий код выводит в окне Output число 3:

```
trace(5 - 2);
```

Оператор *меньше чем* проверяет, какое из двух чисел является меньшим или какая из двух букв раньше встречается в алфавите:

```
if (3 < 300) {
  // Сделать что-то...
}

if ("a" < "z") {
  // Сделать что-то еще...
}
```

Сочетания, сравнения, присваивание и прочие манипуляции, совершаемые операторами, называются *операциями*. Арифметические операции понять легче всего, потому что они выполняют элементарные математические действия: сложение (+), вычитание (–), умножение (*) и деление (/). Но некоторые операторы покажутся вам не столь понятными, так как они выполняют специальные задачи программирования. Возьмем, к примеру, оператор *typeof*. Он сообщает о том, какого типа данные хранятся в переменной. Поэтому, если создать переменную x и присвоить ей значение 4, можно спросить у интерпретатора, какой тип данных содержится в x:

```
var x = 4;
trace (typeof x);
```

Когда эта строка кода будет выполнена во Flash, в окне Output будет получено слово «number». Обратите внимание, что оператору *typeof* передается значение, над которым он должен действовать, но при этом не используются скобки: `typeof x`. Поэтому может показаться непонятным, является ли x *аргументом typeof*. На самом деле, x играет ту же роль, что и аргумент (это вспомогательные данные, необходимые при выполнении фразы кода), но в контексте оператора x, так похожий на аргумент, принято называть *операндом*. Операнд – это то, над чем действует оператор. Например, в выражении 4 + 9 числа 4 и 9 являются операндами оператора +.

В главе 5 «Операторы» подробно рассказано обо всех операторах ActionScript. Сейчас нужно просто запомнить, что операторы связывают фразы кода в некоторых преобразованиях.

Соединяем все вместе

Повторим теперь все изученное. Вот снова первая строка:

```
var message = "Hi there, Flash!";
```

Ключевое слово *var* сообщает интерпретатору, что мы объявляем (создаем) новую переменную. Слово `message` является именем нашей переменной. Знак равенства является оператором, присваивающим текстовую строку («Hi there, Flash!») переменной `message`. Текст «Hi there, Flash!» становится в результате значением переменной `message`. Наконец, точка с запятой (`;`) сообщает интерпретатору, что мы завершили свое первое предложение.

Вторая строка почти такая же, как первая:

```
var firstName = "здесь ваше имя";
```

Здесь мы присваиваем текстовую строку, набранную вместо *здесь ваше имя*, переменной `firstName`. Точка с запятой завершает наше второе предложение.

Затем в третьей и четвертой строках мы используем переменные `message` и `firstName`:

```
trace (message);
trace ("Hi there, " + firstName + ", nice to meet you.");
```

Ключевое слово *trace* сообщает интерпретатору, что нужно вывести в окне Output некоторый текст. Текст, который нужно вывести, передается как аргумент. Открывающая круглая скобка обозначает начало аргумента. В четвертой строке в сам аргумент входят две *операции*, каждая из которых использует *оператор* «плюс». Первая операция присоединяет свой первый *операнд*, `"Hi there, "`, к значению второго операнда, `firstName`. Вторая операция присоединяет `", nice to meet you."` к результату первой. Закрывающая круглая скобка обозначает конец аргумента, а точка с запятой снова указывает на конец предложения.

Вот и готова ваша первая программа ActionScript! Это хорошее предзнаменование и важная веха.

Другие понятия ActionScript

Вы уже познакомились со многими базовыми элементами ActionScript: данными, переменными, операторами, предложениями, функциями и аргументами. Прежде чем глубже погрузиться в эти темы, обрисуем остальные главные возможности ActionScript.

Программы Flash

Для большинства пользователей компьютеров слово *программа* (*program*) является синонимом *приложения* (*application*), например Adobe Photoshop или Macromedia Dreamweaver. Очевидно, это не то, что мы создаем, программируя на Flash. С другой стороны, программисты определяют программу как совокупность строк кода («последовательность предложений»), но это будет лишь частью того, что мы строим.

Фильм Flash – это больше, чем последовательность строк кода. Во Flash код перемежается элементами фильма, такими как кадры и кнопки. Мы прикрепляем свой код к этим элементам, чтобы он мог с ними взаимодействовать.

В конечном счете, не существует такой вещи, как «программа» Flash в классическом смысле этого слова. На ActionScript мы пишем не законченные программы, а *сценарии* (*scripts*) – участки кода, которые задают программируемое поведение нашего фильма, так же как сценарии JavaScript задают программируемое поведение документов HTML. То, что мы создаем, является не программой, а законченным фильмом (содержащим код, временную диаграмму, зрительные и звуковые элементы и прочие возможности).

Наши сценарии содержат большую часть того, что можно увидеть в обычных программах, но без всех этих вещей, что пишут на языках типа C++ или Java на уровне операционной системы, чтобы вывести на экран графику или воспроизвести звуки. Мы избавились от необходимости заниматься техническими деталями программирования графики и звука, что позволяет сосредоточиться на разработке поведения наших фильмов.

Выражения

Предложения сценария, как мы выяснили, содержат инструкции сценария. Но инструкции обычно бесполезны при отсутствии данных. Когда мы, например, задаем переменную, то присваиваем некоторые данные в качестве ее значения. Когда мы используем команду *trace( )*, то передаем данные в качестве аргумента для вывода в окне Output. Данные являются содержимым, которое мы обрабатываем в своем коде ActionScript. В своих сценариях вы будете извлекать, передавать, запоминать и вообще раскидывать всюду массу данных.

В программе любая фраза кода, выдающая во время выполнения одиночное значение, называется *выражением*. Число 7 и строка «Добро пожаловать на мой сайт» являются простейшими выражениями. Они представляют простые данные, которые будут использованы в неизменном виде при выполнении программы. Поэтому такие выражения называются *символьными выражениями*, или *литералами*.

Литералы – лишь один из типов выражений. Переменная тоже может быть выражением (переменные считаются выражениями, поскольку замещаются данными). Выражения становятся еще более интересными, когда объединяются между собой с помощью операторов. Например, выражение 4 + 5 имеет два операнда, 4 и 5, но из-за оператора сложения все выражение принимает одиночное значение 9. Сложные выражения сами могут содержать другие, более короткие, если кодовая конструкция в целом может быть преобразована в одиночное значение.

Ниже показана переменная `message`:

```
var message = "Hi there, Flash!";
```

При желании можно объединить переменное выражение `message` с символьным выражением « How are you?» следующим образом:

```
message + " How are you?"
```

что во время выполнения программы превратится в «Hi there, Flash! How are you?». При работе с арифметикой вы заметите, что длинные выражения часто включают в себя более короткие, например:

```
(2 + 3) * (4 / 2.5) - 1
```

Начиная программировать, важно разобраться с выражениями, потому что термин «выражение» часто используется при описании понятий программирования. Например, можно было бы написать: «Чтобы присвоить значение переменной, введите ее имя, затем знак равенства и произвольное выражение».

Два важных типа предложений: условные и циклические

Почти во всех программах используются *условные предложения* (*conditionals*) для внесения логики в программы и *циклы* (*loops*) для выполнения повторяющихся задач.

Осуществление выбора с помощью условных предложений

Одной из действительно стоящих черт программирования Flash является возможность сделать свои фильмы умными. Что я имею в виду? Представьте себе, что есть девушка по имени Уэнди, которая не любит, чтобы ее одежда промокала под дождем. Каждое утро, перед тем как выйти из дома, Уэнди смотрит в окно и проверяет, какая на улице погода. Если идет дождь, она берет с собой зонтик. Уэнди умная. Она использует элементарную логику – способность видеть ряд вариантов и принимать решение в зависимости от обстоятельств. Такую же эле-

ментарную логику мы используем при создании интерактивных фильмов Flash.

Вот несколько примеров логики в фильмах Flash:

- Пусть в нашем фильме есть три части. Когда пользователь переходит в одну из частей, мы используем логику, чтобы решить, показывать ли ему введение для этой части. Если он уже был в этой части, мы пропускаем введение, в противном случае показываем его.
- Предположим, что доступ к части фильма ограничен. Для входа в зону с ограниченным доступом пользователь должен ввести пароль. Если пароль правильный, мы показываем содержимое зоны с ограниченным доступом, если неправильный – нет.
- Допустим, что мы перемещаем шарик по экрану и хотим, чтобы он отскакивал от стены. Если шарик проходит некоторую точку, мы меняем направление его движения. В противном случае позволяем шарику продолжить движение в том же направлении.

Такие примеры логики в фильме требуют использования особого типа предложений, называемых *условными предложениями*. Условные предложения позволяют задать условия, при которых некоторый раздел кода должен (или не должен) выполняться. Вот пример условного предложения:

```
if (userName == "James Bond") {
  trace ("Добро пожаловать, 007.");
}
```

Общая структура условного предложения такова:

```
if (это условие выполняется) {
  тогда выполнить эти строки кода
}
```

Подробнее о синтаксисе вы узнаете из главы 7 «Условные предложения». А пока запомните, что условные предложения позволяют Flash принимать логические решения.

Выполнение повторяющихся задач с помощью циклов

Мы хотим, чтобы наши фильмы не только принимали решения, но и делали за нас утомительные повторяющиеся задачи. Т. е. пока они не захватили весь мир и не поработили нас, начав выращивать в маленьких энергетических стручках, как … Стоп… забудьте то, что я сейчас сказал… Допустим, вы хотите вывести в окне Output последовательность из пяти чисел, начинающуюся с определенного числа. Если начальное число 10, можно вывести последовательность так:

```
trace (10);
trace (11);
trace (12);
```

```
trace (13);
trace (14);
```

Но если нужно начать последовательность с 513, то придется заново ввести все числа:

```
trace (513);
trace (514);
trace (515);
trace (516);
trace (517);
```

Можно избежать перепечатывания, сделав команды *trace( )* зависимыми от переменной, например:

```
var x = 1;
trace (x);
x = x + 1;
trace (x);
x = x + 1;
trace (x);
x = x + 1;
trace (x);
x = x + 1;
trace (x);
```

В первой строке мы присваиваем переменной x значение 1. Затем во второй строке посылаем это значение в окно Output. В строке 3 мы говорим: «Возьми текущее значение x, добавь к нему 1 и помести результат обратно в переменную x», поэтому x становится равным 2. Затем мы снова посылаем значение x в окно Output. Этот процесс повторяется еще три раза. В результате в окно Output оказывается выведена последовательность из пяти чисел. Прелесть в том, что если теперь нужно изменить начальное число последовательности, достаточно лишь изменить начальное значение x. Поскольку весь последующий код использует x, при выполнении программы меняется вся последовательность.

Это усовершенствование нашего первоначального подхода работает достаточно успешно, когда нужно вывести лишь пять чисел, но оказывается непригодным, если чисел становится 500. Для выполнения многократно повторяющихся задач используется *цикл* – предложение, заставляющее блок кода повторяться произвольное количество раз. Существует несколько типов циклов, каждый из которых имеет собственный синтаксис. Один из самых распространенных – цикл типа *while*. Вот как может выглядеть наш пример с использованием цикла *while* вместо ряда повторяющихся предложений:

```
var x = 1;
while (x <= 5) {
  trace (x);
```

```
  x = x + 1;
}
```

Ключевое слово *while* указывает на то, что мы хотим начать цикл. Выражение (*x <= 5*) определяет, сколько раз должен исполниться цикл (пока `x` остается меньше или равным 5), а предложения *trace (x);* и *x = x + 1;* выполняются при каждом повторении (или *итерации*) цикла. В данном случае цикл экономит нам только пять строк кода, но он мог бы сэкономить сотни строк, если бы мы считали до чисел более высокого порядка. И кроме того, наш цикл гибок. Чтобы сосчитать до 500, нужно лишь изменить выражение (*x <= 5*) на (*x <= 500*):

```
var x = 1;
while (x <= 500) {
  trace (x);
  x = x + 1;
}
```

Как и условные предложения, циклы являются одним из наиболее часто используемых и важных типов предложений в программировании.

Модульный код (функции)

До сих пор ваш самый длинный сценарий состоял из четырех строчек кода. Но пройдет немного времени, и эти 4 строки превратятся в 400, а то и 4000 строк. Рано или поздно вы станете искать способы, как управлять кодом, сократить объем работы и облегчить применение вашего кода в различных сценариях. Тогда вы впервые действительно полюбите *функции*. Функция – это объединенная в один пакет последовательность предложений. На практике функции обычно являются многократно используемыми блоками кода.

Предположим, что вы хотите написать сценарий, который быстро подсчитывает площадь прямоугольника. Без использования функций ваш сценарий мог бы выглядеть так:

```
var height = 10;
var width = 15;
var area = height * width;
```

А теперь допустим, что вам нужно вычислить площади пяти прямоугольников. Ваш код увеличивается в размерах впятеро:

```
var height1 = 10;
var width1 = 15;
var area1 = height1 * width1;
var height2 = 11;
var width2 = 16;
var area2 = height2 * width2;
var height3 = 12;
var width3 = 17;
```

```
var area3 = height3 * width3;
var height4 = 13;
var width4 = 18;
var area4 = height4 * width4;
var height5 = 20;
var width5 = 5;
var area5 = height5 * width5;
```

Так как мы многократно повторяем расчет площади, лучше один раз поместить его в функцию и затем выполнять ее много раз:

```
function area(height, width){
  return height * width;
}
area1 = area(10, 15);
area2 = area(11, 16);
area3 = area(12, 17);
area4 = area(13, 18);
area5 = area(20, 5);
```

Сначала мы создали функцию расчета площади с помощью предложения *function*, которое определяет (объявляет) функцию, подобно тому как слово *var* определяет переменную. Затем мы дали нашей функции имя `area`, подобно тому как мы даем имена переменным. Между круглыми скобками мы перечислили аргументы, которые получает наша функция при каждом своем использовании: `height` и `width`. А в фигурные скобки ({ }) заключены команды, которые должна выполнять наша функция:

```
return height * width;
```

Создав функцию, мы можем выполнять код, который она содержит, из любого места нашего фильма, указывая там ее имя. В нашем примере мы вызвали функцию *area( )* пять раз, каждый раз передавая ей значения `height` и `width`, которые она должна получить: *area(10, 15)*, *area(11, 16)* и т. д. Результаты всех вычислений возвращаются нам, и мы записываем их в переменные от `area1` до `area5`. Мило и точно, и здесь значительно меньше работы, чем в бесфункциональной версии кода.

Не волнуйтесь, если у вас возникли вопросы в связи с этим примером функции: мы узнаем о функциях больше в главе 9. Пока просто запомните, что функции дают нам чрезвычайно мощный способ создания сложных систем. Функции помогают повторно использовать код и упаковывать его функциональность, позволяя расширить границы того, что можно построить.

Встроенные функции

Обратите внимание, что функции принимают аргументы, подобно тому как это делает процедура *trace( )*. Вызов функции *area(4, 5);* очень

похож на подачу команды *trace( )*, например *trace (x);*. Это сходство не является просто совпадением. Как отмечалось выше, многие действия (Actions), в том числе *trace( )*, фактически являются функциями. Но это особый тип функций, которые встроены в ActionScript (в противоположность функциям, определяемым пользователем, таким как наша *area( )*). Поэтому допустимо (и технически более точно) говорить «вызвать функцию *gotoAndStop( )*», а не «выполнить Action *gotoAndStop*». Встроенная функция – это просто многократно используемый блок кода, который для удобства мы получаем вместе с ActionScript. Встроенные функции позволяют делать все – от математических вычислений до управления клипами фильмов. Все встроенные функции перечислены в части III «Справочник по языку». Мы также столкнемся со многими из них, изучая основы ActionScript.

Экземпляры клипов фильмов

Я надеюсь, что за всеми этими разговорами об основах программирования вы не забыли об основах Flash. Одним из ключевых понятий визуального программирования на Flash являются *экземпляры* (*instances*) клипов. Как проектировщик или разработчик для Flash вы должны уже быть знакомы с клипами, но, возможно, не воспринимаете их как инструменты программирования.

Для каждого клипа в библиотеке фильма Flash имеется определение символа. Можно добавлять несколько копий, или *экземпляров*, одного символа клипа в фильм Flash, перетаскивая клип из библиотеки на рабочий стол. В значительной мере развитое программирование Flash заключается просто в управлении экземпляром клипа. Например, прыгающий мячик – это лишь экземпляр клипа, положение которого на столе изменяется периодически. С помощью ActionScript можно изменять местоположение экземпляра, его размер, текущий кадр, угол поворота и т. д. во время воспроизведения нашего фильма.

Если вы не знакомы с клипами и экземплярами, то прежде чем продолжить работу с данной книгой, почитайте документацию по Flash или справочные файлы.

Модель выполнения, управляемого событиями

Последняя тема, которую мы должны рассмотреть в нашем обзоре основ ActionScript, – это модель выполнения, которая определяет, в какой момент выполняется код в фильме. Код может быть прикреплен к различным кадрам, кнопкам и клипам по всему фильму. Но когда фактически он выполняется? Чтобы ответить на этот вопрос, совершим краткую прогулку по истории компьютерных вычислений.

На заре развития программирования инструкции программы выполнялись последовательно в порядке своего появления, начиная с первой строки и заканчивая последней. Программа должна была совер-

шить какие-то действия и затем остановиться. Такого рода программы, называемые *пакетными* (*batch*), не могут обеспечить интерактивность, необходимую для программной среды, *управляемой событиями* (*event-based*), такой как Flash.

Программы, управляемые событиями, не выполняются линейным образом, как пакетные программы. Они выполняются непрерывно (в *цикле событий, event loop*), ожидая, когда что-нибудь произойдет (*событие, event*), и выполняя фрагменты кода в ответ на эти события. В языках, предназначенных для использования в интерактивной визуальной среде (например, ActionScript или JavaScript), событиями обычно являются действия пользователя, такие как щелчок мышью или нажатие клавиши.

Когда происходит событие, например, щелчок мышью, интерпретатор подает сигнал тревоги. Программа может реагировать на этот сигнал тем, что попросит интерпретатор выполнить некоторый участок кода. Например, если пользователь щелкает мышью в фильме, можно выполнить какой-нибудь код, который покажет другую часть фильма (классическая навигация) или передаст переменные в базу данных (классическая подача формы).

Но программы будут реагировать на события только тогда, когда мы создадим *обработчики событий* (*event handlers*). Вот некий псевдокод, который в общем виде показывает, как задается обработчик события:

```
когда (произойдет это событие) {
  выполнить эти строки кода
}
```

Обычно это записывается в общей форме:

```
on (событие) {
  предложения
}
```

На практике обработчик события для кнопки, перемещающей воспроизводящую головку на кадр 200, будет выглядеть так:

```
on (press) {
  gotoAndStop(200);
}
```

Поскольку управляемые событиями программы всегда выполняют цикл событий в готовности реагировать на очередное событие, они подобны живым системам. События составляют решающую часть фильмов Flash. Без событий наши сценарии ничего бы не выполняли, за исключением одного: Flash выполняет любой код в кадре, когда воспроизводящая головка входит в этот кадр. Неявное событие состоит во